28

STUDIES IN LOGIC,
GRAMMAR AND RHETORIC

UNDER THE AUSPICES
OF THE POLISH ASSOCIATION FOR LOGIC AND PHILOSOPHY OF SCIENCE

From Insight to Proof

Festschrift in Honour of Andrzej Trybulec

Edited by ROMAN MATUSZEWSKI
ANNA ZALEWSKA

10 (23) UNIVERSITY OF BIAŁYSTOK 2007

FROM INSIGHT TO PROOF

Series: STUDIES IN LOGIC, GRAMMAR AND RHETORIC 10(23)

Under the Auspices of the Polish Association
for Logic and Philosophy of Science

FROM INSIGHT TO PROOF

Festschrift in Honour of Andrzej Trybulec

Guest Editors Roman Matuszewski
Anna Zalewska

University of Białystok
Białystok 2007

Guest Editors

Roman Matuszewski
University of Białystok
Białystok, Poland
http://mizar.org/people/romat/

Anna Zalewska
University of Białystok
Białystok, Poland
e-mail: zalewska@uwb.edu.pl

Series: STUDIES IN LOGIC, GRAMMAR AND RHETORIC 10(23)
http://logika.uwb.edu.pl/studies/

Series Editor: Halina Święczkowska
University of Białystok, Faculty of Law, Section of Semiotics

in collaboration with Kazimierz Trzęsicki
University of Białystok, Faculty of Mathematics and Physics
Chair of Logic, Informatics and Philosophy of Science
e-mail: logika@uwb.edu.pl

Refereed by Kazimierz Trzęsicki
Cover design: Krzysztof Tur
Typesetting: Camera-ready by authors.

ISBN 978-83-7431-128-1
ISSN 0860-150X

WYDAWNICTWO UNIWERSYTETU W BIAŁYMSTOKU
15-097 Białystok, ul. Marii Skłodowskiej-Curie 14, tel. +48-85 745 70 59
http://wydawnictwo.uwb.edu.pl e-mail: ac-dw@uwb.edu.pl

Printed in Poland by: MZGraf
http://www.mzgraf.pl tel. +48-86 275 41 31

This volume is dedicated to Andrzej Trybulec on the occasion of his 65th birthday

Preface

This volume is dedicated to Andrzej Trybulec on the occasion of his 65th birthday

Andrzej Trybulec created the Mizar language and has been the leader of the Mizar Project since its conception. He obtained his PhD in mathematics (topology) but has also accomplished significant results in algebraic linguistics. He has continuously pursued his passions in mathematics and linguistics since 1973, when he started researching computer-oriented formalization of mathematics. One of the goals of this research has been a better understanding of the process of mathematical creation – from insight to proof.

Andrzej Trybulec is a pioneer in building a computer-checked database of mathematical knowledge and a visionary in the application of automated reasoning to real mathematical practice. For over 30 years, his results have been essential to world-wide research in this field. He has greatly advanced our understanding of rigorous formalized mathematics. The impact of his work is visible in several articles in this volume.

The 65th birthday of Andrzej Trybulec in 2006 prompted his students and colleagues to reflect on his achievements by publishing this volume. In doing so, we recognize Andrzej as an influential scientist and at the same time express our appreciation for his stimulating personality. This Festschrift consists of 21 articles covering three thematic categories:

- Logic,
- The Language of Mathematics,
- Computerized Mathematical Knowledge.

In the Table of Contents, however, we have not conducted a formal division – within the articles, themes, as always, intermingle with one another. Some researchers who could not contribute to this volume have forwarded congratulations to Andrzej. Their names are included in "Tabula Gratulatoria".

The editors would like to thank the following people for their substantial help in preparing this publication: Grzegorz Bancerek, Pauline N. Kawamoto, Adam Naumowicz, Piotr Rudnicki, Jarosław Sokołowski, Kazimierz Trzęsicki, Josef Urban, Freek Wiedijk.

March 2007

Roman Matuszewski
Anna Zalewska
The Editors

Tabula Gratulatoria

Robert S. Boyer, Austin

Czesław Byliński, Białystok

Andrzej Grzegorczyk, Warsaw

Witold Marciszewski, Warsaw

Yuri Matiyasevich, St. Petersburg

Bengt Nordström, Göteborg

Lawrence C. Paulson, Cambridge

Piotr Rudnicki, Edmonton

Dana S. Scott, Berkeley

Josef Urban, Prague

Table of Contents

Chiron: A Multi-Paradigm Logic

William M. Farmer

McMaster University
Hamilton, Ontario, Canada
wmfarmer@mcmaster.ca

Abstract. Chiron is a derivative of von-Neumann-Bernays-Gödel (NBG) set theory that is intended to be a practical, general-purpose logic for mechanizing mathematics. It supports several reasoning paradigms by integrating NBG set theory with elements of type theory, a scheme for handling undefinedness, and a facility for reasoning about the syntax of expressions. This paper gives a quick, informal presentation of the syntax and semantics of Chiron and then discusses some of the benefits Chiron provides as a multi-paradigm logic.

1 Introduction

One of the great challenges of the information age is to mechanize mathematics. Mechanizing mathematics does not mean replacing mathematicians with machines. It means building software systems to help mathematics practitioners—engineers, scientists, mathematicians, etc.—to do mathematics. The principal objective of *mechanized mathematics* is to develop widely accessible *mechanized mathematics systems* (MMSs) that support various parts of the mathematics process. Contemporary MMSs include computer theorem proving systems such as Coq [5], Isabelle [23], and PVS [22], computer algebra systems such as Axiom [18], Maple [3], and Mathematica [27].

A key component of an MMS is its underlying logic. By a *logic*, we mean a language (or a family of languages) that has a formal syntax and a precise semantics with a notion of logical consequence.[1] (A logic may also have a proof system, but it is not a required constituent.) By this definition, a theory in a logic—such as Zermelo-Fraenkel (ZF) set theory or von-Neumann-Bernays-Gödel (NBG) set theory in first-order logic—is itself a logic. Of course, a predicate logic—such as first-order logic or simple type theory (classical higher-order logic) [10]—is also a logic.

A practical, general-purpose MMS needs a well-designed logic that is both theoretically expressive and practically expressive. The *theoretical expressivity* of a logic is the measure of what ideas can be expressed in the logic without regard to how the ideas are expressed. The *practical expressivity* of a logic is the measure of how readily ideas can be expressed in the logic. Traditional "off-the-shelf" logics,

[1] The underlying logics of contemporary computer algebra systems have a formal syntax but usually lack a precise semantics.

ISBN 978-83-7431-128-1 ISSN 0860–150X

like those we mentioned in the previous paragraph, lack the practical expressivity that it needed for an MMS. This is because they are designed to be used *in theory*, not *in practice*.

For example, ZF set theory has very high theoretical expressivity. Indeed most mathematicians believe that essentially all of mathematics can be formalized in ZF—at least in theory. However, ZF does not contain an operator for forming a term that denotes the application of a set f representing a function to a set a that represents an argument to f. Moreover, even if such an application operator were added to ZF, there is no special mechanism for handling "undefined" applications. As a result, statements involving functions and undefinedness are much more verbose and indirect when expressed in ZF than they need to be, and reasoning about functions and undefinedness is usually performed in the metalogic of ZF instead of in ZF itself. (For examples of this kind and other kinds of reasoning that, for the sake of convenience, is usually done in the metalogic of ZF, see K. Kunen's *Set Theory* [19].)

Chiron is a derivative of NBG set theory that is intended to be a practical, general-purpose logic for mechanizing mathematics. It is designed to have both high theoretical and high practical expressivity. As a derivative of NBG, it has the same theoretical expressivity as the ZF and NBG set theories. However, Chiron has a much higher level of practical expressivity than traditional logics. It achieves this in large part by integrating several reasoning paradigms.

A *reasoning paradigm* is a style of reasoning analogous to a *programming paradigm* [25], a style of computer programming. A reasoning paradigm includes such things as background assumptions about semantics, favored modes of expression, and sanctioned reasoning rules. Just as programming languages are designed to support certain programming paradigms, logics are designed to support certain reasoning paradigms. Here is a short list of some reasoning paradigms that are especially relevant to mechanized mathematics:

1. *Classical.* The classical paradigm assumes that formulas have only two possible truth values: true and false. Ideas are expressed using the usual machinery of predicate logic: terms that denote values, predicates applied to terms, the standard propositional connectives, and the existential and universal quantifiers. Reasoning is performed using the standard laws of predicate logic such as modus ponens and universal instantiation including nonconstructive principles such as the law of excluded middle. The classical paradigm is dominant in mathematical practice. Most traditional logics—including first-order logic, simple type theory, ZF, and NBG—support it.
2. *Constructive.* On the surface, the constructive paradigm is very similar to the classical paradigm. However, the underlying semantics is quite different and only constructive principles of reasoning are permitted. The constructive paradigm is not widely followed in mathematical practice. However, the underlying logics of several computer theorem proving systems support the constructive paradigm. For example, the logic of Coq, the Calculus of Inductive Constructions [2, 6], supports this paradigm.
3. *Assured Definedness.* The assured definedness paradigm is based on the assumption that every term is defined and thus denotes some value. As a result, definedness checking is unnecessary and all functions are considered to be total. Nearly all logics used for mechanized mathematics support the assured definedness paradigm including first-order logic, simple type theory, ZF, and NBG. However, this paradigm is not commonly employed in mathematical practice.
4. *Permitted Undefinedness.* The converse of assured definedness, the permitted undefinedness paradigm allows terms to be undefined and thus to be nondenoting. This paradigm is very widely used in mathematical practice. Although traditional logics do not support this reasoning paradigm, we have shown that, if a traditional logic is modified slightly, it can support this paradigm [8, 9, 11, 13].
5. *Set theory.* The set theory paradigm reduces all mathematical reasoning to reasoning about sets. It is deeply entrenched in mathematical practice. ZF and NBG are two of the most popular set-theoretic logics that support this paradigm. They are both based on the same intuitive model of the iterated hierarchy of sets. The key difference between them is that ZF does not admit proper classes, while NBG does. (Proper classes are collections of sets that are too large to be sets themselves. They include useful entities like the class of all sets and the cardinality function.)
6. *Type theory.* The type theory paradigm uses a hierarchy of *types* to classify expressions and their values. Most type theories provide strong support for reasoning with functions. Type theories—both classical and constructive—are popular logics for mechanized mathematics systems. In particular, several leading computer theorem proving systems [1, 14, 17, 20, 22, 23] are based Church's type theory [4], a version of simple type theory which employs lambda-notation. Type theories are seldom used in mathematical practice outside of the computing field.
7. *Formalized Syntax.* The formalized syntax paradigm integrates reasoning about the syntax of expressions with reasoning about what the expressions mean. It is usually used in the metalogic of a logic rather than in the logic itself. For example, sets of axioms and rules of inference are often expressed using syntactic templates called formula schemas. They are formula-like expressions containing syntactic variables that represent collections of formulas with similar syntactic structure, but they usually are not official formulas of the logic. An example is the well-known axiom schema for the induction principle for the natural numbers:

$$(A[x \mapsto 0] \wedge (\forall x \,.\, A \supset A[x \mapsto S(x)])) \supset \forall x \,.\, A$$

where the syntactic variable A ranges over formulas. The formalized syntax paradigm can be used directly in a logic with natural number arithmetic using Gödel's *arithmetization of syntax* via Gödel numbering [15].

Mathematical practice is very rich and varied. There are usually many ways to attack a mathematical problem and many ways to express a solution. We maintain that a logic for mechanized mathematics can benefit from being able to support several reasoning paradigms. Chiron supports in an integrated manner the following five reasoning paradigms:

1. Classical.
2. Permitted undefinedness.
3. Set theory.
4. Type theory.
5. Formalized syntax.

The design of Chiron is novel, but the lion's share of the ideas behind the design come from traditional predicate logic, set theory, and type theory.

This paper gives a quick, informal presentation of the syntax and semantics of Chiron and then discusses some of the benefits Chiron provides as a multi-paradigm logic. The syntax and semantics of Chiron are presented together in section 2. A compact notation for Chiron is given in section 3. Section 4 discusses Chiron's support for its five reasoning paradigms. The paper ends with some concluding remarks in section 5.

2 Syntax and Semantics

We will present the syntax and semantics of Chiron together. The presentation of the semantics will be informal. The reader can find separate formal presentations of the syntax and semantics in [12].

2.1 Values

The official semantics of Chiron is based on *standard models.* (Two alternate semantics based on other kinds of models are given in [12].) A standard model is an elaboration of a model of NBG set theory. The basic values or elements in a model of NBG are classes (which include sets and proper classes). The values of a standard model include other values besides classes, but classes are the most important. We will not give a precise definition of a standard model; the reader can find a proper definition of this notion in [12]. We will instead present the semantics of Chiron with respect to an arbitrary standard model M. As we move along, we will reveal the ingredients of M as needed.

M includes *values* of several kinds: sets, classes, superclasses, truth values, the undefined value, and operations. M is derived from a structure, consisting of a nonempty domain D_{c} of classes and a membership relation $\in$ on D_{c}, that satisfies the axioms of NBG set theory as given, for example, in [16] or [21].

A *class* of M is a member of D_{c}. A *set* of M is a member x of D_{c} such that $x \in y$ for some member y of D_{c}. That is, a set is a class that is itself a member of a class. A class is thus a collection of sets. A class is *proper* if it is not a set. A *superclass* of M is a collection of classes in D_{c}. We consider a class, as a collection of sets, to be a superclass itself. Let D_{v} be the domain of sets of M and D_{s} be the domain of superclasses of M. The following inclusions hold: $D_{\mathrm{v}} \subset D_{\mathrm{c}} \subset D_{\mathrm{s}}$. D_{v} is the universal class (the class of all sets), and D_{c} is the universal superclass (the superclass of all classes).

op	type	formula	op-app	var
type-app	dep-fun-type	fun-app	fun-abs	if
exist	def-des	indef-des	quote	eval
true	false	set	class	expr
expr-op	expr-type	expr-term	expr-formula	in
type-equal	term-equal	formula-equal	not	or

Table 1. The Key Words of Chiron.

T, F, and $\bot$ are distinct values of M not in D_{s}. T and F represent the truth values *true* and *false*, respectively. $\bot$ is the *undefined value* which serves as the value of undefined terms. For $n \geq 0$, an *n-ary operation* of M is a total mapping from $D_1 \times \cdots \times D_n$ to D_{n+1} where D_i is D_{s}, $D_{\mathrm{c}} \cup \{\bot\}$, or $\{\mathrm{T}, \mathrm{F}\}$ for each i with $1 \leq i \leq n+1$. Let D_{o} be the domain of operations of M. $D_{\mathrm{s}} \cup \{\mathrm{T}, \mathrm{F}, \bot\}$ and D_{o} are assumed to be disjoint.

2.2 Expressions

Let $\mathcal{S}$ be a fixed infinite set of symbols that includes the 30 *key words* in Table 1. The key words are used to classify expressions, identify different categories of expressions, and name the built-in operators (see below).

An *expression* of Chiron is defined inductively by:

1. Each symbol $s \in \mathcal{S}$ is an expression.
2. If $e_1, \ldots, e_n$ are expressions where $n \geq 0$, then $(e_1, \ldots, e_n)$ is an expression.

Hence, an expression is an S-expression (with commas in place of spaces) that exhibits the structure of a tree whose leaves are symbols in $\mathcal{S}$. Let $\mathcal{E}$ be the set of expressions of Chiron.

There are four special sorts of expressions: *operators*, *types*, *terms*, and *formulas.* An expression is *proper* if it is one of these special sorts of expressions, and an expression is *improper* if it is not proper. Proper expressions denote values of M, while improper expressions are nondenoting (i.e., they do not denote anything). Operators are used to construct expressions. They denote operations. Types are used to restrict the values of operators and variables and to classify terms by their values. They denote superclasses. Terms are used to describe classes. They denote classes or the undefined value $\bot$. Formulas are used to make assertions. They denote truth values.

A term is *defined* if it denotes a class and is *undefined* if it denotes $\bot$. Every term is assigned a type. Suppose a term a is assigned a type α. Then a is said to be a *term of type* α. Suppose further α denotes a superclass Σ_α. If a is defined, i.e., a denotes a class x, then x is in Σ_α. The value of a nondenoting term is the

1.	(op, true, formula)
2.	(op, false, formula)
3.	(op, set, type)
4.	(op, class, type)
5.	(op, expr, type)
6.	(op, expr-op, type)
7.	(op, expr-type, type)
8.	(op, expr-term, type)
9.	(op, expr-formula, type)
10.	(op, in, (op-app, (op, set, type)), (op-app, (op, class, type)), formula)
11.	(op, type-equal, type, type, formula)
12.	(op, term-equal, (op-app, (op, class, type)), (op-app, (op, class, type)), type, formula)
13.	(op, formula-equal, formula, formula, formula)
14.	(op, not, formula, formula)
15.	(op, or, formula, forumla, formula)

Table 2. The Built-In Operators of Chiron.

undefined value $\bot$, but the value of an nondenoting type or formula is $D_{\rm c}$ (the universal superclass) or F, respectively. That is, the values for nondenoting types, terms, and formulas are $D_{\rm c}$, $\bot$, and F, respectively.

We will use $O, P, Q, \ldots$ to denote operators, $\alpha, \beta, \gamma, \ldots$ to denote types, $a, b, c, \ldots$ to denote terms, and $A, B, C, \ldots$ to denote formulas.

2.3 Operators

A *kind* is the key word type, a type, or the key word formula. Kinds are expressions used to define the signatures of operators. If $s \in \mathcal{S}$ and $k_1, \ldots, k_{n+1}$ are kinds where $n \geq 0$, then

$$(\mathsf{op}, s, k_1, \ldots, k_{n+1})$$

is an n-ary *operator*. s is the *name* and $k_1, \ldots, k_{n+1}$ is the *signature* of the operator. The operator is a *type operator*, *term operator*, or *formula operator* if $k_{n+1} = \mathsf{type}$, k_{n+1} is a type, or $k_{n+1} = \mathsf{formula}$, respectively.

A *language* of Chiron is a set of operators that contains the 15 "built-in" operators in Table 2. Each standard model is *for* a particular language of Chiron. Let L be the language for which M is a model.

An n-ary operator $O = (\mathsf{op}, s, k_1, \ldots, k_n, k_{n+1})$ denotes an n-ary operation o in $D_{\rm o}$ from $D_1 \times \cdots \times D_n$ into D_{n+1} where $D_i = D_{\rm s}$ if $k_i = \mathsf{type}$, $D_i = D_{\rm c} \cup \{\bot\}$ if k_i is a type, and $D_i = \{\mathrm{T}, \mathrm{F}\}$ if $k_i = \mathsf{formula}$ for each i with $1 \leq i \leq n+1$ such that:

1. If $O \not\in L$, then, for all $(d_1, \ldots, d_n) \in D_1 \times \cdots \times D_n$, $o(d_1, \ldots, d_n)$ is $D_{\rm c}$, $\bot$, or F if $k_{n+1} = \mathsf{type}$, k_{n+1} is a type, or $k_{n+1} = \mathsf{formula}$, respectively.

2. If O is a built-in operator, then o is a particular operation of M specified by the definition of a standard model.

Like a function or predicate symbol in first-order logic, an operator in Chiron is not meaningful unless it is applied. Suppose $O = (\mathsf{op}, s, k_1, \ldots, k_n, k_{n+1})$ is an operator and $e_1, \ldots, e_n$ are expressions such that $k_i = \mathsf{type}$ and e_i is a type, k_i is a type and e_i is a term, or $k_i = \mathsf{formula}$ and e_i is a formula for all i with $1 \leq i \leq n$. Then

$$(\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)$$

is an expression called an *operator application* that is a type if $k_{n+1} = \mathsf{type}$, a term of type k_{n+1} if k_{n+1} is a type, and a formula if $k_{n+1} = \mathsf{formula}$. If the value of e_i is $\bot$ or is in the value of k_i for all i for which k_i is a type, then the operator application denotes the result of applying the n-ary operation denoted by O to the n values denoted by $e_1, \ldots, e_n$. Otherwise, the operator application denotes $D_{\rm c}$, $\bot$, or F if O is a type, term, or formula operator, respectively.

Many sorts of syntactic entities can be formalized in Chiron as operators. Examples include logical connectives; individual constants, function symbols, and predicate symbols from first-order logic; base types and type constructors including dependent type constructors; and definedness operators.

Example 1. $O = (\mathsf{op}, \mathsf{class}, \mathsf{type})$ is a built-in 0-ary type operator. The operator application $(\mathsf{op\text{-}app}, O)$ is the type of classes. It denotes the universal superclass $D_{\rm c}$. Similarly, the operator application $(\mathsf{op\text{-}app}, O')$, where $O' = (\mathsf{op}, \mathsf{set}, \mathsf{type})$, is the type of sets. It denotes the universal class $D_{\rm v}$. □

Example 2. The operator

$$O = (\mathsf{op}, \mathsf{in}, (\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{set}, \mathsf{type})), (\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{class}, \mathsf{type})), \mathsf{formula})$$

is a built-in binary formula operator. If a and b are terms denoting the classes x and y, then the operator application

$$(\mathsf{op\text{-}app}, O, a, b)$$

is a formula that asserts $x \in y$. By our remark above, the value of this formula is F if x is not a set. □

Example 3. Suppose N is the type of natural numbers. Let

$$O = (\mathsf{op}, \mathsf{matrix}, \mathsf{N}, \mathsf{N}, \mathsf{type}, \mathsf{type})$$

be a ternary type operator such that, if a, b are terms of type N that denote the natural numbers m, n and α is a type, then the operator application

$$\gamma = (\mathsf{op\text{-}app}, O, a, b, \alpha)$$

denotes the type of $m \times n$ matrices of elements of type α. Since the type γ depends on the values of a and b, γ is a *dependent type* and O is a *dependent type constructor*. □

2.4 Variables

If $x \in \mathcal{S}$ and α is a type, then

$$(\mathsf{var}, x, \alpha)$$

is a term of type α called a *variable*. Variables are used mainly in conjunction with the five variable binders in Chiron: dependent function type, function abstraction, existential quantification, definite description, and indefinite description. They bind variables in the traditional, naive way. It would be possible to use other more sophisticated variable binding mechanisms such as de Bruijn notation [7] or nominal datatypes [24, 26].

As in traditional predicate logic, a variable may occur either *bound* or *free* in a proper expression. The value of a bound occurrence of a variable $(\mathsf{var}, x, \alpha)$ is restricted to classes in the superclass denoted by α. The value of a free occurrence is either a class in this same superclass or $\bot$. The latter case happens, for example, when α denotes the empty set, i.e., the empty collection of classes.

Two variables $(\mathsf{var}, x, \alpha)$ and $(\mathsf{var}, x', \alpha')$ are considered to be identical only if x and x' are identical symbols and α and α' are identical types. In particular, if α and α' are not identical types, then $(\mathsf{var}, x, \alpha)$ and $(\mathsf{var}, x, \alpha')$ are not identical variables even if α and α' denote the same superclass.

2.5 Logical Connectives

Chiron includes the usual logical connectives. The propositional connectives for negation and disjunction are represented by the built-in formula operators named not and or. Truth value constants for true and false are represented by applications of the built-in 0-ary operators named true and false. If $(\mathsf{var}, x, \alpha)$ is a variable and B is a formula, then

$$(\mathsf{exist}, (\mathsf{var}, x, \alpha), B)$$

is a formula called an *existential quantification*. It is for expressing existential assertions. There are three sorts of equality—one for each of types, terms, and formulas—represented by the built-in operators named $\mathsf{type\text{-}equal}$, $\mathsf{term\text{-}equal}$, and $\mathsf{formula\text{-}equal}$. (The universal quantifier will be introduced in the next section by a notational definition, and other propositional connectives can be introduced by operator definitions (see [12]).)

Example 4. Let O be the built-in ternary operator named $\mathsf{term\text{-}equal}$:

$$\begin{array}{ll}(\mathsf{op}, \mathsf{term\text{-}equal}, & (\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{class}, \mathsf{type})),\\ & (\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{class}, \mathsf{type})),\\ & \mathsf{type},\\ & \mathsf{formula}).\end{array}$$

(We will soon see why the operator for term equality is ternary instead of binary.) Also let a, b be terms, α be a type that denotes the superclass Σ_α, and C be the type of classes. $(\mathsf{op\text{-}app}, O, a, b, \alpha)$ asserts that a and b denote the same class in Σ_α. $(\mathsf{op\text{-}app}, O, a, b, \mathsf{C})$ asserts that a and b denote the same class. $(\mathsf{op\text{-}app}, O, a, a, \alpha)$ asserts that a is *defined in* α, that is, that a denotes a class in Σ_α. And $(\mathsf{op\text{-}app}, O, a, a, \mathsf{C})$ asserts that a is defined, that is, that a denotes some class. □

If A is a formula and b, c are terms, then

$$(\mathsf{if}, A, b, c)$$

is an if-then-else term called a *conditional term*. Its value is the value of b if A is true and is the value of c if A is false. If b and c are of the same type α, then the conditional term is also of type α; otherwise its type is the type of classes. An if-then-else term could also be constructed by an operator application, but the term's type would have to be the last member of the signature of the operator. Thus, the typing of conditional terms is more convenient than the typing of operation applications representing if-then-else terms.

Since variables denote classes, they can be called *class variables*. There are no operation, superclass, or truth value variables in Chiron. Thus direct quantification over operations, superclasses, and truth values is not possible. Direct quantification over the undefined value $\bot$ is also not possible. As in standard NBG set theory, only classes are first-class values in Chiron.

2.6 Functions

A *function* of M is a class f of M such that:

1. Each $p \in f$ is an ordered pair $\langle x, y \rangle$ where x, y are in D_{v}.
2. For all $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in f$, if $x_1 = x_2$, then $y_1 = y_2$.

A function represents a mapping from sets to sets. A function may be partial, i.e., there may not be an ordered pair $\langle x, y \rangle$ in a function for each x in D_{v}. Every function that is a set is partial, and every function that is total is a proper class. Since functions are classes, they are first-class values, unlike operations.

Let D_{f} be the domain of functions of M. For f, x in D_{c}, $f(x)$ denotes the unique y in D_{v} such that f is in D_{f} and $\langle x, y \rangle \in f$. ($f(x)$ is undefined if there is no such unique y in D_{v}.) For Σ in D_{s} and x in D_{c}, $\Sigma[x]$ denotes the class of all y in D_{v} such that, for some f in both Σ and D_{f}, $f(x) = y$.

If α is a type and a is a term, then

$$(\mathsf{type\text{-}app}, \alpha, a)$$

is a type called a *type application*. If α denotes the superclass Σ and a denotes the class x, then the type application denotes $\Sigma[x]$. If a is undefined, the type application denotes D_{c}.

If $(\mathsf{var}, x, \alpha)$ is a variable and β is a type, then

$$(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$$

is a type called a *dependent function type*. It denotes the superclass of functions $f \in D_{\mathrm{f}}$ such that:

1. For all sets z in the superclass denoted by α, if $f(z)$ is defined, then $f(z)$ is in the superclass denoted by β when the value of $(\mathsf{var}, x, \alpha)$ is z.
2. For all sets z not in the superclass denoted by α, $f(z)$ is undefined.

Dependent function types are commonly known as *dependent product types.*

If f is a term of type α and a is term, then

$$(\mathsf{fun\text{-}app}, f, a)$$

is a term of type $(\mathsf{type\text{-}app}, \alpha, a)$ called a *function application.* If the value of f is a function g, the value of a is a set z, and $g(z)$ is defined, then the value of the function application is $g(z)$. Otherwise the function application is undefined.

If $(\mathsf{var}, x, \alpha)$ is a variable and b is a term of type β, then

$$(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$$

is a term of type $(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$ called a *function abstraction.* If the collection g of pairs $\langle z, z' \rangle$, where z is a set in the value of α and z' is a set equal to the value of b when the value of $(\mathsf{var}, x, \alpha)$ is z, is a function in D_{f}, then g is the value of the function abstraction. Otherwise the function abstraction is undefined.

The dependent function type

$$\gamma = (\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$$

is a generalization of the more common function type $\alpha \rightarrow \beta$. If f is a term of type $\alpha \rightarrow \beta$ and a is a term of type α, then the application $f(a)$ is of type β—which does not depend on the value of a. In Chiron, however, if f is a term of type γ and a is a term of type α, then the term

$$(\mathsf{fun\text{-}app}, f, a),$$

the application of f to a, is of the type

$$(\mathsf{type\text{-}app}, \gamma, a),$$

the type formed by applying the type γ to a—which generally depends on the value of a.

Example 5. This example continues Example 3. Suppose R is the type of real numbers and v is the variable $(\mathsf{var}, x, \mathsf{N})$. Then

$$\gamma = (\mathsf{dep\text{-}fun\text{-}type}, v, (\mathsf{op\text{-}app}, O, v, v, \mathsf{R}))$$

is a dependent function type that denotes the superclass of functions that map a natural number n to an $n \times n$ matrix of real numbers. Suppose $\mathsf{zero\text{-}matrix}$ is a term of type γ that denotes the function that maps a natural number n to the $n \times n$ zero matrix. If a term a of type N denotes the natural number 17, then

$$(\mathsf{fun\text{-}app}, \mathsf{zero\text{-}matrix}, a)$$

is a term of type $(\mathsf{type\text{-}app}, \gamma, a)$ that denotes the 17×17 zero matrix. Notice that $(\mathsf{type\text{-}app}, \gamma, a)$ and $(\mathsf{op\text{-}app}, O, a, a, \mathsf{R})$ are different types that denote the same superclass, namely, the type of 17×17 matrices of real numbers. □

2.7 Definite and Indefinite Description

If $(\mathsf{var}, x, \alpha)$ is a variable and B is a formula, then

$$(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$$

is a term called a *definite description* and

$$(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$$

is a term called an *indefinite description.* The definite description denotes *the* value of $(\mathsf{var}, x, \alpha)$ that satisfies B; it denotes $\bot$ if there is no value or more than one value of $(\mathsf{var}, x, \alpha)$ that satisfies B. The indefinite description denotes *some* value of $(\mathsf{var}, x, \alpha)$ that satisfies B; it denotes $\bot$ if there is no value of $(\mathsf{var}, x, \alpha)$ that satisfies B.

Definite descriptions are quite common in mathematics but often occur in a disguised form. For example,"the limit of $\sin\frac{1}{x}$ as x approaches 0" is a definite description. Indefinite description is also employed in mathematics but less so than definite description. Definite and indefinite description works very smoothly in a logic, like Chiron, in which terms are allowed to be undefined [11].

In traditional logic, definite descriptions are usually formed using an *iota operator* (ι), while indefinite descriptions are usually formed using an *epsilon operator* (ϵ).

2.8 Quotation and Evaluation

M contains a total, injective mapping $H : \mathcal{S} \rightarrow D_{\mathrm{v}}$ such that, for each $s \in \mathcal{S}$, $H(s)$ is neither an ordered pair nor the empty set $\emptyset$. $\widehat{H}$ is the total, injective mapping of $\mathcal{E}$ into D_{v} defined inductively by:

1. If $s \in \mathcal{S}$, then $\widehat{H}(s) = H(s)$.
2. If $e = (\,) \in \mathcal{E}$, then $\widehat{H}(e) = \emptyset$.
3. If $e = (e_1, \ldots, e_n) \in \mathcal{E}$ where $n \geq 1$, then

$$\widehat{H}(e) = \langle \widehat{H}(e_1), \widehat{H}((e_2, \ldots, e_n)) \rangle.$$

$\widehat{H}(e)$ is a set, called the *construction* of e, that represents the syntactic structure of the expression e. Notice that a construction has the same structure as a list data structure in the Lisp programming language with the empty set playing the role of nil.

A construction is an *operator construction, type construction, term construction,* or *formula construction* if it represents an operator, type, term, or formula, respectively. The class of constructions is represented by the application of the built-in 0-ary operator named expr. That is,

$$(\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{expr}, \mathsf{type}))$$

is the type of expression constructions. Similarly, the class of operator constructions, type constructions, term constructions, or formulas constructions is represented by the application of the built-in 0-ary operators named $\mathsf{expr\text{-}op}$, $\mathsf{expr\text{-}type}$, $\mathsf{expr\text{-}term}$, or $\mathsf{expr\text{-}formula}$, respectively.

If e is any expression, proper or improper, then

$$(\mathsf{quote}, e)$$

is a term of type

$$(\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{expr}, \mathsf{type}))$$

called a *quotation*. The value of the quotation is $\widehat{H}(e)$, the construction of e. Thus a proper expression e has two different meanings:

1. The *semantic meaning* of e is the value denoted by e itself.
2. The *syntactic meaning* of e is the construction denoted by (quote, e).

Example 6. Suppose zero and one are terms that denote the natural numbers 0 and 1, respectively. Define a *(binary) numeral* to be an expression $(a_1, \ldots, a_n)$ where $n \geq 1$ and a_i is zero or one for each i with $1 \leq i \leq n$. As defined, a numeral is an improper expression, and thus it denotes nothing. However, if n is a numeral, then (quote, n) is a proper expression that denotes the construction of n. Since constructions are sets that can be manipulated just like any other sets, numerals can be manipulated (e.g., added or multiplied) in Chiron by manipulating their quotations. (Of course, in order to know what these manipulations mean arithmetically, we would need to define a term that denotes an appropriate function from numerals to natural numbers.) This example shows that quotation enables the manipulation of syntactic objects to be performed in Chiron using all of the machinery that Chiron provides. □

If a is a term and k is a kind, then

$$(\mathsf{eval}, a, k)$$

is an expression called an *evaluation* that is a type if $k = \mathsf{type}$, a term of type k if k is a type, and a formula if $k = \mathsf{formula}$. Roughly speaking, if a denotes a construction that represents an expression e, then the evaluation denotes the value of e.

Suppose $k = \mathsf{type}$. If a denotes a construction that represents a type α in which the symbol eval does not occur, then the evaluation denotes the value of α. Otherwise the evaluation denotes D_{c}. Suppose k is a type. If a denotes a construction that represents a term b in which the symbol eval does not occur and the value of b is in the value of k, then the evaluation denotes the value of b. Otherwise the evaluation denotes $\perp$. And suppose $k = \mathsf{formula}$. If a denotes a construction that represents a formula A in which the symbol eval does not occur, then the evaluation denotes the value of A. Otherwise the evaluation denotes F. In each of the three cases the condition concerning the presence of the symbol eval is needed to block the liar paradox and similar semantically ungrounded expressions (see [12]).

See Example 7 in section 4.5 for an illustration of how the law of beta reduction can be formalized in Chiron using evaluation. Quotation and evaluation in Chiron are inspired by the quote and eval operators in the Lisp programming language.

3 Compact Notation

In this section we introduce a compact notation for proper expressions—which we will use in the rest of the paper whenever it is convenient. The first group of definitions in Table 3 defines the compact notation for each of the 13 proper expression categories we defined above.

Compact Notation	**Official Notation**
$(s :: k_1, \ldots, k_{n+1})$	$(\mathsf{op}, s, k_1, \ldots, k_{n+1})$
$(s :: k_1, \ldots, k_{n+1})(e_1, \ldots, e_n)$	$(\mathsf{op\text{-}app}, (\mathsf{op}, s, k_1, \ldots, k_{n+1}), e_1, \ldots, e_n)$
$(x : \alpha)$	$(\mathsf{var}, x, \alpha)$
$\alpha(a)$	$(\mathsf{type\text{-}app}, \alpha, a)$
$(\Lambda\, x : \alpha \,.\, \beta)$	$(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$
$f(a)$	$(\mathsf{fun\text{-}app}, f, a)$
$(\lambda\, x : \alpha \,.\, b)$	$(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$
$\mathsf{if}(A, b, c)$	(if, A, b, c)
$(\exists\, x : \alpha \,.\, B)$	$(\mathsf{exist}, (\mathsf{var}, x, \alpha), B)$
$(\iota\, x : \alpha \,.\, B)$	$(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$(\epsilon\, x : \alpha \,.\, B)$	$(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$\lceil e \rceil$	(quote, e)
$[\![a]\!]_{\mathrm{ty}}$	$(\mathsf{eval}, a, \mathsf{type})$
$[\![a]\!]_{\alpha}$	$(\mathsf{eval}, a, \alpha)$
$[\![a]\!]_{\mathrm{te}}$	$(\mathsf{eval}, a, (\mathsf{op\text{-}app}(\mathsf{op}, \mathsf{class}, \mathsf{type})))$
$[\![a]\!]_{\mathrm{fo}}$	$(\mathsf{eval}, a, \mathsf{formula})$

Table 3. Compact Notation

The next group of definitions in Table 4 defines additional compact notation for the built-in operators and the universal quantifier.

We will often employ the following abbreviation rules when using the compact notation:

1. A matching pair of parentheses in an expression may be dropped if there is no resulting ambiguity.
2. A variable $(x : \alpha)$ occurring in the body e of $(\star\, x : \alpha \,.\, e)$, where $\star$ is Λ, λ, $\exists$, $\forall$, ι, or ϵ may be written as x if there is no resulting ambiguity.
3. $(\star\, x_1 : \alpha_1 \ldots (\star\, x_n : \alpha_n \,.\, e) \cdots)$, where $\star$ is Λ, λ, $\exists$, or $\forall$, may be written as

   $$(\star\, x_1 : \alpha_1, \ldots, x_n : \alpha_n \,.\, A).$$

   Similarly, $(\star\, x_1 : \alpha \ldots (\star\, x_n : \alpha \,.\, e) \cdots)$, where $\star$ is Λ, λ, $\exists$, or $\forall$, may be written as

   $$(\star\, x_1, \ldots, x_n : \alpha \,.\, e).$$
4. If we fix the type of a variable symbol x, say to α, then a term of the form $(\star\, x : \alpha \,.\, e)$, where $\star$ is Λ, λ, $\exists$, $\forall$, ι, or ϵ, may be written as $(\star\, x \,.\, e)$.

Using the compact notation, expressions can be written in Chiron so that they look very much like expressions written in mathematics textbooks and papers.

Compact Notation	Defining Expression
T	$(\mathsf{true} :: \mathsf{formula})()$
F	$(\mathsf{false} :: \mathsf{formula})()$
V	$(\mathsf{set} :: \mathsf{type})()$
C	$(\mathsf{class} :: \mathsf{type})()$
E	$(\mathsf{expr} :: \mathsf{type})()$
E_{op}	$(\mathsf{expr\text{-}op} :: \mathsf{type})()$
E_{ty}	$(\mathsf{expr\text{-}type} :: \mathsf{type})()$
E_{te}	$(\mathsf{expr\text{-}term} :: \mathsf{type})()$
E_{fo}	$(\mathsf{expr\text{-}formula} :: \mathsf{type})()$
$(a \in b)$	$(\mathsf{in} :: \mathsf{V}, \mathsf{C}, \mathsf{formula})(a, b)$
$(\alpha =_{\mathrm{ty}} \beta)$	$(\mathsf{type\text{-}equal} :: \mathsf{type}, \mathsf{type}, \mathsf{formula})(\alpha, \beta)$
$(a =_{\alpha} b)$	$(\mathsf{term\text{-}equal} :: \mathsf{C}, \mathsf{C}, \mathsf{type}, \mathsf{formula})(a, b, \alpha)$
$(a = b)$	$(a =_{\mathsf{C}} b)$
$(A \equiv B)$	$(\mathsf{formula\text{-}equal} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A, B)$
$(\neg A)$	$(\mathsf{not} :: \mathsf{formula}, \mathsf{formula})(A)$
$(a \notin b)$	$(\neg(a \in b))$
$(a \neq b)$	$(\neg(a = b))$
$(A \vee B)$	$(\mathsf{or} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A, B)$
$(\forall x : \alpha \,.\, A)$	$(\neg(\exists x : \alpha \,.\, (\neg A)))$

Table 4. Additional Compact Notation

4 Reasoning Paradigms

We will now briefly describe how Chiron supports the five reasoning paradigms listed at the end of the Introduction.

4.1 Classical

Chiron fully supports the classical paradigm. As we mentioned in section 2.5, Chiron has the usual logical connectives: propositional connectives for negation and disjunction, an existential quantifier, constants for true and false, and an equality for each of types, term, and formulas. Chiron also includes an if-then-else term constructor. The universal quantifier is introduced by a notational definition, and other propositional connectives can be introduced by operator definitions (see [12]).

4.2 Set Theory

Chiron also fully supports the set theory paradigm. As a derivative of NBG set theory, Chiron can be used to reason about both sets and proper classes. The usual set-theoretic operators such as union, intersection, complement, etc. and constants such as the empty set and the universal class can be introduced by operator definitions (see [12]). Unlike traditional set theories formalized in first-order logic, Chiron is equipped with a rich language for forming typed terms that denote sets and classes. This language includes mechanisms for expressing function application and abstraction and definite and indefinite description.

4.3 Permitted Undefinedness

Undefined expressions are handled in Chiron according to the *traditional approach to undefinedness* [11]. Terms are allowed to be undefined. This means that a term that has no natural denotation—such as an application of a function to an argument outside of its domain—denotes the undefined value $\bot$ instead of a class. An undefined type—such as the application of a type to an undefined term—denotes the universal superclass D_{c}. And an undefined formula—such as an out-of-range formula evaluation—denotes F.

As mentioned in Example 4, the assertion that a term a is defined in a type α is explicitly expressed by the formula $(a =_{\alpha} a)$. This same assertion can be implicitly expressed in various ways. For example, if a is a term of type α, the formula $(a = b)$ implies that a is defined in α. The use of implicit definedness assertions enables statements involving partial functions and definite and indefinite descriptions to be expressed very concisely [11].

4.4 Type Theory

Chiron can be used as a type theory. Every term has a unique type determined by its syntax. The Chiron type system contains three sorts of types:

1. A *type operator application* constructs a type from a list of proper expressions which may be types, terms, or formulas. Many kinds of types can be formed in this way as shown by the following examples. An application of a type operator with the signature type introduces a base type such as V, C, or E. An application of a type operator with a signature $\mathsf{type}, \ldots, \mathsf{type}$ of length $n + 1$ where $n \geq 1$ introduces a type constructed from a list of n other types. An application of a type operator with a signature that contains a type introduces a dependent type.
2. A *dependent function type* constructs a dependent function type from a variable and a type.
3. A *type application* constructs a type from a type (which is usually a dependent function type) and a term (to which the type is applied).

Each type in Chiron denotes a superclass that is a subcollection of D_{c}, the universal superclass that contains all classes. Thus, viewed semantically, types are allowed to freely "overlap". We say that a type α is a *subtype* of another type β if the value of α is a subcollection of the value of β. It is possible for a type to be empty, that is, to denote the empty set. For example, if α is a type that denotes a superclass Σ that does not contain any functions and a is a defined term, then the type $\alpha(a)$ is empty.

Although each term has a unique type assigned to it, a term can be *defined in* many types. Suppose a is a term of type α. If a is defined, it is defined in α, the type C of classes, and every other type that denotes a superclass containing the value of a. If a is undefined, it is not defined in any type.

4.5 Formalized Syntax

Quotation is used in Chiron to refer to a set called a construction that represents the syntactic structure of an expression. Analogously to a *Gödel number* that encodes an expression as a number, a quotation in Chiron is a *Gödel set* that encodes an expression as a set. Constructions can be probed and formed using the set-theoretic machinery of Chiron. Evaluation is used in Chiron to refer to the value of the expression that a construction represents. To avoid the liar paradox and similar semantically ungrounded expressions, evaluation can only be applied to terms that denote constructions representing expressions that do not contain the symbol eval.

Quotation and evaluation together enable many common syntactic devices that are usually expressed metalogically—such as syntactic side conditions, schemas, and syntactic transformations used in deduction and computation rules—to be expressed directly in Chiron. We will give two illustrations: (1) how schemas can be formalized in Chiron and (2) how metalogical reasoning can be "reflected" in Chiron.

Schemas A *schema* is an expression that contains special variables called *schema variables* that range over expressions. An instance of a schema is obtained by simultaneously replacing each schema variable with an appropriate expression. A schema is thus a representation of the set of its instances. Schemas are used in traditional logic to describe deduction rules and sets of axioms. However, schemas usually cannot be directly formalized in a logic but are instead expressed in the logic's metalogic.

Schemas can be directly formalized in Chiron. Schema variables are formalized by variables of type E, the type of expression constructions, or by variables of the subtypes of E: E_{op}, E_{ty}, E_{tc}, and E_{fo}. Syntactic side conditions are formalized as conditions about constructions. Syntactic transformations are formalized as applications of functions that map constructions to constructions. Evaluation is used to state the value of the result of the transformation. And instances of a formalized schema are obtained by instantiating the schema variables with quoted expressions.

Example 7. There are two laws of beta reduction in Chiron, one for the application of a dependent function type and one for the application of a function abstraction. Without quotation and evaluation, the latter beta reduction law would be expressed as the formula schema

$$(\lambda\, x : \alpha\, .\, b)(a) \simeq b[(x : \alpha) \mapsto a]$$

where a is free for $(x : \alpha)$ in b. The expressions x, α, b, a are schema variables, "a is free for $(x : \alpha)$ in b" is a syntactic side condition, and $b[(x : \alpha) \mapsto a]$ is the syntactic transformation that substitutes a for each free occurrence of x in b.

Using constructions, quotation, and evaluation, a general law of beta reduction can be formalized in Chiron as a *single formula* that merges the law of beta reduction for dependent function types and the law of beta reduction for functions:

$$\begin{array}{l}
\forall\, e : \mathsf{E}_{\mathrm{te}}\ . \\
\quad (\mathsf{is\text{-}redex}(e) \wedge \mathsf{free\text{-}for}(\mathsf{redex\text{-}arg}(e), \mathsf{redex\text{-}var}(e), \mathsf{redex\text{-}body}(e))) \\
\qquad \supset \\
\quad [\![e]\!]_{\mathrm{te}} \simeq [\![\mathsf{sub}(\mathsf{redex\text{-}arg}(e), \mathsf{redex\text{-}var}(e), \mathsf{redex\text{-}body}(e))]\!]_{\mathrm{te}}
\end{array}$$

Here $\mathsf{is\text{-}redex}$, $\mathsf{free\text{-}for}$, etc. are defined operators without their signatures. The syntactic side condition (corresponding to "a is free for $(x : \alpha)$ in b") is incorporated into the hypothesis of the formula's implication, while the syntactic transformation (corresponding to $b[(x : \alpha) \mapsto a]$) is incorporated into the conclusion. The beta reduction law is applied to an application α of a dependent function type or an application a of a function abstraction by instantiating the variable $(e : \mathsf{E}_{\mathrm{tc}})$ with the quotation $\ulcorner \alpha \urcorner$ or $\ulcorner a \urcorner$. See [12] for details. Many other axiom schemas and rules of inference rules can be formalized in Chiron using schemas of this style. □

Metalogical Reflection Reasoning that is normally done in the metalogic of a logic—such as defining and applying a proof system—can be "reflected" in Chiron using its facility to reason about syntax. For example, a *theory* T can be defined as a set of formula constructions. A schema for a set of logical axioms for Chiron can be defined as a Chiron-style schema. A rule of inference for Chiron can be defined as Chiron-style schema that involves a *provability relation* $\vdash$ between theories and formula constructions. Then a formula $T \vdash a$ would assert that the formula represented by the construction a is provable from the formulas represented by the constructions in T using the Chiron-style schemas that define the logical axioms and rules of inference for Chiron.

5 Conclusion

Chiron is a logic designed to be a logical foundation for mechanized mathematics. Derived from NBG set theory, it is based on familiar principles from predicate logic, set theory, and type theory. It supports in an integrated manner five reasoning paradigms that are commonly employed either in mathematical practice or in contemporary MMSs. As a result, Chiron has a high level of practical expressivity—it offers the user easy access to a rich mixture of popular reasoning styles.

The design of Chiron is the first step of a long-range research program. The second step is to design a proof system for Chiron. Part of this task will be to determine Chiron's exact relationship to NBG. It is our conjecture that there is a faithful interpretation of NBG in Chiron. The third step is to develop an implementation of Chiron and its proof system. The final, and most important step, will be to develop a series of applications to demonstrate Chiron's reach and level of effectiveness.

Acknowledgments

The author is grateful to Marc Bender and Jacques Carette for many valuable discussions on the design and use of Chiron. Over the course of these discussions, Dr. Carette convinced the author that Chiron needs to include a powerful facility for reasoning about the syntax of expressions. The author is also grateful to Volker Sorge for reading a preliminary draft of the paper.

References

1. P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfennig, and H. Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16:321–353, 1996.
2. G. Bancerek and P. Rudnicki. Inductively defined types. In P. Martin Löf and G. Mints, editors, *COLOG-88: Proceedings of the International Conference on computer Logic*, volume 417 of *Lecture Notes in Computer Science*, pages 50–66. Springer-Verlag, 1990.
3. B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, 1991.
4. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
5. Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 8.0*, 2006. Available at `http://coq.inria.fr/doc/`.
6. T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.
7. N. G. de Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.
8. W. M. Farmer. A partial functions version of Church's simple theory of types. *Journal of Symbolic Logic*, 55:1269–91, 1990.
9. W. M. Farmer. STMM: A Set Theory for Mechanized Mathematics. *Journal of Automated Reasoning*, 26:269–289, 2001.
10. W. M. Farmer. The seven virtues of simple type theory. SQRL Report No. 18, McMaster University, 2003. Revised 2006.
11. W. M. Farmer. Formalizing undefinedness arising in calculus. In D. Basin and M. Rusinowitch, editors, *Automated Reasoning—IJCAR 2004*, volume 3097 of *Lecture Notes in Computer Science*, pages 475–489. Springer-Verlag, 2004.
12. W. M. Farmer. Chiron: A set theory with types, undefinedness, quotation, and evaluation. SQRL Report No. 38, McMaster University, 2007.
13. W. M. Farmer and J. D. Guttman. A set theory with support for partial functions. *Studia Logica*, 66:59–78, 2000.
14. W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.
15. K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.
16. K. Gödel. *The Consistency of the Axiom of Choice and the Generalized Continuum Hypothesis with the Axioms of Set Theory*, volume 3 of *Annals of Mathematical Studies*. Princeton University Press, 1940.

17. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.
18. R. D. Jenks and R. S. Sutor. *Axiom: The Scientific Computation System.* Springer-Verlag, 1992.
19. K. Kunen. *Set Theory: An Introduction to Independence Proofs.* North-Holland, 1980.
20. Lemma 1 Ltd. *ProofPower: Description*, 2004. Available at `http://www.lemma-one.com/ProofPower/doc/doc.html`.
21. E. Mendelson. *Introduction to Mathematical Logic.* Chapman & Hall/CRC, fourth edition, 1997.
22. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Computer Aided Verification: 8th International Conference, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer-Verlag, 1996.
23. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
24. A. M. Pitts. Nominal Logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
25. P. Van Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming.* MIT Press, 2004.
26. C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In R. Niewenhuis, editor, *Automated Deduction—CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53. Springer-Verlag, 2005.
27. S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, 1991.

Remarks on Non-Fregean Logic

Mieczysław Omyła

Institute of Philosophy
University of Warsaw
Poland
m.omyla@uw.edu.pl

1 Introduction

In 1966 famous Polish logician Roman Suszko red a manuscript written by Professor Bogusław Wolniewicz titled *"Rzeczy i fakty. Wstęp do pierwszej filozofii Wittgensteina (Things and facts: An introduction to Wittgenstein's first philosophy)"* published in Polish in 1968. This monograph inspired Suszko to create a new logical calculus called by him *Non-Fregean Logic.* To explain what non-Fregean logic is, let me remind you that Gottlob Frege in his essay "Über Sinn und Bedeutung" [2] formulated and, to some extent, justified the view that sentences in the logical sense of the word, have referents. Frege believes that we have to identify the denotations of sentences with their truth values. Let me quote Frege himself:

> " ... Every declarative sentence concerned with the reference of its words is therefore to be regarded as a proper name, and its reference, if it has one, is either the True or the False. ... If our supposition that the reference of a sentence is truth-value is correct, the latter must remain unchanged if a part of the sentence is replaced by an expression having the same reference. And this in fact the case.
> ... What else but the truth value could be found, that quite generally to every sentence if the reference of its is relevant, and remains unchanged by substitutions of the kind in question."

The following principle that:

> "all true (and, similarly all false) sentences describe the same state of affairs, that is, they have a common referent"

Roman Suszko has called the semantic version of Fregean axiom. Roman Suszko in his formalization of the fragment of the ontology of Wittgenstein's *"Tractatus Logico-Philosophicus"* in [7], [8] offers a continuation of Frege's program without adopting Fregean axiom.
The name non-Fregean logic is justified by the fact that in this logic there are no theorems asserting how many semantic correlates of sentences there can be. In classical logic sentential variables run over two-elements Boolean algebra of truth-values. Because theorems of this logic are the following formulas:

ISBN 978-83-7431-128-1 ISSN 0860–150X

$$(p \vee \neg p), \qquad \neg(p \wedge \neg p)$$
$$(p \leftrightarrow q) \vee (q \leftrightarrow r) \vee (p \leftrightarrow r)$$

Non-Fregean logic is extensional, logically two-valued but it is not two-valued ontologically, because it does not follow from the theorems of logic alone how many semantic correlates of sentences there can be. Following Wittgenstein, R. Suszko calls the semantic correlates of sentences "situations". To say that two sentences describe the same situations Suszko uses a connective identity "$\equiv$", that is not a truth-functional connective.

2 Non-Fregean Semantic Frameworks

I would like to present the semantic principles which constitute foundations of the non-Fregean sentential calculus. This sentential calculus is the main part of the non-Fregean logic. In my opinion the set of all sentences of any language may be interpreted according to non-Fregean semantics.
To state the principle of non-Fregean interpretation of sentences of any language L we apply the following notations:

(i) Let S be the set of all sentences of language L, and let Cn will be a consequence operation on S. We will use letters: $\alpha, \beta, \gamma, ...$ as variables running over the set S.

(ii) Let U be an arbitrary non-empty set. The elements of U will represent the situations. Intuitively speaking elements of U will be treated as given by corresponding sentences of L. For simplicity, elements of the set U will be called of situations. We will treat letters: $p, q, r, ...$ as variables running over the universe of situations U. The sentential variables: $p, q, r, ...$ are not necessarily in the alphabet of L. However, the sentential variables are needed in the alphabet of the semantic metalanguage of L, in which theorems are formulated about the set of semantic correlates of L.

(iii) Next, let H be an arbitrary relation which has as its domain the set S of sentences of language L and its counter-domain is the set U, i.e., $H \subset S \times U$. If $H(\alpha, p)$ holds we will say that the sentence α refers to the situation p, or α describes the situation p.

(iv) A sentence may be true or false, or it may be necessary or only possible or probable, determined or undetermined, sensible or senseless. In several logical systems, the mentioned properties are logical values of sentences.
Let (V, V_1) be an arbitrary pair of sets such that
$$\emptyset \neq V_1 \subset V \text{ and } V_1 \neq V$$
i.e. V_1 is non-empty set and V_1 is properly included in V. The elements of V will be called logical values, and elements of V_1 the designated logical values.

(v) An arbitrary function v is such that:
$$v : S \to V$$
and
$$\emptyset \neq v^{-1}(V_1) \neq S.$$

The function v will be called a logical valuation of L. The family of the sets $\{v^{-1}(V - V_1),\ v^{-1}(V_1)\}$ is called the fundamental partition of the set of sentences of language L determined by the logical valuation v.

Definition 1. (Non-Fregean Semantic Framework)
An arbitrary sequence of the form:

$$(*) \qquad \langle (S, Cn), U, (V, V_1), H, v \rangle$$

is named non-Fregean semantic framework for the sentential fragment of language L iff the sequence satisfies the following seven postulates called semantics principles of non-Fregean sentential logic:

(P1) Principle of Correlation:
For each sentence α of L there is at least one situation p, such that α refers to p, symbolically $\forall_\alpha \exists_p\ H(\alpha, p)$.

(P2) Principle of Univocality:
Each sentence α of language L refers to at most one situation i.e.
If $H(\alpha, p)$ and $H(\alpha, q)$, then $p = q$.

(P3) Principle of Stability:
For any $\alpha, \beta \in S$, if for any situation $p \in U; (H(\alpha, p) \leftrightarrow H(\beta, p))$, then for each sentence $\gamma \in S$, and any situation $q, (H(\gamma, q) \leftrightarrow H(\gamma[\alpha/\beta], q)$.

(P4) Principle of Logical Bivalence:
The set of logical values V is two elements i.e. $V = \{0, 1\}, 0 \neq 1$.

(P5) Principle of Maximality of Truth:
For any $X \subseteq S$, if $v^{-1}(V_1) = X$,
then $X \neq S$, and for any $\alpha \notin X, Cn(X \cup \{\alpha\}) = S$.

(P6) Principle of Subordination to Fundamental Partition:
For any sentences α, β, if for any situation $p, (H(\alpha, p) \leftrightarrow H(\beta, p)$,
then $v(\alpha) = v(\beta)$.

(P7) Principle of Contextual Differentiation:
If for any sentences γ of the language $L, v(\gamma) = v(\gamma[\alpha/\beta])$,
then for any situation $p \in U, H(\alpha, p) = H(\beta, p)$.

Observation 1.
In view of both principles: (P1) and (P2), we may define the function:
$$h : S \to U$$
by the formula:
$$p = h(\alpha) \leftrightarrow H(\alpha, p).$$
In this case we say the situation p is the semantic correlate of α or the sentence α refers to the situation p.

Observation 2.
The function h is not an arbitrary function but it must satisfy certain conditions. In particular, if S is an algebra on the set S with respect to connective of language L

(i.e. any connective of the language L is additionally treated as algebraic operation on the set sentences S) then h is a homomorphism from S to the corresponding algebra on the set $h(S) \subseteq U$.

Corollary 1.
In arbitrary semantic framework (*) the logical syntax of language L, imposes upon the set references of the sentences of L the structure of an algebra similar to the algebra of sentences of language L.

The principle (P4) asserts, that the set logical values has two elements.
If $v(\alpha) = 1$, then we say that α is true in the considered semantic frame. If $v(\alpha) = 0$ we say that α is false in this frame.

It follows from principles (P4) and (P5) that logical valuations are characteristic functions of complete theories in (S, Cn). By contraposition from principle (P6) we obtain: if any sentences α, β have different logical values then there is at least one situation which is semantic correlate for only one of these sentences i.e.

$$v(\alpha) \neq v(\beta) \rightarrow \exists p\ \neg[H(\alpha, p) \leftrightarrow H(\beta, p)].$$

These observations allow us to simplify the notion of a non-Fregean semantic framework namely, henceforth as quadruple:

$$< (S, Cn), U, h, v >$$

where:
(S, Cn) is any sentential calculus,
U is arbitrary set such that $|U| \geq 2$,
v is a characteristic function of any arbitrary complete theory in (S, Cn),
$h : S \rightarrow U$ such that for arbitrary sentences: α,$\beta \in S$ following three conditions are satisfied:
(P3') if $h(\alpha) = h(\beta)$, then $\forall_{\gamma \in S}\ [h(\gamma) = h(\gamma[\alpha/\beta])]$
(P5') if $h(\alpha) = h(\beta)$, then $v(\alpha) = v(\beta)$
(P7') if $h(\alpha) \neq h(\beta)$, then $\exists_{\gamma \in S}\ (v(\gamma) \neq v(\gamma[\alpha/\beta])$.

One direct consequence of definition 1, remarks, observations 1, 2, and corollary 1 is the following theorem:

Theorem 1.
The quadruple:
(**) $< (S, Cn), U, h, v >$
is non-Fregean semantics framework iff the following conditions are satisfied:
(1) (S, Cn) is arbitrary sentential calculus
(2) U is arbitrary set of at least two elements
(3) v is characteristic function of some complete theory in (S, Cn)
(4) h is function

$$h : S \rightarrow U$$

such that for arbitrary sentences: α, $\beta \in S$, condition

$$h(\alpha) = h(\beta) \leftrightarrow \forall_{\gamma \in S}\ [v(\gamma) = v(\gamma[\alpha/\beta])]$$

is satisfied.

The proof of this theorem is in [5].

According to theorem 1 any two sentences: α, β language L have the same semantic correlate in the non-Fregean semantic framework (**) if and only if they are mutually interchangeable in any sentential context γ without changing logical value this context.

In any semantic framework (**) we may define relation $\approx_v$ on the set S in the following way: for any: α, $\beta \in S$:

$$\alpha \approx_v \beta \Leftrightarrow \forall_{\gamma \in S}\ [v(\gamma) = v(\gamma[\alpha/\beta])].$$

Thus

$$\alpha \approx_v \beta \Leftrightarrow h(\alpha) = h(\beta).$$

Definition 2. (Suszko [8], [9])
The functor "$\equiv$" of the language L is the identity connective of sentential calculus (S, Cn) iff the following rules:
(r_0) $\vdash \alpha_1 \equiv \alpha_2$, when α_1, α_2 differ at most in bound variables,
(r_1) $\alpha \equiv \beta \vdash \gamma \equiv \gamma[\alpha/\beta]$
(r_2) $\alpha \equiv \beta, \gamma[p/\alpha] \vdash \gamma[p/\beta]$
are the rules of the consequence operation Cn.

The direct consequence this definition is that: in classical logic, the truth- functional connective of equivalence is also the identity connective.

Theorem 2.
If (S, Cn) is the language with identity connective "$\equiv$" then for any non-Fregean semantic framework $< (S, Cn), U, h, v >$ the following condition is satisfied:

for any sentences α, $\beta \in S$, $h(\alpha) = h(\beta)$ if and only if $v(\alpha \equiv \beta) = 1$.

Theorem 3.
If (S, Cn) is the language with connective "$\equiv$" such that, for any non-Fregean semantic framework $< (S, Cn), U, h, v >$ the following condition is satisfied:

for any sentences α, $\beta \in S$, $h(\alpha) = h(\beta)$ if and only if $v(\alpha \equiv \beta) = 1$, then the connective "$\equiv$" is identity connective the calculus (S, Cn) i.e. the rules: (r_0), (r_1), (r_2) are rules of consequence Cn.

The proof those theorems 2 and 3 may be find in [5].

3 Non-Fregean Logics

Following L.Wittgenstein, R. Suszko calls the semantic correlates of sentences "situations". To speak in formal way about the structure of universe of situations and universe of objects, Suszko introduced to literature a family of logic languages which he called W–languages (in honor of L. Wittgenstein). In the alphabet of these languages, there are:

1. two kinds of variables, sentential variables: p, q, r, ... and nominal variables: x, y, z, ... ,
2. truth-functional connectives: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), $\leftrightarrow$ (equivalence),
3. predicate-letters: P_1, P_2, ..., P_n,
4. function-symbols: F_1, F_2,.., F_m,
5. symbols identity: identity connective and identity predicate which are both symbolized by the sign "$\equiv$",
6. quantifiers: $\forall$, $\exists$ binding both kinds of variables.

Each of the quantifiers can bind both a sentential variable or a nominal one, depending on which variable follows directly after it. Similarly the context uniquely determines whether we have to do with the identity connective or the identity predicate since the expression $x \equiv p$ is not a formula of the language discussed. A detailed description of the syntax of the W-kind languages has been presented in the papers [1] and [3].

Given an W-language L, let S be the set of all sentential formulas, and N be the set of nominal formulas of language L. Sentential formula α is a sentence iff α has no free variables. By S^+ will be denoted the set of all sentences of language L. We consider two consequence operation on S, Cn and Cn^*, analogous as Suszko considered in [11] for open W-language. The constructions presented here extend what has been done in [11] for SCI-language.

Operation Cn is generated by the Modus Ponens rule and the schemas of logical axioms, but operation Cn^* is generated by the rules: Modus Ponens, and the rule of generalization for universal quantifier and logical axioms too. In logical literature operation Cn is called structural, but Cn^* is named invariant consequence operation. To describe the consequence Cn and Cn^* in W-languages the following notations are introduced:

Letters: v, w, v_1, w_1, v_2, w_2, ... will be metalanguage variables denoted depending on the context, either sentential variables or the nominal variables. By the letters: α, β, γ, ... will be denote any sentential formulas, by the letters: ξ, ζ, τ,... we denote any nominal formulas, and finally φ, ϕ, ψ, ... denote sentential formulas or nominal ones, depending on the context. Symbols $\alpha[v/\varphi]$ denote result substitution in formula $\alpha(v)$ for free variable v the expression φ. The result of prefixing the formula α by any finite number of universal quantifiers, i.e. $\forall_{v_1}\forall_{v_2}.........\forall_{v_n}\alpha$, where $n \geq 0$ is called generalization of the formula α. For any set of sentential formulas X, by $Gen(X)$ will be denoted the set of all generalizations of formulas in the set X. The formulas of the form $\varphi \equiv \psi$, are called equations.

3.1 Structural non-Fregean consequence operation

The structural consequence Cn in the language L was described and investigated by S. L. Bloom in [1]. The structural version non-Fregean logic in W-language L is introduced by accepting logical axioms and the only inference rule Modus Ponens. The logical axioms are those formulas which are generalizations of any formula of the following sorts:

A1. Axiom Schemata for truth-functional connective:

(S1) $\alpha \rightarrow (\beta \rightarrow \alpha)$

(S2) $[\alpha \rightarrow (\beta \rightarrow \gamma)] \rightarrow [(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma)]$

(S3) $\neg\alpha \rightarrow (\alpha \rightarrow \beta)$

(S4) $(\neg\alpha \rightarrow \alpha) \rightarrow \alpha$

(S5) $(\alpha \rightarrow \beta) \rightarrow [(\beta \rightarrow \alpha) \rightarrow (\alpha \leftrightarrow \beta)]$

(S6) $(\alpha \leftrightarrow \beta) \rightarrow [\alpha \rightarrow \beta]$

(S7) $(\alpha \leftrightarrow \beta) \rightarrow (\beta \rightarrow \alpha)$

(S8) $(\alpha \vee \beta) \leftrightarrow (\neg\alpha \rightarrow \beta)$

(S9) $(\alpha \wedge \beta) \leftrightarrow \neg(\alpha \rightarrow \neg\beta)$

A2. Axiom Schemata for quantifiers:

(S10) $\forall_v\alpha \rightarrow \alpha[v/\varphi]$

(S11) $\alpha \rightarrow \forall_v\alpha$ (if v is not free in α)

(S12) $\forall_v(\alpha \rightarrow \beta) \rightarrow (\forall_v\alpha \rightarrow \forall_v\beta)$

(S13) $\exists_v\alpha \leftrightarrow \neg\forall_v\neg\alpha$

A3. Axiom Schemata for identity connective and predicate:

A3.1. Congruence axioms. All formulas of the form:

(S14) $\varphi \equiv \phi$ (when φ, ϕ vary in at most bound variables)

(S15) for every functor Ψ we accept the invariance axiom:
$\varphi_1 \equiv \phi_1 \wedge \varphi_2 \equiv \phi_2 \wedge ...\varphi_n \equiv \phi_n \rightarrow \Psi(\varphi_1, \varphi_2, ..., \varphi_n) \equiv \Psi(\phi_1, \phi_2, ..., \phi_n)$

(S16) $\forall_v(\alpha \equiv \beta) \rightarrow (Q_v\alpha \equiv Q_v\beta)$, where $Q = \forall, \exists$.

A3.2. Axiom schema for identity:

(S17) $(\varphi \equiv \psi) \rightarrow (\alpha[v/\varphi] \rightarrow \alpha[v/\psi])$

The set logical axioms **LA** is the sum of three sets: **A1**, **A2**, **A3** i.e.

$$\mathbf{LA} = \mathbf{A1} \cup \mathbf{A2} \cup \mathbf{A3}.$$

A subset $X \subseteq S$ is called Cn–theory iff X contains **LA** and X is closed under rules MP. A set of all the sentential formulas which are derivable from any set X and from logical axioms in any finite number of steps through the application of MP rule is called theory and is denoted by $Cn(X)$. Theory T is called invariant if $Gen(T) \subset T$. A formula α is called logical theorem of non-Fregean logic iff $\alpha \in Cn(\emptyset)$.

In [1] and [3] it is proved that the structural non-Fregean logic has following metatheorems property:

1. The deduction theorem.
   For any set X of formulas of L, and any formulas α, β of L,
   $$\beta \in Cn(X \cup \{\alpha\}) \Leftrightarrow (\alpha \rightarrow \beta) \in Cn(X)$$
   This comes from the fact that theorems of non-Fregean logic are all formulas of the form:
   $\alpha \rightarrow \alpha$,
   $\alpha \rightarrow (\beta \rightarrow \alpha)$,
   $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$

and because we have that the Modus Ponens rule MP is the only rule for proving theorems.

2. The set of logical theorems $Cn(\emptyset)$ is invariant theory.
3. If $\ulcorner\alpha(v)\urcorner \in Cn(\{\alpha_1, \alpha_2, ..., \alpha_n\})$,
   where the variable v are not free in α_1, α_2, ..., α_n,
   then $\ulcorner\forall_v\alpha(v)\urcorner \in Cn(\{\alpha_1, \alpha_2, ..., \alpha_n\})$.
   This follows from the deduction theorem and from the fact that $Cn(\emptyset)$ is an invariant theory.

3.2 Invariant non-Freegan consequence operation

In L invariant non-Fregean consequence operation Cn^* is generated by the rules: Modus Ponens of the scheme:

MP. $\qquad \alpha, \alpha \to \beta \vdash \beta,$

and generalization rule of the scheme:

Gen. $\qquad \alpha \vdash \forall_v\alpha,$

and by the logical axioms divided into three groups:

A1. Axioms for the truth-functional connectives:

(1) $p \to (q \to p)$

(2) $[p \to (q \to r)] \to [(p \to q) \to (p \to r)]$

(3) $\neg p \to (p \to q)$

(4) $(\neg p \to p) \to p$

(5) $(p \to q) \to [(q \to p) \to (p \leftrightarrow q)]$

(6) $(p \leftrightarrow q) \to (p \to q)$

(7) $(p \leftrightarrow q) \to (q \to p)$

(8) $(p \vee q) \leftrightarrow (\neg p \to q)$

(9) $(p \wedge q) \leftrightarrow \neg(p \to \neg q)$

A2. Axiom Schemata for quantifiers:

(10) $\forall_v\alpha \to \alpha[v/\varphi]$

(11) $\alpha \to \forall_v\alpha$ (if v is not free in α)

(12) $\forall_v(\alpha \to \beta) \to (\forall_v\alpha \to \forall_v\beta)$

(13) $\exists_v\alpha \leftrightarrow \neg\forall_v\neg\alpha$

A3. Identity axioms:

A3.1. Congruence axioms. All formulas of the form:

(14) $\varphi \equiv \phi$ (where φ, ϕ differ at most bound variables)

(15) for every functor Ψ we accept the invariance axiom:

$$v_1 \equiv w_1 \wedge v_2 \equiv w_2 \wedge ... \wedge v_n \equiv w_n \to \Psi(v_1, v_2, ...v_n) \equiv \Psi(w_1, w_2, ..., w_n)$$

(16) $\forall_v(\alpha \equiv \beta) \to (Q_v\alpha \equiv Q_v\beta)$, where $Q = \forall, \exists$

A3.2. Special identity axiom:

(17) $(p \equiv q) \to (p \to q)$.

The set logical axioms LA^* of invariant non-Fregean logic Cn^*, is the sum of three sets: A1, A2, A3 i.e.

$$LA^* = A1 \cup A2 \cup A3.$$

A subset $X \subseteq S$ is called Cn^*–theory iff X contains LA^* and X is closed under rules MP i Gen.

The invariant non-Fregean logic satisfies the following properties:

1. For any set $X \subseteq S$, the theory $Cn^*(X)$ is the least invariant theory containing set X.
2. Deduction property: For invariant consequence the following deduction theorem may be proved:
   For any set $X \subseteq S$ and for any sentence $\alpha \in S^+$ and any formulas $\beta \in S$:
   $$\beta \in Cn^*(X \cup \{\alpha\}) \Leftrightarrow (\alpha \to \beta) \in Cn^*(X).$$

It may be interesting to consider operation consequence Cn^+ given by only one rule: Modus Ponens and the set axiom:

A1. Axioms for the truth-functional connectives:

(1) $\forall_p\forall_q[p \to (q \to p)]$

(2) $\forall_p\forall_q\forall_r[p \to (q \to r)] \to [(p \to q) \to (p \to r)]$

(3) $\forall_p\forall_q[\neg p \to (p \to q)]$

(4) $\forall_p[(\neg p \to p) \to p]$

(5) $\forall_p\forall_q[(p \to q) \to [(q \to p) \to (p \leftrightarrow q)]$

(6) $\forall_p\forall_q[(p \leftrightarrow q) \to (p \to q)]$

(7) $\forall_p\forall_q[(p \leftrightarrow q) \to (q \to p)]$

(8) $\forall_p\forall_q[(p \vee q) \leftrightarrow (\neg p \to q)]$

(9) $\forall_p\forall_q[(p \wedge q) \leftrightarrow \neg(p \to \neg q)]$

A2. Axiom Schemata for quantifiers:

(10) $\forall_v\alpha \to \alpha[v/\varphi]$

(11) $\alpha \to \forall_v\alpha$ (if v is not free in α)

(12) $\forall_v(\alpha \to \beta) \to (\forall_v\alpha \to \forall_v\beta)$

(13) $\exists_v\alpha \leftrightarrow \neg\forall_v\neg\alpha$

A3. Identity axioms:

A3.1. Congruence axioms. All formulas of the form:

(14) $\varphi \equiv \phi$ (where φ, ϕ differ at most bound variables)

(15) for every functor Ψ we accept the invariance axiom:

$$v_{\ 1} \equiv w_1 \wedge v_2 \equiv w_2 \wedge ... \wedge v_n \equiv w_n \to \Psi(v_1, v_2, ..., v_n) \equiv \Psi(w_1, w_2, ..., w_n)$$

(16) $\forall_v(\alpha \equiv \beta) \to (Q_v\alpha \equiv Q_v\beta)$, where $Q = \forall, \exists$.

A3.2. Special identity axiom:

(17) $\forall_p\forall_q[(p \equiv q) \to (p \to q)]$.

A subset $X \subseteq S$ is called Cn^+– theory iff X contains

$$LA^+ = A1^+ \cup \ Gen.(A2) \cup \ Gen.(A3.1) \cup \ (A3.2)$$

and X is closed under rules MP.

4 Semantic Remarks

The intended interpretation of W–languages is such that nominal variables range over the universe of objects while the sentential variables run over the universe of situations. All other symbols in these languages, except the sentential and nominal variables, are interpreted as symbols of some functions both defined over the universe of situations and the universe of objects. The identity connective corresponds to the identity relation between situations, and the identity predicate corresponds to the identity relation between objects. It is obvious that the language of the ordinary predicate calculus with identity is a part of the W-language excluding sentential variables, but the most frequently used sentential languages are the part of the W-language without nominal variables and the identity predicate.

If a language does not contain sentential variables then according to Suszko it does not fit for a full formalization of a theory of situations, but at most for that of a theory of events considered as reified equivalents of situations (see Suszko [10]). In contemporary science the majority of studied theories are those which are expressed in languages without sentential variables. The appearance of the Fregean sentential semantics, Leśniewski's prothothetics, Wittgenstein's Tractatus and the Non-Fregean Logic has been an important step in the development of logic-philosophical reflection, because these theories require a language with sentential variables. According to Suszko, situations are primitive with respect to events, for the latter are objects abstracted from the former. In Suszko [10] it is proved that:

(i) certain theories of situations are mutually translatable into theories of events,

(ii) certain algebras of situations are isomorphic with algebras of events.

Suszko posed the question:

> " ... *What, therefore, is the cause of the fact that our thought and the natural language to a certain degree discriminate sentence variables, and particularly, general and existential sentences about situations?*
>
> ..., *Why, therefore, should we prefer the theory of events to that of situations?*"

And in the same paper Suszko answer:

> "*It is probably the symptom of some deep, historically determined attribute our thought and of natural language, the examination and explication of which will certainly be long and arduous*".

These features of our thinking induce an account of the world rather as the universe of objects having certain properties and connected by certain relations, not - as the totality of facts obtaining a logical space as in Wittgenstein's *Tractatus.*

References

1. Bloom, S. L., *A Completeness Theorem for 'Theories of Kind W'*, Studia Logica 27, 1971, pp. 43–55.
2. Frege, Gottlob, *Über Sinn und Bedeutung,* Zeitschrfit fur Philosophie und philosophishe Kritik, 1892, pp. 25–50.
3. Omyła, M., *Translatability in Non-Fregean Theories,* Studia Logica 35, 1976, pp. 127–138.
4. Omyła, M., *Principles of Non-Fregean Semantics for Sentences*, Journal of Symbolic Logic 55, 1990, pp. 422–423.
5. Omyła, M., *Zasady niefregowskiej semantyki zdań a zasady semantyczne Fregego i Wittgensteina,* Szkice z semantyki i ontologii sytuacji, Warszawa 1991, pp. 99–114.
6. Omyła, M., *Non-Fregean Semantics for Sentences,* Philosophical Logic in Poland, 1994, pp. 153–165.
7. Suszko, R., *Ontology in the Tractatus of L. Wittgenstein,* Notre Dame Journal of Formal Logic, 9, 1968, pp. 7–33.
8. Suszko, R., *Non-Fregean Logic and Theories,* Analele Universitatii Bucuresti, Acta Logica, 11, 1968, pp. 105–125.
9. Suszko, R., *Identity Connective and Modality,* Studia Logica 27, 1971, pp. 7–39.
10. Suszko, R., *Reifikacja sytuacji,* Studia Filozoficzne 2, 1971, pp. 65–82.
11. Suszko, R., *Quasi-completeness in non-Fregean Logic,* Studia Logica 29, 1971, pp. 7–16.
12. Suszko, R., *Abolition of the Fregean Axiom,* Lecture Notes in Mathematics, nr. 453, pp. 169–239.
13. Wolniewicz, B., *Rzeczy i fakty. Wstęp do pierwszej filozofii Wittgensteina,* PWN 1968, pp. 221
14. Wolniewicz, B., *Situations as the Reference of Propositions,* Dialectics and Humanism 1, 1978, pp. 171–182.

Some Logical Aspects of Mathematical Reasoning

Xiquan Liang, Fuguo Ge and Li Yan

Qingdao University of Science and Technology
Qingdao, China

Abstract. This article discusses the applications of logic structure in Mizar Language, and discusses mathematical principles and methods, such as concepts and definitions, judgments and propositions, reasoning and proof, etc. These principles and methods are strictly followed by mathematical axiomatic methods, so does Mizar language system.

On the 65th birthday of Professor Andrzej Trybulec, all the teachers and students in our Institute of Mathematics and Physics in Qingdao University of Science and Technology express our sincere wish to his health and happiness.

We want to show our admiration to professor Andrzej Trybulec for his outstanding work in the field of formalized mathematics especially in the field of Mizar, and we also want to thank him for his instructions to us for many years.

Xiquan Liang met Professor Andrzej Trybulec and Professor Yatsuka Nakamura (Shinshu University, Nagano) in 1997 when he was studying and working in the Tokyo Institute of Technology and Shinshu University. From then on, he began to learn Mizar Language and under the guidance of Professor Yatsuka, published his first Mizar article named "Solving Roots of Polynomial Equations of Degree 2 and 3 with Real Coefficients" [4].

During learning and using the Mizar Language, we generally realized the importance of formalized mathematics, and then took the further research of the theories of formalized mathematics. It is an important field to use programmed language to prove mathematical problems. The essential of formalized mathematics is to formalize the intelligent course of reasoning and deduction. Proving theorems by computer is one of the primary directions involving artificial intelligence. It is a great breakthrough of modern mathematics to prove theorems by machine automatically instead of by person themselves.

Xiquan Liang came back to China and worked in Qingdao University of Science and Technology in August, 2000. At the same time, he organized a scientific research group including professors, young teachers and postgraduates to continue the study in this field. On the base of discussing plenty of Mizar articles, we combined Mizar Language with concrete research directions, such as, mathematics morphology, matrix theory, fluxionary calculus theory, etc. The academic atmosphere was strong in our group, and we discussed with each other when we met problems, trying to

ISBN 978-83-7431-128-1 ISSN 0860–150X

improve our levels in this course. In order to improve our comprehension and applications of Mizar, we invited Professor Andrzej and Professor Yatsuka in 2004 and 2006. They explained the knowledge and applications of Mizar Language to our young teachers and postgraduates and meanwhile, they illustrated the confusions and difficulties of writing the papers vividly.

We have proved many theorems using Mizar Language from 2004, and accomplished some achievements. All the proved theorems were accepted by Mizar mathematical library. We plan to do the research and make some breakthroughs in the field of manifolds, algebra, number analysis, etc.

After 30 years of development, many scholars in Poland and Japan have made the great achievements in Mizar Language. But in Qingdao, it's a new field. We have constructed the good cooperative connection with Białystok University and Shinshu University, we hope to strengthen scientific collaboration of the three universities in the future, and promote the development of Mizar Language by our collective efforts.

1 Concept and Definition

First, we'll discuss the meaning, connotation and extension of concepts, and further discuss the methods and rules of definition with Mizar language.

1.1 Concept

Concept is our thinking form reflecting essential attributes of objective things. Like other concepts, mathematical concepts are also abstracted from the realistic world. For example, in geometry, point, line, plane, body are abstracted from the spacial form which objects occupy; in algebra, natural number, fraction, rational number, irrational number, etc., are abstracted from quantitative relationship between investigative objects.

The abstraction of the concept mainly expresses essential attributes and interior relations of investigative objects, instead of the phenomenon and external relations of investigative objects. These essential attributes of investigative objects and general comprehension are just the significate meaning of the scientific abstraction.

Attributes of objects can be divided into "essential attributes" (or "proper attributes") and "nonessential attributes". Essential attributes reflect characteristics of objects, namely they only belong to a certain type of objects, and don't belong to other types. Thus we can distinguish this type of objects from others, and infer other nonessential attributes of objects according to essential attributes. For example, the attribute that quadrangles are equilateral and equiangular is enough for us to distinguish square from all quadrangles, but other attributes, such as diagonals are equal; diagonals are vertical and halved each other; diagonals and sides are not commensurable; quadrangles can make the circumscribed circle and the inscribed circle, etc., are nonessential attributes.

1.2 Connotation and Extension of Concept

Concept is a basic form of thought and reasoning. Concept must be unambiguous; otherwise it can't correctly reflect the objective things and their characteristics. Therefore, we can't correctly judge, reason and prove in the thinking process. How on earth can the concept be unambiguous? This problem involves connotation and extension of the concept.

Connotation of the concept is the summations of all the essential attributes of the investigative objects.

For instance, connotations of the parallelogram are that it is a quadrilateral and its two sets of subtenses are parallel respectively. Connotations of the square are that it is a quadrilateral, its four sides are equal and its two adjacent angles are equal.

Extension of the concept is the summations of all investigative objects. In other words, it is inherent scope of the concept.

For example, triangles include many varieties. According to the angles, they are classified into acute triangles, obtuse triangles and right angled triangles. According to the sides, they are classified into non-equilateral triangles, isosceles triangles and equilateral triangles, etc. All the triangles in these varieties belong to the extension of the concept – triangle. In the same way, the extension of the quadrangle is all the quadrangles.

According to relationship between species and genus-difference, quadrangles can be divided as follows.

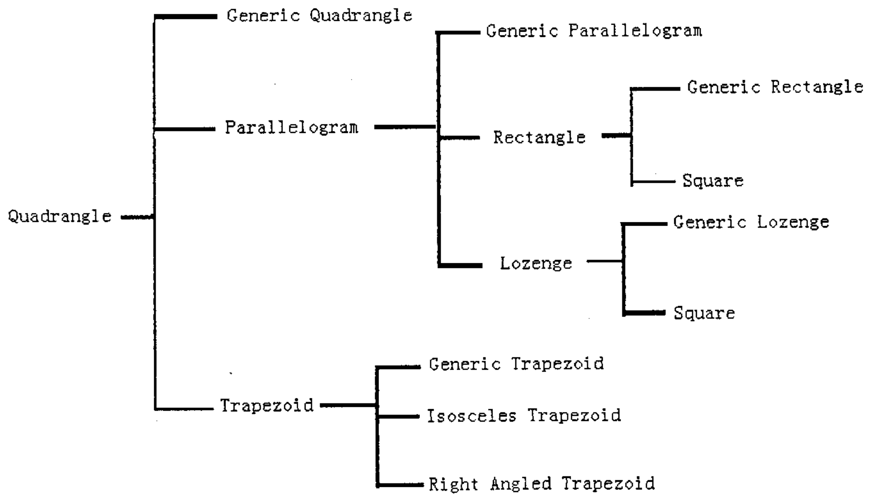


Fig. 1.

All quadrangles are extension of the concept – quadrangle. Generic parallelograms, rectangles, lozenges and squares constitute the extension of the parallelogram. Generic rectangles and squares constitute the extension of the rectangle. Generic lozenges and squares constitute the extension of the lozenge. In this way, we can see the rule that the extension of concept will decrease if its connotation increases. On the contrary, the extension of concept will increase if its connotation decreases.

Connotation and extension of the concept are two parts of the concept. A concept is unambiguous, which means its connotation and extension must be unambiguous and also means its essential attributes must be unambiguous. Therefore, we can use the concept correctly.

1.3 Species and Genus of Concept

If the extension of concept B belongs to the extension of concept A, the A is called species and B is called genus. The two kinds of concepts are constituted to have relationship between species and genus.

Species and genus are relative. The same concept is a species concept for one concept, but is possibly a genus concept for another concept. Every concept with bigger extension is a species concept for the concept with smaller extension. And every concept with smaller extension is a genus concept for the concept with bigger extension. For instance, in geometry, parallelogram is a genus concept for quadrangle, but it is a species concept for rectangle or square.

Circle and geometric figure have relationship between species and genus, but circle and quadrangle don't have relationship between species and genus. Both pentagon and hexagon are genus concepts for polygon, but they don't have relationship between species and genus.

An object possibly has many species concepts. The species that is the closest to it is called the closest species of it. For example, polygon, quadrangle and parallelogram are all species of rectangle. Here, the closest species of rectangle is parallelogram, but not the polygon or quadrangle. There are two closest species of square, which are rectangle and lozenge.

1.4 Definition

Defining a concept is to disclose the essential attributes of the concept, and also to disclose the connotation of this concept. When the concept has an explicit definition, we can distinguish it from other concepts in essence.

"Genus-difference" is the differences in the essential attributes between the concept being defined and other genus concepts paratactic with it. For example, generic quadrangle, parallelogram and trapezoid are paratactic genus concepts corresponding to the species concept quadrangle. But parallelogram has two sets of parallel subtenses, and trapezoid has only one set of parallel subtenses and another set of unparallel subtenses, and generic quadrangle has two sets of unparallel subtenses. These differences are the genus-difference among them. Another example: triangle, quadrangle and pentagon are parallel genus concepts for polygon, and their genus-difference is that their numbers of sides are different.

We usually apply a method to disclose the essential attributes of the concept and correctly define it. This method is "the closest species" plus "genus-difference". Its formula is:

Defined concept = genus-difference + the closest species.

According to the above formula, trapezoid and parallelogram can be defined respectively as follows:

A parallelogram (the nearest species) is called trapezoid, if its one set of subtenses is parallel and another set of substenses is not parallel (the genus-difference).

A parallelogram (the nearest species) is called parallelogram, if its two sets of subtenses are parallel respectively (the genus-difference).

Definitions have to obey the following rules:

Rule 1. Definitions must be both corresponding and matching with, which means the extension of the defined concept must be equal with the extension of the definition.

Breaking this rule may lead us to make the mistakes that the scope of the definition is too wide or too narrow. For example, there is a definition as follows:

"Two polygons that their corresponding angles are equal respectively are called similar polygons."

In this definition, the extension is too wide. It includes not only all of similar polygons but also many nonsimilar polygons, because according to this definition, all rectangles and squares are similar with each other. There is another definition as follows:

"The polygons that their corresponding angles are equal respectively and the ratios of corresponding sides are equal with a certain positive integer, are called similar polygons."

In this definition, the extension is too narrow. Because the similarity ratio is limited in the scope of positive integers, this definition eliminates such similar polygons that their similarity ratio does not equal a positive integer (such as, positive fractions).

Rule 2. The definition can not be circulated.

This rule means to avoid two kinds of mistakes. One kind of mistakes is that concept A is defined by concept B, then concept A is used to define concept B again.

2 Judgment and Proposition

We discuss the meaning and common types of judgments first, and then discuss the mathematical judgment - propositions, including compositions, changes of a proposition and rules used to judge the equivalent relation of propositions, etc.

2.1 Judgment

Judgment is a thinking form expressing certain affirmation or negation of the objects.

Judgment is a thinking form understanding objective things on the foundation of concepts. Different from concept and other thinking forms, judgment is to disclose the relationship between objects and a certain special attribute by affirmation or negation. It isn't a judgment if it doesn't affirm anything or negate anything.

Concepts are realized by predicates. But judgments are realized by sentences. At the same time, the logical judgment has own structure.

Judgments can be divided into positive judgments and negative judgments by nature. They are expressed by "is" or "isn't".

There are some kinds of judgments.

2.2 Categorical Judgment

The categorical judgment reflects the simple contacts or differences between objects. It is the most common and the simplest and the most basic judgment. It can be divided as follows:

1. Single Judgment. It is the simplest form of the judgment. It affirms or negates a certain property of a single object. It includes as follows:
   - Single Positive Judgment. Its formula is "S is P".
   - Single Negative judgment. Its formula is "S isn't P".
2. Special Judgment. It affirms or negates properties of some objects in a certain class. It includes as follows:
   - Special Positive Judgment. Its formula is "some S are P".
   - Special Negative Judgment. Its formula is "some S aren't P".
3. Full Judgment. It affirms or negates properties of all objects in a certain class. It includes as follows:
   - Full Positive Judgment. Its formula is "all S are P".
   - Full Negative Judgment. Its formula is "all S aren't P".

2.3 Proposition

The proposition is a sentence describing a certain judgment in mathematics. Both theorems and axioms are propositions.

Condition proposition is the manifestation of mathematical proposition.

Condition proposition reflects more complicate relationship or causality between objects. Theorems are condition propositions generally. Their formula is "if S is P, then R is Q". For example, if point P is in the exterior of a circle O, then OP<r (O is a center of the circle O, r is the radius).

2.4 Compositions of Proposition

Many mathematical propositions are condition propositions. They are constituted by two or more than two judgments. If the former judgment "S is P" of the condition proposition is written as A, the latter one "R is Q" is written as B, then this kind of propositions is written as "if A, then B". Here A is called the premise, B is called the conclusion. In other words, a mathematical proposition is constituted by premise and conclusion, and its standard format is "if A, then B". It is written as "→" or "if A, then B" or "if A, prove B".

2.5 Four Kinds of Forms of Proposition

A proposition has four kinds of changes, then it has four kinds of forms.

We can get the converse proposition of an original proposition if we exchange its condition and conclusion. We can get the negative proposition of an original proposition if we negate its condition and conclusion at the same time. We can get the converse-negative proposition of an original proposition if we negate its negative proposition again or exchange premise and conclusion of its negative proposition. Its standard form is as follows:

Original proposition: if A, then B.

Converse proposition: if B, then A.

Negative proposition: if not A, then not B.

Converse-negative proposition: if not B, then not A.

Four kinds of forms of a proposition are expressed as follows:

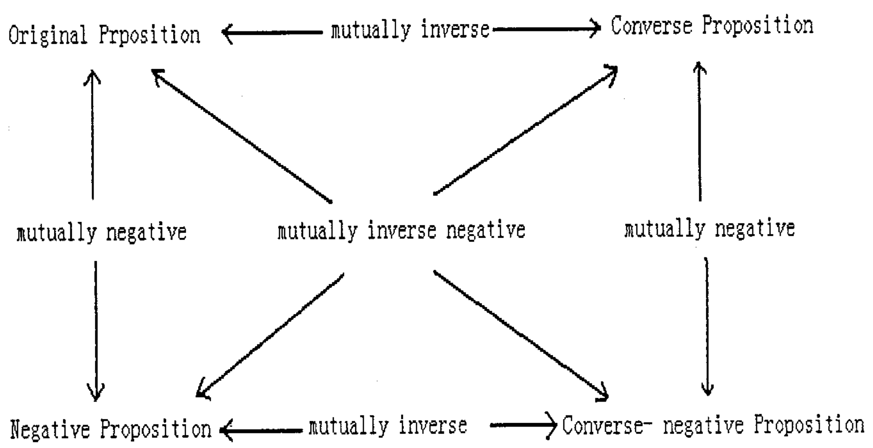


Fig. 2.

2.6 Equivalence in Four Kinds of Forms of Proposition

Proposition α and proposition β in the four kinds of forms of a proposition are called equivalent propositions, if α is true, then β is true; on the contrary, if β is true, then α is true. Equivalent propositions must be true or false simultaneously.

Propositions are not always true. So four kinds of forms of a proposition are not all true.

It is important to study the equivalent relation of transformations of propositions. It helps us to understand the effects of propositions and prove propositions. There are some rules to judge equivalent propositions:

1. Rule of Converse-negative Proposition

From 3.5, we can see that original proposition and converse-negative proposition, converse proposition and negative proposition are mutually inverse negative respectively. They are always true or false simultaneously. So two mutually inverse negative propositions are equivalent. This rule is called "Rule of converse-negative proposition". It is the theoretical foundation of the reduction to absurdity (it is also called "Contradictory Method") which is a indirect proof method.

2. Rule of Identity

Negative propositions and original propositions are not always true or false simultaneously. If premise and conclusion of a proposition exist uniquely, its original proposition and converse proposition must be equivalent. So we call this proposition satisfies the rule of identity. For example, the proposition that the bisector of vertex angle of an isosceles triangle is the middle line of the base obviously satisfies the rule of identity. For a certain isosceles triangle, its bisector of vertex angle and the middle line of the base are existential and unique, and its converse proposition, namely, the middle line of the base of an isosceles triangle is the bisector of vertex angle, is apparently true. There is another example. The proposition that if a straight line goes across a point and is vertical with the radius across this point, this straight line is tangent with the circle at this point also satisfies the rule of identity. Because the vertical line and the tangent across the endpoint of a radius are existential and unique, its converse proposition, if a straight line is tangent with a circle at a point, this line is vertical with the radius across this point, is also true.

3. Segmented Proposition

A proposition is called a segmented proposition, if it is constituted by several propositions, and the contents that are expressed by condition and conclusion of these propositions are elaborate in every aspect (all probable situations among objects are listed exhaustively), but not consistent. For example,

The theorem that in a triangle, if two angles are equal, their corresponding sides are also equal; if two angles are not equal, their corresponding sides are not equal either, and the big angle corresponds the big side and the small angle corresponds the small side.

This proposition is constituted by three propositions in fact, and in the triangle ABC, can be expressed by the symbols as follows:

(i) If $\angle$A=$\angle$B, then BC=CA;
(ii) If $\angle$A>$\angle$B, then BC>CA;
(iii) $\angle$A<$\angle$B, then BC<CA;

Conditions of the above three propositions include all situations of the relations of sizes of two angles. These situations include "equal", "greater than" and "lessen than". At the same time, conclusions of three propositions also include all situations of the relations of the corresponding sides of these two angles. That is to say, conditions and conclusions of propositions are elaborate.

2.7 Condition of Proposition is As Follow:

In mathematics whether many propositions are true or not has something to do with the condition of it. This kind of condition is the hypothesis of the proposition. According to the function of condition A to conclusion B, the condition of proposition has two characters and three forms.

Two characters:

Sufficient character – if we can get conclusion B from condition A, then the condition A is called the sufficient condition of the conclusion B.

Necessary character – if we can get condition A from conclusion B, then the condition A is called the necessary condition of the conclusion B.

Three instances:

1. Sufficient but not necessary condition. This condition has the character of sufficient but not the character of necessary.
2. Necessary but not sufficient. This condition has the character of necessary but not the character of sufficient.
3. Sufficient and necessary. This condition has not only the character of sufficient but also the character of necessary.

3 Inference and Proof

3.1 Inference

Inference is a thinking form of getting a new judgment from one or several known judgments. It's a significant thinking form for people to know the objects. But where does the judgment come from? Some are got from direct observations and experimentations; some are got from inference, which is a more advanced thinking form than judgment.

Inference is generally constituted by two parts: the premise and the conclusion. In the process of inference, the premise is the known judgments from which we can get a new one. And the new judgment which is got from the premise is called the conclusion. The special relation among the judgments which constitute inference is also the relation of premise and conclusion. The inferences which are frequently used are inductive inference, deductive inference and analogy inference etc.

1. Inductive Inference

Inductive inference is a thinking method from special to general, it is also the inference method to get general conclusion from the premise of the individual judgment or special judgment.

Usually, there are two kinds of inductive inference being used in mathematics: complete inductive inference and incomplete inductive inference.

Complete inductive inference: The amounts of same kind of objects which are investigated is finite, every object will be investigated, generalized and then concluded.

The conclusion which comes from true premise is reliable by complete inductive inference.

Incomplete inductive inference: The amounts of the same objects which are investigated are infinite. Even they are finite, people can't be investigate them one by one. So we can only investigate part of them to speculate more general conclusion.

The conclusion got from incomplete inductive inference may be wrong, and it also may be correct, but it can help people to get the new regular patterns. Because when people are trying to get the new principle of facts, first they'll recognize some individual facts of practice which may be the clue for them, then people will get some general conclusions and at last, they'll get the final confirmation by inference or other method.

2. Deductive Inference

Deductive inference is a kind of thinking method from general to special. What we frequently meet in deductive inference is syllogism. Syllogism is a kind of inferable method that is to get a new judgment from two categorical judgments; moreover one of the categorical judgments must be full name judgment.

Syllogism is made up of three parts: assumption (full name judgment), minor premise (special name judgment), and conclusion (final judgment).

For example the diagonals of rectangle are equal (assumption).

Square is rectangle (minor premise).

So the diagonals of square are equal (conclusion).

3. Analogical Inference

Analogical inference is such a kind of inference as follows: We can infer that two objects may be same in other attributes from the fact that the two objects are same in some attributes. Such as the object A has the attributes of a, b, c, dthe object B has the attributes of a, b, c, then we can imagine that object B also has the attribute of d.

Analogical inference is a sort of probable reasoning, it can only provide clues to people, inspire people to consider and find problem; but whether the conclusion is right or not must be validated by other methods.

There are some common characters and differences between analogical inference and incomplete inductive infêrence. The common character is that the conclusion inferred by them is presumption; the difference is that the former is from special to special, the latter is from special to general.

3.2 Proof

Proof is a thinking form which states the judgment is actual and correct. That is a process to make use of some actual and reliable judgments as the gist to illuminate the authenticity of some judgments by one or several inference. There are logicality and continuity in these illations, which means that the conclusion of the former illation is usually the premise of the next illation, until getting the correctness of special judgment to be proved by inference.

The difference between proof and inference is: in the inference process, the procedure is from premise to conclusion. The premise is the judgment which is known beforehand, and then reason to get a new conclusion. The procedure of proof is reverse to the inference, firstly given a judgment, people will try to find the sufficient excuses which state that the judgment is actual. Only based on these excuses, can we can state the authenticity of the judgment which is provided beforehand by several reasoning. Proof is a logic form and method which is more complex and abstruse than inference.

The proof in mathematics can help us confirm the authenticity of some propositions or theories, and it is an important method to institute mathematics theory like inference.

3.3 Constitution of Proof

Three parts constitute the proof:

1. Thesis is the judgment that is needed to prove its authenticity, it's the thesis needed to be proved in mathematics.
2. Arguments are those judgments which are used to prove the authenticity of thesis. These arguments are definitions which are known beforehand and axioms and theorems which are put forward before the thesis needing to be proved.
3. Reasoning is the inference process from argument to conclusion.

3.4 Several Rules of Proof

A correct proof must obey the following rules:

1. The thesis must be correct and the change is forbidden, which means we can not change the content of hypothesis and conclusion.
2. The argument must be real and reliable. That is to say every argument we use must be the adequately confirmed fact.
3. The argument can not be inferred from thesis in one proof. Otherwise to get thesis through argument by reasoning and to get argument through thesis by reasoning, this will lead to the error of vicious circle.
4. The argument must reason the thesis. If the arguments have no relation with the thesis or the arguments are not enough, the whole proof have no effect.

3.5 Methods of Proof

Owing to different aspects, there are several methods of proof in mathematics.

It can be divided into the proof of deductive method and inductive method by the adoption of the different inference.

It also can be divided into the proof of analyze method and synthesize method because of the adoption of different thinking of "from unknown to known" and "from known to unknown".

It can be divided into the direct proof if we begin our thesis directly and indirect proof if we begin our thesis indirectly.

These methods of are classified according to the different views. Classifying in this way is in order to master the characters of every method of proof conveniently. By the reason that every proof is completed by inference, the deductive inference and the inductive inference are the base. By the reason that every proof can not be separated from the thinking process of analysis and synthesize, the analysis and synthesize are the key points to solve the problems; while whether starting the thesis directly or indirectly is the starting point of the whole proof, in fact every proof is the result of the associated effect of using several methods of proof. We should not only master the characters of every method but also apply them unitedly to complete each proof. All these proof methods have extensive applications in Mizar language.

References

1. Yongchun Liang, New Edition of Logic. Peking University Press (2005)
2. Yingcan He, Yilian Peng, Introduction of Logic. East China Normal University Press (1988)
3. Xiaoling Zuo, Discrete Mathematics. Shanghai Science and Technology Press (1982)
4. Xiquan Liang, Solving Roots of Polynomial Equations of Degree 2 and 3 with Real Coefficients. In Roman Matuszewski - editor, *Formalized Mathematics* 9(**2**), 2001, pp.347–350

Jordan's Proof of the Jordan Curve Theorem

Thomas C. Hales*

University of Pittsburgh

Abstract. This article defends Jordan's original proof of the Jordan curve theorem.

The celebrated theorem of Jordan states that every simple closed curve in the plane separates the complement into two connected nonempty sets: an interior region and an exterior. In 1905, O. Veblen declared that this theorem is "justly regarded as a most important step in the direction of a perfectly rigorous mathematics" [13]. I dedicate this article to A. Trybulec, for moving us much further "in the direction of a perfectly rigorous mathematics."

1 Introduction

Critics have been unsparing in their condemnation of Jordan's original proof. According to Courant and Robbins, "The proof given by Jordan was neither short nor simple, and the surprise was even greater when it turned out that Jordan's proof was invalid and that considerable effort was necessary to fill the gaps in his reasoning" [2]. A web page maintained by a topologist calls Jordan's proof "completely wrong." Morris Kline writes that "Jordan himself and many distinguished mathematicians gave incorrect proofs of the theorem. The first rigorous proof is due to Veblen" [8]. A different Kline remarks that "Jordan's argument did not suffice even for the case of a polygon" [7].

Dissatisfaction with Jordan's proof originated early. In 1905, Veblen complained that Jordan's proof "is unsatisfactory to many mathematicians. It assumes the theorem without proof in the important special case of a simple polygon and of the argument from that point on, one must admit at least that all details are not given" [13]. Several years later, Osgood credits Jordan with the theorem only under the assumption of its correctness for polygons, and further warns that Jordan's proof contains assumptions.[1]

Nearly every modern citation that I have found agrees that the first correct proof is due to Veblen in 1905 [13]. See, for example, [9, p. 205].

* This research has been supported by NSF grant 0503447.

[1] "Es sei noch auf die Untersuchungen von C. Jordan verwiesen, wo der Satz, unter Annahme seiner Richtigkeit für Polygone, allgemein für Jordansche Kurven begründet wird. Jordan beweist hiermit mehr als die Funktionentheorie gebraucht; dagegen macht er Voraussetzungen, welche diese Theorie streng begründet wissen will" [11].

ISBN 978-83-7431-128-1 ISSN 0860-150X

My initial purpose in reading Jordan was to locate the error. I had completed a formal proof of the Jordan curve theorem in January 2005 and wanted to mention Jordan's error in the introduction to that paper [3]. In view of the heavy criticism of Jordan's proof, I was surprised when I sat down to read his proof to find nothing objectionable about it. Since then, I have contacted a number of the authors who have criticized Jordan, and each case the author has admitted to having no direct knowledge of an error in Jordan's proof. It seems that there is no one still alive with a direct knowledge of the error.

The early criticisms from Veblen and Osgood are in fact rather harmless. True, Jordan did not write out a proof for polygons, but then again, the proof for polygons is widely regarded as completely trivial. For example, the book *What is Mathematics* "presupposes only knowledge that a good high school course could impart," and yet it presents two different proofs of the polygon case of the Jordan curve theorem. The second of these proofs is sketched in eight lines, with "the details of this proof is left as an exercise" [2]. If Veblen and Osgood had stronger evidence to discredit Jordan, why was their only explicit objection such a trivial one?

We wonder whether Kline's objection that "Jordan's argument did not suffice even for the case of a polygon" might merely be a reiteration of Veblen's objection that the case of polygons was omitted. These authors are correct in stating that Jordan stated the polygon version of the Jordan curve theorem without proof. However, a careful analysis of his proof (which we provide below) shows that Jordan does not make essential use of the Jordan curve theorem for polygons. Rather, he relies instead on the considerably weaker statement that there is a well-defined parity function that counts the number of times a ray crosses a polygon.

I have found one supporter of Jordan's proof. A 1996 paper gives a nonstandard proof along the lines of Jordan's original article [6]. It does not make sense for an essentially flawed approach to shed its defects when translated into another language, any more than it would for pulp to become great literature in translation. Puzzled, I contacted Reeken, one of the authors of the nonstandard proof. He replied that "Jordan's proof is essentially correct... Jordan's proof does not present the details in a satisfactory way. But the idea is right and with some polishing the proof would be impeccable" [12].

At the same time, Veblen's proof has suffered with the passage of time. His proof was part of his larger project to axiomatize *analysis situs* as an isolated field of mathematics. The model for this project was Hilbert's axiomatization of the foundations of geometry in 1899 [4]. This work precedes the rise of set theory as an axiomatic discipline. (Zermelo's first paper on the axioms of set theory appeared three years after Veblen's proof, in 1908.) Veblen's system of axioms was later abandoned when R. L. Moore showed in 1915 that his axioms describe nothing but the ordinary Euclidean plane. According to one account, the "results published in 1915 by Moore were rather devastating" for Veblen's line of research [10]. Thus, after a century, the entire framework of Veblen's proof is largely forgotten.

In view of the fundamental importance of the Jordan curve theorem to geometry, I present Jordan's proof anew. I have brought the terminology and language up to date without changing any essential ideas. In this way, I hope to preserve all of Jordan's major ideas, while avoiding its minor shortcomings. By presenting his ideas once again, we revive an elegant argument that has been unfairly condemned. Ultimately, if history has filled the gaps so completely that it becomes a serious challenge for us to discern them, then this speaks all the more forcibly in favor of Jordan.

There is another reason to take particular interest in Jordan's proof. This reason is the proof of the isoperimetric inequality for a general rectifiable Jordan curve J. The proof of the isoperimetric inequality for polygons is quite simple. To deduce the general version of the isoperimetric inequality from the special case of polygons, one must construct a polygon approximation to J whose length is no greater than that of J, and whose enclosed area exceeds that of J by no more than ϵ (for any $\epsilon > 0$). This is precisely what Jordan constructs in his proof. Hence, the isoperimetric inequality comes as a corollary to Jordan's proof. As far as I am aware, this corollary of Jordan's proof has gone unnoticed for over a century.

2 All About Polygons

2.1 Basic Definitions

Let d be the standard metric on $\mathbb{R}^2$.

Definition 1. *A simple closed curve J, also called a Jordan curve, is the image of a continuous one-to-one function from $\mathbb{R}/\mathbb{Z}$ to $\mathbb{R}^2$. We assume that each curve comes with a fixed parametrization $\phi_J : \mathbb{R}/\mathbb{Z} \to J$. We call $t \in \mathbb{R}/\mathbb{Z}$ the time parameter. By abuse of notation, we write $J(t) \in \mathbb{R}^2$ instead of $\phi_J(t)$, using the same notation for the function ϕ_J and its image J.*

We say that I is a (short) *interval* in $\mathbb{R}/\mathbb{Z}$ if there is an interval $[t, t']$ in $\mathbb{R}$ with $0 < t' - t < 1/2$ such that I is the image of $[t, t']$ under the canonical projection to $\mathbb{R}/\mathbb{Z}$. The upper bound on $t' - t$ ensures that an interval is uniquely determined in $\mathbb{R}/\mathbb{Z}$ by its endpoints.

Remark 1. We adopt the following useful convention, when two distinct parameter values $t, t' \in \mathbb{R}/\mathbb{Z}$ are used in a symmetrical manner, we swap them as necessary and identify them with real representatives such that $0 < t' - t \leq 1/2$.

Definition 2. *A polygon is a Jordan curve that is a subset of a finite union of lines. A polygonal path is a continuous function $P : [0, 1] \to \mathbb{R}^2$ that is a subset of a finite union of lines. It is a polygonal arc, if it is $1 - 1$.*

2.2 Parity Function for Polygons

The Jordan curve theorem for polygons is well known. We will only need a weak form, essentially saying that the complement of a polygon has at least two connected components. For this, it is enough to construct a non-constant locally constant function (the parity function).

Lemma 1. *Let P be a polygon. There exists a locally constant function on the complement of P in $\mathbb{R}^2$ that takes two distinct values.*

Proof. If the vertical line through $p \in \mathbb{R}^2 \setminus P$ does not contain any vertices of P, define its parity to be the parity of the number of intersections of P with downward directed vertical ray starting at p.

This function extends to a function on $\mathbb{R}^2 \setminus P$. In fact, if the downward vertical ray through p intersects P at a vertex, any sufficiently small horizontal displacement of the ray will not pass through a vertex of P.

The parity is independent of the small displacement. In fact, let V be the semi-infinite vertical strip between the two rays. The intersection of P with V consists of a finite number of disjoint polygonal arcs, with endpoints along the bounding rays. Since each polygonal arc has two endpoints, the total number of intersections of P with the two vertical rays is even. Thus, the parity is independent of the small displacement.

The fact that the parity is independent of small displacements is also used to establish the local constancy of the parity function.

The parity function takes two distinct values, as is seen by considering values along a ray that intersects P.

In reference to polygons, terms such as 'interior' or 'inside' mean odd parity and terms such as 'exterior' or 'outside' mean even parity. By using these terms, we do not assume the Jordan curve theorem for polygons, which makes the stronger assertion that the interior and exterior sets are each connected.

Now that we have established the existence of a parity function, we can compute the parity in degenerate situations where the vertical ray passes through a vertex of the polygon.

There are various related properties of the parity function on the complement of a polygon. The following statements about polygonal paths are easily established by the same method of 'displacement' that is used in the preceding lemma.

- We can compute the parity with respect to a ray in any direction through a point p and get the same value for the parity. It is not necessary to use the downward directed rays.
- Any point in the unbounded component has even parity.
- If a polygonal arc L from p to q crosses the polygon P transversally (meaning that L and P do not meet at a vertex of either one) m times, then p and q have the same parity with respect to P if and only if m is even.

2.3 Parity Function for Arcs

Let A be a polygonal arc in the plane with endpoints p and q. Let R_p and R_q be the vertical rays emanating from p and q directed upwards. Then by arguments similar to those used for a polygon, we see that there is a well-defined locally constant parity function $x \mapsto \pi_A(x)$ on

$$\mathbb{R}^2 \setminus (A \cup R_p \cup R_q)$$

that counts the crossing parity of A with a downward directed ray emanating from a point x.

If L is a line that does not meet p or q, then the parity of $\pi_A(x)$ is independent of x for all $x \in L$ with sufficiently large second coordinate. Thus, we may speak of the crossing parity between a line and a polygonal arc A, provided the line does not pass through the endpoints of of A.

If B is a second polygonal arc with endpoints p and q such that $A \cup B$ forms a simple closed curve, then the conditions $\pi_A(x) = \pi_B(x)$ and $\pi_A(x) \neq \pi_B(x)$ extend from

$$\mathbb{R}^2 \setminus (A \cup B \cup R_p \cup R_q)$$

to locally constant conditions on

$$\mathbb{R}^2 \setminus (A \cup B).$$

We have $\pi_A(x) = \pi_B(x)$ if and only if x has even parity with respect to the polygon $A \cup B$.

If A is a polygonal arc joining p and q, and L is a vertical line that does not meet p or q, then the line L meets A an odd number of times if and only if L separates p and q.

2.4 Adding a Polygonal Arc to a Polygon

Let A, B, and C be polygonal arcs that have the same endpoints p, q, but do not meet except at the endpoints. We then have polygons $P(A,B)$, $P(B,C)$, and $P(A,C)$ formed by pairs of arcs. Every point (other than the endpoints) of A (resp. B, C) has the same parity with respect to $P(B,C)$ (resp. $P(A,C)$, $P(A,B)$). So we may speak of the parity of any one of A, B, C with respect to the polygon formed by the other two.

Write π_A, π_B, π_C for the functions of $x \in \mathbb{R}^2 \setminus (A \cup B \cup C \cup R_p \cup R_q)$ giving the parities of crossing with A, B, and C with a downward directed ray starting from a point x.

Lemma 2. *Let x be a point of the plane in the complement of $A \cup B \cup C$ that lies outside $P(A,C)$. Then x lies outside $P(A,B)$ if and only if it lies outside $P(B,C)$.*

Proof. To say that x lies outside $P(A,C)$ means more precisely that $\pi_A = \pi_C$, and so forth. Then the conclusion follows directly from the following boolean tautology:

$$(\pi_A = \pi_C) \Rightarrow [(\pi_A = \pi_B) \Leftrightarrow (\pi_B = \pi_C)].$$

Lemma 3. *Exactly, one of A, B, C has odd parity with respect to the polygon formed by the other two.*

Proof. Assume first that C has odd parity for $P(A,B)$. We show that A has even parity for $P(B,C)$. Starting from a point in the plane where all parities are equal, approach $A \cup B \cup C$. We first meet a point of A or B. If we meet A first, the result is immediate. Say we meet B first. This means that $\pi_A = \pi_C$ along B. Also, we are given that $\pi_A \neq \pi_B$ along C. Analyzing a small disk around a point where A, B, and C meet, we see that these conditions give $\pi_B = \pi_C$ along A.

In the other direction, we show that if C has even parity for $P(A, B)$, and B has even parity for $P(A, C)$, then A has odd parity for $P(B, C)$. This is established in a similar manner, by examining a small disk around a point where A, B, and C meet.

2.5 Joining a Polygon Top to Bottom

In the following lemma, we join the "top" segment of a polygon to the "bottom" part by a linear segment in the interior region.

Lemma 4. *Let P be a polygon. Let L be a vertical line in the plane such that there exists a point $p \in P$ to the left side of L and a point q to the right side of L. Let A and B be the two polygonal arcs with endpoints at p and q (so that $P = A \cup B$, and $\{p, q\} = A \cap B$). Then there is a segment $L' \subset L$, that (except for its two endpoints on P) lies in an interior region formed by the polygon and such that one endpoint of L' lies on A and the other on B.*

We have seen from Lemma 1 that there is a well-defined parity function for crossings of L with P. That is, parities are well-defined even if L passes through a vertex of P.

Proof. As we move along L, we intersect one of the branches of the curve (A or B) with a certain parity, then the other branch with a certain parity, then the first again, and so forth. We write this sequence of parities as

$$a_1, b_1, a_2, b_2, \ldots$$

(where we swap A and B if necessary to make a_1 first in this sequence). Since A runs from p to q which are on opposite sides of L, the sum of the parities a_i is odd. Similarly, the sum of the parities b_i is odd. This means that we can split the sequence of parities after the first odd term. For example, if a_1 and b_1 are even, but a_2 is odd, we split the sequence into a_1, b_1, a_2 and $b_2, a_3, \ldots$. Geometrically, let L' be the segment of L that connects the last crossing of the first group (say, a_1, b_1, a_2) with the first crossing of the second group (say $b_2, a_3, \ldots$). It is clear that L' has the desired properties.

2.6 Interior of Perturbed Polygons

This subsection shows that an interior point of a polygon is also an interior point to a second polygon obtained by perturbing the first.

Lemma 5. *Let P be a (time-parameterized) polygon. Let p lie in the complement of P. Let D be the distance from P to p. Let P' be a (time-parameterized) polygon such that for all t,*

$$d(P(t), P'(t)) < D.$$

Then the parity of p with respect to P equals the parity of p with respect to P'.

Proof. By perturbing p to a nearby point (which we continue to call p) in such a way to preserve the inequality of the hypothesis, we may assume that the horizontal line L_H through p meets P in finitely many points. Let these points be

$$q_1 = P(t_1), \quad q_2 = P(t_2), \ldots$$

listed in cyclic order around P. The polygonal arc $P_i \subset P$ from q_i to q_{i+1} meets the vertical line L_V through p with parity π_i. Let $q'_i = P'(t_i)$, P'_i, π'_i be the corresponding quantities for P'.

Each of the points q_i is at distance greater than D to the left or to the right of p. Since P' approximates P within D, each of the points q'_i is to the left or right of L_V, according to whether q_i is to the left or right. Thus, the parity π_i, which is the number of left-right crossings of L_V, is equal to π'_i.

Since q_i and q_{i+1} are consecutive points on L_H, all of the crossings of L_V and P_i lie distance $> D$ above L_H or all lie distance $> D$ below L_H. Since P' approximates P within D, all the crossings of $P'_i \cap L_V$ are above or below L_H according to what happens to P_i.

Since $\pi_i = \pi'_i$, and all of the crossings of L_V are above or below in the same for both P_i and P'_i, for each i, it follows that the parity of p with respect to P is the same as the parity of p with respect to P'.

3 Tubes

In this section we construct a system of tubes around the edges of a polygon P. Let $r > 0$.

Let e be an edge of P. On both sides of e, construct an edge parallel to e at distance r and of the same length as e. Then cap both ends with a semicircle. This is T_e. This set is precisely the set of points that have distance r from e. Let T be the union of all the tubes, for all the edges of P.

We will use these tubes only for generic values of r. This means that through any point x in the plane, there are at most two tubes that pass through x and the intersection of these two tubes is transverse (an intersection of two curves, each a semicircle or line segment). Furthermore, for generic r, there exist only finitely many points x that meet more than one tube. (Call these points *jump points.*) The generic points are dense in the positive reals.

Let U be a component of the complement of T with the property that it contains a point at distance greater than r from P. We claim that every point of U has distance greater than r from P. Otherwise, there is a point $p \in U$ that lies in the convex hull of some T_e. (When we speak of the interior of a tube, we mean the set of points in the convex hull of T_e but not in T_e itself; there is no parity function for tubes.) If p is interior to the tube, then the entire component is interior to the tube, contrary to the assumption that some point of U has distance greater than r from P.

If q lies on the boundary of U, then it lies on some tube T_e and its distance to e is exactly r. Its distance to P is at least r, so this means that the closest point to q on P is a point on e. If q is not a jump point, then this closest point on P is unique. If q is a jump point, then q has distance r from exactly two points of P.

The segment from a point on the boundary of U to its closest point on P does not meet P, except at that closest point.

Since U is connected and does not meet P, the parity function with respect to P is constant on the connected set U.

The boundary of U is formed by a finite collection of circular arcs and line segments. We may assume that these arcs and segments extend as far as possible along the boundary of U, so that each circular arc and segment is terminated on both ends by a jump point (or by an endpoint of a tube's semicircle).

Remark 2. Let p and q be two distinct points having distance exactly r from P. Let $\bar{p}$ and $\bar{q}$ be points on P at distance r from p and q, respectively. We claim that the line segments $p\bar{p}$ and $q\bar{q}$ do not meet (except at $\bar{p}$, when $\bar{p} = \bar{q}$). In fact, since the distance from p (resp. q) to P is exactly r, we see that the distances from p to $\bar{q}$ and from q to $\bar{p}$ are at least r. Thus, the perpendicular bisector of the segment $\bar{p}\bar{q}$ separates $p\bar{p}$ from $q\bar{q}$.

3.1 Separating Polygons

We continue in the same context, with a polygon P, a generic r, a union of tubes T, and a component U of the complement of T that has a point of distance greater than r from P. Let V also be such a component.

Lemma 6. *Assume that the parity function takes the same value on U and V. Let p and q lie on the boundaries of U and V respectively. Assume that $p \neq q$. Suppose that there exists $m \in P$ such that its distance to p is r, and its distance to q is also r. Then one of the following two options holds:*

- *$U = V$ and p and q are connected by a circular arc in the boundary of U.*
- *There exists a polygonal arc C of length at most $2r$ with endpoints on P and not otherwise meeting P, such that p and q have different parities with respect to each of the two polygons formed by C and P (that is, polygons $P(A, C)$ and $P(B, C)$ of Section 2.4).*

(Note that in the second option, every point of C has distance at most r from P. In particular, it cannot meet a component such as U whose points have distance greater than r from P.)

Proof. The point m lies on the edge e and p and q lie on the tube T_e, for some e.

If p is not on the tube's semicircles, then there are exactly two points on T_e at distance r from m. These points are p and q. The line segment pq meets e transversally at m, and does not otherwise meet P. This shows that p and q have opposite parities with respect to P. This is contrary to the assumption that U and V have the same parity.

Thus, p and q both lie on the same semicircle. We can follow the contour of the semicircle from p to q, staying in the boundary of the same component U, unless there is a jump point s between them on the semicircle. If the first case of the conclusion of the lemma fails to hold, we will use the jump point s to show the second case of the conclusion holds. Let m and m' be the points of P closest to s. The polygonal arc from m' to s to m then joins P to P, has length $2r$, and separates p from q (except in the degenerate case when $s = p$ or $s = q$). In the degenerate case when $s = p$ or $s = q$, we need to take a small polygonal perturbation of this path – still of length at most $2r$ – to avoid passing directly through s. This can be done, since r is generic, and the two tubes passing through s have overlapping interiors.

3.2 Single Component

We continue in the context of the preceding subsection, with a polygon P, a generic r, a union of tubes T, and a component U of the complement of T that has a point of distance greater than r from P. Let V also be such a component.

The next lemma is a key result. It gives a set of conditions that are sufficient to ensure that two components U and V of the same parity are equal. The idea is quite simple. We trace out the boundary of U. As we do this, we trace out time parameters t of the closest points on P to these boundary points. There is a jump in time parameter at every jump point of the boundary of U. However, the hypotheses of the lemma ensure that these time jumps are small. The time jumps are so small that they cannot contain a time parameter t coming from a boundary point of U or V. In this way, we find that the boundary of U "seizes essential control of all the time parameters." Consequently, the time parameter of a boundary point of V can then be matched with a time parameter of a boundary point of U. This allows us to show that U and V have a boundary point in common. For a generic tube size r, this implies that the components U and V are actually equal.

In my view, the biggest (and rather harmless) omission in Jordan's original proof was in failing to state the details about why the "boundary of U seizes essential control of the time parameters." (This is Case 3 in the proof provided below.)

Lemma 7. *Let U, V, r, T, P be as above. Let $R > 0$. Assume U and V have the same parity with respect to P. Assume that V contains a point that has distance greater than $\max(r, R)$ from P. We make the following additional three assumptions:*

1. *There is an interior point of P that has distance greater than R from P.*
2. *U contains a point that has distance greater than $\max(r, R)$ from P.*
3. *For every polygonal arc C with endpoints $(P(t_1), P(t_2))$ on P, not otherwise meeting P, of the same parity as U, and of length at most $2r$, the time parameters t_1, t_2 satisfy $t_1 < t_2$ and $t_2 - t_1 < 1/2$ (after interchanging t_1 and t_2 if necessary). Moreover, if we let A be the image of $[t_1, t_2]$ under P, then the polygon $P(A, C)$ formed by A and C lies in a disk of diameter R.*

Then under these assumptions, $U = V$.

Proof. Let X be the set of $t \in \mathbb{R}/\mathbb{Z}$ such that there is a point p on the boundary of U such that $P(t)$ is a closest point on P to p. Each line segment in the boundary of U contributes an interval to X. Each circular arc on the boundary contributes a point to X. The set X is closed. Each point on the boundary of U that is not a jump point gives a well-defined point on X. Each jump point gives precisely two points t_1 and t_2 on X. The points $P(t_1)$ and $P(t_2)$ both have distance r from the jump point, and are thus connected by a polygonal arc C satisfying the third assumption of the lemma. Thus, we may assume that $t_1 < t_2$ and $t_2 - t_1 < 1/2$. Let $Y \subset \mathbb{R}/\mathbb{Z}$ be the union of these intervals (t_1, t_2) as we run over the jump points on the boundary of U.

Let q lie on the boundary of V, and let $P(t_q)$ be a point on P closest to q. We consider three cases $t_q \in X$, $t_q \in Y$, and $t_q \notin X \cup Y$.

Case 1: Assume that $t_q \in X$. By the definition of X, there is a point p on the boundary of U that also gives $t_q \in X$. By Lemma 6, there are two possibilities. One is that q is joined to p by a circular arc in the boundary of U. In particular, q lies in the boundary of both U and V. Since r is generic, this implies that the components U and V are equal. The other possibility is that there is a polygonal arc C of length at most $2r$ joining P to P and separating p from q. By parity arguments, one of V or U lies inside $P(A, C)$, hence inside a disk of diameter R. This is impossible, since both U and V have points farther than R from P. Thus, $t_q \notin X$.

Case 2: Assume that $t_q \in Y$. To dismiss this case, we show more generally that for any q' in the boundary of $V' \in \{U, V\}$, and such that $P(t_0)$ a closest point to q', we have $t_0 \notin Y$. In particular, X and Y are disjoint. Assume to the contrary, that $t_0 \in Y$. This means that $t_0 \in (t_1, t_2)$, where $P(t_1), P(t_2)$ are the closest points on P to a jump point p on the boundary of U. The two line segments $pP(t_i)$ form a polygonal arc C joining P to P of length $2r$. The component V', and the polygonal arc C have the same parity with respect to P.

We have $P(t_0) \in A$, where A is the image of $[t_1, t_2]$ under P. Let B be the other branch of P from $P(t_1)$ to $P(t_2)$, so $P = A \cup B$. The assumptions on the diameters of U, V and $P(A, B)$ prevent V' from lying in the interior of $P(A, C)$. By Lemma 2, V' has the same parity with respect to $P(B, C)$ as it does with respect to $P = P(A, B)$.

The line segment from q' to $P(t_0) \in A$ does not meet C (by Remark 2). This line connects V' to A without crossing $P(B, C)$. Thus, A (excluding endpoints) has the same parity with respect to $P(B, C)$ as V'. We conclude that the parity of A with respect to $P(B, C)$ is the parity of C with respect to $P(A, B)$. We can now apply By Lemma 3, which forces the parity of B to be odd with respect to $P(A, C)$. Thus, A, B, and $P = A \cup B$ are all contained in a disk of radius R. This is inconsistent with the first of the list of three assumptions of the Lemma.

Case 3: Finally, consider the case $t_q \notin X \cup Y$. This implies that $X \cup Y \neq \mathbb{R}/\mathbb{Z}$. The set $X \cup Y$ is constructed as a union of closed intervals, with endpoints in X. This implies that there is $t' \in X$ such that $P(t')$ is a closest point to a jump point p' in the boundary of U and such that there exists an open interval in $\mathbb{R}/\mathbb{Z}$ that has endpoint t' and is disjoint from $X \cup Y$. We consider two cases, depending on whether t' is an isolated point in X, or the endpoint of a non-trivial closed interval.

If t' is an isolated point in X, then it comes from a circular arc of the boundary of U. Along the boundary of U, both endpoints of this circular arc are jump points, and these two jump points give time parameters $t'', t''' \in X$ with $(t'', t'), (t', t''') \subset Y$. We may assume that $t'' < t' < t'''$. (For example, we cannot have $t'' < t''' < t'$, because Y and X are disjoint by Case 2, so $t''' \notin (t'', t')$.) We have $(t'', t''') \subset X \cup Y$. This is contrary to the hypothesis that t' lies on the boundary between $X \cup Y$ and its complement.

If t' is the endpoint of an interval I in X, then it comes from a line segment in the boundary of U. Since t' is a jump point, there exists t'' such that $P(t'')$ is a second point on P closest to that boundary point of U. Again, by Case 2, t'' must lie on the opposite side of t' from I. The sets $(t', t'') \subset Y$ and I cover a neighborhood of t', contrary to the hypothesis that t' lies on the boundary between $X \cup Y$ and its complement.

4 Polygon Approximation

We are now at an advanced stage of the proof, and yet everything so far has been about polygons. To proceed further, we need to prove that every Jordan curve can be approximated by polygons. That is the purpose of this section.

4.1 Uniform Continuity

To avoid a proliferation of deltas and epsilons, we introduce a special notation, c and c', for two of the deltas. The uniform continuity of a Jordan curve J can be expressed by the existence of a delta (which we call c) (depending on J) such that

$$\forall \epsilon\ t\ t'.\ \epsilon > 0 \wedge d(t, t') < c(\epsilon) \ \Rightarrow\ d(J(t), J(t')) < \epsilon.$$

By redefining $c(\epsilon)$ to be even smaller if necessary, we may assume that for all $\epsilon > 0$, we have $c(\epsilon) < 1/2$. J is a homeomorphism from its domain to its image. The uniform continuity of J^{-1} can be expressed by the existence of a delta c' such that

$$\forall \epsilon\ t\ t'.\ \epsilon > 0 \wedge d(J(t), J(t')) < c'(\epsilon) \ \Rightarrow\ d(t, t') < \epsilon.$$

The following lemma tells us how to ensure that an arc of a Jordan curve lies near $J(t)$ by controlling the location of its endpoints.

Lemma 8. *Let J be a Jordan curve. Let $\epsilon > 0$, and let t, t' be given that satisfy*

$$d(J(t), J(t')) < c'(c(\epsilon)).$$

Then, (adopting our conventions that $c(\epsilon) < 1/2$, $t < t'$, and $t' - t \leq 1/2$) for all $t'' \in [t, t']$, we have

$$d(J(t), J(t'')) < \epsilon.$$

Proof. Combine the uniform continuity of J and J^{-1}.

4.2 Approximation

Lemma 9. *For every (time-parameterized) Jordan curve J and $\epsilon > 0$, there is a time-parameterized polygon P such that $d(J(t), P(t)) < \epsilon$ for all t.*

Proof. Conceptually, the proof is very simple. Form a polygonal path P_N by joining $J(i/N)$ to $J((i+1)/N)$ (with each separate line segment parameterized with constant speed) for some large N and $i = 0, \ldots, N-1$. This path needn't be simple, but there is a subset of the image of P_N that avoids self-intersections and approximates J to within ϵ.

First we note that for any positive ϵ, there is N sufficiently large, so that the polygon P_N approximates J to within ϵ at each time t.

Using the conventions for parameter values t, t', we see that if $d(J(t), J(t')) < c'(c(\epsilon))$, then $d(J(t), J(t'')) < \epsilon$ for all $t'' \in [t, t']$. More strongly, for $\epsilon > 0$, we can find N such that for any double point $P_N(t) = P_N(t')$, we have $d(P_N(t), J(t'')) < \epsilon$ for $t'' \in [t, t']$. That is, the curve J stays within ϵ of the double point $P_N(t)$ throughout that interval. We excise the closed loop of P_N on $[t, t']$ making the path stop at the double point $P_N(t)$ during the time period $[t, t']$. The resulting map remains within distance ϵ of J at all times.

To make a consistent excision of all the self-intersections, from the finite collection of all such intervals, and pick one that is not contained as a subinterval of any other. Excise it, then repeat the process from the beginning. By excising an interval at each stage that is nested in no other, the iterative process picks out a disjoint collection of intervals to excise. The process terminates because the number of self-intersections drops with each iteration.

The resulting subset of $\mathbb{R}^2$ is the desired polygon, but poorly parameterized: a path that stops for a moment each time it reaches an excised interval is not simple. However, this is a simple matter to fix. Near each excised interval, reparameterize so that instead of remaining constant the values progress in a strictly monotone manner, changing the parameter values by such a small perturbation that the ϵ-approximation still holds. This completes the proof.

5 Constructing an Interior Point

Let J be a Jordan curve. The purpose of this section is to construct a point θ_0 that lies in the interior region of every sufficiently close polygonal approximation to J.

5.1 Constants β and w

In this section, we attach positive real numbers β and w to a Jordan curve J, as well as two special points p and q. These will be used later in this section.

Given a Jordan curve J, we define $w = w_J > 0$ to be the width of the narrowest vertical strip containing J. Fix this strip.

Pick p on J along the left edge of the strip and q on J along the right edge of the strip. There are distinct arcs A and B in J that connect p to q.

If we pick coordinates so that the y-axis is the left end of the strip, we can find a constant $\beta > 0$ such that for every point $a \in A$ and $b \in B$ whose x-coordinates lie in $[w/3, 2w/3]$, we have $d(a, b) \geq \beta$. Making β smaller if necessary, we can assume that

$$w/3 > \beta.$$

5.2 Constructing θ_0 and θ_∞

Lemma 10. *Let J be a Jordan curve. Attach data β, w, p, q, A, B to J as above. Pick ϵ so that $0 < \epsilon < \beta/2$. Let P be an ϵ-approximation to J. Set $p' = P(t)$ and $q' = P(t')$, where $p = J(t)$ and $q = J(t')$. Let A' and B' be the polygonal arcs in the image of P, from p' to q' (corresponding to A and B for J). Let L be a vertical line bisecting the vertical strip fitting J. Let L' be the segment on L determined by Lemma 4 (applied to P, L, p', q', A', B'). Then some point θ_0 along L' has distance at least $\beta/2 - \epsilon$ from A' and B'.*

Proof. Each point along L' has distance greater than $w/6 > \beta/2$ from points on J outside the vertical strip between $[w/3, 2w/3]$ to the right of p. Let U_A (resp. U_B) be the subset of the plane consisting of points at distance less than $\beta/2$ from A (resp. B). U_A and U_B are open. Both have nonempty intersection with L', because each contains an endpoint of L' by construction of Lemma 4. Their intersection along L' is empty, because such a point would put A at distance less than β from B within the vertical strip $[w/3, 2w/3]$. By the connectedness of L', we conclude that there is some point θ_0 of L' that is not in $U_A \cup U_B$. It has distance at least $\beta/2$ from A and B, so distance at least $\beta/2 - \epsilon$ from A' and B'.

Let θ_0 be the point that the lemma shows to exist (for some small ϵ). By Lemma 5, it lies in the interior of P' for all sufficiently close approximations P' of J.

Let θ_∞ be a point "at infinity", that is, any point far away from "all the action" in the proof. It will then be in the exterior region of P' for all sufficiently close approximations P' of J.

We let α be an index that runs over the set $\{0, \infty\}$, and write θ_α for the corresponding points.

6 Constructing the Interior and Exterior

We are now ready to give the main argument in the proof of the Jordan curve theorem. Let J be a Jordan curve. Let θ_α be the points constructed in the previous section.

For each $n \in \mathbb{N}$, and $\alpha \in \{0, \infty\}$, we will construct nonempty connected open regions U_α^n as the connected component of θ_α in the complement of some finite union T^n of tubes. The regions will have the following key properties:

1. $U_0^n \cap U_\infty^n = \emptyset$
2. $U_\alpha^n \cap J = \emptyset$

3. $U^n_\alpha \subset U^{n+1}_\alpha$
4. $\mathbb{R}^2 \setminus J \subset \bigcup_{n,\alpha} U^n_\alpha$.

Theorem 1. *(Jordan curve theorem) Let J be a Jordan curve. The complement of the image of J is the union of two disjoint nonempty open connected sets.*

In the words of Jordan, *"toute courbe [fermée] continue [et sans point multiple] divise le plan en deux régions, l'une extérieure, l'autre intérieure, cette dernière ne pouvant se réduire à zéro"* [5, p. 99].

Lemma 11. *If regions U^n_α can be constructed satisfying these four properties, then the Jordan curve theorem holds.*

Proof. Set $V_\alpha = \bigcup\{U^n_\alpha \mid n\}$. We claim that V_0 and V_∞ are the desired disjoint nonempty open connected sets. First, V_α is open because it is the union of open sets. It is nonempty, because it contains the point θ_α. The set V_α is connected because it is the union of connected sets containing the common point θ_α. The sets lie in the complement of J by Property (2), and give the full complement by Property (4). Finally, V_0 and V_∞ are disjoint by Properties (1) and (3).

We pick a sequence of positive real numbers r_n tending to 0. Pick r_0 small enough for the construction of the point θ_0 to work. Then pick

$$r_{n+1} < r_n/5.$$

Take P^n to be a $r_n/4$-approximation to J given by Lemma 9, and T^n to be the union of tubes around P^n for a generic parameter $r'_n \in (r_n/2, r_n)$. Take U^n_α to be the component of the complement of T^n_α complement containing θ_α.

To complete the proof of the Jordan curve theorem, it is enough to prove four lemmas, showing that Properties (1)–(4) hold for these choices.

6.1 Verifying the Four Properties

Lemma 12. *Property (1) holds for these choices. That is, $U^n_0 \cap U^n_\infty = \emptyset$.*

Proof. Points of U^n_0 and U^n_∞ have different parities with respect to the polygons P^n.

Lemma 13. *Property (2) holds for these choices. That is, $U^n_\alpha \cap J = \emptyset$.*

Proof. U^n_α is the θ_α-component of $\mathbb{R}^2 \setminus T^n$. Every point of U^n_α has distance greater than $r'_n > r_n/2$ from P^n (as in Section 3). However, every point of J lies within distance $r_n/4 < r_n/2$ from P^n.

Lemma 14. *Property (3) holds for these choices. That is, $U^n_\alpha \subset U^{n+1}_\alpha$.*

Proof. We claim that every point of the boundary of U^{n+1}_α lies strictly within distance r'_n from P^n. In fact, to bound this distance, go from a boundary point to P^{n+1}, then to J, then to P^n, which have respective distances bounded by

$$r'_{n+1} + r_{n+1}/4 + r_n/4 < 5r_{n+1}/4 + r_n/4 < r_n/4 + r_n/4 < r'_n$$

by the recursion inequality for r_n. However, the points of the boundary of U^n_α have distance exactly r'_n from P^n. So $U^n_\alpha \subset U^{n+1}_\alpha$.

Lemma 15. *Property (4) holds for these choices. That is, $\mathbb{R}^2 \setminus J \subset \bigcup_{n,\alpha} U^n_\alpha$. More precisely, for any $\epsilon > 0$, there exists an n such that U^n_α contains all points x of $\mathbb{R}^2 \setminus J$ whose distance from J is at least ϵ and whose parity with respect to P^n is the same as that of θ_α.*

Proof. Fix α. Pick $\epsilon > 0$. We may assume that ϵ is less than the distance between θ_β and J, for $\beta = 0, \infty$. We pick n large enough so that the following conditions hold. (These conditions are listed in matching order with the hypotheses of Lemma 7, with R instantiated to ϵ.)

- The interior point θ_0 of P^n has distance greater than ϵ from P^n. (This can be arranged for large n because the distance of θ_0 to J is greater than ϵ.)
- The point $\theta_\alpha \in U^n_\alpha$ has distance greater than $\max(r'_n, \epsilon)$ from P^n. (For example, take n large enough that $r'_n < \epsilon$, then use the facts that the distance from θ_α to J is greater than ϵ, and that P^n approximates J.)
- Every pair of points $P^n(t_1), P^n(t_2)$ on P^n whose separation is at most $2r'_n$ satisfies $t_1 < t_2$ and $t_2 - t_1 < 1/2$ (after interchanging t_1 and t_2 if necessary). (This is just the uniform continuity properties of J^{-1} from Section 4.1, combined with the fact that P^n gives a sequence of approximations to J.) Moreover, for every polygonal arc C with endpoints $P^n(t_1), P^n(t_2)$, of the same parity as U^n_α, and of length at most $2r'_n$, the image A of P^n on $[t_1, t_2]$ and C lie in a disk of diameter ϵ. (To get this condition, apply Lemma 8 applied to J, and polygonal arcs of length less than some δ, to get a closed curve inside a disk of diameter $\epsilon' < \epsilon$. Then sufficiently close approximations P^n will yield data inside a disk of diameter ϵ.)

Fix n satisfying these conditions. Let V be the connected component in $\mathbb{R}^2 \setminus T^n$ of an element x as given in the statement. Applying Lemma 7 to U and V, we see that $U = V$. So $x \in U$.

7 The Isoperimetric Inequality

We briefly sketch a proof of the isoperimetric inequality, based on Jordan's proof of the Jordan Curve Theorem.

Corollary 1. *Let J be a Jordan curve of finite one-dimensional Hausdorff measure b. Let a be the area of the interior region. Then*

$$4\pi a \leq b^2.$$

Proof. It is enough to prove the weaker inequality for all $\epsilon > 0$:

$$4\pi(a - \epsilon) \leq b^2.$$

A lower bound on the length of J is the length of the piecewise linear curve obtained by choosing a finite list of points S sequentially along J and joining each consecutive pair of points of S by a line segment. Each polygon approximation P^n to J is obtained by removing a finite number of closed loops from such a piecewise linear curve for some $S = S_n$. (See Section 4.2.) Thus, the length of each polygon approximation P^n to J is no greater than the length of J. Hence the length b_n of each P^n is at most b. The regions U_0^n lie in the interior of P^n and J.

Let $\epsilon' = \epsilon/(\pi b)$. Cover J with finitely many disks whose radii r_i are less than ϵ' and that sum to less than b. (The sum of the diameters can be brought arbitrarily close to b. The sum of the radii can be brought arbitrarily closed to $b/2$.) Using Lemma 15, pick n large enough that the U_0^n and the disks cover the interior of J. Let a_n be the area of the interior of P^n. Then

$$a \leq a_n + \sum \pi r_i^2 < a_n + \pi \epsilon' \sum r_i \leq a_n + \epsilon.$$

The isoperimetric inequality now follows from the isoperimetric inequality for polygons:

$$4\pi(a - \epsilon) \leq 4\pi a_n \leq b_n^2 \leq b^2.$$

The isoperimetric inequality usually includes the statement that circles are the only rectifiable Jordan curve for which equality is obtained. Again, this is not difficult to prove, once the inequality is known. The deepest part of the proof of the isoperimetric inequality is the existence of a suitable polygon approximation, as provided by Jordan.

References

1. J. W. Alexander, A Proof and Extension of the Jordan-Brouwer Separation Theorem, Trans. AMS, 23(4), 1922, 333-349.
2. R. Courant and H. Robbins, What is Mathematics, second edition, Oxford, 1996.
3. T. Hales, The Jordan Curve theorem, both formally and informally, to appear in the Amer. Math. Monthly.
4. D. Hilbert, Foundations of Geometry, Open Court, 1971.
5. C. Jordan, Cours D'Analyse de l'École Polytechnique, Paris, second edition, 1893.
6. V. Kanovei and M. Reeken, A nonstandard proof of the Jordan curve theorem, preprint 1996.
7. J. R. Kline, What is the Jordan Curve Theorem? Amer. Math Monthly, 49, no. 5, 281–286.
8. M. Kline, Mathematical thought from ancient to modern times. Vol. 3. Second edition. The Clarendon Press, Oxford University Press, New York, 1990.
9. M. H. A. Newman, Elements of the Topology of Plane Sets of Points, Dover, 1992.
10. D. Reginald Traylor, Creative Teaching: The Heritage of R. L. Moore http://at.yorku.ca/i/a/a/b/21.htm
11. W. F. Osgood, Lehrbuch der Funktionentheorie, Teubner, 1912.
12. M. Reeken, private communication, 21 Feb 2005.
13. O. Veblen, Theory on Plane Curves in Non-Metrical Analysis Situs, Trans. AMS, 6, No. 1 (1905), 83–98.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 10 (23) 2007

Some Remarks on The Language of Mathematical Texts*

Zinaida Trybulec and Halina Święczkowska

University of Białystok
Białystok, Poland
hswieczkowska@wsap.bialystok.pl

From the authors

We want to remind the readers of an article published nearly fifteen years ago in the joint issue 10/11 of "Studies in Logic, Grammar and Rhetoric". The decision about its reprint in the same periodical is not due to publishing of a special edition of "Studies..." devoted to the Mizar project and its author Andrzej Trybulec, even though dr Trybulec is the central figure of this text. It was him who initiated the subject and was the critical consultant of the whole article, which in spite of its limited influence due to a restricted circulation of the issue and its local character at that time, was quoted and referred to especially in texts of researchers working on formalization of the language of mathematics. Even if we treat this text as a yet historical contribution to the discussion on properties of the mathematical discourse, we are convinced that some of the ideas contained in it are still topical and can become a subject of a detailed analysis anew, particularly in the domain of studies concerned with standardization of the language of mathematics.

In mathematical literature, in popular science texts, in the teaching of mathematics, in the literature concerned with scientific methodology, and in handbooks of stylistics we encounter the concept of "the language of mathematics".

* The present paper is intended as a discussion of selected problems related to the language of mathematical texts and the ways of their organization. The complexity of mathematical statements – at both the language level and the text level – prevents us from dealing in this paper with all detailed issues related to the main subject matter. Hence we focus our attention on finding a proper definition of the language used to describe mathematical objects and their properties. We also try to answer the question about the universal nature of the language of mathematical texts by analysing its semantics and the properties of mathematical symbolism (mainly exemplified by variables).

ISBN 978-83-7431-128-1 ISSN 0860-150X

The conviction that mathematics has a language of its own is so common among mathematicians that the self-evident distinctive characteristics of speaking about mathematical objects absolves those who use that term from offering its precise definition. Some authors formulate an approximate meaning of that term, but their formulations are not in full agreement with one another and they partly obscure the picture of that self-evident reality. It is said that the language of mathematics is a set of formulas intermixed with sentences in natural language [6]. It is emphasized that the language of mathematics is a largely heterogeneous product, a set of languages, each of which refers to a different branch of mathematics [5]. It is also claimed that the language of mathematics is a set of symbols and the rules of using them, completed by the words and grammar drawn from natural language [3]. From some others, finally, it is merely a formalized language with a precisely defined vocabulary and syntax.

The distinctive manner of speaking about, or describing, mathematical objects accounts for the fact that the language of mathematics is often opposed to languages used in other disciplines, which – in the opinion of mathematicians – avail themselves of a freedom of expression which is denied to them. "The terms and expressions in those works (i.e., pertaining to domains not related to mathematics – Z.T. & H.Ś.) need not be precisely defined, and theorems should be only roughly true. Mathematicians suffer from their conviction that terms without precise definitions are meaningless, and statements which are not true are false" [11]. We shall not engage in a polemic with the opinion of the author quoted above, but we have to note that in works on stylistics the style of mathematical texts is singled out as something distinctive. On the other hand, no references are made to the languages of such disciplines as medicine, geology, or agronomy.

The concept of the language used in described mathematical reality is understood in various ways, according, among other things, to the purposes for which language is studied. Definitions of language usually, if not always, are constructed for natural language or for an artificial language as something which in some sense is opposed to ethnic languages.

If we want to speak about the language of mathematics, then we must be convinced about its systemic nature. If this is so, then it should be subsumed under the general definition of language used by linguists. According to one such definition, language is a system of signs used to convey thoughts, that is serving the purpose of communication among members of a given community. This definition is applied to natural language, but one could hardly claim that it suffices to describe the language of mathematics. First of all, even if we agree that by community we mean the community of mathematicians only, we face the fact that researchers who specialize in a single domain of mathematics need not be familiar with symbolism used in other branches of mathematics. This is so because it is claimed that in mathematics we do not have to do with any single language, and that in fact every branch of mathematics uses a language of its own and its specific symbolism. And even if one accepts the opinion that the language of mathematics is merely a specialized fragment of natural language, then that opinion is weakened by the statement that language is even less known than natural language and that the tools used in linguistics prove insufficient for the analysis of its properties [5].

Difficulties emerge already when we try to fix the alphabet of that language. It may be said to consist of all letters of a given ethnic language plus special symbols and letters drawn from alphabets of other languages. eg., Greek, Latin, Hebrew, and Gothic letters, useful in denoting functions, variables, constants, etc. The vocabulary of the language of mathematics is a set of expressions which differ from both those occurring in the everyday language and those used in other disciplines. It includes strictly mathematical terms, terms drawn from everyday language but used in a new, specialized sense, and words drawn from everyday language useful in the construction of mathematical statements. Note in this connection the characteristic processes in the sphere of word-formation and the productivity of some formants, such as **sub-** in such terms as **subset**, **subgroup**, **subdomain**, **subbundle**. Peculiarities are also observed in expressions which are at variance with standard syntactic rules, such as "**f** is a mapping onto ...", "**f** is a mapping into ...", "**f** is a transformation onto ...". Such constructions, plus expressions such as "for almost all", "there are arbitrarily large natural numbers", "the function takes on the value zero almost everywhere", "if and only if", have a fixed meaning in the language of mathematics. Hence, it seems, it is not always correct to compare the vocabulary of the language of mathematics with units from the morphological level of any ethnic language. If we include grammar, then we may merely say that the so-called language of mathematics maps or assumes the grammatical system of natural language in those its fragments in which mathematical statements meet the criteria of linguistic correctness imposed upon it by the grammar of the ethnic language in which those statements are formulated.

It seems neither useful nor correct to treat the language of mathematics as natural language. Nor would it be right to consider the language of mathematical texts as a formalized language in the sense of a language with a precisely definite and explicitly described syntax, in spite of the endeavours to write mathematical texts using such a language. Most mathematical texts show a considerable freedom in expressing mathematical ideas, which is due to the fact that if we know well what are we talking about, then it is less important how we convey that.

*

*   *

The term "language of mathematics", as has been shown above, has no sufficiently precise meaning. But we have at our disposal a wealth of empirical data in the form of mathematical texts written in various ethnic languages. It may be assumed that those texts, for all the differences of content, are marked by a certain similarity. But it would be rather risky to claim that mathematical texts comply with a linguistic system that might be termed the system of the language of mathematics. That would imply the necessity of describing that language in accordance with linguistic criteria or by methods adopted for the description of formal systems.

In view of definitional problems we want to study the language used in the formulation of mathematical statements. We accordingly suggest, for that purpose, to replace the term "the language of mathematics" by the term "the language of mathematical texts". Terminological decisions necessitate the explanation of the goal of the study of the subject matter so defined. Note that the study of texts in natural language may, on the one hand, serve the purpose of constructing a model of the linguistic system of a given language, but on the other, the data available may serve as the foundation for the formulation of a theory of text. If we assume that the language of mathematics does not exist in such a form as an arbitrary ethnic language, and that it is not legitimate to speak about it as a formalized language, we thereby preclude the possibility of constructing a system. In our opinion, mathematical texts are utterances which comply not so much with a certain linguistic system (even though it is obvious that they reproduce fragments of some systems) as with certain patterns of texts, a universal structure filled with content suitable for the described fragment of mathematical reality. Our task is to reveal that structure and to show features which are common to all kinds of mathematical statements. Hence the present paper has its place in that trend of study which is usually termed text theory.

The language of mathematical texts is a language in which people formulate statements pertaining to the various branches of mathematics. Such statements are sequences of signs, usually in a written form, although they may take on the form of, say, a spoken lecture. When referring to mathematical texts we shall hereafter mean their written form. Mathematical texts take on several stylistic variations, whose choice depends on the scope of the problem presented by a given author and the intended group of readers. These variations are: paper (article), monograph, and handbook. We want here explicitly to oppose strictly mathematical texts, that is those whose content refers to mathematical objects, to texts in which the mathematical apparatus is used to explain or to interpret non-mathematical domains.

Let us reflect here on the criteria which make it possible to distinguish mathematical texts from the remaining ones, drawn from other disciplines. It is common knowledge that in mathematical texts symbols play an essential role. The proper choice of such symbols is no less important, because an incorrect choice of symbols not only makes mathematical texts illegible, but makes them fail to perform their main function: it is not conductive to solving intricate problems. A contemporary mathematical text lacking symbolism altogether is unthinkable, either.

Hence it might seem that it is symbolism which is the main element (or even discriminant) of mathematical texts, and an average reader who looks on shelves for a non-mathematical text is guided above all by that criterion. He divides texts into those who lack symbolism and classes them as non-mathematical ones, and those which include it, and treats them as mathematical ones.

The above criterion, simple in application as it is, does not, however, seem satisfactory. It is true that mathematical texts include symbolism, but not every text which includes it is a mathematical one. For instance, chemical texts abound in symbolism, as also do those concerned with medicine, physics, and biology, and the same even applies to linguistic texts, seemingly so free from mathematical elements. It suffices to inspect some philosophical texts to treat them as mathematical one on the strength of that criterion, and yet we would protest against consider chemical and philosophical texts as mathematical ones.

An even cursory inspection of mathematical texts allows us to conclude that in addition to symbolism to be found in them, they are formulated in some their parts in a language which is a variety of a given ethnic language. Even such a superficial contact with the language of mathematical texts shows that those parts which are recorded in symbols and those recorded in an ethnic language are in various proportions. One can encounter texts written almost like texts concerned with other disciplines, also completely deprived of symbolism or including a very small amount of symbols, as well as other texts abounding in symbolism so much that they are outright illegible to a person not already familiar with the domain to which they pertain. Mathematical texts, next to symbols and parts recorded in an ethnic language, may include diagrams and drawings, which must also be recognized as elements of the language of mathematics.

*

* *

In this paper by a mathematical text we mean, following L. A. Kaluzhnin [4], a sequence of symbols and verbal text in a definite ethnic language used in contemporary mathematical literature. Unlike scientific texts in other disciplines, a mathematical text has certain specific features. Note, for instance, that regardless of the ethnic language in which it is formulated, a reader who has at least an elementary mathematical training will recognize its organization, its structure, and a mathematician, even if he does not know the ethnic language in question, can also recognize its content.

As follows from the above, every mathematical text can be split into two parts. One of them is formal (representable in a formalized form), and the other, informal. The formal text, called the proper text by some authors, is usually presented in a language which is a mixture of a formalized language and an ethnic one. This is how a large number of theorems are recorded. The recording of the text proper in a formalized language alone makes that text difficult to read, because a large amount of information is concentrated in its small fragments. That is why the proper text is usually written in a language which might be termed hybrid. Definitions, theorems and proofs are such a formal text. Next to it in mathematical texts we find fragments of text written usually in an ethnic language; they convey the author's intuitions, interpretations of some fact, and serve as a *sui generis* connective tissue of the proper text.

Thus in the structure of a mathematical text we shall single out two strata: the formal (consisting of the formalized text) and the informal (intuitive). Following A. Trybulec ([12], [13]) we shall term the former the objective stratum, and the latter, the subjective one.

The objective stratum in a mathematical text differs quite visibly from the subjective one. It begins where it has its own name (definition, theorem, lemma, conclusion, postulate, proof) or is printed in a different type. The end of a proof is usually indicated by "q.e.d." or by some other sign. Some authors do not do that and confine themselves to stating in the paragraph that follows the proof that the proof has been carried out. Very often both definitions and theorems are numbered to make references to them easier in further parts of the text. For instance, in A. Grzegorczyk's *Outline of Theoretical Arithmetic* (in Polish), [2] definitions are preceded by abbreviations D1, D2, ...; axioms, by A1, A2, ...; theorems, by T1, T2, ...; rules, by R1, R2, ...; lemmas, by L1, L2 ... The beginning of the proof is marked "dowód", which in Polish means 'proof'.

The objective stratum of a mathematical text, and its subjective stratum, are – as in the case of symbolism and natural language – in various proportions. Some mathematical texts have the former stratum very expanded, as the just quoted text by A. Grzegorczyk (this usually applies to monographs and papers); others have a rich subjective stratum (handbook, teaching aids, popular papers).

Note that the primary goal of every mathematical text is to present its formal stratum, the secondary goal consisting in offering the reader a method which would allow him to incorporate that formal stratum with the knowledge he has already had and to keep it there as part of his working intellectual endowment. That secondary goal is attained by those parts of the text which we have termed the informal stratum.

The informal stratum of a mathematical text consists of those fragments of that text which include motivations, analogies, examples, and non-mathematical explanations. N. R. Steenrod [11], by analysing L. C. Young's work *Lectures on the Calculus of Variations* and the book by Hurewicz and Wailman, *Dimension Theory*, drew a list of the informal material which can be found in mathematical works. That list covers the introductory material, which should include a brief review of the basic material which forms the content of the formal structure. The intuitive stratum should also present motivations and an analysis of examples which suggest hypotheses. There is also a place in it for a cursory description of the results that will be arrived at, and the methods that will be used. The informal material should also include a review of the content of the book. It is worth noting, too, those fragments of the intuitive stratum which usually follow the formal material and discuss connections with other subjects or alternative approaches to the same problem. The formal material may be presented in various ways. The author is not required formally to expound the alternative solutions, but it is assumed that in such cases a brief description of the idea of the alternative solution should be included. If the work is intended as a handbook, then it usually has at the end material for exercises.

The intuitive stratum of the text is written in a language which relatively less often includes formulas and symbols used in the formal part. The principle of not using formulas and symbolism in the informal material is observed in particular in mathematical handbooks. It is claimed outright that the best way of expounding those fragments is to avoid all symbols [3]. A good example of the application of those principles can be seen in the handbook by J. Słupecki, H. Hałkowska and K. Piróg–Rzepecka, *Elements of Theoretical Arithmetic* (in Polish), Warsaw 1980.

It must be emphasized that some fragments of the intuitive stratum, especially examples and analogies, may be presented in a formalized form. This is due to the fact that we have there, for instance, to do with the substitution of a theorem proved earlier, or an equivalent, though merely outlined, way of presenting the formal material. The formalization of the entire intuitive stratum is usually difficult. On the other hand, the possibility of paraphrasing fragments of the formalized text in natural language is a certain regularity.

*
* *

Let us reflect on what accounts for the fact that the language of mathematical texts is also comprehensible to mathematicians who only poorly, or not at all, know that ethnic language in which a given text has been written. Why is it so that since the early years of the development of science it has been accepted that precisely that language should be the universal language of science, and why no ethnic language can become such a language (on which there is a general consent)? Where are we to look for the essential difference between the language of mathematical texts and ethnic languages? Has the language in which mathematical texts are written some peculiar properties of which other languages are deprived, and if so, then what these properties are? And what role in that language is played by symbolism, without which we can hardly imagine a contemporary mathematical text? Our working hypothesis is: the semantics of the language of mathematical texts accounts for its universal comprehension. That semantics differs essentially from the semantics of any ethnic language. One might also risk the statement that ethnic languages determine various perspectives of the world, whereas the language of mathematics has only one and the same such perspective[1].

The above hypothesis of course requires an at least superficial verification and argumentation. We assume that natural language makes use of continuous (analog) semantics, opposed to discrete (digital) semantics. Terms used in ethnic languages usually have a fuzzy (blurred) meaning (e.g., vague terms). This can be illustrated by the well-known fact that we are unable to indicate a precise boundary showing the use of colours in ethnic languages. We usually resort to gradation to indicate differences between colours that do not differ much from one another (for instance, *blue*, *dark blue*, *light blue*). Moreover, indicating such a boundary is impossible because there is a margin in which the boundaries would differ from one another even if they were described as precisely as possible. It may be said without exaggeration that, for instance, not to every Pole the blue colour is the same blue colour. Hence, it may be said in most general terms that the continuity of semantics finds reflection in non-eliminable vagueness. When speaking about the continuity of semantics we do not mean ambiguity, so characteristic of everyday language, because ambiguity does not cause fuzziness of meanings of words. Ambiguity occurs systematically in both natural language and mathematics. But that does not deprive mathematical texts of their semantic clarity. Ambiguity does not make semantics discrete, and the type of the semantics with which we have to do is determined by whether the boundaries between meanings of words are fuzzy or not.

That, for all ambiguities, the semantics of the language of mathematics remains discrete is illustrated, for instance, by the concept "kernel of transformation", which

[1] When speaking about the perspectives of the world determined by ethnic languages we refer to the hypothesis of linguistic relativity, formulated by Sapir and Whorf. That hypothesis includes the supposition that the ways of classifying objects in extralinguistic reality may be different in different languages. The Sapir-Whorf hypothesis states that languages as a social product shape our way of perceiving the world around us, and in view of the differences among linguistic systems people who think in those languages perceive that world in different ways. That hypothesis has not, it is true, turned into a theory in view of lack of convincing experiments that would confirm it, but has served as a stimulus for experimental research in field previously not covered by linguists.

in mathematics occurs in many meanings. We shall consider here several examples of its use.

(i) *The Encyclopaedia of Physics* [1] states: "KERNEL, in mathematics the function $K(x, y)$ in the following integral transformation

$$\varphi(x) = \int\limits_a^b K(x, y) f(y) dy,$$

which defines the relationship between the function $f(y)$ and the function $\varphi(x)$". Compare the entry *TRANSFORMATION* in the *Comprehensive Universal Encyclopaedia* [14].

(ii) In L. S. Pontriagin's *Topological Groups* [8] on p. 187 we find the information that "in the integral equation

$$\varphi(x) = \lambda \int k(x, y)(y) dy$$

the function $k(x, y)$ is termed the kernel of that equation".

(iii) Z. Semadeni and A. Wiweger in their *Introduction to the Theory of Categories and Functors* [10] state that "the counterimage

$$Ker\, \alpha = \alpha^{\leftarrow}(0) = ((\alpha \in A : \alpha(a) = 0))$$

of the natural (zero) element 0 of a group is the kernel of the morphism α".

(iv) In *Linear Algebra* by A. Mostowski and M. Stark [7] the definition of kernel is: "A set of vectors α, for which $f\alpha = 0$, is a linear space $\Sigma(f)$ termed the kernel or the zero space of the transformation f".

Obviously, in (i) and (ii) on the one hand, and also in (iii) and (iv) on the other, we have to do practically with one and the same concept. In (i) and (ii) it is interpreted as a function which determines an integral transformation, whereas in (iii) and (iv), as the counterimage of the zero element. Formally, however, all the four meanings are different. That ambiguity does not cause any misunderstandings in the interpretation of the texts quoted above. It is to be noted that in mathematical texts such ambiguity is encountered as frequently as in natural language.

The semantics of the language of mathematical texts is discrete. That is secured above all by definitions used in those texts, which, unlike in natural language, are never definitions by examples. This guarantees a lack of vagueness of the terms used and consistence in their use. Before any change in the meaning of a term used earlier the author ought to indicate the new meaning of that term by stating it explicitly. The semantic of the language of mathematical texts is usually too intricate for being expressed by the means at the disposal of natural language. That necessitates the use of stronger tools: that role is taken over by mathematical symbolism.

It is true that contemporary mathematics is unthinkable without the use of symbols, but these appeared in mathematical texts relatively late. It is legitimate to assume, for instance, that the use of letters (as variables) introduced in mathematics in the early 17th century made the birth and development of algebra possible. That led in turn to the emergence of analytic geometry, and the application of adequate symbolism in solutions of problems in the differential calculus turned the latter into an independent branch of mathematics.

The introduction and consistent use of the system of symbols in mathematics were in a close connection with works on the systems of symbols used in logical calculi. It is assumed that the rapprochement between logics and mathematics was bilateral. On the one hand, logic was coming closer to mathematics by a more and more general interpretation of logical relationships and operations, and on the other, mathematicians, following the development of their discipline, were forced to become interested in its logical structure. The development of studies in the logical foundations of mathematics must have accordingly had important consequences for logic as a whole. On the other hand, the development of mathematical content induced people to develop symbolism and brought out the usefulness of certain standard formulations serving the precise expression of thoughts. Verbal expressions were being replaced by a non-ambiguous and transparent system of symbols, free from the chance and obscurity of everyday language.

The task of constructing a theory of mathematical reasonings, to be presented in a symbolic form, was set themselves at the close of the 19th century by G. Peano in Italy and G. Frege in Germany. The thesis on the reducibility of mathematics to logic was demonstrated by A. N. Whitehead and B. Russell in their *Principia Mathematica*.

Arriving at mathematical theories which would comply with the logical requirements of formal correctness became the goal of the formalist programme of D. Hilbert and his school. That programme, striving for the axiomatization of the whole of mathematics, when verified by the works of Gödel, Church and others, and also those of Tarski, proved unattainable in its philosophical aspect. But it marked a certain step forward towards the construction of a standard language of mathematics when it comes to certain techniques used in that discipline. The concept of the language of mathematics was, in the context of metalogical and metamathematical studies, identified with that of the formalized language. From the point of view of the language of mathematics, both the works of Hilbert and the earlier studies of Peano, Whitehead, Russell, and others show that any mathematical theory with its primitive concepts and proofs, can be recorded in a formal manner. It has also been demonstrated that the exposition of mathematics by totally or almost totally abstaining from the use of a verbal text is possible when one resorts to a symbolism specially worked out for that purpose.

It turned out in this connection, which is stressed by some authors, that as in the 16th and the 17th century, when algebra was being counched in symbols, two hundred years later the formalization of the mathematical language was not due to any new principles, but to the precision and standardization of the ordinary lexical and syntactical means of language: logical relationships replaced conjunctions, function symbols were used in describing the relations between the subject and the predicate, and the discovery of the usefulness of quantifying expressions, as a natural consequence of the development of the symbolism of the values of variables in mathematical analysis and geometry, preceded the introduction of quantifiers.

It seems that the idea of the formalization of mathematics is being achieved by the imposition upon natural language of limitations enforced by classical logic. One could, by the way, hardly imagine another sequence of events. Everyday language is that language in which all scientific theories, in particular mathematical ones, were formulated in a natural manner. The limitations imposed upon everyday language were, it seems, mainly due to the nature of the semantics of that language. But it should be assumed that the shortcomings of natural language resulting from a lack of precision in the expression of certain ideas, were the inspiration for adjusting that language to the needs of science, mathematics in the first place. It seems characteristic of mathematics to used bivalued logic, which entailed the adoption of the extensional nature of conjunctions (in technical language renamed "sentential connectives"), and also the fixing of the meanings of quantifying expressions, which, by the way, came rather late. The formal interpretation of the latter was essential in the sense that in natural language those expressions were never used in the meaning adopted in formal logic. The formal recording of logical reasonings was the next stage in formalization, which essentially influenced the formalization of mathematics.

How far recourse to symbolism in definitions facilitates their formulation and use is shown by the example drawn from Webster's dictionary, where under the entry metric we find: "a mathematical function that associates with each pair of elements of a set a real non-negative number constituting their distance and satisfying the conditions that the number is zero only if the two elements are identical, the number is the same regardless of the order in which the two elements are taken, and the number associated with one pair of elements plus that associated with one member of the pair and a third element is equal to or greater than the number associated with the other member of the pair and the third element" (*Webster Dictionary* 1977, p. 724–725). It is to be doubted whether that text is comprehensible to a non-mathematician, and even a mathematician will find it difficult to identify it as the clearly and precisely formulated definition of distance, commonly used in mathematical texts. That definition reads:

the function $D(x,y) \geq 0$ is termed distance if and only if it satisfies the following conditions:

(1) $D(x,y) = 0 \leftrightarrow x = y$,
(2) $D(x,y) = D(y,x)$,
(3) $D(x,y) + D(y,z) \leq D(x,z)$.

The above example finely illustrates another element of the language of mathematics, very important in our opinion, namely the common occurrence of variables. Variables appeared in the language of mathematics more or less simultaneously with the first attempts to formalize mathematics, and their role cannot be overestimated. It seems that the "invention" of variables was decisive in the process of formalization, for in natural languages variables do not occur in such a form as in mathematical texts, and their place is occupied by other natural language expressions, most frequently pronouns[2], but not only them, as the example drawn from Webster has shown. There the role of variables is played by numerals. While the definition of distance formulated in natural language can be interpreted correctly with some difficulty, and possible to apply, many other definitions cannot in any way be sensibly formulated in natural language. Here is one of the axioms of Moebius' geometry (where *circumference* stands for *circumference of a circle*:

Given eight different points A, B, C, D, E, F, G, H, then
if there is a circumference which passes through A, B, C, D, and
if there is a circumference which passes through E, D, C, H, and
if there is a circumference which passes through E, D, A, F, and
if there is a circumference which passes through F, A, B, G, and
if there is a circumference which passes through G, B, C, H, then
there is also a circumference which passes through E, F, G,H.

In the formulation of the above axiom there are eight objects of the same type, namely points. There are too many of them to be sensibly distinguished from one another using means of natural language. It is true that this objects can be distinguished in the way suggested by Webster, namely by replacing letters with ordinal numerals, e.g., instead of *point C* to say *the first* point, instead of *point H, the second point*, and so on. The numerals in this case allow us to distinguish the points in question, but that is an inconvenient way of replacing letter variables by "descriptive" ones. It seems that recourse to letter variables is the only practical way of recording the axiom under consideration. Note that the variables referred to also occur in the same for instance in chemical texts. But similarity is merely apparent. Chemistry has a strongly expanded system of nouns used as technical terms, and letter symbols do not play there the role of variables but function rather as definite descriptions. The problem of variables does not arise even when very many objects are involved if they are of different kinds. In such a case in natural language it is just nouns which are used to describe them. The objection might be raised that it is not only in mathematics that we have to do with a rich terminology, and yet that does not require the use of symbolism abounding in variables. It is true that other disciplines have an expanded terminology and yet do not use variables in the manner typical of mathematics, but there is no such need because terminology, for instance in biology, serves different purposes than it does in mathematics. We have there to do with taxonomy, and hence with a classification of objects of various types. It is only in mathematics that so many objects of the same type are used, which requires a way of speaking about them without obscuring their picture. It is legitimate to think that the underestimated fact – the introduction of variable in mathematical texts – has been one of the greatest achievements of the symbolism used in the language of mathematical texts, and at the same time the element which marks an essential difference between the language of mathematics, on the one hand, and not only everyday language but the languages of other, non-mathematical, disciplines as well.

*

* *

The possibility of making mathematical statements uniform in the sense of a system of symbols created specially for that purpose does not imply its common acceptance. When one reads the majority of mathematical texts one finds that they make use to a little extent or not at all of the symbolism applied in mathematical logic, even though its use together with a verbal text increases the precision of exposition of the subject matter. Moreover, some authors are convinced that the

[2] The opinion that pronouns in natural language may be treated as variables was voiced, among others, by W V. O. Quine in his paper "Logic and the reification of universals", (in: [9]).

symbolism of formal logic, while indispensable in the discussion of mathematical logic, becomes an intricate code when used as a means of conveying ideas (Steenrod 1978, Shiffer 1978). At the same time attention is drawn to the fact that formalized languages are too poor to cover large mathematical theories. For instance, logical and mathematical symbolism does not suffice in expounding the theory of differential equations. Verbal additions are necessary and, moreover, the need of availing one self of verbal text manifests itself in nearly all mathematical works. Without it many of them prove too difficult for the readers or even outright incomprehensible.

Note that a large part of the verbal text tends to be standardized: the vocabulary of the mathematical language (except for specific mathematical terms) is poor. There is a fixed number of operators which are used as formulas ("let us consider", "to do so it suffices to demonstrate" etc.).

Since the language of mathematical logic, combined with the language presently used in mathematics, does not suffice for a standard recording of mathematical contents, the need emerges of constructing a new language, which would meet the requirements of simplicity and adequacy and thus materialize the idea of a standard language of mathematics. The suggestions in that matter, known in the contemporary literature of the subject, are as follows:

(1) A precise statistical and logical analysis of the language of contemporary mathematical texts that would bring out its tendencies for standardization.

(2) Construction on that basis of a specialized and simplified mathematical language which, combined with mathematical symbolism and the symbolic means of mathematical logic, would suffice for the description of the results of contemporary mathematical research. According to L. A. Kaluzhnin, the simplification of that language in the first phase should consist in the selection and fixing of a not too rich vocabulary; the fixing of precise rules of the introduction of specialized terms; the simplification of grammar (e.g., in the sense of eliminating grammatical exceptions). This is based on the fact that even a superficial knowledge of the grammar of a given foreign language suffices one for understanding a mathematical text written in such a language.

The idea of creating a standard mathematical language is not new and emerges systematically in the literature of the subject, and the advances in information science and the resulting new prospects have stimulated those ideas a new. Theories of the construction of such a language and the properties which it should have are known in international literature, but in practice they have proved to be a difficult undertaking. The only standard mathematical language we know, used in recording mathematical texts and in automatically checking the correctness of mathematical proofs, namely Mizar, meets in practice the requirements set to a standard language. Its author is A. Trybulec, and it has already been successfully applied in practice both in Poland and abroad.

References

1. *Encyklopedia fizyki* (The Encyclopaedia of Physics), PWN (Polish Scientific Publishers), Warsaw 1972.
2. Grzegorczyk A., *Zarys arytmetyki teoretycznej* (Outline of Theoretical Arithmetic), PWN (Polish Scientific Publishers), Warsaw 1971.
3. Halmos P. R., *Jak pisać teksty matematyczne* (How to write mathematical texts), Wiadomości Matematyczne, XXI.1, PWN (Polish Scientific Publishers), Warsaw 1978.
4. Kaluzhnin L. A., *O języku informacyjnym nauki* (The Information Language of Science), Wiadomości Matematyczne, VII.2, PWN (Polish Scientific Publishers), Warsaw 1964.
5. Kuzincheva Z. A., *O niekotorikh problemakh osnovaniy matematiki sviazanykh so spetsifikoy yazyka matematiki*, in: *Zakonomiernosci rozvitia sovriemennoy matiematiki*, M. I. Pavlov (ed.), Nauka, Moscow 1987.
6. Marinov V., *Computer understanding of mathematical proof*, Studies in Logic, Grammar and Rhetoric, II, Białystok 1982.
7. Mostowski A., Stark M., *Algebra liniowa* (Linear Algebra), PWN (Polish Scientific Publishers), 1973.
8. Pontriagin L. S., *Grupy topologiczne* (Topological Groups), PWN (Polish Scientific Publishers), 1961.
9. Quine W. V. O., *From a Logical Point of View*, Cambridge (Mass.) 1953.
10. Semadeni Z., Wiwiger A., *Wprowadzenie do teorii kategorii i funktorów* (Introduction to the Theory of Categories and Functors), PWN (Polish Scientific Publishers), 1976.
11. Steenrod N. E., *Jak pisać teksty matematyczne* (How to write mathematical texts), Wiadomości Matematyczne, VU.2, PWN (Polish Scientific Publishers), Warsaw 1964.
12. Trybulec A., *Modelowanie procesów informacyjnych nauki* (Modelling information process in science), Studia Filozoficzne, No. 2 (75), 1972;
13. Trybulec A., *New standard of Mizar MSE syntax*, Mizar News, Vol. 2, No. 1 (2), Stockholm 1987.
14. *Wielka Encyklopedia Powszechna* (The Comprehensive Universal Encyclopaedia) PWN (Polish Scientific Publishers), 1970.

Informal and Formal Representations in Mathematics

Manfred Kerber[i] and Martin Pollet[ii]

[i]School of Computer Science
University of Birmingham
Birmingham B15 2TT, England
www.cs.bham.ac.uk/~mmk

[ii]Fachbereich Informatik
Universität des Saarlandes
66041 Saarbrücken, Germany
www.ags.uni-sb.de/~pollet

Abstract. In this paper we discuss the importance of good representations in mathematics and relate them to general design issues. Good design makes life easy, bad design difficult. For this reason experienced mathematicians spend a significant amount of their time on the design of their concepts. While many formal systems try to support this by providing a high-level language, we argue that more should be learned from the mathematical practice in order to improve the applicability of formal systems.

> *It is not easy to say precisely what we learnt in the meantime, if we learnt anything at all. I believe that the most important results are:*
> 1. *we need experiments with much more advanced mathematics than already done*
> 2. *a system for practical formalization of mathematics probably will not be a simple system based on small number of primitive notions*
> 3. *integration with a computer algebra systems may be necessary or at least a feasible system must have bigger computational power.*
>
> Andrzej Trybulec [33]

1 Introduction

There is an old debate on the foundations of mathematics, which goes at least back to the time when Alfred North Whitehead and Bertrand Russell did their epochal work, which also resulted in a much narrower view on what "foundations" of mathematics should mean. Russell articulated the idea in 1903 in the *Principles of Mathematics* [29]: to reduce mathematics to formal logic. He and Whitehead carried it through in the famous *Principia Mathematica* [35]. Russell was at this time also in inspiring discussions with Ludwig Wittgenstein and strongly acknowledges Wittgenstein's contribution to his thoughts. He writes in [30] (quoted from the reprint in [31, p.178]):

> *As I have attempted to prove in* The Principles of Mathematics, *when we analyse mathematics we bring it all back to logic. It all comes back to logic in the strictest and most formal sense.*

ISBN 978-83-7431-128-1 ISSN 0860-150X

Although there is evidence that Wittgenstein shared Russell's view, he later took the opposite stance and attacked Russell's approach. In particular he discusses the important notion of proof (quoted from [36, p. 143]):

> *'A mathematical proof must be perspicuous.' ... I want to say: if you have a proof-pattern that cannot be taken in, and by a change in notation you turn it into one that can, then you are producing a proof, where there was none before.*

One of the reasons why the Principia is so difficult to read is that the main ideas of the proofs are no longer visible in very long and very detailed proofs. Wittgenstein continues (p. 176f) to question the idea to try to reduce everything to a very small number of primitives (had the resolution calculus already been invented at that time his attack might have been to try to reduce everything to one single rule):

> *Mathematics is a* MOTLEY *of techniques of proof. – And upon this is based its manifold applicability and its importance. ...*
> *Now it is possible to imagine some – or all – of the proof systems of present-day mathematics as having been co-ordinated in such a way with one system, say that of Russell. So that all proofs could be carried out in this system, even though in a roundabout way. So would there then be only the single system – no longer the many? – But then it must surely be possible to shew of the* one *system that it can be resolved into the many. – One part of the system will possess the properties of trigonometry, another those of algebra, and so on. Thus one can say that different techniques are used in these parts.*

He continues then to counter the argument that Russell and Whitehead have constructively shown the possibility to reduce everything to one single system (p. 185)[1]:

> *If someone tries to shew that mathematics is not logic, what is he trying to shew? He is surely trying to say something like: – If tables, chairs, cupboards, etc. are swathed in enough paper, certainly they will look spherical in the end.*
> *He is not trying to shew that it is impossible that, for every mathematical proof, a Russellian proof can be constructed which (somehow) 'corresponds' to it, but rather that the acceptance of such a correspondence does not lean on logic.*

Who is right? On the one hand, Whitehead and Russell could claim that in more than a hundred years no significant counterexample to their original claim has been found.[2] Wittgenstein, on the other hand, could claim that mathematical practice is far from the logicistic approach.

[1] By the way, Gödel's proof that formal systems like the Principia are necessarily incomplete is irrelevant for this argument, since not only the Principia but any other powerful system suffers from the same problems.

[2] Gödel's incompleteness theorem again could be considered as one, but seems for this discussion less relevant than it looks on first view.

As a good cabinet maker spends a lot of time on the design of tables, chairs, and cupboards, a good mathematician spends great care on the design of their concepts. We will discuss this in more detail in Section 3. Before we do that we want to clarify what we mean by "design", in Section 2. In Section 4 we discuss some of the most important approaches in formal reasoning systems to provide adequate design possibilities.

A lot of what we say in the following will look completely self evident to many. However, we feel that in the traditional theorem proving community, which we consider as our home community, the logicistic view has too strongly dominated ever since its start more than 50 years ago. A seemingly strong argument against new approaches in theorem proving has been of the type "Everything you can do in your system X, I can do in Y," where Y stands typically for standard first-order logic. This kind of thought is so strong that proponents of the conventional approach to theorem proving seem to fail to even understand why this argument – albeit it may be true – is unhelpful and misleading.

For instance, Pat Hayes said in 1974 ([12] quoted from [5, p.18]):

> *A more recent attack on conventional theorem-proving ... is that it is too concerned with "machine-oriented" logic, and not enough with "human oriented" logic. I confess to being quite unable to understand what this could possibly mean.*

In this paper we try to clarify why the argument, although it may be technically correct, is pragmatically flawed. The argument is pragmatically wrong, since it is meant to say "Your system X is redundant and uninteresting, since we have system Y already, which suffices for everything you want to do." Since this argument was widely accepted the focus in the field was set much too narrow on the study of foundational systems. For other communities, our observations in this paper may be trivial.

Of course there are exceptions and we claim in no way that we are the first to have a look at this relationship of mathematical practice and fundamental systems. We cannot give a comprehensive overview of this type of work here but we want to mention some work in this direction, which we think provides very important starting points to the support of the design of mathematical concepts. In AUTOMATH, N.G. de Bruijn developed the idea of a mathematical vernacular [6], which should allow to write everything mathematicians do in informal reasoning, in a computer assisted system as well. In this tradition, Hugo Elbers looked in [10] at aspects of connecting informal and formal reasoning, in particular the integration of computations in formal proofs. Francis Jeffry Pelletier [24] as well as Henk Barendregt and Arjeh Cohen [2] discuss related philosophical questions on the nature of proof. Proof planning [7] in general can be viewed as an attempt to simulate informal reasoning (and integrate it with formal reasoning). Following this paradigm, Alan Bundy made first steps towards a Science of Reasoning [8], which goes beyond a narrow focus on a particular calculus. Ursula Martin took in [19] a close look at the mathematical practice and its relationship to computer algebra and computer-assisted reasoning. Michael Beeson presented in [3] a system that combines deductive and computational reasoning steps. He proposed to use the mathematical standard for checking the correctness of proofs generated by the system, namely peer reviews. Claus Zinn looked at the understanding of informal mathematical discourse [37].

We are not aware, however, of work in which the design process in mathematics is compared to the design possibilities of computer based mathematical support systems. In this paper, we look at design problems, which current automated reasoning systems suffer from.

We try to exemplify in the rest of the paper why these matters are crucial for automated theorem proving systems. One important issue is that of acceptance among mathematicians. Current automated theorem provers do not find wider acceptance among mathematicians, while computer algebra systems do. The initial investment that a mathematician has to do before he/she gets any benefit from a theorem prover is still quite high. This is different for computer algebra systems, partly since in many computer algebra systems things are *as they should be, as an inexperienced but mathematically educated user would expect them to be.* That is, these systems are typically *well-designed.* In theorem proving systems they are too often *not as they should be, not as an inexperienced user would them expect to be.* We will argue that this is *not* just a deficiency of the user interface, but that the problem with automated theorem provers is much deeper, it goes to the core of these systems, namely to the formal representation of mathematical knowledge and the reasoning that can be performed with this knowledge.

2 Good and Bad Design

Before we look at representation issues in mathematics we want to widen our point of view and have a more general look at design issues. Donald Norman gives a fascinating introduction into "The Design of Everyday Things" [22]. As the title of his book says, it is mainly on everyday things such as light switches or door handles. Norman gives impressive insights why when we face difficulties in everyday situations this is often due to bad design; and conversely when we do not face such difficulties this is often due to good design. His insights are of a very general nature and we will discuss in the next section that the principles for good design hold in mathematics as well and form another facet of mathematics which go in some aspects beyond logic.

Let us look at a simple example how bad design can make life difficult. Very often light switches are carelessly installed so that in a particular situation it is not possible to intuitively know which one to use. For instance, in the office of one of the authors there are two light switches next to each other, the left one for the upward directed light and the right one for the downward directed light, or is it the other way around? Since nobody can guess or remember this, the switches are labelled "up" and "down". Unfortunately these labels cannot be read in the darkness before you switch the light on. This is bad design. A good design for the problem would be to locate the two switches not next to each other but on top of each other, the upper one for the up-light, the lower one for the down-light. A similarly bad design is used in almost all cookers with four hot plates arranged in a 2 × 2 form. The four switches belonging to the four hot plates are arranged in a straight line so that labels are necessary to remember which of the two left (or right) hot plates are controlled by which of the two left (or right) switches. If the switches for the back plates were only slightly moved up out of the straight line, the problem would go.

Although Norman does not relate design issues to mathematics, most of his observations can be translated to a mathematical context. For instance, design principles allow us to answer questions like "Why and how do we find a proof without major search effort, although it is a difficult one and we do not know it?" Norman states four principles of good design which help us to get certain things right, although we do not know precisely what to do [22, p.55]: *"Precise behavior can emerge from imprecise knowledge for four reasons.*

1. *Information in the world. Much of the information a person needs to do a task can reside in the world. Behavior is determined by combining the information in memory (in the head) with that in the world.*
2. *Great precision is not required. Precision, accuracy, and completeness of knowledge are seldom required. Perfect behavior will result if the knowledge describes the information or behavior sufficiently to distinguish the correct choice from all others.*
3. *Natural constraints are present. The world restricts the allowed behavior. The physical properties of objects constrain possible operations: the order in which parts can go together and the ways in which an object can be moved, picked up, or otherwise manipulated. Each object has physical features – projections, depressions, screwthreads, appendages – that limit its relationship to other objects, operations that can be performed to it, what can be attached to it, and so on.*
4. *Cultural constraints are present. In addition to natural, physical constraints, society has evolved numerous artificial conventions that govern acceptable social behavior. These cultural conventions have to be learned, but once learned they apply to a wide variety of circumstances."*

Norman [22, p.188f] develops in the sequel seven principles of (good) design for transforming difficult tasks into simple ones:

1. *Use both knowledge in the world and knowledge in the head.*
2. *Simplify the structure of tasks.*
3. *Make things visible: bridge the gulfs of Execution and Evaluation.*
4. *Get the mappings right.*
5. *Exploit the power of constraints, both natural and artificial.*
6. *Design for error.*
7. *When all else fails, standardize.*

Following such principles it will be possible to design light switches and door handles well. Without doubt good designers follow such principles, but not only in their relationship to everyday objects, but also in their relationship to mathematical objects. In the next section we will have a closer look at one example where careful design of a mathematical entity helps avoiding errors, so-called multiplication tables. We will also briefly mention further examples.

3 Design and Representation in Mathematical Practice

We will take a closer look at multiplication tables, a concept that seems on a first view easy, and on a second difficult to model in existing theorem proving systems. Multiplication tables are part of rigorous mathematics in the sense that they appear not only as comments or illustrations in mathematical textbooks, but are usually introduced in definitions and their properties are stated as theorems.[3] We believe that the features of the concept "multiplication table" as well as those we present in the sequel are not only a matter of presentation but that they are used to encode and retrieve information about mathematical concepts in an efficient way and that they ease the actual process of finding and presenting proofs. To represent a mathematical object such as a multiplication table requires careful design.

3.1 Multiplication Tables

$$\begin{array}{c|ccc} \circ & d_1 & \cdots & d_n \\ \hline d_1 & c_{11} & \cdots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ d_n & c_{n1} & \cdots & c_{nn} \end{array}$$

Multiplication tables were introduced by Cayley to represent the operation of finite abstract groups. The information encoded in a table is that the operation is a binary operation, defined on $\{d_1, \ldots, d_n\} \times \{d_1, \ldots, d_n\}$ and with range $\{c_{11}, \ldots, c_{nn}\}$, the operation is discrete and has a finite domain and codomain.

The table has its own notion of well-formedness, that is, all d_i have to occur and have to be different, the table must be fully filled. Here you find Norman's principles 1, 3, and 4. Multiplication tables are designed in a way that their structure puts "information in the world" that makes it difficult to violate well-formedness. It is hard to imagine when you have got the task to define a specific operation starting with an empty multiplication table that you forget a case, since that would leave a hole in the structure. The table itself is of the form that it constrains the possibilities. For instance, it is impossible to enter more than one entry per field. This prevents any over-specification of $\circ$. Furthermore, although the order of the d_i in the columns and rows could in principle be different, cultural conventions prevent that.

Note that there are particular reasoning methods connected to the representation. To check the basic property of closedness, one has to go through the elements of the table and check whether for all elements holds $c_{ij} \in \{d_1, \ldots, d_n\}$. The commutativity of $\circ$ is checked by verifying that the table is symmetric with respect to the diagonal. This intuitive form of reasoning depends on the cultural convention to use the same order for rows and columns. Another cultural convention is to write a (potential) unit element as first element (or second element in the presence of a zero, which typically goes in the first place). With this convention, it is checked that d_1 is a unit element by establishing the equality of the columns under $\circ$ and d_1 and the rows right to $\circ$ and d_1. Inverse elements can be checked by establishing that each column and each row contain the neutral element exactly once. These reasoning patterns follow partly the design principle number 2, since they are natural and easy to reconstruct. From the group properties only associativity requires a logic level proof.[4]

Of course, it is possible to define the same operation in a logic, possible formalizations are for example:

- First order: extend the signature by a function constant $\circ$ and add the assumptions $d_1 \circ d_1 = c_{11} \wedge d_1 \circ d_2 = c_{12} \wedge \ldots \wedge d_n \circ d_n = c_{nn}$.
- Higher order: use the description operator[5] to define the operation as $\circ \equiv \lambda x.\iota y.(x = (d_1, d_1) \wedge y = c_{11}) \vee \ldots \vee (x = (d_n, d_n) \wedge y = c_{nn})$.

While the special representation of multiplication tables can be translated into these general logical formalisms, parts of the information stored in the mathematical representation are lost. In the first case the compoundness of the table representation is hard to reconstruct. We are speaking about a set of equations suitable for equational reasoning steps, but to recognize that the set of equations is suitable for an abstract method for closedness or commutativity as mentioned before is not so obvious. Also the special reasoning methods for proving commutativity, inverse elements, and neutral element are not so obvious, but require search in a set of formulae. From a human interface point of view, the lack of structure in the formulae puts the burden of guaranteeing well-definedness on the human. He or she has to be careful not to forget the definition of one element or not to over-define one expression by inserting two formulae like, for instance, $d_1 \circ d_1 = a$ and at a different place $d_1 \circ d_1 = b$.

Although, the higher order formalization of the operation seems preferable over the first order one since it encodes $\circ$ as one compound object, here as well it is hard to recognize what kind of function is encoded. Actually, there is a proof obligation to be shown in an application of the function to arguments, namely that there really is a unique element with these properties.

In the rest of this section we look at further examples for mathematical concepts and procedures which are difficult to represent directly in a foundational system. Although we do not relate them in detail to the design principles discussed in the previous section, it would not be hard to establish similar relationships here as well.

3.2 Representations Adapted to Objects

Similarly to multiplication tables there are other objects which typically have specialized representations. Quite related are matrices. A matrix is a two dimensional array that may contain numbers or more complex objects. It has similarities with multiplication tables and a possible formal definition as lists of lists is the same as the one of multiplication tables. However its usage is very different, and human

[3] We use here the word "rigorous" and not "formal" in order to distinguish it from "formal logic" and mechanical systems. Mathematicians would probably use the word "formal," since they are happy with these concepts as a level of formalization that clarifies concepts unambiguously.

[4] Surely, for people experienced with this type of proof, there is not anything to prove anymore, but it boils all down to trivial computations, which according to the Poincaré principle [2, 15] can be considered as not being a part of the proof. This is different for beginners, whose perspective we have taken here.

[5] The description operator ι returns the element of a singleton. $\iota y.P[y]$ denotes the unique element c such that $P[c]$ holds, if such a unique element exists.

mathematicians have different methods attached to these concepts. Matrices are typically used to represent linear mappings and other transformations in vector spaces. The equation[6]

$$\left(\begin{array}{c|ccc} \alpha & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & T^{-1} & \\ 0 & & & \end{array}\right) \cdot \left(\begin{array}{c|c} 1 & uT \\ \hline 0 & \\ \vdots & BT \\ 0 & \end{array}\right) = \left(\begin{array}{c|c} \alpha & \alpha uT \\ \hline 0 & \\ \vdots & T^{-1}BT \\ 0 & \end{array}\right)$$

is taken from a proof about the tridiagonalization of matrices.

In principle matrices over a field F can be represented in logic by functions from an index set into F. However, this representation does not lend itself to the definition of multiplication of matrices in which product elements are calculated by traversing the left matrix left-right and the right matrix of the product top-down. The generalization of this principle makes it easy to establish the relationship accounted for above. Note that this form of abstraction is not just a matter of analogy which helps to find a proof, since it would be quite difficult to state – let alone prove – the property above without abstraction. Furthermore the explicit matrix representation exhibits advantages mentioned above for multiplication tables. We have presented some work on this in [25], which was extended in [32].

Note that matrices are available in Computer Algebra Systems (CASs) as primitives and that direct manipulations are possible. This, however, does not make it obsolete to introduce them directly into automated theorem proving systems as well. For instance, the matrices used in the equation above cannot be easily defined in a CAS, since they represent more than one instance, they are generalized matrices representing any matrix that has the same 'form.' Establishing the equation itself cannot be done by computation, but requires reasoning. Once established it can become a further computation rule.

3.3 Dynamic Representations

A feature of mathematical representations is that they are dynamic in the sense that as new knowledge is available the basic representation of objects may change. For instance, the existence of inverse elements of a *group* G can be formalized by $\forall_{x \in G} \exists_{y \in G}\ x \circ y = e \wedge y \circ x = e$ with the unit element e. Since for each group element there exists exactly one inverse element, the inverse of an element x is usually denoted with the help of a function as $inv(x)$. Whereas the introduction of a function denoting the inverse elements is possible in most of the interactive theorem proving systems, the question whether the concept group should be introduced as $group(G, \circ)$, $group(G, \circ, e)$, $group(G, \circ, e, inv)$ seems to be more subtle.

The first formalization eases the possibility to inherit properties of subsumed concepts with $group(G, \circ) \Rightarrow monoid(G, \circ)$. The latter formalization makes the unit elements and inverse elements directly accessible but already contains the uniqueness of the inverse element of an element in form of the inverse function. The process of the actual exploration of basic principles of group theory that starts with the tuple $(G, \circ)$ and later introduces the neutral and inverse elements as useful parameters of the concept can hardly be modelled in current automated reasoning systems. Rather it is necessary to choose one formalization and to stick to it. This way it is not only the case that the introduction of a concept cannot be modelled adequately, but perhaps more seriously re-representations of concepts are not supported. However, while one representation may be best suited for one task, it may turn out to be unsuitable for a different one. In the latter situation mathematicians change representations. Different formalizations model different views on something that is one single mathematical concept to a mathematician. If a mathematician had to use one of the standard theorem proving systems, he or she would need to know in advance which choice of representation to make, since the choice of a good formalization is crucial for the success. But how do you know which formalization is best, when you start to explore something?

Another example of dynamic re-representation, which is very simple, but for which the different reasoning complexities are striking, is when associativity holds for an operation $+$. Once associativity is established, no mathematician would still use brackets, but the notation for a term like $((1 + x) + y)$ would change to $1 + x + y$. The property is encoded into the notation and can be retrieved from the given term. On a reasoning level this simple shift in representation can make a dramatic difference. For a term with $n + 1$ summands there are $\frac{1}{n+1}\binom{2n}{n}$ different ways to put brackets in. That means for a medium sized expression with just 10 summands there are already 4862 ways to represent it. If all these representations are part of the search process it is no surprise that automated theorem provers find such expressions difficult. The design of these systems does not allow for a change in representation, a user must write down unnecessary brackets in order to be syntactically correct, the brackets, however, do not help but unnecessarily confuse the reasoner. These all are signs of bad design. In the next sub-section we will look more closer at the change of representation.

3.4 Change of Representations

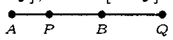
Sometimes an appropriate reformulation of a problem into another representation is already the key step to find a proof. Different representations allow to apply knowledge from different sources to a problem. The importance of re-representation was pointed out by George Pólya [27, vol.2, p.80]:

> *When you are handling material things (for instance, when you are about to saw a limb off a tree) you automatically put yourself in the most convenient position. You should act similarly when you are facing any kind of problem; you should try to put yourself in such a position that you can tackle the problem from the most accessible side. You turn the problem over and over in your mind; try to turn it so that it appears simpler. The aspect of the problem that you are facing at this moment may not be the most favourable: Is the problem as simply, as clearly, as suggestively expressed as possible? Could you restate the problem?*

[6] Mathematicians store information even in the letters they choose for their objects. Even without further information it is relatively easy to reconstruct that T^{-1}, B, and T should represent submatrices since they make use of upper-case letters, while the Greek α stands for a scalar, and u denotes a vector.

> *Of course you want to restate the problem (transform it into an equivalent problem) so that it becomes more familiar, more attractive, more accessible, more promising.*

We exemplify this importance by different forms of re-representations:

- *functions:* Given an Euclidean space $\mathcal{R}$ with a metric function $|\ | : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$ then for each pair $A, B \in \mathcal{R}$ of disjoint points there exists exactly one distance preserving function $g_A^B : \mathbb{R} \to \mathcal{R}$ with $g(0) = A$, $g(|AB|) = B$. With the help of this function the lines in Euclidean space can be interpreted as images of the real numbers. Notions of the real numbers, as intervals, correspond to notions in the abstract Euclidean space, namely line segments.
- *representation theorems:* Poincaré's model for the hyperbolic plane, where a line has infinitely many parallel lines through one point, is a unit disk, where circle segments correspond to hyperbolic lines. With this representation it becomes possible to re-represent constructions in the hyperbolic plane as constructions in the Euclidean plane.
- *theory change:* Geometric constructions can be represented as field extensions. The possibility to construct a square equal in area to a circle using compass and ruler can be re-represented to the question whether π belongs to a particular class of algebraic numbers (which it does not since it is transcendental).
- *inheritance:* The properties of monoids and groups are inherited by the multiplicative and additive substructures of rings and fields.
- *diagrams:* $P \in [AB], B \in [AQ] \Rightarrow P \in [AQ], B \in [PQ]$ which is obvious when this situation is expressed in a diagram: 

And of course there exist re-representations between different theories. Some of them changed the structure of mathematics itself:

- *Cartesian geometry:* arithmetic representation for geometry. This makes it possible to reduce geometrical problems to arithmetic problems and solve them arithmetically. This is actually the starting point of Descartes' idea to take arithmetic as a foundational system so that all problems should be translated to arithmetic and then solved by equation solving.
- *set theory:* mathematical concepts are representable as sets. Set theory is another foundational system on which most of mathematics can be based *in principle.*
- *group theory:* the group concept can be used to represent geometric transformations, permutations, and the solvability of polynomials.

Certain forms of representations are very important to form new concepts. The theorem that every permutation can be decomposed into transpositions, that is, can be represented as a product of transpositions, makes the definition of even and odd permutations suggestive. It is hard to imagine how to define the concept without this particular representational form. Once these concepts have been formed they become the starting point for the introduction of other concepts like the alternating group, which is defined as permutation group containing only permutations with an even number of transpositions.

Let us look at a different example, the concept of real numbers. Real numbers can be defined as Dedekind cuts or Cauchy sequences. However, Cantor's second diagonalization proof that the reals are uncountable is difficult to imagine without having the representation in the decimal (dual, or another) number system.

Often the opposite of a change in the representation happens in mathematics, that is, so-called overloading is used. The use of the same notation for different concepts, e.g. $|\ |$ in the example above for the concept of distance, $+$ for operations that behave like the well-known addition on natural numbers, etc. Even formally incorrect notation, such as equality for isomorphisms, is used to enable the transfer of knowledge from a known concept to a new concept. Certain *sort* and *type* systems allow for some kind of overloading, but they do not offer the kind of knowledge transfer that humans achieve this way in using overloaded symbols in an analogical way.

3.5 Structured Knowledge

A mathematical textbook is usually highly structured with *definitions*, *theorems* and *proofs* as main categories. This categorization is reflected in the formalizations for deduction systems. A closer look reveals that there exists a finer classification. For instance, definitions can be simple, inductive, or implicit. Some theorems are explicitly introduced as tools which can be applied in other situations or contain equivalence statements that are useful for re-representation. A proof can contain subproofs that will be repeatedly used for other theorems and that are emphasized to be important by the author, and a proof may contain an algorithm for the construction of mathematical objects.

A typical schema for the introduction of mathematical concepts is that a definition is followed by theorems giving simple properties and typical examples for this concept. One could argue that this structure has no significance for deduction systems because it is solely beneficial for the human process of learning and understanding. We try to show that the structure can become important as a basis for the reasoning process.

Take as an example the continuity of functions. The basic properties that are usually provided after this definition are that continuity is preserved for the sum, product, and composition of continuous functions. Usually as an example the identity on the reals will be given. Now suppose it is required to show that any polynomial is continuous. Instead of going back to the definition itself, the basic properties of the concept and the example of the identity function are used to prove the continuity of polynomials. The basic properties can be seen as attached to the definition and preferably used to prove simple proof obligations.

Of course it is possible to argue that the attached properties can be retrieved from a database that consists only of the plain structure of definitions and theorems. However, it is then difficult to query for a property like continuity of $+$. Should the query consist of all theorems containing the symbol 'continuous' or '+', or both? The structure that is lost in the plain logic representation would have to be reconstructed through elaborated query techniques that give useful results.

Note that the standard mathematical practice to build hierarchies of mathematical concepts is a very important means to reduce complexity in theorem proving. For instance in set theory, it is possible to define symbols like $\subset$, $\cup$, $\cap$ using the symbol $\in$. When you build a topology on top, using functions like $\circ$ for inner and $^{-}$ for closure, you can carry through most proofs on a level where you make use of abstract properties of $\subset$, $\cup$ and $\cap$, and can avoid totally to go down to the level of elements that involves $\in$.

Another category of mathematical knowledge form examples. They are crucial for the understanding of concepts. This is not only important to give semantics to syntactically defined concepts, but it can also be very relevant on the level of syntactic proof construction. We exemplify this for the theorem that "Groups G of order greater than two have non-trivial automorphisms." It seems to be hard to synthesize automorphisms directly only from the given assumption that G is a group. In this case, the standard human heuristic to try one of the typical examples, namely $x \mapsto gxg^{-1}$ and $x \mapsto x^{-1}$, provides already the crucial idea to prove this theorem.

3.6 Limited Expressiveness of Mathematical Tasks in Logic

When we take mathematical textbooks as basis for what is part of mathematics and what not, we can find statements that are not expressible in formal languages at all. Naturally comments or diagrams, and a number of models are not represented. Historic statements, statements about the relevance of properties and concepts and so on are part of mathematics, but not part of logic, although they are often important for a deeper understanding and an informed proof search. As Robinson cites himself in [28]: "Logic deals with what follows from what. ... The correctness of a piece ... does not depend on what the reasoning is about." That is a strength and a weakness of logic at the same time.

But even at the level of problem formulation, it is not always possible to apply a formal system in a straightforward way. Let us look for instance at the mathematical task (provided in some context): "Determine the maximum of $f(x)$ in the interval $[a, b]$." Does the obvious formalization "$\exists_{x\in[a,b]}$ max(f, x)" reflect the meaning of the sentence? Not necessarily, assume somebody comes up with the argument "Since f is a continuous function on a compact interval it has a maximum." This might be a correct argument to prove the logical formulation but it would not solve the original task. On a closer look adequacy of the logical statements depends on the logic used. If interpreted in a constructive way the logical formulation is adequate; if interpreted classically it is not. While there are constructive and classical systems around, it seems to be inappropriate that one has to decide for a constructive system once and for all, in order to be able to formulate a standard task like the one above.

While one can hardly cover all aspects of mathematics in a computer-based system, it seems for many applications – like the recently emerging applications in education – important to find a coverage which is as broad as possible.

3.7 A 'Natural' Calculus

The suggestions for a 'Mathematical Vernacular' [6] seem not to question that all concepts of mathematics are *sufficiently* expressible in a language consisting of functions and relations, potentially enriched by types or sorts. The design of the Vernacular seems not to be focused on the objects mathematicians are interested in, but on the reasoning framework.

The strength of a formal system can be measured by the de Bruijn factor, that is, the ratio of the length of the proof in the formal system compared to its version in a mathematical textbook.[7] A reassuring observation is that in the experiments with AUTOMATH and MIZAR the de Bruijn factor remained constant for the proofs of a wide range of differently complex theorems.

Even if we agree with the conclusion that the proofs constructed in formal calculi are already in principle an approximation of standard mathematical proofs, it is important to note that such a comparison is based on the *output* of mathematical work, the language used by mathematicians to communicate proofs in textbooks and articles. While a comparison of such completed proofs, proofs after all search is finished may be interesting, they often do not resemble the proof construction. As an example look at standard ϵ-δ-proofs. In the proof construction you compute a sufficient criterion on δ, choose δ as a function of ϵ and then you prove that with this δ the difference of the function values is indeed smaller than ϵ. In a finished proof a crucial part of the proof construction – the construction of δ – is redundant and hence not presented. For this reason it is impossible to understand prima facie why δ has been selected as it is.

Traditionally, the formulations of final proofs are minimalist and mathematicians repress their original ideas, even the order of steps can be different, in favour of an objective rigorous style. As Pólya [26, p. vi], pointed out:

> *We secure our mathematical knowledge by demonstrative reasoning, but we support our conjectures by plausible reasoning ... Demonstrative reasoning is safe, beyond controversy, and final. Plausible reasoning is hazardous, controversial, and provisional. ... In strict reasoning the principal thing is to distinguish a proof from a guess, a valid demonstration from an invalid attempt. In plausible reasoning the principal thing is to distinguish a guess from a guess, a more reasonable guess form a less reasonable guess. ... [plausible reasoning] is the kind of reasoning on which his [a mathematician's] creative work will depend.*

The 'demonstrative reasoning' corresponds to a formulation in reasoning steps that were investigated by logicians. In this sense current deduction systems are suitable as proof checkers for existing and well-understood parts of mathematics but lack to act as proof assistants for the exploration and construction of new mathematical knowledge. How can 'plausible reasoning' be modelled? Proof planning follows the paradigm of proof search on an abstract level and can be seen as

[7] The notion is not without problems, since mathematical proofs are not standardized with respect to their detailedness. Furthermore there are other aspects for the quality of a proof than its length.

an important step into this direction. But as described in [4] even proof planning depends on structural restrictions of the underlying calculus and uses a formal language as the only representation for mathematical concepts.

There might be the view that we can come up with a more powerful logic which provides the best possible representation. Marvin Minsky gave a strong argument why this would not be the case in artificial intelligence in general, but why multiple representations are necessary. He recommends [21]:

> *Everywhere I go I find people arguing about which representation to use. One person says, "It is best to use Logic." The next person says, "No, logic is too inflexible. Use Neural Networks." The third person says, "No, Neural Nets are even less flexible, because you have to reduce everything to mere numbers. Instead, you should use Semantic Networks. Then, the different kinds of things can be linked by concepts instead of mere numbers!" But then the first person might complain, "No, Semantic Nets are too arbitrary and undefined. If you use Formal Logic, that will remove those ambiguities." What is the answer? My opinion is that we can make versatile AI machines only by using several different kinds of representations in the same system! This is because no single method works well for all problems; each is good for certain tasks but not for others. Also different kinds of problems need different kinds of reasoning."*

As we have seen for multiplication tables different types of representations allow for specialised and efficient reasoning methods connected to them, opposed to search on the logic level. Mathematicians carefully design their concepts to keep the search spaces small. We need to understand this aspect of mathematical reasoning much better in order to understand the versatility of the reasoning capabilities of human mathematicians.

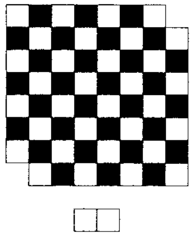
Historically the development of many concepts went the way that certain meta expressions were introduced, which were later reified and became object expression. The development of number systems might illustrate this. Having natural numbers and fractions, irrational numbers and negative numbers as well were considered as odd ones out, which only later became first class citizens. Then imaginary numbers were the odd ones and negative square roots were considered as strange entities which were used only for convenience. Likewise, functions were first concrete in nature, and only much later it was possible to speak about them.

In the next section we will contrast mathematical representations, which are very flexible and extendible, to formal computer systems, which still lack the required flexibility to design concepts to the level of sophistication which is achieved in informal mathematics.

4 Formal Representations of Mathematics

Up to here we have strongly argued how important a broad range of specialized constructs is for the adequate representation of mathematical knowledge and for proof construction. Representations find their analogues in data structures used in existing implementations of mathematical software systems, a range of general purpose and specialized theorem proving systems, computer algebra systems, and educational software. These data structures provide functionality and the ability to implement mechanisms working on them. In this section we want to take a brief look at some important features of systems from which ideas can be borrowed to realize a flexible system of the kind we envisage.

In a case study [16], we looked at different formalizations of the so-called mutilated checkerboard problem in different systems.



The challenge is to prove that *"It is impossible to cover the mutilated checkerboard shown in the figure with dominoes like the one in the figure. Namely, a domino covers a square of each color, but there are 30 black squares and 32 white squares to be covered."*

In the past a multitude of formalizations have been developed, for each system (be it an abstract system or an actual running system) at least one that is particularly well suited for that system. McCarthy states already in his original challenge paper [20] two different formalizations, one for first order logic without equality and one for first order with equality. At the second QED workshop organized by the Mizar group in Warsaw in 1995, McCarthy himself came back to the problem and presented a different formalization which makes use of set-theoretical notions and basic integer arithmetic. He gave this as a challenge to the community, namely to build a system to which one can give the essential hint to colour the mutilated checkerboard and to recognize that it has more white than black tiles but that any covering would cover equally many white and black tiles. The Mizar group took up the challenge and built directly a proof of the problem in Mizar using a formalization which is close to McCarthy's challenge [1]. There are also formalizations in Isabelle [23] and Coq [13].

While these systems show quite a flexibility in formalizing this problem and proving it, there are a number of mathematical concepts which need further structures. In particular the choice of representation is outside the formal system and the search for a good representation is not supported by the systems. We discuss some important ones in the following, in which special constructs have been made available to allow for a good representation.

4.1 Sorted Extensions to Logic

Sorted logic is a good example to exemplify the importance of careful design. It is an old insight that sorted logic (more carefully put, some classes of sorted logics) can have significant advantages over unsorted logic. Let us just consider the case of standard first-order logic with and without its order sorted extension. Sorts can be considered as special unary predicate symbols. Typical formulations in sorted logic are $\forall x_{\mathsf{Human}}.\mathsf{Mortal}(x)$, $\mathsf{socrates} \lessdot \mathsf{Human}$, $\neg\mathsf{Mortal}(\mathsf{socrates})$. The equivalent in unsorted logic is $\forall x.\mathsf{Human}(x) \Rightarrow \mathsf{Mortal}(x)$, $\mathsf{Human}(\mathsf{socrates})$, $\neg\mathsf{Mortal}(\mathsf{socrates})$.

The translation from the sorted to the unsorted logic is called relativization. The possibility to relativize any sorted problem formulation can be and has been used as an argument *against* the use of sorted logic in line with the standard argument "Everything you can do in your system of *sorted logic*, I can do in *unsorted logic*." This argument is – although correct – unhelpful because of the counterargument "Everything you can do in your *unsorted logic*, I can do in *sorted logic*, but in a smaller search space!"

The reason for the smaller search space is due to a better design realized by the sorted system. Although the formalism of sorted logic is equivalent in strength to the unsorted one, a concrete formulation (which makes use of sorts in a non-trivial way, that is, whose relativization is not equal to itself) is *not*. It is actually *weaker*, since certain things can *not* be derived in sorted logic which can be derived in unsorted logic (concretely, formulae like $\mathsf{Human}(1) \Rightarrow \mathsf{Mortal}(1)$ are syntactically possible in unsorted logic, but the equivalent is rejected in sorted logic). To generalise this observation: A particular design is *better* than an alternative one if certain redundant, or heuristically uninteresting derivations cannot be made.

The integration of sorts in a system also demonstrates how subtle design issues can be. We cannot discuss this in detail here, but we will give some indication. It should be noted that sorts are not necessarily exclusively beneficial.[8] If we take standard order-sorted logic we have for each relation like Human and Mortal to decide whether we want to formalize it by a sort symbol or by a predicate symbol. If we want to prove some statement like $\neg\mathsf{Human}(\mathsf{pegasus})$, we cannot formalize Human as a sort symbol. This is a serious flaw in standard sort systems, also it is not possible to model the genesis of concepts in an adequate way. In order to perform the reasoning above we need to know $\mathsf{socrates}{<}\mathsf{Human}$ a priori. It is not possible to infer that Socrates is a human being later in the reasoning process. This unduly limits the flexibility of the system. Ideally you would want to have the advantages of a sorted formulation whenever possible, but also benefit from the flexibility of the unsorted formulation when necessary. To our knowledge only Christoph Weidenbach's system [34] offers these possibilities.

4.2 Data Structures in Computer Algebra Systems

CASs offer many more primitive data structures such as matrices than standard theorem proving systems. With these structures it is possible to cover large parts of the 'computational' part of mathematics. As we tried to show, there is a grey area in which deduction and computation go hand in hand, since any structure in a CAS is concrete, while mathematical expressions often make use of ellipses, for instance, expressions which contain dots like $x_1, x_2, \ldots, x_n$, or the multiplication table and matrix in Sections 2 and 3.2. A promising first approach in the direction of formalizing ellipses in reasoning can be found in [9].

[8] For a detailed discussion why sorts/types may be harmful see [18].

4.3 Data Structures for Reasoning

Koedinger and Anderson [17] introduce a representation different from a purely logical formalization called diagram configuration model (DC) for diagrammatic reasoning in geometry. The representation is based on DC schemas that encode typical geometric situations that were identified through observations on the problem solving behavior of experts in this domain. The schemas contain the main property of the situation, the subsumed properties and the different ways under which the schema can be established. The level of abstraction allows for an efficient inference algorithm that introduce the DCs as inference steps.

Mateja Jamnik describes in [14] a diagrammatic representation for theorems and inference steps based on this data structure that allow to infer theorems in arithmetic. The proofs constructed in the diagrammatic representation can be translated to proof planning and then formally verified with a theorem prover.

These examples are important in our context since they show that at least for particular domains it is possible to build special reasoners for diagrammatic reasoning which are distinct from a purely logic based system, but which can be formally linked to such a system. An adequate data structure for diagrams seems to be the key point for the success of both approaches. Only this data structure allows the implementation of an efficient inference mechanism.

4.4 Little Theories

Another important logic-based approach which approximates mathematical reasoning well is the little theory approach of IMPS [11]. As we can see in textbooks from different areas of mathematics, there is no fixed hierarchy of theories. Each textbook presumes knowledge from different areas of mathematics and by this induces a partial order of theories. That there exists a total order of theories is rather a theoretical result than used in everyday mathematics. The attempt to realize this total order prohibits the flexibility to use theorems constructed for one area in another area. The little theory approach also allows to relate different formulations with each other without necessitating a hierarchy.

5 Conclusion

In this paper, we presented aspects of mathematical representations that we think are highly relevant for mathematical theorem proving and that can – if at all – be implemented in traditional theorem proving systems only with great difficulty. While systems which make use of a language which is comparatively close to mathematical practice are less of a problem than systems whose input languages are close to mathematical logic, we think that the lack of acceptance of theorem proving systems (compared to the success of computer algebra systems) is at least partly due to unsolved problems of mathematical design.

We have summarized some approaches to good design in Section 4. More work in this direction is necessary to offer well-designed tools to mathematicians and other people interested in computer assisted mathematics. While certain parts might turn out to be straightforward other aspects will require a much deeper understanding. To give one example for the size of the task, if we wanted to integrate multiplication tables as a primitive in automated reasoning systems, it should be relatively easy to offer concrete multiplication tables with all the advantages mentioned in sections 2 and 3 (as it is relatively easy to offer order-sorted sorts). However, it will be difficult to offer general type multiplication tables which contain ellipses, and flexible ways to reason about them. Human beings have an amazing capability to reify structures in their space of discourse. We seem not to have yet a deep understanding of these capabilities.

What is way forward? Let us go back to the quote from the start of this paper, since it clearly shows such a way forward. Andrzej Trybulec says that firstly, *"we need experiments with much more advanced mathematics than already done"*. Only complex examples show the complex problems and show ways to their solution. Addressing examples such as reasoning with matrices presents new challenges and when we solve those we can improve the corresponding systems.

Secondly, *"a system for practical formalization of mathematics probably will not be a simple system based on small number of primitive notions"*. We could not agree more. Good design is not done once and for all, it remains a continued task that mathematicians have to master. While many systems are carefully designed and makes life easy for many problems, we as a community have not yet solved the problem to give mathematicians the tools at hand to provide and adapt this design for themselves. This will be a big challenge for the future.

Thirdly, *"integration with a computer algebra systems may be necessary or at least a feasible system must have bigger computational power"*. The field of reasoning systems can certainly learn a lot from that of computer algebra systems, but also an integration of different activities is very important. Up to now strong systems often are quite specialized. To integrate them in a flexible way and to still keep them maintainable is unsolved.

Acknowledgements We thank Mateja Jamnik for fruitful discussions. Martin Pollet thanks Peter Pröhle for the insights he gave him into the process of mathematical problem solving.

References

1. Grzegorz Bancerek. The mutilated chessboard problem – checked by Mizar. In Roman Matuszewski, editor, *The QED Workshop II*, pages 37–38, 1995. Available from `http://www.mcs.anl.gov/qed/index.html`.
2. H. Barendregt and A.M. Cohen. Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants. *Journal of Symbolic Computation*, **32**(5):3–22, 2001.
3. M. Beeson. Automatic generation of epsilon-delta proofs of continuity. In J. Calment and J. Plaza, editors, *Artificial intelligence and symbolic computation*, pages 67–83. Springer Verlag, Berlin, Germany, LNCS 1476, 1998.
4. C. Benzmüller, A. Meier, E. Melis, M. Pollet, J. Siekmann, and V. Sorge. Proof planning: A fresh start? In M. Kerber, editor, *IJCAR-Workshop: Future Directions in Automated Reasoning*, pages 30–37, Siena, Italy, 2001.

5. R. Brachman and H. Levesque, editors. *Readings in Knowledge Representation.* Morgan Kaufmann Publishers, Los Altos, California, 1985.
6. N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. Nederpelt, J. Geuvers, and R. de Vrijer, editors, *Selected Papers on Automath*, pages 865–935. Elsevier, North-Holland, Amsterdam, The Netherlands, 1994. Studies in Logic, Volume 133.
7. A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. of the 9th CADE*, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany, LNCS 310.
8. A. Bundy. A science of reasoning. In *Computational Logic: Essays in Honor of Alan Robinson.* MIT Press, 1991.
9. A. Bundy and J. Richardson. Proofs about lists using ellipsis. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th LPAR*, pages 1–12. Springer Verlag, Berlin, Germany, LNAI 1705, 1999.
10. H. Elbers. *Connecting Informal and Formal Mathematics.* PhD thesis, Eindhoven University of Technology, 1998.
11. W. Farmer, J. Guttman, and F. Thayer. Little theories. In D. Kapur, editor, *Proc. of the 11th CADE*, pages 567–581, Saratoga Springs, New York, USA, June 1992. Springer Verlag, Berlin, Germany, LNAI 607.
12. P. Hayes. Some problems and non-problems in representation theory. In *Proc. of the AISB Summer Conference*, pages 63–79, Sussex, 1974.
13. Gérard Huet. The mutilated checkerboard (Coq library), 1996. `http://coq.inria.fr/contribs/checker.html`.
14. M. Jamnik. *Mathematical Reasoning with Diagrams: From Intuition to Automation.* CSLI Press, 2001.
15. Manfred Kerber. A dynamic Poincaré principle. In Jonathan M. Borwein and William M. Farmer, editors, *Mathematical Knowledge Management – 5th International Conference, MKM 2006*, pages 44–53, Wokingham, UK, 2006. Springer, LNAI 4108.
16. Manfred Kerber and Martin Pollet. A tough nut for mathematical knowledge management. In Michael Kohlhase, editor, *Mathematical Knowledge Management – 4th International Conference, MKM 2005*, pages 81–95, Bremen, Germany, 2006. Springer, LNAI 3863.
17. K. Koedinger and J. Anderson. Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science*, 14:511–550, 1990.
18. L. Lamport. Types are not harmless. Technical report, 1995.
19. U. Martin. Computers, reasoning and mathematical practice. In U. Berger and H. Schwichtenberg, editors, *Proceedings of the NATO Advanced Study Institute on Computational Logic, Marktoberdorf, Germany, July 29 - August 10, 1997.* Springer Verlag, 1999.
20. John McCarthy. A tough nut for proof procedures, 1964. Stanford Artificial Intelligence Project Memo No. 16, 1964. Available from `http://www-formal.stanford.edu/jmc/`.
21. M. Minsky. Future of AI technology. *Toshiba Review*, 1992. `http://web.media.mit.edu/~minsky/papers/CausalDiversity.txt`.
22. D. Norman. *The Design of Everyday Things.* The MIT Press, London, 1998.
23. Lawrence C. Paulson. A simple formalization and proof for the mutilated chess board. *Logic Journal of the IGPL*, 9(3):475–485, 2001. Also published as Technical Report Computer Laboratory, University of Cambridge, 394, May 1996.
24. F.J. Pelletier. The philosophy of automated theorem proving. In John Mylopoulos and Ray Reiter, editors, *Proc. of the 12th IJCAI*, pages 538–543, Sydney, 1991. Morgan Kaufmann, San Mateo, California, USA.

25. Martin Pollet, Volker Sorge, and Manfred Kerber. Intuitive and formal representations: The case of matrices. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *Mathematical Knowledge Management. Third International Conference, MKM*, Białowieża, Poland, September 19-21, 2004. Springer, LNCS 3119.
26. G. Pólya. *Mathematics and Plausible Reasoning.* Princeton University Press, New Jersey, USA, 1954.
27. G. Pólya. *Mathematical Discovery - On Understanding, Learning, and Teaching Problem Solving.* Princeton University Press, New Jersey, USA, 1962/1965.
28. A. Robinson. Proof = Guarantee + Explanation. In S. Hölldobler, editor, *Intellectics and Computational Logic*, pages 277–294. Kluwer Academic Publishers, 2000.
29. B. Russell. *The Principles of Mathematics.* George Allen & Unwin Ltd, London, UK, 2nd edition, 1937 edition, 1903.
30. B. Russell. The philosophy of logical atomism. *The Monist*, 28/29:495–527/32–63,190–222,345–380, 1918/1919. Republished in [31, p.177-281].
31. B. Russell. *Logic and Knowledge.* Allen & Unwin, London, 1956.
32. Alan P. Sexton and Volker Sorge. Processing textbook-style matrices. In Michael Kohlhase, editor, *Mathematical Knowledge Management, 4th International Conference, MKM 2005, Bremen, Germany, July 15-17, 2005, Revised Selected Papers*, volume 3863 of *Lecture Notes in Computer Science*, pages 111–125, 2006.
33. Andrzej Trybulec. Towards practical formalization of mathematics. In Fairouz Kamareddine, editor, *Abstracts of the invited talks at the Workshop on 35 years of Automath*, 2002. `http://www.cee.hw.ac.uk/~fairouz/automath2002/abstracts/abstracts.html`.
34. C. Weidenbach. Extending the resolution method with sorts. In *Proc. of the 13th IJCAI*, pages 60–65, Chambéry, France, 1993.
35. A.N. Whitehead and B. Russell. *Principia Mathematica.* Cambridge University Press, Cambridge, UK, 1910.
36. L. Wittgenstein. *Remarks on the Foundations of Mathematics.* Basil Blackwell, Oxford, England, third edition edition, 1956.
37. Claus Zinn. *Understanding Informal Mathematical Discourse.* PhD thesis, Universität Erlangen Nürnberg, 2004.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 10 (23) 2007

Gradual Computerisation/Formalisation of Mathematical Texts into Mizar

Fairouz Kamareddine, Manuel Maarek, Krzysztof Retel, and J. B. Wells

ULTRA group, Heriot-Watt University
`http://www.macs.hw.ac.uk/ultra/`

Abstract. We explain in this paper the gradual computerisation process of an ordinary mathematical text into more formal versions ending with a fully formalised Mizar text. The process is part of the MathLang–Mizar project and is divided into a number of steps (called aspects). The first three aspects (CGa, TSa and DRa) are the same for any MathLang–TP project where TP is any proof checker (e.g., Mizar, Coq, Isabelle, etc). These first three aspects are theoretically formalised and implemented and provide the mathematician and/or TP user with useful tools/automation. Using TSa, the mathematician edits his mathematical text just as he would use LaTeX, but at the same time he sees the mathematical text as it appears on his paper. TSa also gives the mathematician easy editing facilities to help assign to parts of the text, grammatical and mathematical roles and to relate different parts through a number of mathematical, rethorical and structural relations. MathLang would then automatically produce CGa and DRa versions of the text, checks its grammatical correctness and produce a dependency graph between the parts of the text. At this stage, work of the first three aspects is complete and the computerised versions of the text, as well as the dependency graph are ready to be processed further. In the MathLang–Mizar project, we create from the dependency graph, the roles of the nodes of the graph, and the (preamble) of the CGa encoding, a Mizar Formal Proof Sketch (FPS) skeleton. The stage at which the text is transformed into a Mizar FPS skeleton has only been explained through transformation hints, and is yet to be theoretically developed into an aspect that can be implemented and developed into a partially-automated tool. Finally, the Mizar FPS skeleton of the text is transformed (currently by hand as any Mizar expert would do and without any computerised tools) into a correct Mizar FPS and then into a fully formalised Mizar version of the text. Although we have tested our process on a number of examples, we chose to illustrate it in this paper using Barendregt's version of the proof of Pythagoras' theorem. We show how this example text is transformed into its fully formalised Mizar version by passing through the first three computerised aspects and the transformation hints to obtain Mizar FPS version. This version is then developed by hand into a fully formalised Mizar version.

ISBN 978-83-7431-128-1 ISSN 0860–150X

1 Introduction

The past forty years have seen a growing number of uses of the computer in the daily routine of the mathematician. These uses range from authoring tools (e.g., LaTeX, MathML), to computation and calculation aids (e.g., Mathematica) to proof checking tools (e.g., Mizar). Proof checking tools have had the least uses by ordinary mathematicians since they are completely different from traditional mathematical authoring, and remain difficult to use by non experts. Even if the language behind the proof checking tool closely mimics the common mathematical language (CML – the language and style mathematicians use to write their mathematics), the formalisation process remains very long, labor-intensive and will require expertise in at least programming and logic. Furthermore, for a mathematical text to be fully verified by a proof checking tool, all its informal parts and proofs need to be rewritten in sufficient details before being processed for correctness. Mathematicians do not like writing proofs or details that they consider to be obvious or trivial. Furthermore, mathematicians prefer developing new or studying existing mathematical theories rather than proof checking using the computer existing theories. And so, the gap between the mathematician and the computer proof checker remains large.

Recent years have seen many attempts to bridge this gap. For example, some work has been done on computerising mathematical texts without fully formalising or computer proof checking them. Such computerisations are not sufficiently detailed for correctness verification but are used as *skeletons* in the full formalisation (see Wiedijk's work [25]). Although the computerised text remains at a low level to be fully automatically checked, it has a precise notion of correctness: it is *syntactically* correct according to the grammar language but according to the proof language it contains steps that are not sufficiently justified. However, in order to create these skeletons, the user still needs to be an expert in the final destination language. For example, for a mathematician to carry out the work as outlined in [25], he/she needs to be an expert in Mizar.

In this paper, we allow the ordinary mathematician to do a reasonable amount of work on computerising mathematical texts that lead to computerised versions that can be passed to any expert in any proof checker to be fully formalised in that proof checker. We choose Mizar to be our target proof checker, and we consider G. H. Hardy and E. M. Wright's skeletons as one of the steps to reach the final Mizar version. However, the skeleton and the Mizar version are obtained not from CML texts, but from versions computerised by the mathematician which have gone through a number of automatic checking and manipulations. These computerised versions are easier to transform into skeletons and final Mizar version.

Our approach is sketched in Figure 1. Instead of a Mizar expert transforming the CML text into a fully formalised Mizar version following one of the paths:

- ⓒ: immediately create the fully formalised Mizar version of the text;
- ⓑ-ⓔ: first create the Mizar Formal Proof Sketch skeleton and then the fully formalised Mizar version of the text,

we believe that each of the formalisation paths ⓒ and ⓑ-ⓔ could be divided into a number of smaller steps as in the path ⓐ-ⓓ-ⓔ of Figure 1 where all the levels at step ⓐ are done by the mathematician and where the sub-path ⓓ-ⓔ is done by the Mizar expert. This approach has a number of advantages:

- It gives a better view at the process of computerisation.
- It helps build computer programs that can assist humans along the computerisation/formalisation processes. In fact, at the TSa, CGa and DRa levels, the user already enjoys numerous automated help which makes his work almost minimal. We also aim for partial automations of steps ⓓ and ⓔ.
- It shows that the mathematician can benefit by first authoring the text, and later on 'tagging' it within the MathLang system and checking its grammatical correctness (see Figure 2 and Section 2.1).
- As expressed in the description of Figure 1, different formalisation paths involve different levels of the expertise required by the user.

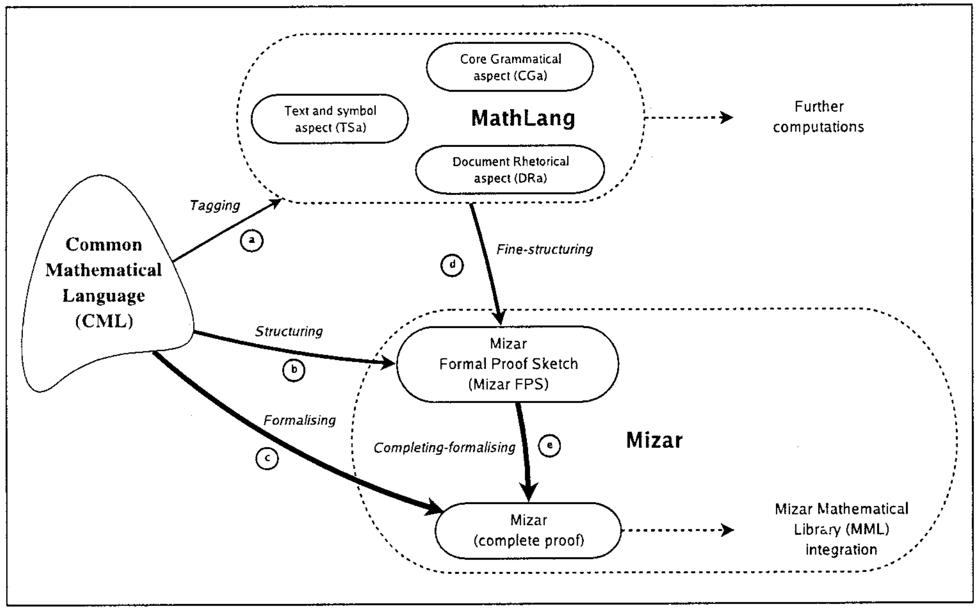

Core Grammatical aspect (CGa)
Text and symbol aspect (TSa)
MathLang
Document Rhetorical aspect (DRa)
Further computations
Tagging
ⓐ
Common Mathematical Language (CML)
ⓓ
Fine-structuring
Structuring
ⓑ
Mizar Formal Proof Sketch (Mizar FPS)
Formalising
ⓒ
Completing-formalising
ⓔ
Mizar
Mizar (complete proof)
Mizar Mathematical Library (MML) integration

Fig. 1. Computerisation/formalisation paths from CML to Mizar.
The labeled arrows shows the computerising paths from CML to Mizar. In this paper we mainly focus on the path ⓐ-ⓓ-ⓔ. We also briefly compare it with the path ⓑ-ⓔ and the path ⓒ. The width of the arrow representing each path segment increases accordingly to the expertise required to achieve the path segment. The dashed arrows illustrate further computerisation that one can envision.

We have chosen Mizar as the final destination since it comes with the biggest library of mathematical knowledge verified by computer. This is important since, most CML texts assume that the reader has at least the fundamental mathematical knowledge required to follow the topic in the document. Therefore, if we want to formalize a mathematical text, we have to refer at some point to that fundamental knowledge. Mizar has an impressive library of mathematical knowledge: the Mizar Mathematical Library (MML). In addition to its MML, Mizar is the most accessible of the proof checkers (in terms of readability and write-ability) by mathematicians as expressed by the Mizar creator Trybulec:

> Experience has shown that many people with some mathematical training develop a good idea about the nature of the Mizar language just by browsing through a sample article. This is no big surprise, since one of the original goals of the project was to build an environment which supports the traditional ways that mathematicians work. *[20, pp.2]*

This said, the user will still need to have an expert knowledge if he/she wants to deal with the whole mechanism (i.e. the Mizar system, MML, the Mizar Language, the MML search engines) and create a new Mizar document.

1.1 Example

The example we use in this paper is taken from [26] where Wiedijk used Hardy and Wright's version [8, Ch. IV] of Pythagoras' theorem of irrationality of $\sqrt{2}$ to compare computer based theorem provers. Barendregt wrote a textual version [3] (reproduced in Figure 8, page 118) of this proof which is said to be *"informal"* in contrast to the formal versions of theorem provers as in [26]. Wiedijk's comparison illustrates the need to assist the mathematician non-expert in theorem provers. We have already used this example in [11] to obtain what was then a CGa version. In this paper, we use this example to show all the versions of the proposed path (TSa, CGa, DRa, Mizar FPS skeleton, Mizar FPS and finally Mizar).

1.2 Notations, contributions and outline

Notations. We adopt some rules to facilitate reading this paper. We use different font styles to differentiate MathLang and Mizar syntax and notions. MathLang's abstract syntax is written in the sans-serif font style, in contrast to Mizar's syntax, which is highlighted using the `typewriter` font style. MathLang jargon words are written using the **boldface** family font. Specific Mizar jargon words are written using the *slanted* font style.

	MathLang	Mizar
Syntax	sans-serif	`typewriter`
Jargon	**boldface**	*slanted*

Contributions. Our contributions can be summarised as follows:
1. *A gradual computerisation/formalisation into Mizar.* We propose a new approach for the gradual computerisation/formalisation of a CML document into its more formal versions ending with a fully formalised Mizar version. This gradual approach shows the increasing level of expertise required to achieve the fully formalised Mizar version, and allows the mathematician and the Mizar expert to collaborate in the process (see the explanation of Figure 1).
2. *Transformation hints.* Through the example, we give transformation hints that allows us to create a skeleton of a document in a fully formal Mizar language.
3. *A short comparison between MathLang and Mizar constructs.* Through the example, we compare different MathLang constructions with their counterparts in Mizar. This brief comparison provides hints and ideas on which knowledge is required to understand the original document, how this document could be stored in MathLang and in Mizar where identifiers are taken from MML.

Outline. In Section 2 we briefly explain the various boxes shown in Figure 1. We also explain the transformation path we are following to build a formal document in the Mizar language. In Section 3 we explain how the DRa annotation on the CML text helps us to build a rough skeleton of the Mizar document. In Section 4 we give hints as to how mathematical identifiers and their CGa presentation could be used to narrow their representation in Mizar and give further transformation hints of the MathLang document into Mizar. In Section 5 we reflect on the different formalisation paths. Finally, in Section 6 we conclude and describe related and future work.

2 Background

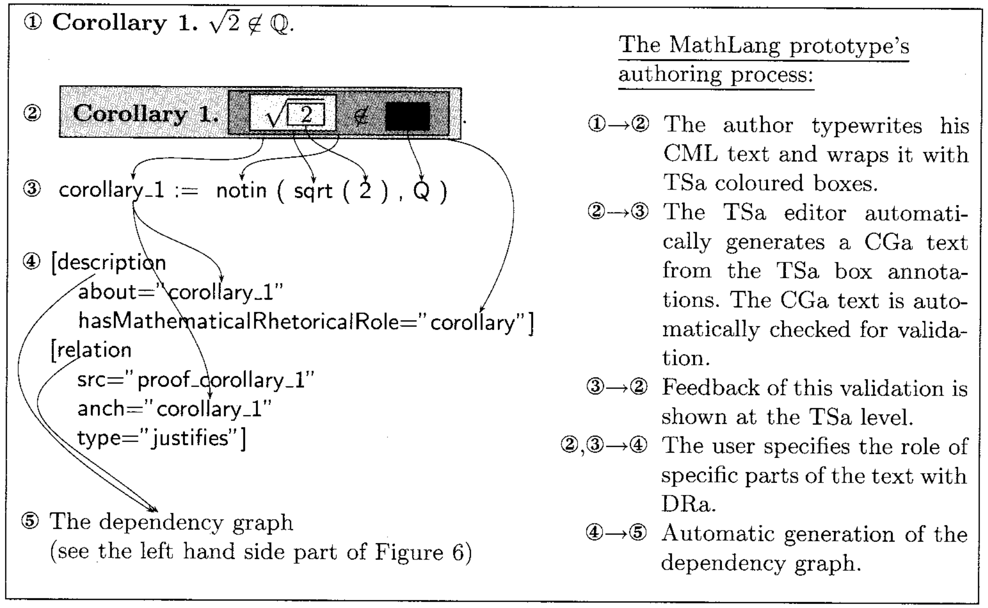
2.1 MathLang and its aspects

Since 2001, the ULTRA group has been developing as part of the MathLang project, a number of prototypes for computerising mathematics. The project MathLang aims to give *alternative* and *complete* paths which transform mathematical texts into new computerised and/or formalised versions. These paths are intended to accommodate different degrees of formalisation, different mathematical editing/checking tools and different proof checkers. Dividing the formalisation of mathematical texts into a number of stages was first proposed by N.G. de Bruijn to relate CML to his Mathematical Vernacular [6] (MV) and his proof checking system Automath. We call this principle *de Bruijn's path.*

> The work may be subdivided. One can think of a first stage where a person with some mathematical training inserts a number of intermediate steps whenever he feels that further workers along the belt might have trouble, and a second stage where the logical inference rules are supplied and the actual coding is carried out. For the latter piece of work one might think of a person with just some elementary mathematics training, or of a computer provided with some artificial intelligence. But we should not be too optimistic about that: programming such jobs is by no means trivial.*[5]*

MV was proposed as a formal substitute for parts of CML. Nederpelt refined MV into another formal substitute for parts of CML, Weak Type Theory (WTT) whose underlying proof theory was developed by Kamareddine [17,14]. MathLang started from de Bruijn's path idea and Nederpelt's WTT and was faced with the huge challenge of how to really create a path from original mathematical texts into fully formalised ones and how would this path differ for different choices of texts, text editors, logical frameworks, and proof checkers. Soon after a number of prototypes were built, it became obvious that the stages of the path and the formal substitute of CML need to be seriously revised.

The MathLang language expressiveness has been increased and its description simplified in comparison with MV and WTT. Moreover, MathLang adopted to decompose the computerisation process by means of knowledge components. Each element of this decomposition is defined in what we call, an **aspect**. In the current development of MathLang we have defined three aspects (CGa, TSa and DRa) which we explain below. Figure 2 illustrates with a sentence from our example (see Section 1.1) the viewpoint each aspect gives to the same text.


① Corollary 1. $\sqrt{2} \notin \mathbb{Q}$.
② Corollary 1. $\sqrt{2} \notin$.
③ corollary_1 := notin (sqrt (2) , Q)
④ [description
about="corollary_1"
hasMathematicalRhetoricalRole="corollary"]
[relation
src="proof_corollary_1"
anch="corollary_1"
type="justifies"]
⑤ The dependency graph
(see the left hand side part of Figure 6)

The MathLang prototype's authoring process:

- ①→② The author typewrites his CML text and wraps it with TSa coloured boxes.
- ②→③ The TSa editor automatically generates a CGa text from the TSa box annotations. The CGa text is automatically checked for validation.
- ③→② Feedback of this validation is shown at the TSa level.
- ②,③→④ The user specifies the role of specific parts of the text with DRa.
- ④→⑤ Automatic generation of the dependency graph.

Fig. 2. Example of mathematical authoring using MathLang's aspects.

To illustrate the decomposition of mathematical knowledge with MathLang's aspects, we use the example in ①: the definition of Corollary 1 which states the irrationality of $\sqrt{2}$. We identify in this sentence the grammatical role of each element of the text: **definition** for the entire sentence, **term** for "$\sqrt{2}$" and "2", **set** for "$\mathbb{Q}$" and **statement** for "$\sqrt{2} \notin \mathbb{Q}$". The author attributes to each element its CGa grammatical role by wrapping it into a coloured box following our colour coding system of Figure 3. The formal interpretation of this sentence is automatically generated from TSa's ② and is printed in ③ using CGa's abstract syntax as defined in [13]. The identifiers corollary_1, notin, sqrt, 2 and Q are provided by the user as arguments for each coloured box of ②. Note that the CGa syntax used in ③ is not meant to be used by the end-user of MathLang, it is only designed for computerisation purposes. The MathLang end-user edits his MathLang document using the view offered by TSa, as shown by ② (TSa plays the role of a user interface for MathLang). The internal syntax used in our implementations follows XML recommendations. In ④ we indicate with DRa what role this sentence plays in the context from which it was taken. Sentence ① plays the role in the entire example (see [3] or Figure 8) of corollary and is being justified by the proof which follows. DRa's ④ makes this explicit (see DRa's role list in Table 1). Finally, we automatically produce a dependency graph ⑤ out of the DRa information (see the left hand side part of Figure 6).

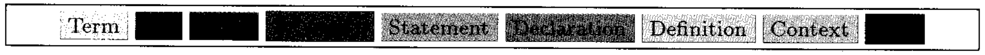


Fig. 3. MathLang CGa's colour coding system.

The Core Grammatical aspect (CGa) is a formal language derived from MV and WTT which aims at expliciting the grammatical role played by the elements of a CML text. The structures and common concepts used in CML are captured by CGa with a finite set of grammatical categories. **Terms** represent mathematical concrete objects such as "$\sqrt{2}$" from our example in Figure 2. A **set** is a collection of objects like the set of rationals "$\mathbb{Q}$". A **noun** is a family of mathematical objects that share common characteristics, "number" is an example of a **noun. Nouns** could be refined by **adjectives** like "even". A valid document according to CGa's grammar is a succession of **phrases** (**statements**, **declarations** and **definitions**). A **phrase** could be combined in a sequence to form a deduction, this sequencing is called a **block**. One can restrict a **statement** and the **declaration/definition** of identifiers to a specific part of the text with a construction called **context** or **local-scoping**. We call **step** an expression which is either a **phrase**, a **block** or a **local-scoping**. MathLang's type system [13] derives typing judgments to check whether the reasoning parts of a document are coherently built. CGa is intentionally elementary and results in a computable low level encoding.

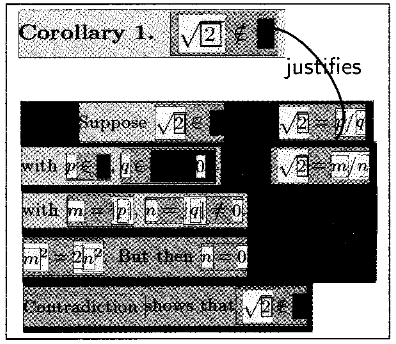

Corollary 1. $\sqrt{2} \notin$
justifies
Suppose $\sqrt{2} \in$
$\sqrt{2} = p/q$
with $p \in$, $q \in$ 0
$\sqrt{2} = m/n$
with $m = |p|, n = |q| \neq 0$
$m^2 = 2n^2$. But then $n = 0$
Contradiction shows that $\sqrt{2} \notin$

Fig. 4. MathLang's encoding of part of Barendregt's version of Pythagoras' theorem. This coloured text is the definition and proof of Corollary 1 as written by Barendregt (see Section 1.1). We used TEXmacs and MathLang's plugin for TEXmacs to write this example. The text was automatically checked by our implementation of the MathLang type system [13]. The arrow on top of this text shows the DRa relation between the corollary's definition and its proof.

The Text and Symbol aspect (TSa) builds the bridge between a CML text and its grammatical interpretation and adjoins to each CGa expression a string of words and/or symbols which aims to act as its CML representation. The CGa grammar provides computable constructions to represent mathematical reasoning. We added to this strict language information on how each CGa element should be printed on paper or on screen. This makes MathLang's encoding of mathematical texts faithful to traditional mathematical authoring [11]. TSa adds on top of a CML text a new dimension to the document. This dimension is rendered in our examples in Figures 2 and 4 with coloured boxes following the colour coding system of Figure 3.

The coloured boxes shown in Figure 4 are added by the MathLang user himself. We implemented TSa in a *plugin* for the scientific text editor TEXmacs (`http://www.texmacs.org/`). The view of our example shown in Figure 4 was authored using TEXmacs and our *plugin*. The *plugin* transforms the authored document (encoded in TEXmacs' data structure extended with TSa boxes) into MathLang CGa internal XML representation. The document is then checked for CGa grammatical validation. In the rest of this paper we avoid mentioning TSa and concentrate instead on the use one can make of the information held by CGa and DRa.

The Document Rhetorical aspect (DRa) extends the knowledge already computerised in CGa by expliciting the subjective judgments that the author gave on the role some text parts play in the document's structure. At the CGa level, a document is decomposed into **steps** either put in a sequence or contextualized by the **context** construction. One would encode *division elements* (such as *chapter*, *section*) and *mathematical labeling units* (such as *axiom*, *theorem*, *proof*) by this unique step construction (see the MathLang encoding examples in [12,11,13]). In our example in Figure 4, the proof paragraph is a **step** composed by several sub-**steps**. To enhance flexibility, CGa does not differentiate between these divisions, labels and any other kind of **step**. DRa provides a method to computerise these labels traditionally attributed to chunks of text. These labels and text elements when used in mathematical textbooks or articles give important hints and indications on how to interpret a chunk of text. Moreover, relations between recognised chunks of text sometimes stay implicit in the original document. DRa annotations allow to express such information since the DRa works as an annotation system for the CGa **step**. This annotation system is summarised in Table 1 and consists of:

1. Structural rhetorical role names for *division elements*.
2. Mathematical rhetorical role names for *mathematical labeled units*.
3. A set of *relations* to express relations between *labeled units*/*division elements*.

Using the DRa annotation system we can capture the role that a chunk of text plays in a document and the relationship that this role imposes on the rest of the document or other chunk of text. This leads to an automatic generation of a dependency graph for the text (see Figure 8) where relations between parts of the text are represented by visible arrows and graph nodes have specified (but not visible) mathematical rhetorical roles. From the annotated narrative feature of a document we receive a *dependency graph* between the chunks of text in a document (e.g. see the left hand side of Figure 6). Those dependencies play an important role in the mathematical knowledge representation. Thanks to those dependencies, the reader can find his own way while reading the original text without the need to understand all its subtleties. Moreover, we will show that these dependencies give the ability to structure the skeleton of a document in a formal language Mizar (see Figure 6).

Description
Instances for the hasStructuralRhetoricalRole *property:* preamble, part, chapter, section, paragraph, *etc.*
Instances for the hasMathematicalRhetoricalRole *property:* lemma, corollary, theorem, conjecture, definition, axiom, claim, proposition, assertion, proof, exercise, example, *etc.*
Relation
Types of relation*:* justifies, subpartOf, uses, exemplifies, inconsistentWith

Table 1. DRa annotations.

2.2 Mizar and Formal Proof Sketch

The Mizar system (`http://mizar.org`) is a system for computer checked mathematics [20,21,19,15]. The ongoing development of the Mizar framework, lead by Trybulec since 1973, has resulted in several things: the Mizar system, the Mizar language, the Mizar library and the Mizar software utilities for working with Mizar documents and the Mizar library.

The Mizar language is used for recording mathematics whereas the Mizar system is used for checking the correctness of texts written in this language. The Mizar language is a language suitable for the practical formalisation of mathematics. It is based on first–order logic with free second-order variables. Proofs are written in the style of natural deduction as proposed by Jaśkowski [9]. The language itself is also an attempt to approximate in a formal way the mathematical vernacular used in publications. On one hand, the Mizar language inherits the expressiveness, naturalness and freedom of reasoning of CML. On the other hand it is formal enough to allow mechanical verification and computer processing.

The Mizar system is accompanied by a library of mathematics – the Mizar Mathematical Library (MML), which is the biggest collection of digitalized mathematical texts verified by computer [24]. MML consists of Mizar documents, which are called *Articles* within the Mizar community. This library is based on two axiomatic *Articles*: `HIDDEN` [4] which consists of built-in notions, and `TARSKI` [22] which presents axioms of the Tarski-Grothendieck set theory. All the other *Articles* of MML are consequences of those axioms and are verified by the Mizar system. The user while writing a new Mizar *Article* reuses the notation, definitions and theorems and other constructs stored in the library. The Mizar system assists the author while formalising new terminology and results. It verifies the claims of the new *Article* and extracts facts and definitions for inclusion into the library. The task of building a rich mathematical library is currently the main effort of the Mizar community. Currently, the library includes 960 *Articles* contributed by 189 authors[1], a number of whom have been active on a long term basis. There is a number of introductory papers and manuals on Mizar [1,20] as well as practical hints for writing Mizar *Articles*.

A Mizar article consists of two parts: the *Environment-Declaration* and the *Text-Proper*. The *Environment-Declaration* begins with `environ` and consists of *Directives*: `vocabularies`, `notations`, `constructors` etc. Roughly, each *Directive* is composed of names of *Articles* from the MML that contain the knowledge required for verifying the correctness of the *Text-Proper*. The *Text-Proper* is a sequence of *Sections*, where each *Section* starts with `begin` and consists of a sequence of theorems and definitions together with their proofs. The division of the *Text-Proper* into *Sections* has no impact on the correctness of the *Article*.

These two parts of the Mizar *Article* are processed by two different programs: *Accommodator* and *Verifier*. The *Accommodator* processes the *Environment-Declaration* and creates the *Environment* in which the knowledge is imported from MML. The *Verifier* has no communication with the library and checks the correctness of the *Text-Proper* using the knowledge stored in the *Environment*.

The Formal Proof Sketch (FPS) notion was introduced by Wiedijk in [25] for declarative systems where the input language of the system is designed to be similar to the language of the informal proofs found in mathematical papers. The FPS notion makes sense for instance for both the Mizar language and the Isar language (used for the Isabelle system). According to Wiedijk:

> A *Formal Proof Sketch* is a text in the syntax of a declarative proof language that was obtained from a full formalization in that language by removing some proof steps and references between steps. The only errors (according to the definition of the proof language) in such a stripped for-

[1] `http://merak.pb.bialystok.pl/` (last accessed on 2007-02-15, MML version: 4.76.959).

> malization should be *justification errors*: the errors that say that a step is not sufficiently justified by the references to previous steps. *[25]*

Even if the above definition states that the FPS version is derived from the full formalization, the process of formalization can start from the informal mathematical document. The process actually consists of two phases: first, one mimics the informal English proof in the formal proof sketch language, second, one fleshes out this formal proof sketch to a full formalization.

The Mizar Formal Proof Sketch (Mizar FPS) is a representation of an informal proof in the formal Mizar language. A text in Mizar FPS is between a fully checkable proof and a statement without any proof at all. It is seen as an incomplete Mizar *Article* that contains holes in the natural deduction reasoning. The application of the Mizar system for a correct Mizar FPS text should result in only one kind of error (the well known *4 error in the Mizar system), which says that justifications do not necessarily justify the steps. A Mizar Formal Proof Sketch can be completed into a correct fully formalised Mizar *Article* by adding steps and filling essential references for the steps to the proofs. However, it may sometimes happen that the Mizar FPS version needs to be changed to be able to reach full formalisation in the Mizar system. In short, the Mizar FPS version and the full formalisation of an informal text are both written in the same formal language – the Mizar language, and are both checked by the same software – the Mizar system; furthermore, Mizar FPS accepts holes in the reasoning.

3 Narrative features vs. Mizar *Text-Proper* skeletons

The purpose of DRa is to discern explicitly the structure of mathematical knowledge for providing a better encoding of its content, see Figure 8. Using the DRa annotation system we indicate where important mathematical statements start and end. One may argue that this information is visible and we do not need to annotate it explicitly. However, although this information is obvious for a human, it has to be explicitly specified for a computer. As described in Section 2.1, the DRa annotations of a text are used to automatically generate the dependency graph of the text where the relationships between different parts of the text are represented by visible arrows and where the graph nodes have well specified (but not visible) mathematical/structural roles (see the left hand side of Figure 6). We advocate that the DRa annotations of a text and the automatic generation of its dependency graph are useful for the computerisation of a mathematical text because it explicits the narrative features of the text. In this section, we explain how the DRa annotations of a text and its automatically generated dependency graph are used to create a Mizar FPS *Text-Proper* skeleton of the text.

3.1 Transformation hints provided by the dependency graph

Note that the DRa dependency graph (see the left hand side of Figure 6) does not impose any logical correctness. For instance, on our example (see Figure 4), the paragraph labeled proof is related by justifies to a mathematical sentence labeled corollary (see Figure 8). However, this does not imply that the proof indeed proves the corollary. This latter affirmation is of a different level of importance, and within MathLang it belongs to a different **aspect** than the labeling and relation information. We expect to get an automatic validation of the coherence of relationships drawn in the document (for instance a block of steps labeled proof can not justifies another step labeled axiom).

In this section we give transformational hints which use the dependency graph of a text and the internal representation of the mathematical/structural roles of its nodes, to create a Mizar FPS *Text-Proper* skeleton of the text. It should be noted that we call this stage transformation hints rather than give it a full blown "aspect" status like CGa, TSa or DRa because we have not completed its formalisation/implementation. Currently, we simply give hints to the user. In the future, any implementation of this desired aspect should ask the user, how each relation is used, and in which order the annotated (boxed) text should be.

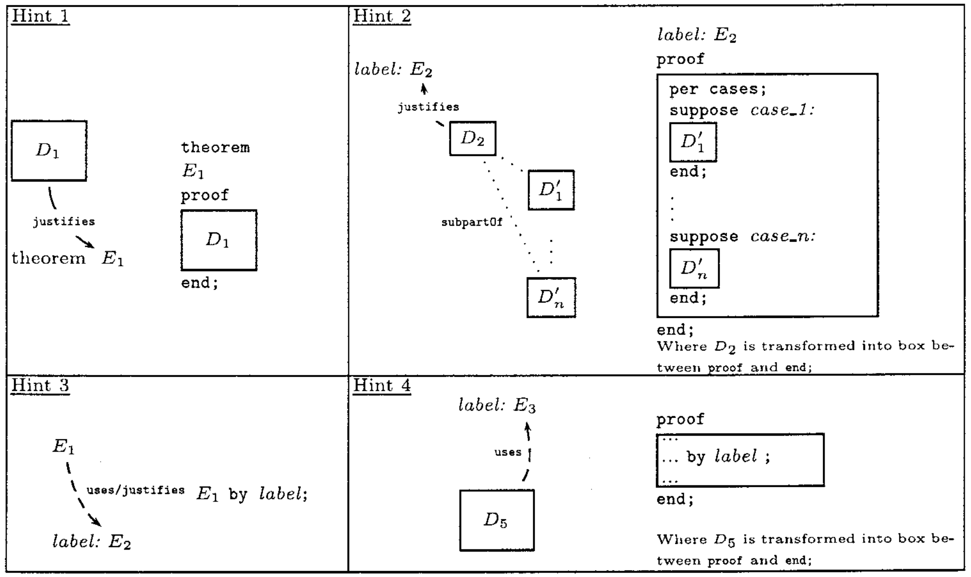


Fig. 5. Four transformation hints provided by the dependency graph.

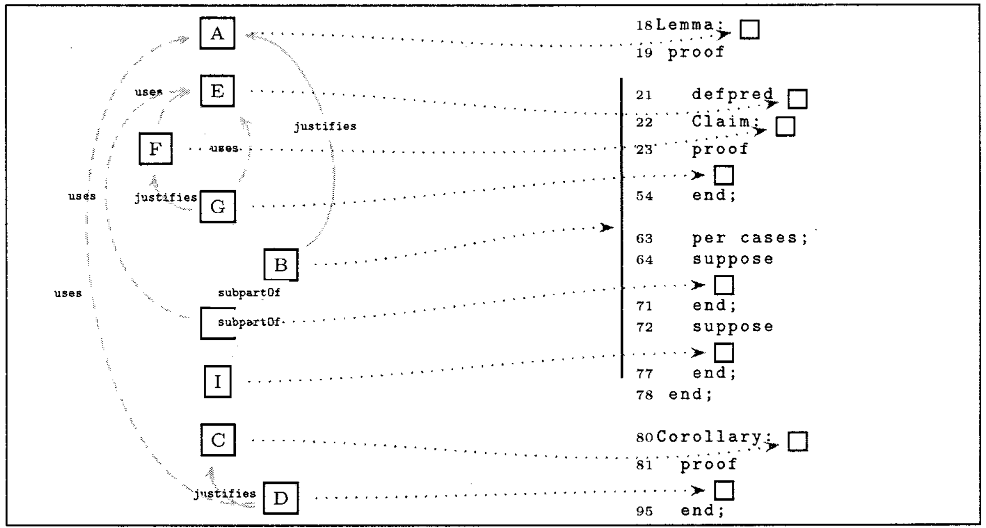
Figure 5 lists four hints that we use to transform our main example. In each of these hints, a dependency graph (on the left hand side of the hints) is transformed into a Mizar specific structure (on the right hand side). For example, in hint 1, if we annotate a box, let say E_1, as a theorem, it could be transformed into Mizar syntax as: `theorem` E_1. Moreover, if we say that a box $\boxed{D_1}$ has the mathematical role proof, then we can transform it into: `proof` $\boxed{D_1}$ `end;`. Moreover, since a block of steps having the mathematical role proof is in relation justifies with a single statement, we can say that this is a particular *Proof Justification* in Mizar, which is transformed into a specific form like the right hand side of hint 1.

Hint 2 deals with proof by cases. If $\boxed{D_2}$ is in relation justifies with a statement E_2, and consists of the parts $\boxed{D'_1}, \ldots, \boxed{D'_n}$, then we can give a user hint that this is a *Proof Justification* in which reasoning is done by all the cases.

In hint 3, the relation uses, could express a Mizar *Straightforward-Justification*, for instance, if a sentence E uses or justifies sentence E, then we can inform the user that this corresponds to a Mizar *Straightforward-Justification*.

In the dependency graph of hint 4, block $\boxed{D_5}$ uses statement E_3. Here, we transform block $\boxed{D_5}$ into a specific Mizar *Proof* block, which contains an expression with *Straightforward-Justification* to statement E_3.

3.2 Applying the transformation hints to our example



```
18Lemma: □
19 proof
21    defpred □
22    Claim: □
23    proof
      □
54    end;
63    per cases;
64    suppose
      □
71    end;
72    suppose
      □
77    end;
78 end;
80Corollary: □
81  proof
      □
95  end;
```

Fig. 6. Transformation into *Text-Proper* skeleton.
The left hand side reproduces the MathLang dependency graph of our example (Figure 8). On the right hand side we show the Mizar *Text-Proper* skeleton of the same example. The arrows from left to right shows how the MathLang dependency graph gives hints on how to build the Mizar *Text-Proper* skeleton.

Using the transformation hints of Figure 5, we can transform the dependency gaph produced for our main example (see the left hand side of Figure 6) into a proper structure of the *Text-Proper* part of our Mizar FPS (see the right hand side of Figure 6). As already discussed, this transformation is not done automatically. It is our intention in the near future, to formalise and implement the transformation into a further aspect of MathLang.

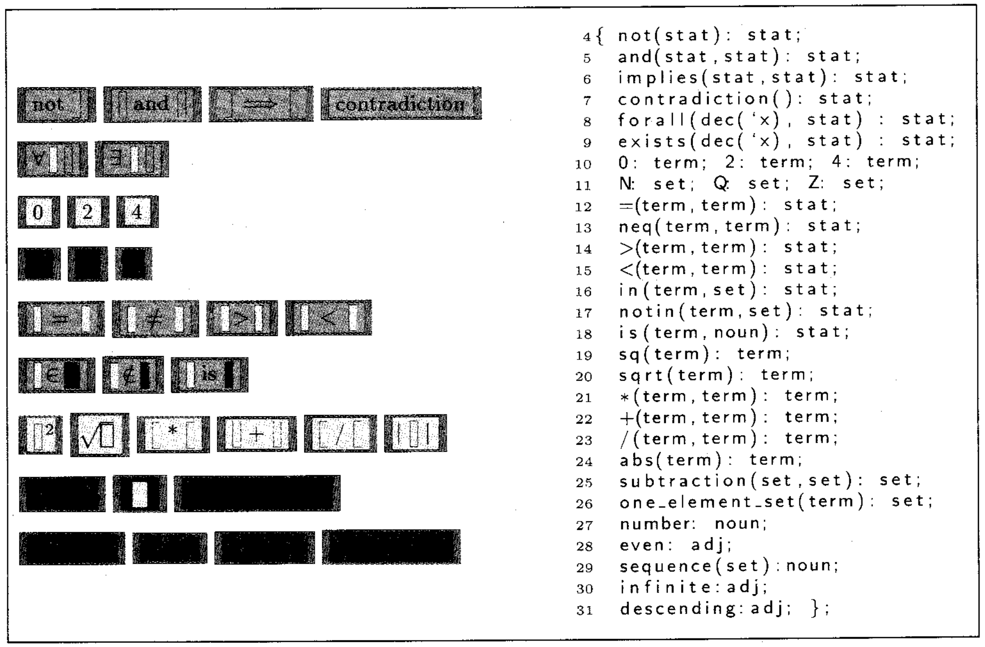
4 Building parts of Mizar FPS from a grammatically annotated document

4.1 The document's background knowledge

When a document has been properly encoded in CGa, all the notions used in the document would be properly declared with the appropriate CGa grammatical categories. This results in a list which declares the identifiers used in the document and which form an important part of the background knowledge required to understand it. For example, the arithmetic operations plus, times or square root are not defined in our main example (see Figure 8) but are assumed to be known by the reader. At the CGa encoding level of our example, these arithmetic operations need to be declared at the start of the encoded document. The common way to do so is to start the document with a **context** containing the list of declarations of all these symbols and notions (see Figure 7). At the DRa level, we identify this list of declarations as a **preamble** by annotating the CGa paragraph containing the left hand side of Figure 7 by:

[description hasStructuralRhetoricalRole="preamble"]

In our example, the identifier even (line 28 of Figure 7) is in the **preamble** because it is used in the original document but not defined. We expect these identifiers to have a proper definition outside the original text. One can understand them with a good mathematical background or with access to the background literature (we omit here the way MathLang adopts to refer to external documents). Each of these externally defined identifiers has to be declared.



```
4{ not(stat): stat;
5  and(stat,stat): stat;
6  implies(stat,stat): stat;
7  contradiction(): stat;
8  forall(dec('x), stat) : stat;
9  exists(dec('x), stat) : stat;
10 0: term; 2: term; 4: term;
11 N: set; Q: set; Z: set;
12 =(term,term): stat;
13 neq(term,term): stat;
14 >(term,term): stat;
15 <(term,term): stat;
16 in(term,set): stat;
17 notin(term,set): stat;
18 is(term,noun): stat;
19 sq(term): term;
20 sqrt(term): term;
21 *(term,term): term;
22 +(term,term): term;
23 /(term,term): term;
24 abs(term): term;
25 subtraction(set,set): set;
26 one_element_set(term): set;
27 number: noun;
28 even: adj;
29 sequence(set):noun;
30 infinite:adj;
31 descending:adj; };
```

Fig. 7. Preamble of MathLang's encoding of Pythagoras' theorem
The left hand side presents the **preamble** as shown by TSa whereas the right hand side shows the corresponding lines in the automatically generated CGa (printed using CGa's abstract syntax as defined in [13]).

In Mizar, the *Environment* plays a similar role to the MathLang **preamble** by describing the background knowledge of the *Article*, however, there is one subtle difference. Namely, the *Environment* in Mizar lists the MML entries that have to be loaded prior to any analysis of the *Text-Proper* part of the *Article* (see Listing 1.1 for our example's *Environment*). The *Articles* to be loaded contain, among other things, the notations and definitions that are used in the *Text-Proper* part. This gives a slightly different constraint to the authoring: in MathLang the author simply needs to list the external identifiers, whereas in Mizar the author needs to select the background MML literature to use. The MathLang CGa is more concerned with the "visible" external identifiers (CGa is about the grammatical completeness and therefore only needs the grammatical signature of each identifier) but Mizar needs to have a complete semantic and logic background (with *Definitions* or *Proofs* associated to each identifier).

Listing 1.1. Mizar FPS *Environment*

```
environ
vocabularies INT_1, SQUARE_1, MATRIX_2, IRRAT_1, RAT_1, ARYTM_3, ABSVALUE,
  SEQM_3, FINSET_1;
notations INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1, XCMPLX_0,
  INT_2, SEQM_3, FINSET_1, REAL_1, PEPIN;
constructors INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1,XCMPLX_0,
   INT_2, SEQM_3, FINSET_1, PEPIN;
requirements SUBSET, NUMERALS, ARITHM, BOOLE, REAL;
registrations XREAL_0, REAL_1, NAT_1, INT_1;
```

The **preamble** of the CGa encoding is crucial in the migration process of encoding into a Mizar FPS version of the text. We treat the information of the **preamble** as a subset of the Mizar *Environment*. In Mizar FPS we use the same symbols and identifiers that were explicitly introduced in the CGa, although some of them have different spelling. We have to remember that at some point we can acquire the CGa encoding of a mathematical text, which could contain identifiers or symbols that have not been defined in the MML yet. In such case we have to define those identifiers in the *Text-Proper* part of the Mizar FPS and introduce their names in the associated *Vocabulary* file. This situation requires much more investigation in the future.

In this section, we use the **preamble** to build two parts of the Mizar FPS *Environment-Declaration*, namely *Directives*: `vocabularies` (which consist of MML entries that store symbols used in the *Text-Proper* part of the *Article*) and `notations` (which consist of MML entries that store notions of symbols used in the *Text-Proper*). The information in the **preamble** gives hints how the identifiers could correspond to Mizar counterparts. By filling only those two *Directives* in the *Environment-Declaration*, we can check the *Text-Proper* part of the Mizar FPS in terms of "grammatical correctness". After those *Directives* are fully filled, we call the Mizar system with a special option (i.e. `accom -p $PATH/file_name.miz` and `verifier -p $PATH/file_name.miz`). If the Mizar system does not return any error, then the *Text-Proper* part of the *Article* is "grammatically correct" according to the Mizar grammar, and the symbols and their *Formats* that are used in the *Text-Proper*. The *Format* describes the number of arguments and the order (infix, prefix or postfix) in which the arguments of a *Constructor Symbol* may be used. Although, this partially filled *Environment-Declaration* allows to check the "grammatical correctness" of the *Text-Proper*, the *Environment-Declaration* needs to be more fully filled to achieve a proper Mizar FPS where the only errors are *Justification* errors.

We do not show in the paper how to build the proper *Environment* and how to search the MML. We only express briefly that identifiers in the CGa correspond more or less to Mizar *Items*. Such correspondence gives the overall idea and hints as to what kind of *Items* we have to search for in MML or to introduce in the *Text-Proper* in case they are not yet defined in MML.

4.2 Mathematical identifiers and their formal counterparts

Mathematical conjunctions. In the **preamble** created in our example (see Figure 7) one can find the introduced identifiers that play the role of statement conjunctions, e.g. *implies* or *and.* These are usually reserved words and terminal in the Mizar Language, and they are used to form *Formulas*, see the table below.

CML	CGa	Mizar
or	*id* (stat, stat): stat;	*Formula* `or` *Formula*
and		*Formula* `and` *Formula*
implies	where the identifier's name *id*	*Formula* `implies` *Formula*
iff	is choosen by the user, e.g. lines 4–7	*Formula* `iff` *Formula*
not	in Figure 7.	`not` *Formula*

Declarations of these conjunctions are presented in CGa as identifiers that take two arguments of type stat and return the same type stat. This typing information allows us to assume that these identifiers are expressed as Mizar reserved words, which have the same spelling as in CML.

CML	∀, ∃

Possible CGa	`forall(dec('x), stat) : stat;`
	`exists(dec('x), stat) : stat;`

Mizar	*Quantified-Formula* = `for` *Qualified-Variables* [`st` *Formula*] (`holds` *Formula* | *Quantified-Formula*) | `ex` *Qualified-Variables* `st` *Formula* (`holds` *Formula* | *Quantified-Formula*)

Binders like '∀', '∃' or '∑' are indispensable parts of CML. In CGa, the user has to declare them (see figure 7). The CGa is flexible and allows any kind of binder. The Mizar user, if he wants to introduce a new binder, has to do so in an indirect way, although the syntax for Mizar binders was proposed in [23]. Nonetheless, the Mizar language offers the two most essential binders: ∀ and ∃ which are given as *Quantified-Formula* in the Mizar syntax (see the table on the right).

CML	$\_^2$
Possible CGa	`sq(term): term;`

Mizar	`definition let x be complex number;`
	`  func x^2 equals`
	`:: SQUARE_1:def 3`
	`  ...`
	`end;`

CML	_ \\ _
Possible CGa	`subtraction(set,set): set;`

Mizar	`definition let X,Y be set;`
	`  func X \ Y -> set means`
	`:: XBOOLE_0:def 4`
	`  ...`
	`end;`

Functions identifiers in CGa are closely related to *Functor-Definitions* in Mizar. The information that we gain from the CGa encoding is the number of arguments and their weak input and output types. For instance sq is a function which takes one argument and return a value with the same type as argument (see the table on the right). However, it is common knowledge that this function corresponds to the mathematical function *square* (usually written as ^{2} in CML). With this information, we can search MML to find the appropriate Mizar function definition counterpart, which is introduced as *Functor-Definition.* Similarly to such specific CGa identifiers, functions in Mizar have to define the types of their arguments and the type of their results. Mizar *functors* are constructors of (atomic) term, i.e. applied to a (possibly empty) list of terms they create a term.

Predicates. In the CGa encoding some identifiers play the role of predicates which take arguments of type term or set and return stat. In Mizar these identifiers are given in terms of *Predicate-Definition* (see the tables on the right). *Predicates* in Mizar are constructors of atomic formulas, can have several predefined properties (e.g. `symmetry`, `reflexivity` etc.) and define the type of their arguments. We claim that some CGa identifiers (with input types: term or set, and output type stat) correspond more or less to Mizar *Predicates*.

CML	_ ∈ _
Possible CGa	`16 in(term,set): stat;`

Mizar	`notation let a,b be ext-real number;`
	` antonym b < a for a <= b;`
	`end;`

CML	_ < _
Possible CGa	`15 <(term,term): stat;`

Mizar	`notation let a,b be ext-real number;`
	` antonym b < a for a <= b;`
	`end;`
	where the original predicate is defined as follows:
	`definition let x,y be ext-real number;`
	`  pred x <= y means`
	`:: XXREAL_0:def 5`
	`  ....`
	`end;`

Nouns in CML are abstract concepts that classify objects according to their characteristics. The CGa notion of **noun** corresponds to the notion of *Types* in Mizar. *Types* in Mizar are defined using either *Mode-Definitions* or *Structure-Definitions*. For example, the identifier number declared as a **noun** in CGa corresponds to *Mode* in Mizar (see the table on the right). One can also define a **noun** in CGa by giving its features with a **step**. This corresponds to the *Definiens* inside either *Mode-Definition* or *Structure-Definition* which helps to find within MML a proper *Type*. For example, a noun description of the identifier group in CGa (see the example in [13]) could help to identify the *Type* `Group` in Mizar.

CML	*number*
Possible CGa	`27 number: noun;`

Mizar	`notation`
	`  synonym number for set;`
	`end;`
	where set is the primitive type (i.e. the widest type) in Mizar introduced as a *Mode-Definition* in the article `HIDDEN`

Adjectives are another essential part of CML. The CGa notion of adjectives corresponds to the notion of *Attributes* in Mizar. Mizar *Attributes* are defined using *Attributes-Definitions*. For example, the identifier even declared as an **adjective** in CGa corresponds to the *Attribute* in Mizar (see the table on the right). Furthermore, adjectives in CML and CGa are used to modify the characteristics of a noun. Similarly, in Mizar we use *Adjectives* to refine *Types* [2].

CML	*even*
Possible CGa	`28 even: adj;`

Mizar	`definition let i be number;`
	` attr i is even means`
	`:: ABIAN:def 1`
	` ...`
	` end;`

Other identifiers. In CGa we have declared some identifiers to be terms, e.g. 0, 2, 4 (see line 10 of Figure 7), whereas in Mizar they are treated as *Numerals*, which have not been introduced inside MML.

Other identifiers, that have been introduced while computerising our example in CGa, are sets, i.e. N, Q, Z (see line 11 of Figure 7). These represent well known mathematical sets of numbers, i.e. $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{Z}$ respectively. In the Mizar Mathematical Library these sets are introduced as *Functors* (via *Functor-Definitions*) with empty lists of terms, using the symbols: `NAT`, `RAT`, `INT` respectively.

4.3 Transforming the document building steps

As already mentioned (see Section 2.1), in MathLang we present **phrase**, **block** and **local-scoping** in terms of **step** and treat a block as a single **step** composed of a sequence of **statements**. Moreover, the CGa **preamble** gives hints how the identifiers should be translated in Mizar and in which Mizar *Format* (i.e. which Mizar symbols and the place and number of arguments). This information is used to put Mizar symbols inside *Formulas*.

In this section we show using a number of examples, how particular **steps** of our main example encoded in CGa are represented in the Mizar language. Although, we do not give hints how each CGa **step** could be transformed into the Mizar language, we show some ideas through small examples.

Atomic statements in CGa correspond to Mizar's *Formulas*.

CML	... that m^2 is even, but ...
Possible CGa	`44 is(sq(m), even number);`
Mizar	`28 m^2 is even ;`

Blocks in MathLang and in Mizar express a sequence of statements/steps: $\{step_1, \ldots, step_n\}$ (see the example below). In MathLang, if a block is accompanied with a particular mathematical rhetorical role (using the DRa annotation system), it could be transformed into a Mizar specific structure. For instance, if a MathLang **block** is annotated as proof using the DRa, it will still be treated as a sequence of **steps** within the CGa. However, in Mizar, it is transformed to a special *Proof Justification* : `proof` *Reasoning* `end;` (see Section 3).

CML	Possible CGa	Mizar

... $m^2 = 2n^2$. But then $n = 0$ by the lemma. Contradiction shows that $\sqrt{2} \in \mathbb{Q}$.	`89 {`
	`90   =(sq(m),*(2,sq(n)));`
	`91   Lemma |> =(n,0);`
	`92   contradiction;`
	`93 };`
		`89    m^2 = 2*n^2;`
		`90::>          *4`
		`91    n = 0 by Lemma;`
		`92::>        *4`
		`93    hence contradiction;`
		`94::>                   *4`

Contexts. In CGa we use **local-scoping**, i.e. $step_1 \rhd step_2$ which makes the declarations, definitions, and assertions inside $step_1$ available inside $step_2$. This allows to build any kind of **context** for another statement or part of the document. For example, we can use **local-scoping** to introduce a new local predicate. In Mizar, this is introduced as a private predicate (via *Private-Predicate-Definition*), and does not need any kind of context (see the table below).

CML	Define on $\mathbb{N}$ the predicate: $P(m) \iff \exists n.m^2 = 2n^2 \;\&\; m > 0$.

Possible CGa	`38   { m1: N; } |>`
	`39     P(m1):= exists(n1: N, and(=(sq(m1),*(2,sq(n1))),>(m1,0)) );`

Mizar	`21   defpred P[Nat] means ex n being Nat st $1^2 = 2*n^2 & $1 > 0;`

Another possible way of presenting the **local-scoping** usage is to make assumptions into a context to be used in the reasoning block (see the table below). Based on such specified assumptions we can provide further deduction.

<table>
<tr><td>CML</td><td>suppose $m^2 = 2n^2$</td></tr>
<tr><td>Possible CGa</td><td><code>{ =(sq(m),*(2,sq(n))); } |></code>
<code>{</code>
<code>...</code>
<code>};</code></td></tr>
<tr><td>Mizar</td><td><code>assume A0: m^2 = 2*n^2;</code></td></tr>
</table>

We can use *local-scoping* to introduce a new (local or global) variable with a statement expressing some property of this variable. In Mizar this is called *Choice-Statement* (see the table below). In such **context** we introduce a local variable, which is actually bounded with the skeleton of the proof, in the sense that it doesn't change the proof. We have treated this as referring to an available proposition (probably being a consequence of reasoning, for instance definition unfolding) "`ex x being T st P[x]`", where `x` is *Term*, `T` is *Type* and `P` is *Predicate*. From this we can write "`consider a being T such that P[a]`". We do this when we want to reuse the introduced variable in further reasoning steps.

<table>
<tr><td>CML</td><td>So $m = 2k$ and we have</td></tr>
<tr><td>Possible CGa</td><td><code>{ k: N; =(m*(2,k)); } |></code>
<code>{</code>
<code>...</code>
<code>};</code></td></tr>
<tr><td>Mizar</td><td><code>consider k being Nat such that m = 2*k;</code></td></tr>
</table>

The above listed examples of the **local-scoping** construct show only ideas how it could be used when computerizing mathematics. The usage of this construction is not limited to these examples and gives a lot of freedom and flexibility when representing mathematical expressions in MathLang. Therefore it is difficult to propose one transformation hint for the Mizar corresponding structure.

5 Different formalisation paths

The direct path from CML to Mizar – (ⓒ of Figure 1). When transforming a CML text directly to Mizar, a number of facts need to hold:

1. The user needs to be a specialist in the Mizar system (including MML and its search engines: MML Query or the `grep` tool). Furthermore, the user's expertise needs to encompass both mathematics and computer science. In fact, even if a Mizar *Article* resembles a CML text, Mizar is much more closer to declarative programming languages (e.g. Pascal).
2. CML texts can be ambiguous, and the user needs to find and clarify those ambiguities when formalising the text.
3. The Mizar user needs to interpret the text, to find the meaning of each part of the document, and to present it in a formal way.
4. Although the Mizar user has a choice (he may first present parts of the text in the Mizar language, or start from a single statement and look in MML for knowledge that allows the presentation of this statement in the Mizar language, and then to rewrite it; if the statement is accompanied with the proof, maybe the user could fully formalize the proof, and move forward to rewriting the rest of the document within Mizar), there is a common way to translate a text into Mizar which is as follows:
   - Start from a single statement, write it in Mizar.
   - If the statement is accompanied with the proof then fully formalize the proof, if the statement is a definition then define it in Mizar with the proper definiens and prove Mizar specific conditions for the definition.
   - Then, move forward within the CML and Mizar translation and finally, reveal the rest of the reasoning structure of the CML text in Mizar.

The path from CML to Mizar FPS to Mizar – (ⓑ-ⓔ of Figure 1). When transforming a CML text to Mizar FPS and then to Mizar, a number of facts needs to hold:

1. The user needs to be a Mizar specialist as well, and requires similar amount of knowledge as the user who follows the direct path ⓒ to Mizar.
2. The difference (see point 4 from the above list) is that first, the user needs to structure the whole CML text in Mizar FPS. At this stage the user actually does not stop to fully formalize a particular definition or theorem. Although such a choice is possible, Mizar FPS is not meant to do that.
3. After structuring the CML text in Mizar FPS, the user needs to complete the formalisation by filling all the gaps in the reasoning (i.e., filling the holes in sentences that were labelled with the error *4 by the Mizar system.)
4. Since the level of ambiguity of a text is the same as in the direct path, the user needs to carry out the same amount of work as above.

At this stage we could say that although step ⓒ might lead to the same result of steps ⓑ-ⓔ, the work done via ⓑ-ⓔ can be more enjoyable for the Mizar user and a bit easier.

Our proposed path from CML to MathLang to Mizar FPS to Mizar – (ⓐ-ⓓ-ⓔ of Figure 1). When transforming a CML text to MathLang, then to Mizar FPS and then to Mizar, a number of facts needs to hold:

1. The first part of the path (ⓐ) is done by a mathematician, who does not require a lot of MathLang knowledge when annotating the text with CGa grammatical categories or assigning the relationships between different parts of the text and the mathematical or structural rhetorical roles different mathematical entities play. The mathematician simply reveals his understanding of the text. This annotation gives some advantages:
   - It explicits all the identifiers used in the text together with a number of arguments and their weak input and output types.
   - It resolves some ambiguities of the text.
   - It allows the document to be grammatically validated via the automatic CGa checker.
   - It specifies the roles of the important chunks of the text, and expresses dependencies between them.
   - The dependencies of the text parts and the internal information about the roles entities play, allow the automatic generation of a dependency graph which gives the reasoning structure of the text.
   - At this point the work of the mathematician is finished.

2. The Mizar specialist takes the tagged document within MathLang and transforms it into Mizar FPS (part ⓓ of the path). Here, the user needs to be a Mizar specialist and to have the same Mizar knowledge as in the direct path and the one via Mizar FPS.
3. However, at step ⓓ, the user has a structure of the CML text (tagged by the mathematician's understanding of the text and the DRa and CGa **steps**) which helps him to build the skeleton of Mizar FPS. Secondly, all the used identifiers, with the number of their arguments, are stored in one place (the DRa explicit annotation of the **preamble**), and could be reused to find counterparts in Mizar MML and to build parts of the *Environment*. The user also gains from resolved ambiguities of the CML text within MathLang. We believe that this makes the work for the Mizar user a lot easier.
4. At this stage we could say that although step ⓑ might lead to the same result of steps ⓐ-ⓓ, the work done via ⓐ-ⓓ gives an active role to the mathematician in the computerisation and allows the mathematician's computerisation to give a number of useful hints to the Mizar user to create the Mizar FPS skeleton and the Mizar FPS version of the text.

We believe that it is worth following our proposed path. Not every mathematician is interested in fully formalising mathematical texts. Sometimes one may just want a partial formalisation, or even to formalise and verify the correctness of one particular theorem/proof. We believe it is too taxing on mathematicians to ask them to learn the language and specific logic of a proof checker. The advantage of using MathLang as an intermediate step in the proposed path towards Mizar FPS is a guidance for non expert-authors. This guidance mainly helps to extract from the original text an indication of the required background knowledge and an abstraction of the reasoning structure of the text.

6 Conclusions, Related and Future Work

Conclusions. We have presented in this paper our MathLang approach to encoding mathematics on computers. This approach defends the idea that computerisation should come before any formalisation. We briefly showed how MathLang could be used as a useful computerisation tool for the ordinary mathematician who can use it to edit his text (as if he was using LaTeX) and then get a number of automated features and programs that enable him to to create a number of computerised versions of the text. These computerised versions have useful information about the original text, and are then used by the Mizar expert to create first a Mizar FPS version and then a fully formalised Mizar version.

- The main advantage of using MathLang as a mathematical framework is a clear guidance for the non-expert author. This guidance mainly helps to extract from the original text different aspects of mathematical knowledge at different phases of its computerisation.
- The DRa annotation gives useful hints how the skeleton of the Mizar FPS *Article* can be built.

- The CGa **preamble** is treated as a subset of the Mizar *Environment*. CGa identifiers have corresponding counterparts within the Mizar library or could be introduced as Mizar specific *Definitional-Items*. This gives hints about the way we could transform CGa identifiers into their counterparts in Mizar and for which kind of *Symbols* and *Formats* we need to search in MML.

At the time of writing the paper, we used the most recent Mizar system: Mizar system version: 7.8.03 and the MML version: 4.76.959. Due to page constrains we do not attach the complete formalisation of our example in Mizar. However, it is available on-line: `http://www.macs.hw.ac.uk/~retel/pythagoras/`.

Related work. Geleijnse [7] compared WTT and Mizar, presented CML examples in both WTT and Mizar and gave a correspondence between WTT and Mizar identifiers. His main approach was based on comparing these two languages. Our approach is completely different (although of course we are indebted to all the progress in Automath, MV, WTT, FPS and Mizar). Even though inspired by MV and WTT, MathLang's CGa has moved towards an automatically generated structure obtained from the mathematician's editing of the text at the TSa level where the mathematician types his text easier than using LaTeX (in fact, we can claim that this stage is as easy as if the mathematician is writing his text on paper). TSa also gives the mathematician editing features that allows him to assign mathematical, structural and rethorical roles, to entities and chunks of the text and relationships between these chunks. The automatic programs of MathLang create not only the CGa version of the text but also the dependency graph of the text which is then used to create a Mizar FPS *Text-Proper* version of the text. Our path ⓐ-ⓓ-ⓔ of Figure 1 is fully worked out and offers the user much computerised help along the way, and a number of well-formulated hints used in the gradual computerisation and formalisation of the text from the original CML version to a number of computerised versions (CGa, DRa, TSa) followed by a Mizar skeleton followed by Mizar FPS and full Mizar versions. Furthermore, although the de Bruijn path principle (see Section 2.1) has played an influential role in this research, the various levels (or aspects, or stages) of our proposed path are new. Another approach which follows the de Bruijn path principle is discussed by Jojgov and Nederpelt in [10]. However their description of a path from CML to type theory via WTT and type theory with open terms (TTOT) starts from a WTT-text which differs from (but represents) the original CML-text, and then takes the WTT-text into a TTOT version and later into type theory. Our approach starts from the original CML-text (which is the input given by the mathematician into MathLang TSa). The process of moving from the CML-text input into a Mizar FPS skeleton is supported by a number of automated MathLang programs and transformed into the Mizar FPS full version by the Mizar specialist and fully checked by the Mizar system.

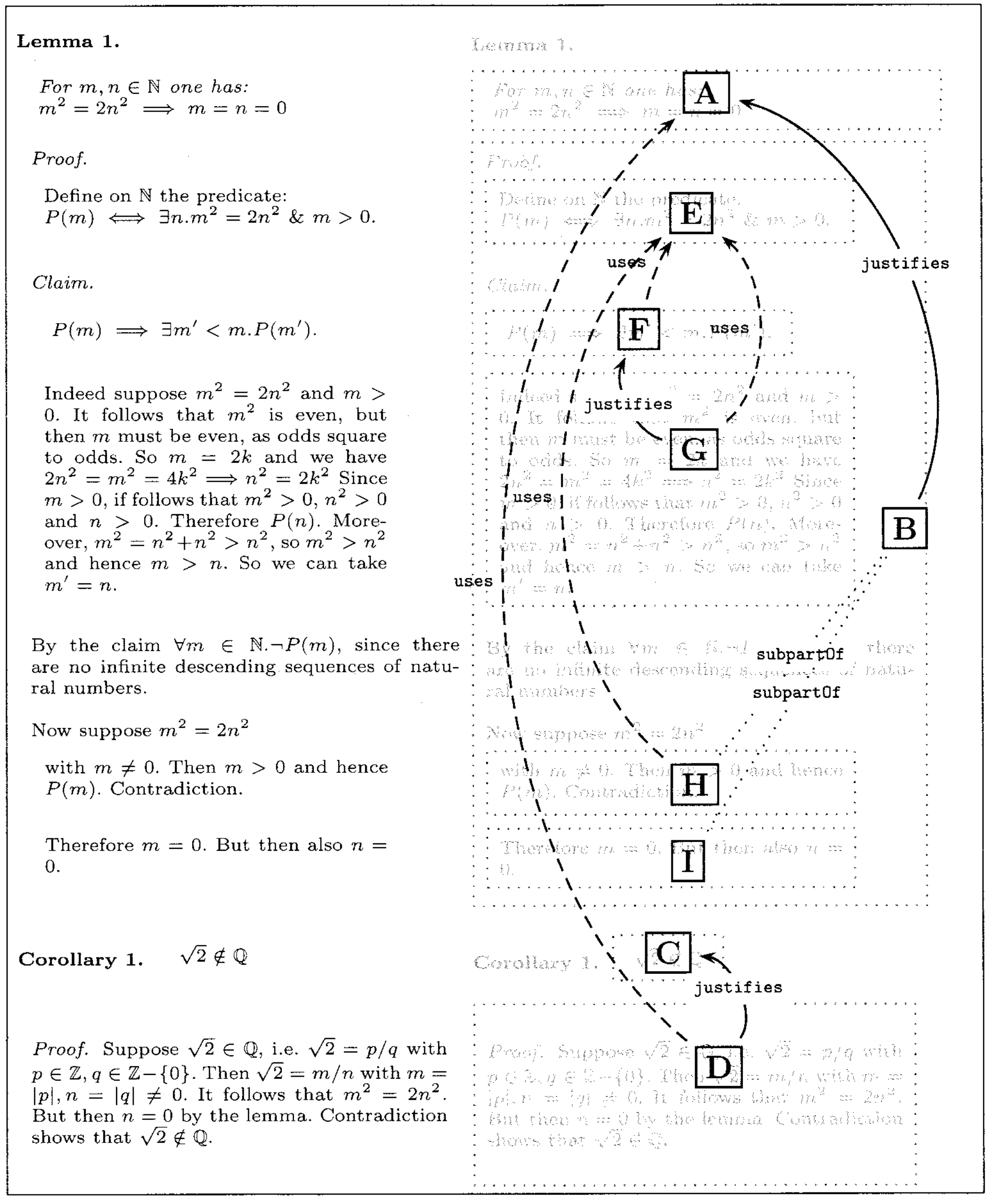
Future Work. MathLang is an ongoing project. As we have seen, the theoretical formalisation and computer implementation of the first three aspects provided a number of useful tools that automatically generate a number of computerised versions of the text each used for a different purpose and each enjoys a different level of formality. As we have also seen, further aspects need to be formalised. For example, it is important to have an aspect that transforms the dependency graph into a Mizar *Text-Proper* skeleton (currently we only provide hints for doing so). It is also important to study in depth the stage where a Mizar FPS version is fully formalised in Mizar. A number of issues need to be investigated:

- How to employ search engines (like grep or semantic mining MML Query) to look up MML in order to find a proper Mizar counterpart for an identifier used in a CML text and explicitly stated in its MathLang CGa version.
- How the MathLang **noun** description construction could be reused to find a counterpart in MML or to define either a *Mode* or a *Structure*.
- How to deal with the freedom that MathLang gives while computerising a common mathematical document.
- We also need to express the hints for transforming a dependency graph into a Mizar FPS *Text-Proper* skeleton, in terms of formal rules which we aim to prove correct and to implement. Moreover, our aim is to start building a computer tool which will support the Mizar specialist with the migration process from a CML+MathLang document to Mizar FPS.

References

1. *Mizar Manuals.* http://mizar.org/project/bibliography.html.
2. G. Bancerek. On the structure of Mizar types. *ENTCS*, 85(7):1–17, 2003.
3. Henk Barendregt. *Informal*, page 10. Volume 3600 of Wiedijk [26], 2006.
4. Library Committee. Mizar built-in notions. *Journal of Formalized Mathematics*, Axiomatics, 1989. http://mizar.org/JFM/Axiomatics/hidden.html.
5. N. G. de Bruijn. Checking mathematics with computer assistance. *Notices of the American Mathematical Society*, 38(1):8–15, 1991. Available at the Automath Archive: http://automath.webhop.net/ Brouwer Institute in Nijmegen and the Formal Methods section of Eindhoven University of Technology.
6. N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In *Workshop on Programming Logic*, 1987.
7. G. Geleijnse. Comparing two user-friendly formal languages for mathematics: Weak type theory and Mizar. Master's thesis, Technische Universiteit Eindhoven, May 2004.
8. Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the Theory of Numbers.* Oxford University Press, 5th edition, April 1980.
9. S. Jaśkowski. On the rules of supposition in formal logic. *Studia Logica*, 1934.
10. G. Jojgov and Rob Nederpelt. A path to faithful formalizations of mathematics. In MKM '04 [16], pages 145–159.
11. Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Flexible encoding of mathematics on the computer. In MKM '04 [16], pages 160–174.
12. Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Mathlang: Experience-driven development of a new mathematical language. In *Proc. [MKMNET] Mathematical Knowledge Management Symposium*, volume 93 of *ENTCS*, pages 138–160, Edinburgh, UK (2003.11.25–29), February 2004. Elsevier Science.
13. Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Toward an object-oriented structure for mathematical text. In Michael Kohlhase, editor, *Mathematical Knowledge Management, 4th Int'l Conf., Proceedings*, volume 3863 of *Lecture Notes in Artificial Intelligence*, pages 217–233. Springer, 2006.
14. Fairouz Kamareddine and Rob Nederpelt. A refinement of de Bruijn's formal language of mathematics. *J. Logic Lang. Inform.*, 13(3):287–340, 2004.
15. R. Matuszewski and P. Rudnicki. Mizar: the first 30 years. Volume 4 of *Mechanized Mathematics and its Applications*, pages 3–24, March 2005. http://mizar.org/people/romat/MatRud2005.pdf.
16. *Mathematical Knowledge Management, 3rd Int'l Conf., Proceedings*, volume 3119 of *Lecture Notes in Computer Science.* Springer, 2004.
17. R. P. Nederpelt and F. Kamareddine. Formalising the natural language of mathematics: A mathematical vernacular. In *4th Int'l Tbilisi Symp. Language, Logic & Computation*, 2001.
18. Rob Nederpelt, J. H. Geuvers, and Roel C. de Vrijer. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, 1994.
19. P. Rudnicki. An overview of the Mizar project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, 1992.
20. P. Rudnicki and A. Trybulec. On equivalents of well-foundedness. An experiment in Mizar. *Journal of Automated Reasoning*, 23(3–4):197–234, 1999.
21. A. Trybulec. The Mizar logic information language. *Studies in Logic, Grammar and Rhetoric*, 1, 1980. Bialystok.
22. Andrzej Trybulec. Tarski Grothendieck set theory. *Journal of Formalized Mathematics*, Axiomatics, 1989. http://mizar.org/JFM/Axiomatics/tarski.html.
23. F. Wiedijk. A proposed syntax for binders in Mizar. http://www.cs.ru.nl/~freek/notes/.
24. F. Wiedijk. Comparing mathematical provers. In *Mathematical Knowledge Management, 2nd Int'l Conf., Proceedings*, volume 2594 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003.
25. F. Wiedijk. Formal proof sketches. In *Proceedings of TYPES'03*, volume 3085 of *LNCS*, pages 378–393. Springer-Verlag, December 2004.
26. F. Wiedijk, editor. *The Seventeen Provers of the World, foreword by Dana S. Scott*, volume 3600 of *LNCS.* Springer Berlin, Heidelberg, 2006.

A Original and DRa-annotated text of our example



Lemma 1.

For $m, n \in \mathbb{N}$ *one has:*
$m^2 = 2n^2 \implies m = n = 0$

Proof.

Define on $\mathbb{N}$ the predicate:
$P(m) \iff \exists n. m^2 = 2n^2 \;\&\; m > 0.$

Claim.

$P(m) \implies \exists m' < m. P(m').$

Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, but then m must be even, as odds square to odds. So $m = 2k$ and we have $2n^2 = m^2 = 4k^2 \implies n^2 = 2k^2$ Since $m > 0$, if follows that $m^2 > 0$, $n^2 > 0$ and $n > 0$. Therefore $P(n)$. Moreover, $m^2 = n^2 + n^2 > n^2$, so $m^2 > n^2$ and hence $m > n$. So we can take $m' = n$.

By the claim $\forall m \in \mathbb{N}. \neg P(m)$, since there are no infinite descending sequences of natural numbers.

Now suppose $m^2 = 2n^2$

with $m \neq 0$. Then $m > 0$ and hence $P(m)$. Contradiction.

Therefore $m = 0$. But then also $n = 0$.

Corollary 1. $\sqrt{2} \notin \mathbb{Q}$

Proof. Suppose $\sqrt{2} \in \mathbb{Q}$, i.e. $\sqrt{2} = p/q$ with $p \in \mathbb{Z}, q \in \mathbb{Z} - \{0\}$. Then $\sqrt{2} = m/n$ with $m = |p|, n = |q| \neq 0$. It follows that $m^2 = 2n^2$. But then $n = 0$ by the lemma. Contradiction shows that $\sqrt{2} \notin \mathbb{Q}$.

Fig. 8. Barendregt's version (without and with dependency graph) of the proof of the Pythagoras' theorem
The original text of Barendregt's version[3] of the proof of the Pythagoras' theorem is reproduced on the left hand side. The right hand side of the figure shows the automatically generated dependency graph for the text where relations between parts of the text are represented by visible arrows and graph nodes have specified (but not visible) mathematical structural roles.

B The Mizar Formal Proof Sketch presentation

Listing 1.2. Encoding of the example from Figure 8 in the Mizar FPS

```
:: This file is verified with the system version:
:: Mizar verifier= 7.8.03,MML = 4.76.959
::
:: Created by Krzysztof Retel {retel@macs.hw.ac.uk}

environ
 vocabularies INT_1, SQUARE_1, MATRIX_2, IRRAT_1, RAT_1, ARYTM_3, ABSVALUE,
   SEQM_3, FINSET_1;
 notations INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1, XCMPLX_0,
   INT_2, SEQM_3, FINSET_1, REAL_1, PEPIN;
 constructors INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1,XCMPLX_0,
    INT_2, SEQM_3, FINSET_1, PEPIN;
 requirements SUBSET, NUMERALS, ARITHM, BOOLE, REAL;
 registrations XREAL_0, REAL_1, NAT_1, INT_1;
begin


Lemma:  for m,n being Nat holds m^2 = 2*n^2 implies m = 0 & n = 0
 proof
   let m,n being Nat;
   defpred P[Nat] means ex n being Nat st $1^2 = 2*n^2 & $1 > 0;
   Claim: for m being Nat holds P[m] implies ex m' being Nat st m' < m & P[m']
   proof
     let m being Nat;
     assume P[m];
     then consider n being Nat such that
     m^2 = 2*n^2 & m > 0;
     m^2 is even ;
::>            *4
     m is even;
::>          *4
     consider k being Nat such that m = 2*k;
::>                                       *4
     2*n^2 = m^2
::>            *4
          .= 4*k^2;
::>              *4
     then n^2 = 2*k^2;
     m > 0 implies m^2 > 0 & n^2 > 0 & n > 0;
::>                                        *4,4,4
     then P[n];
::>          *4,4
     m^2 = n^2 + n^2;
::>                *4
     n^2 + n^2 >  n^2;
::>                 *4
     then m^2 > n^2;
::>               *4
     then m > n;
::>           *4
     take m' = n;
     thus thesis;
::>            *4,4
   end;
   A2: for k being Nat holds not P[k]
   proof
     not ex q being Seq_of_Nat st q is infinite decreasing by Claim;
::>                                                         *4
     hence thesis;
::>             *4
   end;
   assume A0: m^2 = 2*n^2;
   per cases by A0;
```

```
  suppose B1: m <> 0;
    then m > 0;
::>          *4
    then P[m] by B1;
::>            *4
    then contradiction by A2;
    hence thesis;
  end;
  suppose S1: m = 0;
    then n = 0;
::>          *4
    thus thesis by S1;
::>                *4
  end;
end;

Corollary:  sqrt 2 is irrational
 proof
   assume sqrt 2 is rational;
   then ex p,q being Integer st
   q <> 0 & sqrt 2 = p/q;
::>                      *4
   then consider m,n being Integer such that
   A0: sqrt 2 = m/n and m = abs m & n = abs n & n <> 0;
::>                                                    *4
   m^2 = 2*n^2;
::>         *4
   n = 0 by Lemma;
::>       *4
   hence contradiction;
::>                   *4
 end;

::> 4: This inference is not accepted
```

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 10 (23) 2007

The QED Manifesto Revisited

Freek Wiedijk

Institute for Computing and Information Sciences
Radboud University Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Abstract. We present an overview of the current state of formalization of mathematics, and argue what will be needed to make the vision from the QED manifesto come true.

This short and intentionally provocative paper is dedicated to Andrzej Trybulec. When I first wrote about the Mizar system, Andrzej wrote to me:

> *I have looked to pages that you have prepared.* [. . .] *I advertise them, even if they are a bit enemical* :-)

I hope that Andrzej will enjoy this paper, and will not consider it to be too 'enemical'.

1 Why the QED manifesto has not been a success (yet)

One of the most interesting documents about the formalization of mathematics is the *QED manifesto* [2]. This anonymous[1] pamphlet from 1994 paints a future in which most of mathematics will be put in the computer, even the proofs – *especially* the proofs – in such a way that the computer will be able to check it for correctness. The QED manifesto describes the development of a system, the QED system, that mathematicians will adopt for this purpose.

The future that the QED manifesto sketches has not happened. There is no such system as the QED system in regular use by mathematicians today.

There are two main reasons that the future from the QED manifesto currently is not much closer than it was in 1994:

- The most important reason is that *only very few people are working on formalization of mathematics.* If there had been a larger research community interested in that subject, a system like the QED system would have been realized long ago.
  There are various reasons why not many people are working on the vision from the QED manifesto. For one thing formalization has no 'killer application'.

[1] One of the main people behind the QED manifesto was Bob Boyer, but for idealistic reasons the authors of the QED manifesto were not identified in it.

ISBN 978-83-7431-128-1 ISSN 0860–150X

If one writes a formalization of a mathematical result, then one has to work quite hard, and then at the end one has a `tar.gz` file with several computer program-like files in it. However, unlike a computer program, those files have no immediate further use. The fact that they *fully* describe the mathematics has some aesthetic appeal, and it is nice that they make it completely certain that the mathematics is correct, but the unformalized version of the result already was beautiful and understandable, so not much is gained.

There are people who think that *education* is a good application for this kind of technology. The use of proof assistant would teach students what proof is, and it would allow them to work on proofs in a non-threatening, crystal clear environment. I strongly disagree with this point of view. The use of a proof assistant is a way to keep students busy, but it cannot compete with the more traditional way to do mathematics – which is much easier – to actually make them understand something.

An application that *might* become important at some point is the correctness of computer algebra. Computer algebra systems (the most important being, in order of the number of users: Mathematica [22], Maple [15], MuPAD [3] and Reduce [10]) are notorious for occasionally giving wrong answers. This can be very irritating. Therefore, building an integrated system that combines the automation of computer algebra with the correctness of proof assistants might be seen as a real step forward. If that happens, it could make people start using proof assistant technology without them even realizing it.

An aspect where 'QED-like systems' currently are notoriously bad at is *communication.* A formalization is completely useless for communicating the mathematics that is formalized in it. Here is the most to be gained for formal mathematics. In particular the visual side of mathematics is currently completely absent: there are no such things as graphs and diagrams in current proof assistants.[2]

- The other reason that there has not been much progress on the vision from the QED manifesto is that currently *formalized mathematics does not resemble real mathematics* at all. Formal proofs look like computer program source code. For people who *do* like reading program source code[3] that is nice, but most mathematicians, the target audience of the QED manifesto, do not fall in that class.

  To make things worse there are two choices made by large parts of the formalization community that scare mathematicians away even more. Often proof assistants are *constructive* instead of classical, and often they do proof in a *procedural* way instead of in the more recognizable declarative way [7]. If we want to make some progress of getting people actually to use formal mathematics, it has to be close to the way mathematics already is being done for centuries. Improving on tradition is good, but ignoring tradition is stupid. This means that the focus in formal mathematics should be on *classical* & *declarative* systems.[4] From this point of view, among the systems from Section 2 the Mizar and Isabelle/HOL systems – the two systems in that section that are both classical & declarative – are the most interesting.

2 The state of the art in formalization of mathematics

We define the *QED-like systems* to be the five highest ranking systems in an investigation of which theorems already have been formalized from an arbitrary list of 100 well-known theorems.[5] The counts as of March 2007 are:

HOL Light	63
Coq	38
ProofPower	37
Mizar	35
Isabelle/HOL	33
PVS	15
nqthm & ACL2	12
NuPRL & MetaPRL	8

(At that point in time, in all systems together 77 of the 100 theorems have been formalized.) Clearly there is some kind of gap after the fifth system, which is why we put the boundary of being a 'QED-like system' there.

This means that the current QED-like systems are:

- **Mizar** [16, 21]
- HOLs
  - **HOL Light** [8]
  - **Isabelle/HOL** [17, 19]
  - **ProofPower** [14]
- **Coq** [18]

We grouped the three systems from the HOL family together. They have (almost) the same logic and share a very similar architecture.[6]

Projects in the spirit of the QED manifesto have been proposed again and again. For instance, the Automath project from the seventies clearly already was one of

[2] Proof assistants sometimes show diagrams about the structure of the *proof process*, like for instance PVS does, but there never are diagrams about the *mathematics*.

[3] Books that appeal to people like that are for instance Lions' commentary on the Unix kernel source code [13], Donald Knuth's 'TEX: The Program' [12], and John Harrison's forthcoming book on theorem proving [9]. However, proof assistants should appeal to more than just those people.

[4] Georges Gonthier, the author of the most impressive formalization in existence today – the formalization of the Four Color Theorem [6] –, disagrees with this opinion. He uses Coq, which is a constructive & procedural system. He claims that a primarily procedural approach is much more efficient than a declarative proof style, and that for that reason the declarative proof style cannot compete.

[5] The current state of this investigation can be found on the web page ⟨`http://www.cs.ru.nl/~freek/100/`⟩.

[6] In one respect they are not similar: unlike the other HOLs, Isabelle/HOL provides a declarative language called Isar, which is quite close in look and feel to Mizar.

these. Here is a list of projects that still are active and that might be considered to aim at implementing the vision from the QED manifesto. We indicate with each project which QED-like system it uses.

- **MML, the Mizar Mathematical Library** (Mizar)
- **John Harrison's work** on the formalization of mathematics (HOL Light)
- **Georges Gonthier's project** in the Microsoft/INRIA institute (Coq)
- **The Archive of Formal Proofs** (Isabelle/HOL)

The first and the fourth are just the 'collected works' of their respective provers. (The 'contribs' of Coq is another of those, but it does not have a fancy name, so I did not put it in the list.) The second and third are currently just the work of one person. However for the third there are plans to extend it into something that is more, in the joint Microsoft/INRIA institute in Paris.

I do not believe that these four projects are already implementing the vision from the QED manifesto. None of these projects have solved the difficult problem of how to integrate work by multiple people into a nice coherent whole. The first and the fourth project have contributions from multiple people, but they currently fail to build a coherent whole out of it.[7] The second and the third projects have a library that is a nice coherent whole, but these libraries do not yet contain contributions by a significant number of people.

3 A benchmark of four statements

Here are four mathematical statements that most mathematicians will consider to be totally non-problematic[8]:

1. $$\int_0^1 \int_0^1 \sum_{n=0}^{\infty} (xy)^n dx\, dy = \frac{\pi^2}{6}$$
2. *Every field has an algebraic closure.*
3. *Natural transformations are the morphisms of the functor category.*
4. *PFA (the proper forcing axiom) implies*
   $$2^{\aleph_0} = \aleph_2$$

We claim that currently none of the QED-like systems can express all four statements in a good way. Either they do not have good syntax for it (statement 1 in Mizar), or their foundations do not give one enough power to define the notions in these statements in a good way (statement 2 in the HOLs, statement 3 in Mizar, and statement 4 in Coq.) Here is a table that shows which of the QED-like systems have the framework to express which of these four statements:

[7] Try for instance finding a lemma in the MML!

[8] Most of them will probably not know the statement of the proper forcing axiom, but they *will* know the meaning of $\aleph_0$ and $\aleph_2$, and will consider that meaning to be – again – non-problematic.

	Mizar	HOLs	Coq
1. Calculus	−	+	+
2. Abstract algebra	+	−	+
3. Category theory	−	−	+
4. Set theory	+	−	−

Of course one can easily extend the systems with axioms that allow one to write down these statements. However, that really amounts to 'changing the system'. It would mean that both the library and the automation of the system will not be very useful anymore. Classical & extensional reasoning in Coq or abstract algebra in the HOLs by *postulating* the necessary types and axioms will not be pleasant without re-engineering the system.

The first of these statements[9] is from *Proofs from the Book* [1]. There we find the following 'calculation'[10]:

$$\begin{aligned} I = \int_0^1 \int_0^1 \sum_{n\geq 0} (xy)^n dx\, dy &= \sum_{n\geq 0} \int_0^1 \int_0^1 x^n y^n dx\, dy \\ = \sum_{n\geq 0} \Big(\int_0^1 x^n dx \Big) \Big(\int_0^1 y^n dy \Big) &= \sum_{n\geq 0} \frac{1}{n+1} \frac{1}{n+1} \\ = \sum_{n\geq 0} \frac{1}{(n+1)^2} = \sum_{n\geq 1} \frac{1}{n^2} = \zeta(2) \end{aligned}$$

Now the Mizar system has support for 'iterated equalities' like this, but there is no hope of writing this calculation in Mizar in a way that resembles the way it is written in the book. Instead of being able to write

$$\int_0^1 \int_0^1 \sum_{n=0}^{\infty} (xy)^n dx\, dy$$

in Mizar one has to write

$$\begin{array}{lll} \int_0^1 f(y)dy & \textit{where} & f(y) = \int_0^1 g(x,y)dx \\ & \textit{where} & g(x,y) = \sum_{n=0}^{\infty} h(x,y,n) \\ & \textit{where} & h(x,y,n) = (xy)^n \end{array}$$

Since Mizar has no user-definable '*binders*', one has to introduce a *name* for every sub-expression. (Another way of saying this is to say that Mizar is too much of a

[9] Incidentally, computer algebra systems like Mathematica and Maple produce this result without trouble. Now why is there no proof assistant that can do the same? Also, note that the sum $\sum_{n=0}^{\infty}(xy)^n$ diverges when $xy \to 1$, so the inner integral is improper for $y = 1$.

[10] After this calculation it then is shown that $I = \frac{\pi^2}{6}$. The goal in *Proofs from the Book* is not to show the statement that we put in the benchmark (that is just an intermediate result there), but to show that $\zeta(2) = \frac{\pi^2}{6}$. However, for the benchmark we wanted an expression with many nested binders, which is the reason why we selected this intermediate equality.

'first order' system.) This makes doing calculations like the one from *Proofs from the Book* cumbersome in Mizar.

The second of the statements from this section is difficult to state in a nice way in the HOLs. The reason for this is that the HOL type system is too poor. It is not possible in HOL to define a type of *fields* in a uniform way. Therefore, instead of writing the formal equivalent of

> *For all fields ...*

in HOL one has to write

> *For all fields of which the elements are from this given carrier type ...*

The quantification of this carrier type will be implicit. For that reason only universal quantification over these carrier types is possible. Which means that in HOL it is *not* possible to write

> *There exists a carrier type together with a field with that carrier type ...*

Instead one explicitly has to *construct* a carrier type from the already given types. This makes the whole statement clumsy.

Also, one gets a copy of a given field for many different carrier types, and it will need work to navigate these equivalences.

One would hope that with set theory – the foundation of Mizar – one does not have the kind of problem that the HOLs had with the second statement. Unfortunately, this is not true. Mizar has no problems with the statement about fields, but it encounters a similar problem in a different subject.

If one formalizes category theory in set theory, one steps out of the set theoretic universe, and therefore one gets exactly the same kind of problem there that the HOLs have with algebraic structures. Instead of being able to write

> *For all categories ...*

one needs to write

> *For all* small *categories ...*

or maybe

> *For all categories that are from this given universe ...*

In both cases the categories that one talks about will not be proper classes, they will be *small* categories, and therefore the theorems that one proves will not apply to categories like the category of all groups (which is *not* a small category.) This is not acceptable. It means that one cannot talk categorically about the most common kind of algebraic structure.

When one talks about 'natural transformations', one is not only talking about natural transformations where only small categories are involved. Here is a statement that one would like to be able to formalize without making it more complicated than it informally is:

> *Consider the category* **Grp** *of all groups with group homomorphisms as morphisms. The identity functor* $\mathrm{id}_{\mathbf{Grp}} : \mathbf{Grp} \to \mathbf{Grp}$ *is naturally isomorphic to the opposite functor* $\cdot^{\mathrm{op}} : \mathbf{Grp} \to \mathbf{Grp}$.

Here the natural isomorphism that is being talked about is a natural transformation between functors that go from the category of groups to the category of groups. That is, between large categories.

The ideal QED system should allow statements like this without one having to talk (or even to *think*) about universes.

Finally, I have never seen the infinities from set theory – of which $\aleph_0$ and $\aleph_2$ are the first and the third – been formalized in a proof assistant that is based on type theory. For some reason the Coq system has not really been designed to talk about set theoretical notions like the cardinal numbers.

Even worse, also more in general it seems that classical & extensional mathematics is much more difficult in Coq than one would like it to be.[11]

Intermezzo: on constructive proof assistants

A system that implements the QED manifesto should be usable to people who are not aware of the existence of constructive mathematics. The possibility to do constructive mathematics with the system, if at all present[12], should be hidden to people who are not interested in it.

Constructive mathematics seems to be all about being able to point your finger at other people and say 'Oh! They are bad!' It is a very moralistic kind of mathematics. The Coq people point at the NuPRL people and say 'Oh! They are extensional!' ('Type checking of proof terms is not even decidable!') The Agda people point at the Coq people and say 'Oh! They are impredicative!' (Apparently impredicativity is not to be trusted for philosophical reasons.) The other people point at the Agda people and say 'Oh! Their system does not even check termination!' Or maybe someone will say about someone else 'Oh! They use countable choice!' And so on, and so forth.

Instead of trying to prove *as many* true statements as possible, constructive mathematics is about making it *difficult* to prove something. (Of course, *if* you then prove it, the proof contains a bit more information.) Constructivism also takes a strange position with respect to a question like 'does this mathematical statement hold?' The answer often is: 'It depends on how you interpret the statement'; or maybe: 'It depends on what principles you allow yourself.' I guess that this would surprise many classical mathematicians.

Of course for logicians and other philosophers this is all very interesting, but classical mathematicians should not be bothered by these issues. This kind of fine structure of the axiomatics probably does not interest them.

There are various ways in which you can be restrictive:

[11] Of course I would not mind being shown wrong here.

[12] Henk Barendregt talks about a *dial* on a proof assistant that allows people to put it in more or less constructive modes. This intermezzo describes some settings on such a dial.

- First of all you can allow yourself the use of the law of the excluded middle[13] or not. This is the distinction between being *classical* or *constructive.*
- Second, you can be *extensional* or *intensional.* If you are extensional there is *one* equality, that behaves like you would expect equality to behave. If you are not extensional you get into a morass of many different kinds of equality: syntactic identity, convertibility (also called $\beta\delta\iota$-equality), Leibniz equality, John-Major equality[14], setoid equality, etc. etc.
- Third, there is *predicativity.* Predicativity means that you are not allowed to define something by referring to a whole that contains the thing being defined. Although this seems rather clear, logicians do not agree on which systems are predicative and which are not. Some of them claim that the natural formalization of predicativity is a system called *Feferman-Schütte* [11]. Others think that 'inductively generated structures' are also predicative. Here we agree with the second – less strict – opinion, because else no system in our list below would turn out to be predicative.
- Finally there is the *axiom of choice.* Even most classical mathematicians are aware of the fact that one might do mathematics without choice.[15] Now there are various levels of having choice:
  - The weakest version of choice is to take the statement of the axiom of choice

    $$\forall R\,([\forall x\,\exists y\,R(x,y)] \to [\exists f\,\forall x\,R(x,f(x))])$$

    only to give existence of a function f that does *not* have to respect equality. This is called *intensional choice,* and one gets it for free in type theory because of the Curry-Howard-de Bruijn interpretation. (In Coq this is slightly subtle: there this only works for the 'informative' existential quantifier.) In the table below we call this 'weak choice'.
  - Then there is the axiom of choice from set theory. Here one knows that there is a choice function that respects equality, but one cannot necessarily define one explicitly. This axiom is called *extensional choice.* Constructivists think that this axiom is a confused version of their intensional choice axiom.
  - Finally there is the existence of a *Hilbert choice operator* ϵ, an operator that returns a specific element for any given non-empty set.

    In the table below we call both of these last two properties 'strong choice'.[16]

Here is the table that shows which systems satisfy which restrictions:

	Mizar	HOLs	Coq	NuPRL	Agda		
			CIC		ITT	IZF	CZF
Classical	+	+	−	−	−	−	−
Impredicative	+	+	+	−	−	+	−
Extensional	+	+	−	+	−	+	+
Weak choice	+	+	+	+	+	−	−
Strong choice	+	+	−	−	−	−	−

Surprisingly there is no 'bottom' proof assistant that has minuses *everywhere*, one that both does not support extensionality nor any form of the axiom of choice.[17]

4 A criticism of current systems

At one of the meetings of the TYPES project, Georges Gonthier was mentioning a paper from the Mizar community to me in which Mizar was described as a 'state of the art proof assistant'. This seemed to amuse him. As Mizar is certainly in the same ball park as the proof assistant that he uses himself – Coq – I asked him which of the proof assistants in his opinion *are* 'state of the art'. His answer was that 'there is no state of the art proof assistant yet.'

I agree that the QED-like systems that exist today are not good enough to start developing a library as is described in the QED manifesto. I will for each of the three classes of the QED-like systems indicate what in my opinion are their most important weaknesses:

Why Mizar is not acceptable as the QED system yet.

There are three major reasons why Mizar is not yet attractive enough to be taken to be the QED system:

- In the previous section I already addressed the problem that Mizar has no way to define binders. This means that common operations from calculus, like $\sum_{x=a}^{b} f(x)$, $\lim_{x\to c} f(x)$, $\frac{d}{dx} f(x)$, $\int_a^b f(x)\,dx$, etc., can not be properly written down without naming sub-expressions.
- An aesthetic flaw of Mizar that is not very important but that *is* irritating, is that Mizar does not support empty types.[18] For this reason the Mizar library MML has many lemmas that are restricted to non empty sets, while there really is no good mathematical reason for that.

  Also it means that many natural mathematical notions cannot be expressed as a Mizar type. For instance it is not possible to talk about 'the normal form of a term in a given term rewriting system' as a Mizar type (because some terms might not *have* a normal form.)

[13] $p \lor \neg p$.

[14] Also called 'heterogeneous equality'. This is a typed equality in which one allows the terms that are being compared to have different types. Heterogeneous equality generally is a good idea. The equality of the HOLs is homogeneous, but the equality of Mizar is heterogeneous.

[15] If you believe that the union of countably many countable sets is countable, then you do believe in choice.

[16] There are no systems in the table that have extensional choice but do not have a Hilbert choice operator.

[17] The 'logic-enriched type theories' of Peter Aczel [5] are designed to remove the Curry-Howard-de Bruijn interpretation from intensional type theory. However, currently this seems primarily to be used as a logical framework, and not yet as a foundational system for mathematics.

[18] For a type theorist this restriction to non empty types is *very* strange.

- Finally, and I do not know how important this is but I expect it to be very important, Mizar does not support 'user automation'. If you, as a user of the system, know a way to solve a certain class of mathematical problems algorithmically, it is not possible to 'teach' this to the system.
  The only people who can do that kind of automation are the developers of the Mizar system itself, who can add so-called '*requirements*' to the system. This 'closed' architecture is too restrictive.

Why the HOLs are not acceptable as the QED system yet.
There is one important reason why the HOLs are not yet attractive enough to be taken to be the QED system:

- The HOL type system is too poor. As we already argued in the previous section, it is too weak to properly do abstract algebra.
  But it is worse than that. In the HOL type system there are no *dependent types*, nor is there any form of *subtyping*. (Mizar and Coq both have dependent types and some form of subtyping. In Mizar the subtyping is built into the type system. In Coq a similar effect is accomplished through the use of automatic *coercions*.)
  For formal mathematics it is essential to both have dependent types and some form of subtyping.

Why Coq is not acceptable as the QED system yet.
There are two important reasons why Coq is not yet attractive enough to be taken to be the QED system:

- The foundations of Coq are too complicated. They are baroque to say the least. Maybe they are even *beyond* baroque. They might even be called *rococo*.
  There is no paper in which the foundations of Coq are spelled out in full mathematical precision. Bob Solovay told me that this was one of the reasons for him to lose interest in Coq as a system.
  Also, apparently giving a set theoretical interpretation to the foundations of Coq in all its details – to show Coq's consistency relative to some set theory – might not be as easy as it sounds.
  Also, the foundations of Coq are sufficiently complicated that they are tinkered with, and therefore change between versions of the system.
- As already noted in the previous section, Coq is not designed for classical mathematics, which means that doing classical & extensional mathematics in it is not as easy as one would like it to be.
  At the very least one would need to add some axioms for that. For instance axioms that would be needed are:
  - excluded middle
  - the K axiom[19]
  - axioms for extensionality/quotient types
  - choice

  and it is not very clear when enough axioms like that will have been added.

[19] The K axiom states that if two dependent pairs $\langle x_1, y_1\rangle$ and $\langle x_2, y_2\rangle$ are equal , then y_1 has to be equal to y_2. In the Coq logic this is not provable. (It *is* provable that x_1 has to be equal to x_2.) 'Dependent pairs' means that the types of the y_i depend on the x_i.

5 What has to be done?

To realize the vision from the QED manifesto, three things need to happen:

- First of all, *a project should be started that is not based on the expectation that other people will do the work.*
  Currently people often just build a system and then hope that other people will start using it for formalization of mathematics. The QED manifesto is an even more extreme version of this: it just presents a *vision*, and then hopes that other people will *both* make a system and start using it for formalization of mathematics. That approach does not work. If you want something accomplished, you should work on it yourself.
  An important aspect of this project should be that it should involve a *small* number of people,[20] and have a clear focus. Only that way there will be enough coherence to get a good result.
  It seems important to focus on formalization of actual mathematical practice, and *not* also try to 'improve' on the way that people currently do mathematics.
- In this project, *a good system for formalization of mathematics should be developed.*
  As I argued in the previous section, the current crop of systems is not yet good enough to be the 'QED system' in which to build a library of all of mathematics. Ideally, the system should be set up in such a way, that the mathematics that is formalized in it can survive a 'redesign of the foundations'. Suppose that one builds a QED-like library on top of Coq, and then changes one's mind and switches to a HOL-like system. Basically, this will mean that one will have to start from scratch.[21] It seems very arrogant to think that we already know what the best foundations for our formal library should be, and one would therefore prefer not to be 'locked in' to a specific version of the foundations.
  This is one more argument for having a declarative proof style over a procedural proof style. Proofs in a declarative style are much more 'robust' with respect to changes in foundation than proofs in a procedural style. An argument for this is the close similarity between the Mizar and Isar[22] proof languages, despite the wildly different foundations of their underlying systems.
  I do *not* believe that the QED system will consist of many different systems living peacefully together. One of the systems – hopefully the best one – will kill the others. That is how evolution works. It is this system that should be built.[23]

[20] The original Unix system was created by only two people (who shared an office at the time), Ken Thompson and Dennis Ritchie. Starting something very good does not need a large group of people.

[21] Even if one does not mind having to start from scratch like that (taking that as the way that science progresses), it still seems sensible to work with foundations that are not *too* idiosyncratic.

[22] Isar is the proof language of the Isabelle/HOL system [19].

[23] From that point of view, spending time on having different systems be able to communicate their mathematics is lost energy.

- Finally, *a multi-contributor library of formal mathematics should be created that is well-organized.* As I claimed in Section 2, the sociological puzzle has not been solved yet of how to set up a library in such a way that it both admits many contributors, but also stays well-organized. Current attempts that have not yet been very successful in this respect are:
  - *multi-contributor, but not really well-organized:*
    - * MML
    - * Coq contribs
    - * AFP
  - *well-organized, but not really multi-contributor:*
    - * HOL Light library
    - * C-CoRN [4]

  A good first goal for this library will be to formalize 'all of undergraduate mathematics'. This will take more than a hundred man-years [20], which means that it will need the involvement of a non-trivial number of contributors. Also, with something like that it will be clear whether it is well-organized or not.

  The success of Wikipedia suggests that it might be possible to solve the puzzle of having *both* many contributors and still also have a well-organized whole. However, aiming for a 'Wikipedia for formalized mathematics' is for me too much like hoping that someone else will do the work. I will believe that such an approach can be successful when I see it happening.

6 What will happen?

Henk Barendregt always asks people when they expect the future from the QED manifesto to arrive. He tells me that the more experience people have with proof formalization, the more pessimistic they are about this. (The answers seem to range from 'it already is here!' to 'in about fifty years.')

I myself certainly believe that the QED system will come. If we do not blow up the world to a state that mathematics will not matter much anymore, then at some point in the future people will formalize most of their proofs routinely in the computer. And I expect that it will happen earlier than we now expect.

So I *do* believe that in a reasonable time

- a proper system will be created
- a proper basic library for this system is made
- a proper infrastructure is set up for keeping this library well-organized despite it having many contributors

I expect that if the first and third of these items arrive, then the second of these items – the QED library – will flow from that in a natural way.

Acknowledgments. Thanks to Jacques Carette for interesting discussions about the relation between formal mathematics and computer algebra. Thanks to Bas Spitters for explaining to me that there is no consensus on what the word 'predicative' means, by pointing out that there are people who claim that one can predicatively prove Kruskal's theorem. Finally, thanks to Michael Beeson, John Harrison and Bob Solovay for helpful comments on a draft of this paper.

References

1. M. Aigner and G.M. Ziegler. *Proofs from the Book.* Springer-Verlag, second edition, 2001.
2. R. Boyer et al. The QED Manifesto. In A. Bundy, editor, *Automated Deduction - CADE 12*, volume 814 of *LNAI*, pages 238–251. Springer-Verlag, 1994. ⟨http://www.cs.ru.nl/~freek/qed/qed.ps.gz⟩.
3. C. Creutzig and W. Oevel. *MuPAD Tutorial.* Springer-Verlag, second edition, 2004.
4. L. Cruz-Filipe, H. Geuvers, and F. Wiedijk. C-CoRN: the Constructive Coq Repository at Nijmegen. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *Mathematical Knowledge Management, Proceedings of MKM 2004, Białowieza, Poland*, volume 3119 of *LNCS*, pages 88–103. Springer-Verlag, 2004.
5. N. Gambino and P. Aczel. The generalised type-theoretic interpretation of constructive set theory. *Journal of Symbolic Logic*, 71(1):67–103, 2006.
6. G. Gonthier. A computer-checked proof of the Four Colour Theorem. ⟨http://research.microsoft.com/~gonthier/4colproof.pdf⟩, 2006.
7. J.R. Harrison. Proof style. In Eduardo Giménez and Christine Paulin-Möhring, editors, *Types for Proofs and Programs: International Workshop TYPES'96*, volume 1512 of *LNCS*, pages 154–172, Aussois, France, 1996. Springer-Verlag.
8. J.R. Harrison. *The HOL Light manual (1.1)*, 2000. ⟨http://www.cl.cam.ac.uk/users/jrh/hol-light/manual-1.1.ps.gz⟩.
9. J.R. Harrison. *Introduction to Logic and Automated Theorem Proving.* To be published by Cambridge University Press, 2007.
10. A.C. Hearn. *Reduce, User's Manual Version 3.6.* Santa Monica, CA, 1995.
11. K. Schütte. *Proof theory.* Springer-Verlag, 1977.
12. D.E. Knuth. *$\TeX$: The Program.* Addison Wesley, 1986.
13. J. Lions. *Lions' Commentary on UNIX 6th Edition.* Peer-to-Peer Communications, San Jose, CA, 1977.
14. Lemma 1 Ltd. *ProofPower - Description.* Lemma 1 Ltd., 2000. ⟨http://www.lemma-one.com/ProofPower/doc/doc.html⟩.
15. M. Monagan, K. Geddes, K. Heal, G. Labahn, and S. Vorkoetter. *Maple V Programming Guide for Release 5.* Springer-Verlag, Berlin/Heidelberg, 1997.
16. Michał Muzalewski. *An Outline of PC Mizar.* Fondation Philippe le Hodey, Brussels, 1993. ⟨http://www.cs.ru.nl/~freek/mizar/mizarmanual.ps.gz⟩.
17. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS.* Springer, 2002. ⟨http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle2004/doc/tutorial.pdf⟩.
18. The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 2006. ⟨http://pauillac.inria.fr/coq/doc/main.html⟩.
19. M. Wenzel. *The Isabelle/Isar Reference Manual.* TU München, 2002. ⟨http://isabelle.in.tum.de/doc/isar-ref.pdf⟩.
20. F. Wiedijk. Estimating the Cost of a Standard Library for a Mathematical Proof Checker. ⟨http://www.cs.ru.nl/~freek/notes/holl2coq.ps.gz⟩, 2002.
21. F. Wiedijk. Writing a Mizar article in nine easy steps. ⟨http://www.cs.ru.nl/~freek/mizar/mizman.ps.gz⟩, 2007.
22. S. Wolfram. *The Mathematica book.* Cambridge University Press, Cambridge, 1996.

Automated Discovery of Inductive Theorems*

Roy McCasland[1], Alan Bundy[1], and Serge Autexier[2]

[1] School of Informatics
University of Edinburgh
Edinburgh, Scotland, UK
[2] DFKI GmbH & Informatics
Saarland University
Saarbruecken, Germany
rmccasla@inf.ed.ac.uk
A.Bundy@ed.ac.uk
autexier@dfki.de

Abstract. Inductive mathematical theorems have, as a rule, historically been quite difficult to prove – both for mathematics students and for automated theorem provers. That said, there has been considerable progress over the past several years, within the automated reasoning community, towards proving some of these theorems. However, little work has been done thus far towards automatically discovering them. In this paper we present our methods of discovering (as well as proving) inductive theorems, within an automated system. These methods have been tested over the natural numbers, with regards to addition and multiplication, as well as to exponents of group elements.

1 Introduction

There have been considerable advances made over the past fifty years in automated theorem proving, including the proving of inductive[3] theorems. However, regarding automated theorem *discovery* (theorems of any kind), relatively little has been published, other than works such as [2, 5, 8, 10]. Moreover, to our knowledge, there is as yet little or nothing in the literature about automated discovery of inductive theorems.

In this paper we briefly describe some of our methods for discovering (as well as proving) inductive theorems, within an automated system. We have tested these methods over two different representations of the natural numbers, each with respect to the operations of addition and multiplication. In addition, we have applied these methods to basic group theory, regarding (natural number) exponents of group elements. Admittedly, our case studies (see Section 5 and Appendix A)

* This work was supported by EPSRC MathFIT grant GR/S31099, by EPSRC Platform grant GR/S01771, and by EPSRC VR grant EP/E005322

[3] By "inductive", we mean theorems that are proved by mathematical induction – as opposed to inductive reasoning.

ISBN 978-83-7431-128-1 ISSN 0860–150X

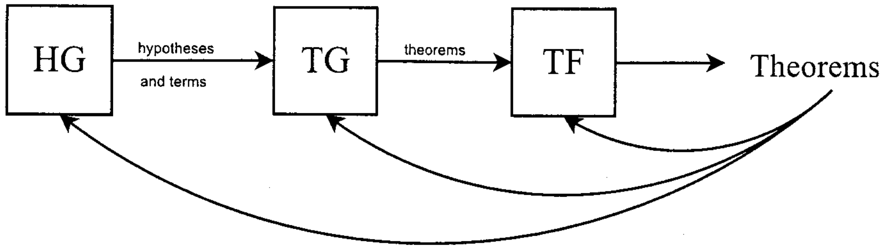


Fig. 1. Component Diagram

have thus far been rather limited; in particular, they have only concerned theorems proven from (what many mathematicians refer to as) the First Principle of Mathematical Induction. Nevertheless, the results are quite promising. In particular, we have obtained the usual associativity, commutativity and distributivity theorems for both of the aforementioned representations of natural numbers, and, within group theory, we have obtained the usual theorems regarding (natural number) exponents.

2 The Main Idea

We should perhaps, at this point, state that we do not claim to have found, nor indeed, do we ever expect to find, a single approach for automatically discovering every inductive theorem that one could want. Instead, we offer an approach for discovering a significant number of the more "routine" inductive theorems – particularly theorems involving equality. For the moment, we only give a rough sketch of our main idea, which actually is really quite simple. The implementation of our ideas, however, is arguably not so simple. This implementation was built onto the three main components already existing in MATHsAiD, as described in [9] (see Fig. 1). These components are the automatic hypothesis generator (HG), the theorem generator (TG), and the theorem filter (TF). In particular, the first two components now have separate, non-inductive and inductive, tracks. In this paper, we of course focus on the latter track.

The key to our approach is to first find an "interesting" proposition $P(n)$ that holds for the "2-case" – bearing in mind, of course, that "2" might well have any one of several representations, depending, in part, on the recursive data-structure. (We will henceforth use TWO to represent the generic "2"). Once $P(\text{TWO})$ is established, the remaining steps obviously are guided by the appropriate induction axiom. That is, determine whether $P(base)$ is also true (where $base$ runs through the list of base elements), and if so, then apply the step case(s) of the induction axiom – e.g., try to prove that $P(k) \implies P(k+1)$. If all this succeeds, and importantly, if $P(n)$ is determined to be non-trivial, then $P(n)$ (for arbitrary n) is added to the database as a Theorem (by which we mean the sort of result that mathematicians would call a theorem, lemma, corollary, etc.). If at any point this process fails, then backtrack, and try again.

We choose to focus initially on the TWO-case, because it seems, on the whole, to be optimal, in terms of predicting non-trivial results, whilst keeping the search space relatively small. As for the 1-case, there are too many statements $P(1)$ that are true, for which the corresponding $P(n)$ is not true in general. One could instead try the 3-case (or any $k > 2$), but this usually results in a much larger search space, whereas any additional rewards, for having gone beyond TWO, are typically minimal.

There are, admittedly, some considerable limitations to this method; in particular, it will likely not find the well-known formula of Gauss, that $1 + \cdots + n = n(n+1)/2$.[4] That said, this method has, in our case studies to date, succeeded in discovering (and proving) all the inductive Theorems that we were expecting/wanting it to find. Note that these Theorems can be found in section 5.

3 Finding a two-Case

Before proceeding further, we remark that, while our work thus far only involves inducting over the natural numbers, we shall endeavour to present our methods in a rather more general context. Thus, for lists, one might think of the TWO-case as a proposition involving a list of length two. One ought, however, to make allowances for the likelihood that some of our ideas will not (easily) be adaptable to certain settings (e.g., binary trees). We intend to continue our work on the automatic discovery of inductive theorems, and hope that others will join us and will also find these ideas a congenial basis to build on.

3.1 Finding an Appropriate Induction Axiom

In our system, called **MATHsAiD**[5], we build theories in layers, in essentially the same manner as is done by mathematicians, and more or less in keeping with the Little Theories approach (see [4]). This of course implies that, whilst investigating one theory, the appropriate induction axiom might reside in a separate theory. Indeed, this is the case in our group theory example.

Nevertheless, for MATHsAiD, this does not present any real difficulty, since the appropriate induction axiom is rather easily identifiable from the functions/constants in the theory being investigated.

We currently limit MATHsAiD to structural induction, which is sufficient for the simple Theorems MATHsAiD currently discovers and proves. Allowing a wider range brings additional search problems and is a subject for further work.

[4] The main reason for not finding this formula is that the term generator in MATHsAiD (see section 3.3) will likely not generate either side of this equation – at least, not without some prompting, perhaps in the form of summation (sigma) notation. (For this, and similar theorems to be discovered, one might hope that an examples-based approach, perhaps akin to that used by Colton [2], would work).

[5] An acronym for **M**echanically **A**scertaining **T**heorems from **H**ypotheses, **A**xioms and **D**efinitions.

3.2 Finding two

In order to find an interesting TWO-case, one ought to first find a useful representation for TWO. Even though this, along with most everything else, is to be done as automatically as possible, the induction axiom provides most of the necessary information. Clearly, TWO is either one or two steps up from the (last) base element (depending on whether "1" or "0" is that element), where "step" is of course determined by the step case(s). For example, in one of our case studies, where the representation of $\mathbb{N}$ (the natural numbers) is given by the Peano Postulates, we have 0 as the base element. In our other chosen representation (as found, for example, in [3]) of the naturals, which we denote by $\mathbb{N}^*$, the base is 1. It must be said that it is not all that crucial to get TWO right – "3" will do, but at a cost. In particular, the discovery of the distributivity Theorem for multiplication over addition would be more difficult to attain, were the THREE case used. (For this reason, we leave it to the reader to determine how best to automatically detect whether "0" or "1" is the base element – but obviously, the name given to this element is quite irrelevant. What *is* relevant, is the set of properties it possesses. Suffice it to say that our means of determining this is tantamount to determining what "addition" is, and whether the base element is an identity for this "addition".) That said, for our purposes, it is important to use the step representation of TWO. That is, in $\mathbb{N}$, TWO takes the form $s(s(0))$, whereas in $\mathbb{N}^*$, it looks like $1 + 1$.

3.3 Generating Terms

As mentioned previously, we are at present primarily interested in Theorems about equalities. Thus, for the remainder of this section, we shall restrict our focus to equational propositions. This is, admittedly, a considerable restriction; nevertheless, it still allows the possibility of significant achievements. Equations do, after all, represent a sizable and important portion of mathematical theorems. Moreover, given that our methods only assume the reflexive, symmetric and transitive properties of equality, one could quite reasonably expect these methods to apply as well to any other equivalence relation, though we have not yet tested them on such.

Moving on, once MATHsAiD has completed the above tasks, it begins generating a sequence of terms (together with appropriate hypotheses, such as $a, b \in \mathbb{N}$, etc. – see Fig. 2), each of which potentially represents the left-hand side of an equation (Theorem). This sequence generation is not random – it is based on an analysis of the given axioms, and an assumption that for certain situations, one looks for certain properties. For example, given a binary operation, one is likely to be interested in the associativity and commutativity properties, and given a pair of these operations, the distributive property – provided that distributivity makes sense.

We remark that in a future version of MATHsAiD, there should be much greater flexibility here, since these "properties of interest" will be dictated more by the axioms provided by the user, rather than by a set programme. Since the term-generator is due to change, we are a bit reluctant to go into great detail about precisely how these terms are at present generated. We will, however, give a rough idea of our plans for the future term-generator, and the reader should understand that the present term-generator is based on a rather restricted version of these plans.

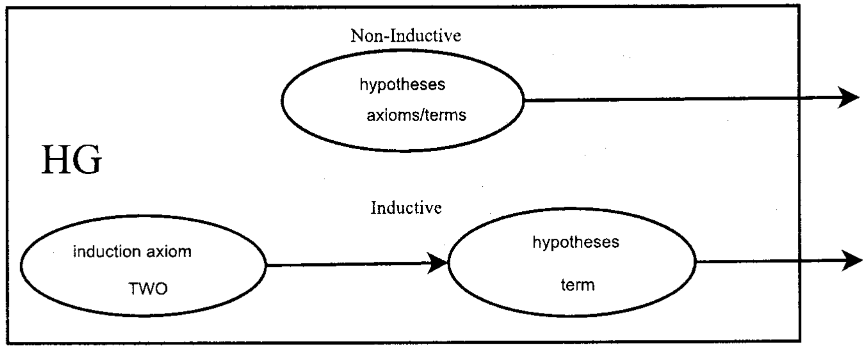


Fig. 2. Hypothesis/Term Generator

The idea is to collect, for each operation/function to be investigated, the equational axioms/definitions for all the properties that are relevant/applicable, either to one of these operations/functions, or to a combination (preferably having no more than two distinct ones) of them – at least, the combinations that make sense. The term-generator should then produce the left (or perhaps the right) hand side of the relevant equation in each of the collected axioms/definitions, as adapted to the appropriate setting. For example, suppose that the operations $+$ and $\cdot$, and the properties of associativity, commutativity, and distributivity (both left-hand and right-hand), have all previously been defined. In this case, the term-generator should produce the terms: $a + (b + c)$, $a \cdot (b \cdot c)$, $a + b$, $a \cdot b$, $a \cdot (b + c)$, $(b + c) \cdot a$, $a + (b \cdot c)$ and $(b \cdot c) + a$, where a, b and c belong to the appropriate set. Moreover, if a constant C has been declared, then the term-generator should also produce terms in which each relevant operation/function has been applied to C (along with however many variables are needed). For example, given the constant 0, then the term-generator should also produce the terms: $a + 0$, $0 + a$, $a \cdot 0$ and $0 \cdot a$.

Note that, in some cases, not all of the terms mentioned above need be investigated. For instance, included in our axioms for $\mathbb{N}$ (see Appendix A.1), is the statement that $a + 0 = a$ (for any $a \in \mathbb{N}$). One would not, therefore, expect to find any Theorem specifically about $a + 0$, and thus, this term can be discarded. On the other hand, axioms which involve a combination of a (binary) operation with a (unary) function, offer an opportunity for further exploration. Note that axiom 8 (respectively, 10) in Appendix A.1, combines $+$ (respectively, $\cdot$) with the successor function. Comparing 8 with axiom 7 (respectively, 9), the successor function effectively replaces 0. This begs the question, what if the variable inside the successor function were replaced with 0? (Note that axiom 13 in Appendix A.3 has a similar combination of the exponent operation and the successor function). Hence, our term-generator of the future will (as the present version already does, to some extent) take into account all the relevant axioms and Theorems in the database, in determining which terms to investigate.

The reader can see, within this paper, most of the terms generated in our studies, as the successful ones (insofar as producing a Theorem is concerned, that is) appear as the left-hand side in the inductive Theorems found in Section 5.[6] Note that each term involves one or more operations and/or other functions described in the axioms, along with an appropriate number (at least one) of "fixed, but arbitrary" variables (to borrow a mathematicians' expression), and occasionally includes one or more constants. One such term that does not appear below, although it was generated by MATHsAiD, is the term $(a * b)^n$, where $a, b \in G$ and $n \in \mathbb{N}$ (see Section 5.3). Since the group in question is not necessarily abelian, then there is essentially nothing that can be proven about this term. Indeed, MATHsAiD tries to find something interesting to prove in this case, but gives up (as it should) after a few seconds.

This last example helps point out that the ordering of this sequence of terms is not, contrary to what one might think, all that critical. We agree that, in order to prove certain inductive theorems, it is important (if not essential) to already have certain lemmas at hand. And indeed, MATHsAiD would not be able to prove, for example, commutativity of multiplication, without having discovered most of the previously found Theorems. Nevertheless, if the given ordering of the sequence of terms does not produce, for example, this commutativity result, then one may simply run MATHsAiD again (and perhaps repeatedly), until all the necessary lemmas have been found.

We should point out that not all of the Theorems found by MATHsAiD are inductive – one ought not expect them to be. The reader should understand that the process for generating these non-inductive Theorems is quite different from what we have been describing here (see [10] and [9] for a further explanation). Suffice it for now to say that the non-inductive-type process, like the inductive one, is automatic.

3.4 Finding an Interesting Case

For each generated term t, one of the variables (call it v) in t is chosen as the induction variable, and is replaced throughout the term (allowing for more than one occurrence of this variable in t) by TWO. MATHsAiD then uses forward chaining, applying whatever axioms and Theorems are at hand, to find another term t', different from t, but such that t' contains TWO and $t(\text{TWO}) = t'(\text{TWO})$. As stated earlier, once such a term t' is found, then MATHsAiD attempts to prove that $t(base) = t'(base)$, and if this succeeds, then it tries to prove the appropriate step case(s), as determined by the relevant induction axiom. (We shall have more to say about the step case, and its proof, in the next section). Even should all this succeed, there is still one more hurdle to clear, before this result is declared to be a Theorem; namely, that $t(v) = t'(v)$ should be a non-trivial result. As indicated earlier, if any stage should fail, then MATHsAiD backtracks, including to the point where v was chosen (see Fig. 3).

[6] Additional information pertaining to MATHsAiD – in particular, the remainder of the generated terms – can be found at http://dream.inf.ed.ac.uk/projects/mathsaid/

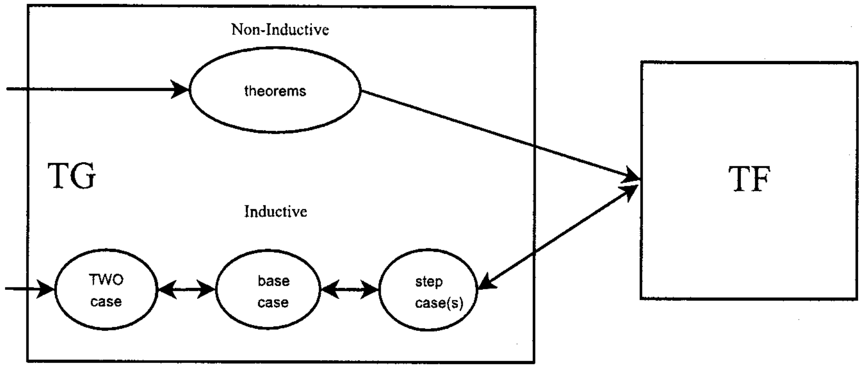


Fig. 3. Theorem Generator

As for precisely what is meant by "non-trivial" (and for that matter, "trivial"), this is an interesting study in itself, and is well beyond the scope of this paper. We tend to equate "trivial" with what we referred to in [10] as "already-known". In more practical terms, for our purposes, "trivial" effectively means, that which MATHsAiD can prove, by using only a specific, limited subset of prescribed procedures and Theorems, along with all the given axioms. The exact makeup of this specific subset can, and perhaps should, vary, depending upon one's objectives. For instance, if one's main goal is to emulate, as much as possible, the human mathematical process, then "trivial" should mean what mathematicians think it means. Alternatively, if one is trying to automatically discover lemmas that might prove useful for an automated theorem prover (ATP), then "trivial" might well mean something quite different. (In future work, we hope to combine MATHsAiD with various ATP's, for this very purpose. Hence, we intend to parameterize our specification of "trivial", in order to make it more adaptable to this, and other situations).

Leaving further discussion of trivialities aside, we return our attention to finding a suitable term t'. As we suggested earlier, it is important that we use the step representation of TWO. One reason for this is that otherwise, MATHsAiD might not be able to make best use of the necessary axioms, required to find t'. Case in point, consider the associativity of addition in $\mathbb{N}^*$. In this situation, TWO $= 1 + 1$, and $t(\text{TWO}) = (a + b) + (1 + 1)$. By repeated application of Axiom 5 (see Appendix A.2), one can obtain (by hand) the following string of equalities:

$$\begin{aligned}(a + b) + (1 + 1) &= ((a + b) + 1) + 1\\ &= (a + (b + 1)) + 1\\ &= a + ((b + 1) + 1)\\ &= a + (b + (1 + 1))\end{aligned}$$

Observe that the last term is the only other term in the sequence (besides t) that contains TWO, and thus is the term we seek. Note that, had we replaced TWO with 2, say, then we would have been stuck. Note also, that the above string of equalities effectively shows the way for proving the step case – another (potential) reason for using the step representation of TWO. This last observation, however, will not necessarily hold in all situations. Hence, we have implemented other means of proving the step case, which we will describe in some detail later.

Not surprisingly, MATHsAiD does not typically arrive at the term t' nearly so easily as might be suggested by the above equations. Indeed, particularly whenever there is an identity involved, the search space can be literally overwhelming. One method we use to limit this search space, is to put a cap C on the size of allowable terms, as measured by the following variant (denoted $m(t)$) on the standard size measurement of terms:

$$m(v) ::= 0;\text{ where } v \text{ is a variable or constant}$$

$$m(f(s_1, \ldots, s_n)) ::= 1 + \sum_{i=1}^{n} m(s_i);\text{ where } f \text{ is a function} \ .$$

(Note that each term t is quantifier-free, because we rely on fixed, but arbitrary variables). This cap is set at

$$C ::= M + m(\text{TWO}) + 2 \ ,$$

where M is initially set at

$$M ::= m(t) \ .$$

The extra cushion of $m(\text{TWO}) + 2$ is allowed, in order to accommodate such results as distributivity, wherein the sought-after right-hand side might be larger than the given left-hand side, and moreover, the induction variable (and hence, TWO) might appear twice.

As for the search itself, we have found that a two-stage approach works quite well. In the first stage, we limit the reach of the forward chaining process, in much the same way as discussed previously, regarding "trivialities". In particular, we collect all "promising" terms s, reachable (subject to our imposed limitations) from t, such that $t(\text{TWO}) = s$ and $m(s) \leq C$. If this does not produce the desired term t', then we add to our collection whatever "promising" terms can be reached from each of the terms s already in our collection, and so on. Along the way, each of these "promising" terms is sent to the second stage, which uses a directed (but still limited) search to see if it is "close to" a term s' that contains TWO. That is, can a term s' be (quickly) found that contains TWO and such that $s'(\text{TWO}) = s$. Moreover, for any promising term s, if $m(s) < M$, then M is reset to $M ::= m(s)$, further restricting the search space.

In the above associativity example, this second stage finds that the term $(a + (b + 1)) + 1$ is indeed "close to" the desired term $a + (b + (1 + 1))$, and in effect bypasses the last but one term.

The process continues in this vein, until either a suitable term t' is found, or no more promising terms s can be found. Note that the transitivity of equals insures the soundness of this approach.

4 Proving the Step Case

Once we reach the stage of trying to prove the step case, we could hand the proof off to one of the ATP's that are designed to handle induction proofs. However, our design philosophy, together with a strong desire to keep everything in-house, required that we built our own equation-prover. But more importantly, there is still the question of whether the resulting general equation is non-trivial, and ATP's, quite understandably, were simply not designed to answer this question. We admit that our own equation-prover could stand some improvement; nevertheless, it has, thus far, succeeded in proving everything we wanted it to.

We built the induction-proof portion of our equation-prover on the (fairly obvious) assumption that, in order to prove the "$k + 1$-case", one is almost certainly going to make use of the knowledge provided in the "k-case". Note that, in our situation, both cases are, in fact, equations. Hence, both the "k-equation" and the "$k+1$-equation" are passed to the equation-prover. Taking the left-hand side of the "$k+1$-equation", which we will denote by $lhs(k+1)$, the prover first uses a process much like the second stage process described above, in order to see if this term is "close to" a term t that contains $lhs(k)$. If so, then $rhs(k)$ (i.e., the right-hand side of the "k-equation") is substituted in the appropriate place in t (provided that substitution is allowable), and an attempt is made to prove that the resulting term equals $rhs(k + 1)$. Should all this succeed, then, of course, the proof is complete.

If each step in this process succeeds quickly, then well and good. If not, then we use a piece-wise search, which is certainly reminiscent of, but somewhat different from, rippling (see, for example, [1]). Here, the target term (e.g., $lhs(k)$) is broken down into subterms (to begin with, just one level down, in terms of the tree structure). A search is then made for a term that is equal to the given term (e.g., $lhs(k + 1)$), but that contains the first subterm of the target term. If this succeeds (including, of course, the case that the given term already contains this subterm), then a subsequent search is made, regarding the next subterm, with the proviso that the previous subterm not be lost. If any search fails, then the relevant subterm is broken down into its subterms, and we proceed as before. These searches continue, until the desired target term is found, or until all searches fail.

Should this still fail, then the prover begins again, but starting from the respective right-hand sides, moving to the left. Should this last attempt fail, then the proof fails.

As an example of the piece-wise search, consider the (left-hand) distributivity Theorem, again in $\mathbb{N}^*$. The equation-prover receives the (assumed) "k-equation"

$$a \cdot (b + k) = (a \cdot b) + (a \cdot k) \ ,$$

along with the (to-be-proved) "$k + 1$-equation"

$$a \cdot (b + (k + 1)) = (a \cdot b) + (a \cdot (k + 1)) \ .$$

In order to make use of the given equation, the prover needs to find a term t such that $t = a \cdot (b + (k + 1))$ and t contains $a \cdot (b + k)$. It could, of course, stumble around, until it (hopefully) eventually succeeded. However, using the piece-wise search, it successively searches for terms t_1 and t_2 such that:

$$t = t_1 \text{ and } t_1 \text{ contains } a$$
$$t_1 = t_2 \text{ and } t_2 \text{ contains } a \text{ and } b + k \ .$$

Clearly t already contains a, but does not contain $b + k$. Thus, a search is made for t_2, and in fairly short order, it finds that $t_2 = (a \cdot (b + k)) + a$ satisfies the requirements. Now $rhs(k)$ can be applied, which gives

$$(a \cdot (b + k)) + a = ((a \cdot b) + (a \cdot k)) + a \ .$$

At this point, the piece-wise search can again be used, with $rhs(k + 1)$ as the target term. The sought-after subterms are $a \cdot b$, which $((a \cdot b) + (a \cdot k)) + a$ has, and $a \cdot (k + 1)$, which it does not have. Once again, the search rather quickly finds the following sequence of equations:

$$\begin{aligned} ((a \cdot b) + (a \cdot k)) + a &= (a \cdot b) + ((a \cdot k) + a) \\ &= (a \cdot b) + (a \cdot (k + 1)) \ , \end{aligned}$$

and the proof is complete.

5 Theorems

We include the Theorems found by MATHsAiD, for both of the aforementioned representations of the natural numbers, and for group theory. The inductive Theorems are designated by *. We remark that MATHsAiD is programmed to determine, for each binary operation, whether the operation is closed. (Of course, if there is an axiom that provides this information, then nothing else need be done.) Hence, in addition to the following results, MATHsAiD also discovered that, for $a, b \in N$ and for $g \in G$, then $a + b, a \cdot b \in \mathbb{N}$ and $g^n \in G$. These results were indeed added to the database, but were not recorded as "Theorems".

The code for MATHsAiD is written in two languages. All of what one might consider the "mathematics", is done in Prolog; everything else is handled in Java. For these experiments, we ran MATHsAiD on a Pentium 4 CPU, 2.40GHz machine, with 512MB RAM. The time taken, rounded to the nearest second, to generate each list of Theorems (including the non-inductive, as well as the inductive results) is provided at the end of each list.

For the convenience of the reader, the data here have been rewritten in standard mathematical notation.

5.1 Theorems in the Natural Numbers

The axioms, from which these Theorems (and the Theorems in the remainder of this section) are derived, can be found in Appendix A.

Assume throughout that $a, b, c \in \mathbb{N}$.

Theorems:	
1.	$a + s(0) = s(a)$
2.*	$(a + b) + c = a + (b + c)$
3.*	$0 + a = a$
4.*	$s(b) + a = s(b + a)$
5.*	$a + b = b + a$
6.	$a \cdot s(0) = a$
7.*	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
8.*	$(b + c) \cdot a = (b \cdot a) + (c \cdot a)$
9.*	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$
10.*	$0 \cdot a = 0$
11.*	$s(0) \cdot a = a$
12.*	$a \cdot b = b \cdot a$

The above list of Theorems was generated in 84 seconds.

5.2 Theorems in the Positive Naturals

Assume throughout that $a, b, c \in \mathbb{N}^*$.

Theorems:	
1.*	$(a + b) + c = a + (b + c)$
2.*	$a + 1 = 1 + a$
3.*	$a + b = b + a$
4.*	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
5.*	$(b + c) \cdot a = (b \cdot a) + (c \cdot a)$
6.*	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$
7.*	$1 \cdot a = a$
8.*	$a \cdot b = b \cdot a$

The above list of Theorems was generated in 14 seconds. Note that the time required in this setting is considerably less than in the preceding setting. This is due primarily to the absence of an additive identity, which in turn leads to significantly smaller search spaces.

5.3 Theorems in Group Theory

Assume throughout that $a, b, c \in G$ and that $m, n \in \mathbb{N}$. Note that $inv(a)$ denotes the inverse of a, and that for this study, we do not consider negative exponents.

Theorems:

1. $inv(inv(a)) = a$
2. $b * a = a \implies b = e$
3. $a * b = a \implies b = e$
4. $b * a = e \implies inv(a) = b$
5. $a * b = e \implies inv(a) = b$
6. $inv(a) * inv(b) = inv(b * a)$
7. $c * a = c * b \implies a = b$
8. $a * c = b * c \implies a = b$
9. $a^{s(0)} = a$

10.* $e^n = e$

11.* $a^m * a^n = a^{m+n}$

12.* $(a^m)^n = a^{m \cdot n}$

The above list of Theorems was generated in 90 seconds. Of that time, only 19 seconds were spent on the "exponent" Theorems (i.e., beginning with Theorem 9, which includes all the inductive results). This includes the discovery/proof that $a^n \in G$, along with the (laudable, but futile) attempt to find something interesting to prove about $(a * b)^n$.

5.4 Significance of These Theorems

The reader might not be fully aware of the difficulties in automatically proving some of the above Theorems. Particularly, the commutativity Theorem for multiplication in $\mathbb{N}$ has been notoriously hard for conventional induction provers to prove. One reason, several intermediate lemmas are normally required to be on hand, before trying to prove the commutativity result. Even with these lemmas provided, some provers still cannot succeed in the proof, without human intervention.

These difficulties exist, even when the provers are told what to prove! The fact that MATHsAiD was not told what to prove, but had to both discover and prove each Theorem, – including the commutativity Theorem – seems to be a rather significant achievement.

6 Related Work

There has been some work towards automated discovery of non-inductive Theorems by groups other than ourselves, notably [2], [5] and [8]. However, in [2] and [8], mere conjectures were formed,[7] whereas in [5], only a very few Theorems were ever found. In any event, none of these systems were able to discover inductive Theorems.

As for automatically discovering inductive theorems, [6], [7] and [11] have had some success. However, in all these cases, the discovery process was initiated only after an attempt had been made to prove a particular theorem, and the search was geared solely towards discovering intermediate lemmas. Note that in our process, no initial theorems or conjectures are required, or even used.

[7] In [2], the Theorems were actually proven by separate automated theorem-provers – not by the discovery system itself.

7 Conclusions and Future Work

We have described our methods, and to some extent, our implementation of these methods, for automatically discovering and proving inductive theorems. We have tested our methods, albeit in somewhat limited fashion, and have included the results of our tests. While our work is still ongoing, the results to date are quite promising.

Besides the previous references to future work, we are quite keen to try out these methods in other theories – particularly for lists, as well as over the integers (including integer exponents of group elements). We anticipate the need for at least some (perhaps only minor) adjustments to our implementation, if not to our methods.

> *We are delighted to have this opportunity to celebrate the achievements of Andrzej Trybulec. He has pioneered the application of automated reasoning to real mathematical practice, building a huge corpus of formally-proved, challenging, mathematical theorems. From the beginning, he recognised that to engage working mathematicians in this project, the Mizar system should fit in with their working practices, rather than impose alien ones. In our work, we have followed Andrzej's lead by trying to emulate human theorem-discovery processes.*

A Axioms

We include the axioms provided to MATHsAiD, for two versions of the natural numbers and for group theory. For the convenience of the reader, the data here have been rewritten in standard mathematical notation.

A.1 The Natural Numbers

The following are the axioms/definitions provided to MATHsAiD, for the natural numbers (based on the Peano Postulates).

Axioms: Given that $a, b \in \mathbb{N}$;

1. $\mathbb{N}$ is a set
2. $0 \in \mathbb{N}$
3. $s(a) \in \mathbb{N}$
4. $s(a) = s(b) \iff a = b$
5. $s(a) \neq 0$
6. If $S \subseteq \mathbb{N}$ such that:
    (i) $0 \in S$; and
    (ii) $k \in S \Rightarrow s(k) \in S$;
   then $S = \mathbb{N}$
7. $a + 0 = a$
8. $a + s(b) = s(a + b)$
9. $a \cdot 0 = 0$
10. $a \cdot s(b) = (a \cdot b) + a$

A.2 The Positive Natural Numbers

The following are the axioms/definitions provided to MATHsAiD, for the positive natural numbers ($\mathbb{N}^*$) (as found, for example, in [3]).

Axioms: Given that $a, b \in \mathbb{N}^*$;

1. $\mathbb{N}^*$ is a set
2. $1 \in \mathbb{N}^*$
3. $a + b \in \mathbb{N}^*$
4. $a \cdot b \in \mathbb{N}^*$
5. $(a + b) + 1 = a + (b + 1)$
6. $a \cdot 1 = a$
7. $a \cdot (b + 1) = (a \cdot b) + a$
8. If $S \subseteq \mathbb{N}^*$ such that:
   (i) $1 \in S$; and
   (ii) $k \in S \Rightarrow k + 1 \in S$;
   then $S = \mathbb{N}^*$

A.3 Group Theory

The following are the axioms/definitions provided to MATHsAiD, for group theory. Note that $inv(a)$ denotes the inverse of a, and that for this study, we do not consider negative exponents.

Axioms: Given that $a, b, c \in G$ and $n \in \mathbb{N}$;

1. G is a set
2. $a * b \in G$
3. $a = b \implies c * a = c * b$
4. $a = b \implies a * c = b * c$
5. $(a * b) * c = a * (b * c)$
6. $e \in G$
7. $a * e = a$
8. $e * a = a$
9. $inv(a) \in G$
10. $inv(a) * a = e$
11. $a * inv(a) = e$
12. $a^0 = e$
13. $a^{s(n)} = a^n * a$

References

1. Bundy, A., Basin, D., Hutter, D., Ireland, A.: Rippling: Meta-level Guidance for Mathematical Reasoning. Volume 56 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2005).
2. Colton, S.: Automated Theory Formation in Pure Mathematics. Springer-Verlag (2002).
3. Dodge, C.W.: Numbers & Mathematics. 2nd edn. Prindle, Weber & Schmidt, Inc. (1975).
4. Farmer, W.M., Guttman, J.D., Thayer, F.J.: Little theories. In Kapur, D., ed.: CADE11. (1992) 567–581.
5. Gao, Y.: Automated generation of interesting theorems. Master's thesis, University of Miami (2004).
6. Ireland, A. and Bundy, A.: Productive use of failure in inductive proof. JAR **16**(1–2) (1996) 79–111. Also available from Edinburgh as DAI Research Paper No 716.
7. Kapur, D., Subramaniam, M.: Lemma Discovery in Automating Induction. Intl. Conf. on Automated Deduction, CADE-13. (1996) 538–552.
8. Lenat, D.B.: AM: An artificial intelligence approach to discovery in mathematics as heuristic search. In: Knowledge-based systems in artificial intelligence, McGraw Hill (1982). Also available from Stanford as TechReport AIM 286.
9. McCasland, R.L., Bundy, A.: Mathsaid: A mathematical theorem discovery tool. In: Proceedings of SYNASC 2006 (to appear).
10. McCasland, R.L., Bundy, A., Smith, P.F.: Ascertaining mathematical theorems. Electronic Notes in Theoretical Computer Science **151**(1) (2006) 21–38.
11. Walsh, T.: A divergence critic for inductive proof. Journal of Artificial Intelligence Research 4, (1996) 209–235.

Formalizing Basic Complex Analysis

Dedicated to Andrzej Trybulec on the occasion of his 65th birthday

John Harrison

Intel Corporation, JF1-13
2111 NE 25th Avenue, Hillsboro OR 97124, USA
johnh@ichips.intel.com

Abstract. We describe the formalization of some of the basics of complex analysis in the HOL Light theorem prover. Besides being a beautiful area of mathematics, this has many potential applications, e.g. in analytic number theory. We have endeavoured to set up the kind of general analytic machinery that would make such applications feasible.

0 Mizar and me: some personal recollections

The first time I saw mention of the Mizar project was in a message from Bob Boyer to the QED mailing list in August 1993:[1]

> Indeed, there have been a good number of QED-like efforts spread over at least the last 27 years, both large scale and small. (I hear rumors that the Polish MIZAR effort may be the largest so far.)

At the time, even though Mizar had a large and thriving user community all over the world, I hadn't heard of it at all, so my curiosity was piqued. The following summer, the second QED Workshop was in Warsaw, and I had the opportunity to meet Andrzej Trybulec in person for the first time. This was a memorable experience in many ways, and immediately after the workshop a few participants travelled to Białystok where we had the opportunity to try out Mizar for ourselves with Andrzej's help.

I was impressed that Andrzej was willing to devote so much time to helping out some complete beginners. But I was equally impressed how little his help was needed! After a few hours, I was able to prove in Mizar the formula for the roots of a quadratic equation. To someone with no experience of proof assistants, that might seem an unspectacular accomplishment, but Mizar seemed to be dramatically easier to use than the other systems I'd tried, particularly HOL [3], with which I was most familiar.

Over the next few months, back in Cambridge (where I was supposed to be finishing my PhD) and in Turku/Åbo in Finland (where I was doing a postdoc under

[1] http://icml.stanford.edu/~uribe/mail/qed.messages/105.html

ISBN 978-83-7431-128-1 ISSN 0860–150X

the EU Human Capital and Mobility scheme), I often thought about why Mizar's proof language seemed much easier to use. I became convinced that Andrzej's design had many key features that we should emulate in other systems, and at the TPHOLs conference in 1996, I presented a paper [4] where I described a simple 'clone' of some features of the Mizar proof language built on top of my own HOL Light prover. During the discussion after my talk, Mike Gordon (the original inventor of HOL) coined the word *declarative* to describe the Mizar approach to proof, in contrast to the *procedural* approach of HOL and most other systems. Those terms, based on a natural analogy with declarative and procedural programming languages, seem to be very appropriate and have stuck.

Before I had even finished coding my 'Mizar Mode for HOL', Don Syme had attended a half-baked talk I gave on the subject in Cambridge and been inspired to write his own declarative prover. Many of the ideas about proof style that Don and I discussed found shape in a paper I gave at the TYPES meeting in Aussois in 1996 [5]. And next year, declarative proof had become a hot topic, with three papers by different people at the same conference [13, 14, 17].

Ironically, though I acted as a channel for Mizar's ideas, I have been little occupied with declarative over the subsequent decade. The most impressive line of work derived from Mizar is Markus Wenzel's "Isar" mode for the Isabelle prover [12]. This has even become Isabelle's default mode in place of the traditional ML tactic scripts, and the structured proof language seems to have contributed to a much wider usage of the Isabelle system. I have mostly continued to do proofs in the good old procedural way in HOL Light. This is partly inertia, but also partly a recognition that this approach has its advantages too. (Just as, for example, point-and-click interfaces and command lines have their place.) Perhaps the ultimate dream is a smooth combination of both procedural and declarative approaches [16].

As he approaches his 65th birthday, I hope Andrzej gains the deserved satisfaction from seeing his ideas spread not just through the user community of Mizar itself, but through the world of proof assistants generally. So far, the intellectual influence has mainly concerned the logical structure of the proof language, but I would not be surprised if other ideas from Mizar, such as its treatment of algebraic structures, begin to propagate into other systems. I would have liked to talk about some of these questions, but I lack the competence to do so. Instead, I draw inspiration from another of Andrzej's notable characteristics: while many people talk and dream about the formalization of mathematics, he and his colleagues really do it. In the same spirit, I want to describe some work I have done on the formalization of complex analysis using my HOL Light prover.

1 Topological and analytical basics

Instead of defining a special type of complex numbers, we use the type $\mathbb{R}^2$ directly, though we define `complex` as an accepted abbreviation for this type. In this way we instantly inherit all the topological and analytic apparatus described in [8], without needing explicit isomorphisms from one type to the other. Nevertheless, we add various special things for the complex numbers, e.g. the mappings `Re` and `Im` for the real and imaginary components of a complex number, the conjugation mapping `cnj`, and multiplication and inversion of complex numbers. We have tried to keep notation compatible with an earlier theory [7]. Readers unfamiliar with a simple type system such as HOL will need to get used to seeing various type injections such as `&` : $\mathbb{N} \to \mathbb{R}$ and `Cx` : $\mathbb{R} \to \mathbb{C}$. Also, although the types $\mathbb{R}^2$ and $\mathbb{C}$ are synonymous, the types $\mathbb{R}$ and $\mathbb{R}^1$ are not, so we use bijections `lift` : $\mathbb{R} \to \mathbb{R}^1$ and `drop` : $\mathbb{R}^1 \to \mathbb{R}$.

More interestingly, we also define the key concept of complex differentiability. The general notion of differentiability for a function $\mathbb{R}^M \to \mathbb{R}^N$ is that there is a local linear approximation, which is also a function $\mathbb{R}^M \to \mathbb{R}^N$, e.g. for a simple limit at a point:

```
|- (f has_derivative f') (at x) ⇔
        linear f' ∧
        ((λy. inv (norm (y - x)) % (f y - (f x + f' (y - x)))) --> vec 0)
        (at x)
```

As discussed in [6], we formalize many such concepts in this relational style. For example, when proving things about limits we generally focus on assertions of the form $x_n \to a$ rather than the informal equivalent $\lim x_n = a$. In the HOL formalization these are not equivalent: $x_n \to a$ implies $\lim x_n = a$ but not conversely. The trouble is that because the limit function lim has some specific type, say $(\mathbb{N} \to \mathbb{R}^N) \to \mathbb{R}^N$, there is no way of encoding whether or not the limit exists; whether it does or not there will be *some* object $\lim x_n$ of type $\mathbb{R}^N$ and the equation $\lim x_n = a$ does not furnish any information about whether it's a "real" limit or just some other arbitrary value. (This problem is less severe in untyped frameworks or those with more flexible types like Mizar, since one can use subtyping to indicate definedness, e.g. setting lim to return something not in $\mathbb{R}^N$ when there is no limit.) Therefore, we quite consistently formalize limiting concepts in the relational fashion, as with complex differentiability and path integrability below.

Compared with general differentiability in $\mathbb{R}^N$, complex differentiability makes the more stringent assumption that the linear mapping just amounts to multiplication by a complex number:

```
|- (f has_complex_derivative f') net ⇔
        (f has_derivative (λx. f' * x)) net
```

It is easy to prove this equivalent to another natural characterization:

```
|- (f has_complex_derivative f') (at a) ⇔
      ((λx. (f(x) - f(a)) / (x - a)) --> f') (at a)
```

Equally, one can derive the Cauchy-Riemann characterization (though we have had no use for it so far):

```
|- f complex_differentiable (at z) ⇔
        f differentiable (at z) ∧
        jacobian f (at z)$1$1 = jacobian f (at z)$2$2 ∧
        jacobian f (at z)$1$2 = --(jacobian f (at z)$2$1)
```

where:

```
|- f complex_differentiable net ⇔
     ∃f'. (f has_complex_derivative f') net
```

We also define complex differentiability throughout a set. We use a restricted notion of limit considering points inside the set only. This is more in line with the general approach to limits within subspaces in topology, and in any case the distinction disappears when, as is usually done, we consider open sets. However, it is important to keep this in mind, since often analyticity on a set is taken to imply analyticity in an open set around each point. (Our motivation in taking the chosen course is that in the past we sometimes found it tedious handling such things as 1-sided limits at the endpoints of intervals with a separate argument.)

```
|- f analytic_on s ⇔
   ∀x. x ∈ s ⇒ ∃f'. (f has_complex_derivative f') (at x within s)
```

The usual theorems about composing limits and derivatives extend easily from general differentiability to this special case. Properties of complex multiplication are derived from general properties of bilinear mappings, though the complex inverse uses a more explicit and tedious argument. (Could this be avoided?)

We also define complex versions of the usual transcendental functions (starting with the Taylor expansion for the exponential function) and derive real versions from them. These are not much used in the first part of this development, but we do need the complex exponential function and its periodicity when we later come to talk about winding numbers. Note that HOL Light already has a theory of real analysis [6], but our long-term goal is to subsume it under this more general analytical theory in $\mathbb{R}^N$ and $\mathbb{C}$.

2 Paths

Cauchy's theorem and related results depend on integration of a complex function over a path (sometimes known as a line, road, curve or contour). A path is considered to be a mapping $\gamma : [0,1] \to \mathbb{C}$ out of some canonical real interval, and the path integral is then defined as

$$\int_\gamma f = \int_0^1 f(\gamma(t))\gamma'(t)\ dt$$

or in our HOL relational formulation:

```
|- (f has_path_integral i) g ⇔
   ((λx. f(g x) * vector_derivative g (at x)) has_integral i)
   (interval[vec 0,vec 1])
```

For most purposes, we want to forget the details of the parametrization using the arbitrary interval $[0,1]$, so we define various natural abbreviations:

```
|- pathstart g = g(vec 0)

|- pathfinish g = g vec 1)

|- closed_path g ⇔ pathstart g = pathfinish g

|- path_image g = IMAGE g (interval[vec 0,vec 1])
```

It's often convenient to stick together two new paths to make a new path. To retain the canonical parametrization we allocate the two intervals $[0,1/2]$ and $[1/2,1]$ to scaled versions of the the two components; formally:

```
|- g1 ++ g2 = (λx. if drop x <= &1 / &2
                   then g1(&2 % x)
                   else g2(&2 % x - vec 1))
```

where `vec` is a direct injection $\mathbb{N} \to \mathbb{R}^1$ and the infix operator '`%`' is scalar-vector multiplication. (We could have written '`Cx(&2) * x`' but the present definition is valid for any $\mathbb{R}^N$ as well as $\mathbb{C}$.) Similarly we often want to consider a path in reverse:

```
|- reversepath g = (λx. g(vec 1 - x))
```

For a path integral as defined above to exist, some restrictions on the functions allowed as paths are needed. The functions f we are concerned with are normally well-behaved, certainly continuous and usually analytic, so it is the nature of γ that is critical. The usual assumption in complex analysis texts is that a path γ should be piecewise continuously differentiable, which ensures that the path integral exists at least for any continuous f, since all piecewise continuous functions are integrable. However, our notion of validity is weaker, just piecewise differentiability:

```
|- valid_path f ⇔ f piecewise_differentiable_on interval[vec 0,vec 1]
```

where piecewise differentiability is continuity plus differentiability except on a finite set:

```
|- f piecewise_differentiable_on i ⇔
      f continuous_on i ∧
      (∃s. FINITE s ∧ (∀x. x ∈ i DIFF s ⇒ f differentiable at x))
```

This choice is influenced by the fact that our underlying theory of integration is the Kurzweil-Henstock theory [9, 10]. In contrast to the Riemann or Lebesgue theory (but in common with a simple notion based on antiderivatives), this can integrate all derivatives. Using the Riemann or Lebesgue theory we would be unable even to integrate a constant function along an exotic non-rectifiable path like $\gamma(t) = t + t^2 \sin(1/t^3)$, which is differentiable everywhere but with unbounded variation in a neighbourhood of 0, and so we could prove little of interest about general paths. With our integration theory, it is only when we need to consider bounds on integrals that we may need to impose stronger restrictions.

Of course, it is easy to establish various basic results about our combinators for paths, such as the following. Once these have been proved (often quite laborious, though intellectual trivial), one does not often have to return to the basic definition of path integral.

```
|- reversepath(reversepath g) = g

|- pathstart(reversepath g) = pathfinish g

|- valid_path(reversepath g) ⇔ valid_path g

|- pathfinish g1 = pathstart g2
   ⇒ (valid_path (g1 ++ g2) ⇔
       valid_path g1 ∧ valid_path g2)

|- (f has_path_integral i1) g1 ∧
   (f has_path_integral i2) g2 ∧
   valid_path g1 ∧ valid_path g2
   ⇒ (f has_path_integral (i1 + i2)) (g1 ++ g2)
```

Two particularly common paths that we use in what follows are a straight-line path from a to b:

```
|- linepath(a,b) = λx. (&1 - drop x) % a + drop x % b
```

and a circular path, traversed counterclockwise, with centre z and radius r (here `ii` denotes the imaginary unit i):

```
|- circlepath(z,r) = λx. z + Cx(r) * cexp(Cx(&2) * Cx pi * ii * Cx(drop x))
```

where `cexp` is the complex exponential function.

3 Cauchy's theorem

The cornerstone of the usual approach to complex analysis is the Cauchy-Goursat integral theorem, although there are ingenious approaches that avoid integration [15]. Our first step is a straightforward corollary combining the 1-dimensional Fundamental Theorem of Calculus and the general chain rule for composition of derivatives:

```
|- (∀x. x ∈ s ⇒ (f has_complex_derivative f'(x)) (at x within s)) ∧
   valid_path g ∧ (path_image g) SUBSET s
   ⇒ (f' has_path_integral (f(pathfinish g) - f(pathstart g))) g
```

This shows that if a function has a primitive (usual terminology for a complex antiderivative) within a set, the integral round any closed curve in that set is zero. Note that as when defining analyticity, we carefully consider limits within the set s, so this is a little sharper than the most common variant where the set s is assumed open.

```
|- (∀x. x ∈ s ⇒ (f has_complex_derivative f'(x)) (at x within s)) ∧
   valid_path g ∧ (path_image g) SUBSET s ∧
   pathfinish g = pathstart g
   ⇒ (f' has_path_integral Cx(&0)) g
```

In order to make progress, we need to establish the existence of line integrals for a more general class of functions. However, we only need to consider integrals along straight-line segments, and in this case the integrability of continuous functions follows from an analogous result for the reals:

```
|- f continuous_on segment(a,b) ⇒ f path_integrable_on (linepath(a,b))
```

Our goal is now to prove the Cauchy-Goursat theorem, which states that if a function is analytic in a set with suitable topological properties, its line integral round any closed path in that set is zero. For the time being, we just aim to prove this for a triangular path, on and inside which the function is analytic. The key argument is a quadrisection step, showing that if the integral were $\geq eK^2$ where K is a bound on the sides, then one of the four triangles obtained by joining all pairs of midpoints would have the same property, and the corresponding K would be halved. Note that we use the notion of convex hull as an easy way of talking about the inside and outside of the triangle.

```
|- f continuous_on (convex hull a,b,c) ∧
   dist (a,b) <= K ∧
   dist (b,c) <= K ∧
   dist (c,a) <= K ∧
   norm(path_integral(linepath(a,b)) f +
        path_integral(linepath(b,c)) f +
        path_integral(linepath(c,a)) f) >= e * K pow 2
   ⇒ ∃a' b' c'. a' ∈ convex hull a,b,c ∧
                b' ∈ convex hull a,b,c ∧
                c' ∈ convex hull a,b,c ∧
                dist(a',b') <= K / &2 ∧
                dist(b',c') <= K / &2 ∧
                dist(c',a') <= K / &2 ∧
                norm(path_integral(linepath(a',b')) f +
                     path_integral(linepath(b',c')) f +
                     path_integral(linepath(c',a')) f)
                 >= e * (K / &2) pow 2
```

This means that if the path integral round a triangle is nonzero, we could generate a decreasing nest of triangles, which by completeness must all contain a common point. (It is easy to see that there is exactly one.) But then from complex differentiability at that point, we obtain the following upper bound on the integral:

```
|- x ∈ s ∧ f continuous_on s ∧
   f complex_differentiable (at x within s) ∧
   &0 < e
   ⇒ ∃k. &0 < k ∧
         ∀a b c. dist(a,b) <= k ∧ dist(b,c) <= k ∧ dist(c,a) <= k ∧
                 x ∈ convex hull a,b,c ∧ convex hull a,b,c SUBSET s
                 ⇒ norm(path_integral(linepath(a,b)) f +
                        path_integral(linepath(b,c)) f +
                        path_integral(linepath(c,a)) f)
                   <= e * (dist(a,b) + dist(b,c) + dist(c,a)) pow 2
```

and for sufficiently small triangles, this is a contradiction, so we obtain our first version of Cauchy's theorem. We use the notion of integration on a 'chain', which is just a list of paths. We could express this directly using the path-combining operator '++', but this proof was designed before we introduced that combinator. It is a trivial corollary that the integral is zero round a closed path (a singleton chain).

```
|- f analytic_on (convex hull a,b,c)
   ⇒ (f has_chain_integral Cx(&0))
      [linepath (a,b); linepath(b,c); linepath(c,a)]
```

Before proceeding to more general regions, we generalize this in two directions. First, by a simple but slightly laborious limiting argument, we can see that the assumption of analyticity can be weakened to continuity on the boundary:

```
|- f continuous_on (convex hull a,b,c) ∧
   f analytic_on interior (convex hull a,b,c)
   ⇒ (f has_chain_integral Cx(&0))
       [linepath (a,b); linepath(b,c); linepath(c,a)]
```

and then we can permit a finite number of exceptional points inside, using a process of trisection at those points:

```
|- f continuous_on (convex hull a,b,c) ∧
   FINITE s ∧
   (∀x. x ∈ interior(convex hull a,b,c) DIFF s
        ⇒ f complex_differentiable (at x))
   ⇒ (f has_chain_integral Cx(&0))
       [linepath (a,b); linepath(b,c); linepath(c,a)]
```

We will eventually be able to show that in such a case the function is differentiable at these interior points anyway, but the weak hypothesis is useful to develop the machinery that will allow us to reach this result.

Now, given any convex set with f analytic on the interior except at a finite number of points, we can fix a point a of the set and define a function $g(x)$ by the line integral of f along linepath(a, x). Cauchy's theorem for a triangle is exactly what we need to show that this is a primitive for f, i.e. $g'(x) = f(x)$. For we have

$$g(x+h) - g(x) = \int_{\text{linepath}(a,x+h)} f - \int_{\text{linepath}(a,x)} f = \int_{\text{linepath}(x,x+h)} f$$

and letting $h \to 0$ we see $(g(x+h) - g(x))/h \to f(x)$. So we can conclude that if a function is analytic at almost all interior points of a convex set, it has a primitive there:

```
|- convex s ∧ FINITE k ∧ f continuous_on s ∧
   (∀x. x ∈ interior(s) DIFF k ⇒ f complex_differentiable at x)
   ⇒ ∃g. ∀x. x ∈ s
             ⇒ (g has_complex_derivative f(x)) (at x within s)
```

Combining this with the first result for line integrals of a function with a primitive, we obtain our main version of Cauchy's theorem:

```
|- convex s ∧ FINITE k ∧ f continuous_on s ∧
   (∀x. x ∈ interior(s) DIFF k ⇒ f complex_differentiable at x) ∧
   valid_path g ∧ (path_image g) SUBSET s ∧
   pathfinish g = pathstart g
   ⇒ (f has_path_integral Cx(&0)) g
```

We can prove this under the slightly weaker assumption that the set is starlike, i.e. there is some point a such that for any x in the set, the line segment from a to x lies entirely in the set. However in that case we do seem to need the traditional hypothesis of openness. In any case, though this is far from the most topologically general version of Cauchy's theorem, it is enough to deduce many interesting and non-trivial results.

4 Winding numbers

So far, we haven't described results guaranteeing the existence of path integrals for a general path unless the function has a primitive throughout the path. (Of course, we can get results for the special case of paths that are continuously differentiable.) However, by chopping up the path sufficiently, it's easy to generalize this to require the existence only of a local primitive in a small neighbourhood of each point:

```
|- (∀x. x ∈ s
        ⇒ ∃d h. &0 < d ∧
                 ∀y. norm(y - x) < d
                     ⇒ (h has_complex_derivative f(y)) (at y within s)) ∧
   valid_path g ∧ (path_image g) SUBSET s
   ⇒ f path_integrable_on g
```

Thanks to our 'local' version of Cauchy's theorem, this implies that path integrals of analytic functions always exist:

```
|- open s ∧ f analytic_on s ∧ valid_path g ∧ path_image g SUBSET s
   ⇒ f path_integrable_on g
```

In particular, we can define the winding number by the usual integral $W(\gamma, z) = \frac{1}{2\pi i} \int_\gamma dw/(w-z)$, and deduce that this is welldefined provided z is not on the path γ:

```
|- valid_path g ∧ ¬(z ∈ path_image g)
   ⇒ ((λw. Cx(&1) / (w - z)) has_path_integral
       (Cx(&2) * Cx(pi) * ii * winding_number(g,z))) g
```

It follows immediately from our simple Cauchy theorem that if a path γ is inside a convex set, the winding number is zero about all points outside the set, because in that case $1/(w - z)$ is analytic throughout:

```
|- valid_path g ∧ convex s ∧ pathfinish g = pathstart g ∧
   ¬(z ∈ s) ∧ path_image g SUBSET s
   ⇒ winding_number(g,z) = Cx(&0)
```

and so in particular this means that the winding number must be zero for sufficiently large z:

```
|- valid_path g ∧ pathfinish g = pathstart g
   ⇒ ∃B. ∀z. B <= norm(z) ⇒ winding_number(g,z) = Cx(&0)
```

It is usual to consider the notion of winding number only in the case of closed paths. However, the definition does not preclude applying it to general paths, and we find this potentially useful. For example, we might want to compute winding numbers for composite paths using additivity:

```
|- valid_path g1 ∧ valid_path g2 ∧
   ¬(z ∈ path_image g1) ∧ ¬(z ∈ path_image g2)
   ⇒ winding_number(g1 ++ g2,z) =
      winding_number(g1,z) + winding_number(g2,z)
```

Of course, it is important to know that the winding number for a closed path is always an integer. The proof we use, apparently due to Ahlfors, is based on the observation that

$$e^{-\int_a^b \gamma'(t)/(\gamma(t)-z)\,dt}(\gamma(b) - z) = \gamma(a) - z$$

or in our formalization:

```
|- g piecewise_differentiable_on interval[a,b] ∧
   drop a <= drop b ∧ (∀x. x ∈ interval[a,b] ⇒ ¬(g x = z))
   ⇒  (λx. vector_derivative g (at x) / (g(x) - z))
       integrable_on interval[a,b] ∧
       cexp(--(integral (interval[a,b])
                        (λx. vector_derivative g (at x) / (g(x) - z)))) *
       (g(b) - z) = g(a) - z
```

The proof is fairly straightforward, although it needs some care because of the generality of our paths. At all points where γ is differentiable, the derivative of the expression $e^{-\int_a^y \gamma'(t)/(\gamma(t)-z)\,dt}(\gamma(y) - z)$ with respect to y is zero, using the Fundamental Theorem of Calculus. And at *all* points this is continuous as a function of y (the Kurzweil-Henstock integral in one dimension is always a continuous function of its upper limit whenever the integral exists). Therefore it is constant on the interval $[a, b]$ and so the values at a and b are equal; since $e^{-\int_a^a \gamma'(t)/(\gamma(t)-z)\,dt}(\gamma(a) - z) = e^0(\gamma(a) - z) = \gamma(a) - z$ the result follows. From this lemma, we can deduce as usual that if the path is closed, i.e. $\gamma(b) = \gamma(a)$, we must have $e^{-\int_a^b \gamma'(t)/(\gamma(t)-z)\,dt} = 1$, so $\int_a^b \gamma'(t)/(\gamma(t) - z)\ dt$ is a multiple of $2\pi i$ and thus the winding number is an integer. In fact, this equivalence works equally well in either direction: the winding number is an integer *if and only if* the path is closed:

```
|- valid_path g ∧ ¬(z ∈ path_image g)
   ⇒ (complex_integer(winding_number(g,z)) ⇔
       pathfinish g = pathstart g)
```

In what follows, we will often want to consider integrals round simple circular paths. It follows immediately from results above that the winding number is zero for points outside the circle. To show that it is 1 for points inside, the most satisfactory approach might be to prove it is clearly positive by considering the integrand, and must be at most 1 because the path is a *simple* curve. However, we haven't actually proved that the winding number cannot exceed 1 in magnitude for a simple closed curve, so we use a more direct approach. Explicit evaluation of the integral easily yields the winding number for a circle about the centre:

```
|- ¬(r = &0) ⇒ winding_number(circlepath(z,r),z) = Cx(&1)
```

Also, the winding number is a continuous function of the point, assuming it is off the curve. This could be proved for general paths, except that since we need to bound the integral, we may finally need to make a stronger assumption about our paths than piecewise differentiability. We just proved it for circles:

```
|- &0 < r ∧ ¬(norm(w - z) = r)
   ⇒ (λw. winding_number(circlepath(z,r),w)) continuous (at w)
```

Since the winding number is an integer, it must therefore be constant on connected components:

```
|- connected s ∧ (s INTER path_image (circlepath(z,r)) = ) ∧
   &0 < r ∧ w1 ∈ s ∧ w2 ∈ s
   ⇒ winding_number(circlepath(z,r),w1) =
      winding_number(circlepath(z,r),w2)
```

and so we deduce as we wanted:

```
|- norm(w - z) < r ⇒ winding_number(circlepath(z,r),w) = Cx(&1)
```

5 Cauchy's integral theorem

We deduce Cauchy's integral formula for a convex set by applying Cauchy's theorem to the function

$$g(w) = \begin{cases} \frac{f(w)-f(z)}{w-z} & \text{if } w \neq z \\ f'(z) & \text{if } w = z \end{cases}$$

Note that here it is useful to be able to assume only that g is *continuous* at z, while being analytic everywhere else. On the other hand, this result can eventually be used to deduce that in such a case the function is in fact differentiable at z anyway. (We have not actually proved such results about removable singularities yet, but this would be one natural next step.)

```
|- convex s ∧ FINITE k ∧ f continuous_on s ∧
   (∀x. x ∈ interior(s) DIFF k ⇒ f complex_differentiable at x) ∧
   z ∈ interior(s) DIFF k ∧
   valid_path g ∧ (path_image g) SUBSET (s DELETE z) ∧
   pathfinish g = pathstart g
   ⇒ ((λw. f(w) / (w - z)) has_path_integral
       (Cx(&2) * Cx(pi) * ii * winding_number(g,z) * f(z))) g
```

For a circular path and an analytic function, we have the simple instance:

```
|- convex s ∧ f analytic_on s ∧ z ∈ interior s ∧ &0 < r ∧
   path_image (circlepath(z,r)) SUBSET s
   ⇒ ((λw. f(w) / (w - z)) has_path_integral
       (Cx(&2) * Cx(pi) * ii * f(z))) (circlepath(z,r))
```

Even this is already enough to deduce a weak version of Liouville's theorem, that an entire function that tends to zero for all sufficiently large values of the argument must be zero everywhere. This is already enough to obtain an easy proof of the Fundamental Theorem of Algebra, by applying it to the inverse of a polynomial without a zero, though this is something which has already been proved by the more direct 'minimum modulus' proof [7].

```
|- f analytic_on (:complex) ∧ (f --> Cx(&0)) at_infinity
   ⇒ ∀z. f(z) = Cx(&0)
```

However, most further results depend on formulas for derivatives in terms of path integrals. The basic lemma that allows us to pass from an expression for values of a function g as an integral to a similar expression for its derivative g' is as follows. Note that we have two a priori different functions g and f, one defined in terms of a path integral of the other; this extra generality makes this quite a flexible lemma. (Here pow is the power function $\mathbb{C} \to \mathbb{N} \to \mathbb{C}$.)

```
|- ¬(k = 0) ∧
   (f continuous_on path_image(circlepath(z,r))) ∧
   (∀w. w ∈ ball(z,r)
        ⇒ ((λu. f(u) / (u - w) pow k) has_path_integral g w)
           (circlepath(z,r)))
   ⇒ ∀w. w ∈ ball(z,r)
          ⇒ (λu. f(u) / (u - w) pow (k + 1)) path_integrable_on
                 (circlepath(z,r)) ∧
             (g has_complex_derivative
              (Cx(&k) * path_integral(circlepath(z,r))
                     (λu. f(u) / (u - w) pow (k + 1))))
             (at w)
```

In particular, setting $k = 1$ and f and g to be the same, we obtain a formula for the first derivative:

```
|- f continuous_on cball(z,r) ∧
   f analytic_on ball(z,r) ∧
   w ∈ ball(z,r)
   ⇒ (λu. f(u) / (u - w) pow 2) path_integrable_on circlepath(z,r) ∧
     (f has_complex_derivative
         (Cx(&1) / (Cx(&2) * Cx(pi) * ii) *
          path_integral(circlepath(z,r)) (λu. f(u) / (u - w) pow 2)))
     (at w)
```

6 Consequences of the integral formula

Our versions of Cauchy's theorem and Cauchy's integral formula are restricted to convex sets. However, even this special case allows us to prove some significant consequences. First of all, the derivative formula immediately yields the existence of higher derivatives on an open set, since we can integrate round a sufficiently small circle:

```
|- open s ∧ f analytic_on s
   ⇒ (complex_derivative f) analytic_on s
```

Also, we easily obtain the full version of Liouville's theorem, because the integral $\int_\gamma f(w)/(w-z)^2\ dw$ must tend to zero as the size of the circle increases if f is bounded.

```
|- f analytic_on (:complex) ∧ bounded (IMAGE f (:complex))
   ⇒ ∃c. ∀z. f(z) = c
```

A slightly more involved application is Weierstrass's convergence theorem: if a sequence of analytic functions (f_n) with derivatives (f'_n) tends uniformly to a limit f, then the limit is also analytic and $f'_n \to f'$. Note that an analogous result fails for differentiability of a function $\mathbb{R}^M \to \mathbb{R}^N$ even for $M = N = 1$, where the series of derivatives may not even converge. (For example, by the Weierstrass approximation theorem, every continuous function, not necessarily differentiable anywhere, is a uniform limit of analytic functions, viz. polynomials.) In our context, however, it is easy to see that the integral formulas defining higher derivatives converge, and we can conclude:

```
|- open s ∧
   (∀n x. x ∈ s ⇒ ((f n) has_complex_derivative f' n x) (at x)) ∧
   (∀x. x ∈ s
        ⇒ ∃d. &0 < d ∧ cball(x,d) SUBSET s ∧
                 ∀e. &0 < e
                     ⇒ eventually (λn. ∀y. y ∈ cball(x,d)
                                              ⇒ norm(f n y - g y) < e)
                                  sequentially)
   ⇒ ∃g'. ∀x. x ∈ s ⇒ (g has_complex_derivative g'(x)) (at x) ∧
                       ((λn. f' n x) --> g'(x)) sequentially
```

This allows us to prove the following, which is a convenient 'one-stop-shop' for defining an analytic function in terms of a series (including but not limited to power series). If the sequence of functions (f_n) are all analytic on a set s with derivatives f'_n, and $|f_n(x)|$ is bounded on s by a summable real series h, then we can conclude that the series (f_n) converges to a function g and (f'_n) converges to its derivative, throughout f. (The set k restricts the terms of the infinite sum, e.g. to allow us to sum just from 1, only for even n or only for prime p.)

```
|- open s ∧
   (∀n x. n ∈ k ∧ x ∈ s ⇒ (f n has_complex_derivative f' n x) (at x)) ∧
   (∃l. (lift o h sums l) k) ∧
   (∃N. ∀n x. N <= n ∧ n ∈ k ∧ x ∈ s ⇒ norm(f n x) <= h n)
        ⇒ ∃g g'. ∀x. x ∈ s
                     ⇒ ((λn. f n x) sums g x) k ∧
                        ((λn. f' n x) sums g' x) k ∧
                        (g has_complex_derivative g' x) (at x)
```

7 Conclusions and future work

Generally, the sequence of results here have been developed without much difficulty, more or less following the usual informal plan. However, they rest on some other theories that were occasionally quite hard work to formalize. For example, even an *affine* change-of-variables theorem for the Kurzweil-Henstock integral (admittedly for integrals on $\mathbb{R}^N$ not just $\mathbb{R}^1$) was quite hard work to formalize, despite the straightforward underlying intuition that all the concepts like interval, partition, gauge and so on scale in an obvious way. We also found it very convenient to have a good library of results about convex sets to rely on, but it was occasionally remarkable how much work it was to prove complete trivialities. One amusing (or depressing) example is the following, which actually took many dozens of lines to prove!

```
|- ¬(a ∈ interior(convex hull a,b,c))
```

We have found it very convenient that HOL Light's programmability lets us set up quite powerful automated proof procedures for special tasks, e.g. a COMPLEX_FIELD rule for simple algebraic manipulations like the following, which would also be tedious by hand:

```
|- ¬(x = u) ∧ ¬(x = w)
   ⇒ Cx(&1) / (x - u) - Cx(&1) / (x - w) =
       (u - w) / ((x - u) * (x - w))
```

The most annoying gap in HOL Light's automation concerns 'triangle law' reasoning, which crops up all the time. We ended up proving a lot of trivial lemmas like the following:

```
|- abs(norm(w - z) - r) = d ∧
   norm(u - w) < d / &2 ∧
   norm(x - z) = r
   ⇒ d / &2 <= norm(x - u)
```

Thanks to a vector-space quantifier elimination procedure due to Bob Solovay [8], we can in principle prove such results automatically. In practice, using this method is not a complete answer because even linear problems like the above give rise to nonlinear problems over the reals, which though decidable are often quite slow. Recently we have observed that almost all the triangle law reasoning we use relies only on the basic norm properties and nothing specific to the Euclidean norm $|x| = \sqrt{x \cdot x}$. Using this fact, a more efficient decision procedure is possible, and it would have saved us some work if we'd come up with this sooner.

We have only scratched the surface of the extensive and beautiful theory of complex functions, and there are many future avenues to explore. One obvious idea would be to generalize Cauchy's theorem to simply connected regions and beyond, perhaps to a global version stated in terms of homotopy or homology of paths. Dixon's proof of the global Cauchy theorem [11] should be relatively easy to prove using the existing machinery.

Another interesting idea is to seek applications to other areas of mathematics. One notable possibility would be a proof of the Prime Number Theorem based on the usual complex-analytic machinery and the nonvanishing of the Riemann zeta function $\zeta(s)$ for $Re(s) \geq 1$. The "elementary" Erdös-Selberg proof has already been formalized [1], but Solovay has suggested the usual analytical proof as a significant challenge for formal verification. We hope to take up this challenge in a future paper.

References

1. J. Avigad, K. Donnelly, D. Gray, and P. Raff. A formally verified proof of the prime number theorem. To appear in the ACM Transactions on Computational Logic, 2006.
2. Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors. *Theorem Proving in Higher Order Logics: 12th International Conference, TPHOLs'99*, volume 1690 of *Lecture Notes in Computer Science*, Nice, France, 1999. Springer-Verlag.
3. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.
4. J. Harrison. A Mizar mode for HOL. In J. v. Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 203–220, Turku, Finland, 1996. Springer-Verlag.

5. J. Harrison. Proof style. In E. Giménez and C. Paulin-Mohring, editors, *Types for Proofs and Programs: International Workshop TYPES'96*, volume 1512 of *Lecture Notes in Computer Science*, pages 154–172, Aussois, France, 1996. Springer-Verlag.
6. J. Harrison. *Theorem Proving with the Real Numbers.* Springer-Verlag, 1998. Revised version of author's PhD thesis.
7. J. Harrison. Complex quantifier elimination in HOL. In R. J. Boulton and P. B. Jackson, editors, *TPHOLs 2001: Supplemental Proceedings*, pages 159–174. Division of Informatics, University of Edinburgh, 2001. Published as Informatics Report Series EDI-INF-RR-0046. Available on the Web at `http://www.informatics.ed.ac.uk/publications/report/0046.html`.
8. J. Harrison. A HOL theory of Euclidean space. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005*, volume 3603 of *Lecture Notes in Computer Science*, pages 114–129, Oxford, UK, 2005. Springer-Verlag.
9. R. Henstock. A Riemann-type integral of Lebesgue power. *Canadian Journal of Mathematics*, 20:79–87, 1968.
10. J. Kurzweil. Generalized ordinary differential equations and continuous dependence on a parameter. *Czechoslovak Mathematics Journal*, 82:418–446, 1958.
11. S. Lang. *Complex Analysis.* Graduate Texts in Mathematics. Springer-Verlag, 3rd edition, 1993.
12. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
13. D. Syme. DECLARE: A prototype declarative proof system for higher order logic. Technical Report 416, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK, 1997.
14. M. Wenzel. Isar - a generic interpretive approach to readable formal proof documents. In Bertot et al. [2], pages 167–183.
15. G. T. Whyburn. *Topological Analysis*, volume 23 of *Princeton Mathematical Series.* Princeton University Press, revised edition, 1964.
16. F. Wiedijk. Mizar light for HOL Light. In R. J. Boulton and P. B. Jackson, editors, *14th International Conference on Theorem Proving in Higher Order Logics: TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 378–394. Springer-Verlag, 2001.
17. V. Zammit. On the implementation of an extensible declarative proof language. In Bertot et al. [2], pages 185–202.

On the Formalization of Lebesgue Integrals

Yasunari Shidama[1], Noburu Endou[2], and Pauline N. Kawamoto[1]

[1] Shinshu University, Nagano, Japan
[2] Gifu National College of Technology, Gifu, Japan

Abstract. In this paper, we report the progress of our work in the Mizar project of creating a library of various theorems relating to Lebesgue integrals. Concepts such as the integration of measurable functions are defined and topics on the linearity of integration operations, etc. are also included in this library.

1 Introduction

In this work, the authors focus on building a portion of the Mizar library dedicated to the area of functional analysis. The theory of Lebesgue integration, as with Riemann integration, is a classical as well as important foundation of analysis and provides a necessary part of the Mizar library in this area. Lebesgue integration is an important tool in probability and engineering fields and it is used widely in many applications. In the Mizar articles MESFUNC1 [7] to MESFUNC5 [10], we formalized the definitions for measurable functions, integration of simple functions, integration of measurable functions, as well as various theorems concerning properties such as linearity. Also, in the work up to MESFUNC5, integrated functions are assumed to take values of $\pm\infty$, but functions normally defined in Mizar are not designed to take $\pm\infty$ values in many cases. For this reason, in MESFUNC6 [12], we treat the definitions of integrability of general real valued functions which do not take values of $\pm\infty$ and their linearity, etc. We are also currently working on the formalization of various theorems related to these items as well as function spaces such as L^p space created by sets of integrable functions. This paper summarizes the work of the authors on the formalization of Lebesgue integration.

2 Outline of Formalization Work

2.1 Specification of Measurable Functions

The formalization of concepts concerning the foundations of σ fields and the theory of measurability was done in Mizar by Józef Białas in library articles MEASURE1 [1] to MEASURE4 [2]. In this work, the topic of measurability of sets is also addressed. However, the definition of measurability by Białas is of σ additive measure and is not always suitable for defining general measurable functions. For this reason, the authors redefined measurability of sets anew to apply only to σ fields. We compare these below.

ISBN 978-83-7431-128-1 ISSN 0860–150X

```
definition
  let X be set,
      S be SigmaField of X,
      M be sigma_Measure of S,
      A be set;
pred A is_measurable M means
:: MEASURE1:def 12
A in S;
end;

definition
  let X be set;
  let S be SigmaField of X;
  let A be set;
pred A is_measurable_on S means
:: MESFUNC1:def 11
A in S;
end;
```

The former is the definition by Białas and in regular mathematical terms A is_measurable M indicates an M-measurable set on measure space (X, S, M). The latter A is_measurable_on S on the other hand indicates a Borel set on measurable space (X, S). (In this sense, it might have been better to define the latter as A is Borel.) In MESFUNC1, the measurability of sets is redefined as mentioned above and the definition of measurable functions is constructed based on this. In the following MESFUNC2 [8], we discuss the sum, difference, etc. of measurable functions. For this, the treatment of algebraic operations and limits of real numbers including ±∞ (ExtREAL) becomes necessary so in EXTREAL1 [5], EXTREAL2 [6] we define the multiplication and division of ExtREAL and prove related theorems. A portion of the work is shown below.

```
definition
  let x,y be R_eal;
func x * y -> R_eal means
:: EXTREAL1:def 1
 (ex a,b being Real st (x = a & y = b & it = a * b)) or
 (((0. < x & y=+infty) or (0. < y & x=+infty) or (x < 0. & y=-infty)
 or (y < 0. & x = -infty)) & it = +infty) or
 (((x < 0. & y=+infty) or (y < 0. & x=+infty) or (0. < x & y=-infty)
 or (0. < y & x = -infty)) & it = -infty) or
 ((x = 0. or y = 0.) & it = 0.);
end;
```

The definition of measurable functions is formalized in MESFUNC1 as shown below by slightly generalizing the standard definition appearing in textbooks.

```
reserve X for non empty set;
reserve x for Element of X;
reserve f for PartFunc of X,ExtREAL;
reserve a for R_eal;

definition
  let X,f,a;
func less_dom(f,a) -> Subset of X means
:: MESFUNC1:def 12
x in it iff x in dom f & ex y being R_eal st y=f.x & y < a;

definition
  let X be non empty set;
  let S be SigmaField of X;
  let f be PartFunc of X,ExtREAL;
  let A be Element of S;
pred f is_measurable_on A means
:: MESFUNC1:def 17
for r being real number holds
    A /\ less_dom(f,R_EAL r) is_measurable_on S;
end;
```

Also, in the generalization here, although in standard textbooks measurable functions are taken to be functions defined on measurable sets (f is Function of A, ExtREAL), in this definition we use PartFunc to formulate the measurability of functions defined on sets that are not measurable. This is because some of the symbols will become complicated in the definition and we want to include cases of functions which are not defined on measurable functions.

The measurability of the sum, difference, and scalar product of measurable functions is as shown below.

```
reserve X for non empty set;
reserve f,g for PartFunc of X,ExtREAL;
reserve S for SigmaField of X;
reserve r for Real;
reserve A for Element of S;

theorem :: MESFUNC2:7
for f,g,A st f is_finite & g is_finite & f is_measurable_on A &
g is_measurable_on A holds f+g is_measurable_on A;

theorem :: MESFUNC2:13
   for f,g,A st f is_finite & g is_finite & f is_measurable_on A &
g is_measurable_on A & A c= dom g holds f-g is_measurable_on A;

theorem :: MESFUNC1:41
   for X,S,f,A,r st f is_measurable_on A & A c= dom f
```

```
holds r(#)f is_measurable_on A;
```

2.2 Formalization of Simple Functions

We defined the simple function in MESFUNC2 as follows.

```
registration
  let X be set;
  let S be SigmaField of X;
  cluster disjoint_valued FinSequence of S;
end;

definition let X be set;
  let S be SigmaField of X;
  mode Finite_Sep_Sequence of S is disjoint_valued FinSequence of S;
end;

definition
  let X be non empty set;
  let S be SigmaField of X;
  let f be PartFunc of X,ExtREAL;
pred f is_simple_func_in S means
:: MESFUNC2:def 5
 f is_finite &
 ex F being Finite_Sep_Sequence of S st
  (dom f = union rng F &
   for n being Nat,x,y being Element of X st
    n in dom F & x in F.n & y in F.n holds f.x = f.y);
end;
```

Here the meaning of F is `Finite_Sep_Sequence of S` is the values of F are a Finite Sequence of disjoint measurable sets. Therefore, to say that f is a simple function means that the domain of f can be divided into disjoint measurable sets of F and that for each $F.n$, f takes a constant value.

2.3 Integration of Simple Functions

In MESFUNC3 [11], we defined the integration of simple functions whose domains take non-empty, non-negative values as follows. The restriction on domains is resolved in MESFUNC5.

```
definition
  let X be non empty set;
  let S be SigmaField of X;
  let M be sigma_Measure of S;
  let f be PartFunc of X,ExtREAL;
    assume f is_simple_func_in S
           & dom f <> {}
           & for x be set st x in dom f holds 0. <= f.x;
  func integral(X,S,M,f) -> Element of ExtREAL means
  :: MESFUNC3:def 2
    ex F be Finite_Sep_Sequence of S,
        a, x be FinSequence of ExtREAL st F,a are_Re-presentation_of f
           & a.1 =0.
           & (for n be Nat st 2 <= n & n in dom a holds
             0. < a.n & a.n < +infty )
           & dom x = dom F
           & (for n be Nat st n in dom x holds x.n=a.n*(M*F).n)
           & it=Sum(x);
  end;
```

As is well known, simple functions can be expressed by dividing the domains of measurable sets (including measure 0) into a countless number of ways and producing an infinite number of expressions. We need to show that integration operations will always produce a unique answer regardless of the function expression used. To do this, we use FUNCTOR for the definition of integration and show its existence and uniqueness. Also, in MESFUNC4 [9] we show the linearity of integration of simple functions.

```
theorem :: MESFUNC4:5
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f,g be PartFunc of X,ExtREAL
  st f is_simple_func_in S & dom f <> {}
   & (for x be set st x in dom f holds 0. <= f.x)
   & g is_simple_func_in S & dom g = dom f
   & (for x be set st x in dom g holds 0. <= g.x)
 holds
    f+g is_simple_func_in S & dom (f+g) <> {}
  & (for x be set st x in dom (f+g) holds 0. <= (f+g).x)
  & integral(X,S,M,f+g)=integral(X,S,M,f)+integral(X,S,M,g);

theorem :: MESFUNC4:6
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f,g be PartFunc of X,ExtREAL, c be R_eal
  st f is_simple_func_in S
   & dom f <> {}
   & (for x be set st x in dom f holds 0. <= f.x)
   & 0. <= c & c < +infty
   & dom g = dom f
   & (for x be set st x in dom g holds g.x=c*f.x)
 holds
    integral(X,S,M,g)=c*integral(X,S,M,f);
```

2.4 Integration of Measurable Functions

To formalize the integration of measurable functions, we follow the method of standard textbooks and in MESFUNC5 we first show that for functions taking non-negative values it is expressed as the limit of a sequence of simple functions as:

```
definition
let X be non empty set,
    H be Functional_Sequence of X,ExtREAL,
    x be Element of X;
func H#x -> ExtREAL_sequence means
:: MESFUNC5:def 13
for n be Nat holds it.n = (H.n).x;
end;
```

```
theorem :: MESFUNC5:70
for X be non empty set, S be SigmaField of X,
    f be PartFunc of X,ExtREAL st
 (ex A be Element of S st A = dom f & f is_measurable_on A) &
 f is nonnegative
holds
  ex F be Functional_Sequence of X,ExtREAL st
   (for n be Nat holds F.n is_simple_func_in S & dom(F.n) = dom f) &
   (for n be Nat holds F.n is nonnegative) &
   (for n,m be Nat st n <=m holds
     for x be Element of X st x in dom f holds (F.n).x <= (F.m).x ) &
   (for x be Element of X st x in dom f holds
     (F#x) is convergent & lim(F#x) = f.x);
```

We define the limit of simple function integration as follows.

```
definition
let X be non empty set;
let S be SigmaField of X;
let M be sigma_Measure of S;
let f be PartFunc of X,ExtREAL;
  assume that
ex A be Element of S st A = dom f & f is_measurable_on A and
f is nonnegative;
func integral+(M,f) -> Element of ExtREAL means
:: MESFUNC5:def 15
ex F be Functional_Sequence of X,ExtREAL,
   K be ExtREAL_sequence st
```

```
   (for n be Nat holds F.n is_simple_func_in S & dom(F.n) = dom f) &
   (for n be Nat holds F.n is nonnegative) &
   (for n,m be Nat st n <=m holds
     for x be Element of X st x in dom f holds (F.n).x <= (F.m).x ) &
   (for x be Element of X st  x in dom f holds
     F#x is convergent & lim(F#x) = f.x) &
   (for n be Nat holds K.n=integral'(M,F.n)) &
   K is convergent &
   it=lim K;
end;
```

Here as well, to show that the limit does not depend on the simple function sequence selected, we use FUNCTOR, just as in the case of simple functions, to create the definition and prove the existence and uniqueness of the integration of a given non-negative measurable function.

Next, for functions which are not non-negative, we use the fact that they can be expressed using the non-negative function `max+f` and the non-positive function `max-f` and create the definition as follows.

```
definition
  let X be non empty set;
  let S be SigmaField of X;
  let M be sigma_Measure of S;
  let f be PartFunc of X,ExtREAL;
  func Integral(M,f) -> Element of ExtREAL equals
:: MESFUNC5:def 16
  integral+(M,max+f)-integral+(M,max-f);
end;
```

After we formalize the definition of integration for measurable functions, we use it to prepare a variety of theorems for Lebesgue integration and the formulation of function spaces such as L^p space. We show the representative theorems below.

- Theorems concerning the division of integration areas

```
theorem :: MESFUNC5:104
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f be PartFunc of X,ExtREAL,
    A,B be Element of S st
 f is_integrable_on M & A misses B holds
  Integral(M,f|(A\/B)) = Integral(M,f|A) + Integral(M,f|B);
```

```
theorem :: MESFUNC5:105
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f be PartFunc of X,ExtREAL,
    A,B be Element of S st
 f is_integrable_on M & B = (dom f)\A
holds
   f|A is_integrable_on M & Integral(M,f) = Integral(M,f|A)
                                            +Integral(M,f|B);
```

- Theorems on the integrability of absolute value functions of integrable functions

```
theorem :: MESFUNC5:106
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f be PartFunc of X,ExtREAL st
 (ex A be Element of S st A = dom f & f is_measurable_on A )
   holds
    f is_integrable_on M iff |.f.| is_integrable_on M;
```

```
theorem :: MESFUNC5:107
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f be PartFunc of X,ExtREAL st
 f is_integrable_on M holds
    |. Integral(M,f) .| <= Integral(M,|.f.|);
```

```
theorem :: MESFUNC5:108
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f,g be PartFunc of X,ExtREAL st
 ( ex A be Element of S st A = dom f & f is_measurable_on A ) &
 dom f = dom g & g is_integrable_on M &
 ( for x be Element of X st x in dom f holds |.f.x .| <= g.x )
holds
   f is_integrable_on M & Integral(M,|.f.|) <= Integral(M,g);
```

- Theorem showing that integrable functions are sets with measure 0 that take ±∞ values

```
theorem :: MESFUNC5:111
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f be PartFunc of X,ExtREAL st
f is_integrable_on M holds
  f"{+infty} in S & f"{-infty} in S & M.(f"{+infty})=0 &
```

```
                     M.(f"{-infty})=0 &
f"{+infty} \/ f"{-infty} in S &
                     M.(f"{+infty} \/ f"{-infty})=0;
```

- Theorems on the linearity of integration operations

```
theorem :: MESFUNC5:114
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f,g be PartFunc of X,ExtREAL st
 f is_integrable_on M & g is_integrable_on M holds
   f+g is_integrable_on M;
```

```
theorem :: MESFUNC5:115
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f,g be PartFunc of X,ExtREAL st
 f is_integrable_on M & g is_integrable_on M holds
  ex E be Element of S st
    E = dom f /\ dom g & Integral(M,f+g)=Integral(M,f|E)
                                        +Integral(M,g|E);
```

```
theorem :: MESFUNC5:116
for X be non empty set,
    S be SigmaField of X,
    M be sigma_Measure of S,
    f be PartFunc of X,ExtREAL,
    c be Real st
 f is_integrable_on M holds
  c(#)f is_integrable_on M &
  Integral(M,c(#)f) = R_EAL c * Integral(M,f);
```

```
definition
let X be non empty set;
let S be SigmaField of X;
let M be sigma_Measure of S;
let f be PartFunc of X,ExtREAL;
let B be Element of S;
func Integral_on(M,B,f) -> Element of ExtREAL equals
:: MESFUNC5:def 18
 Integral(M,f|B);
end;
```

```
theorem :: MESFUNC5:117
for X be non empty set,
      S be SigmaField of X,
      M be sigma_Measure of S,
      f,g be PartFunc of X,ExtREAL,
      B be Element of S st
  f is_integrable_on M & g is_integrable_on M & B c= dom(f+g)
 holds
    f+g is_integrable_on M &
    Integral_on(M,B,f+g) = Integral_on(M,B,f) + Integral_on(M,B,g);

 theorem :: MESFUNC5:118
 for X be non empty set,
      S be SigmaField of X,
      M be sigma_Measure of S,
      f be PartFunc of X,ExtREAL,
      c be Real,
      B be Element of S st
  f is_integrable_on M & f is_measurable_on B holds
   f|B is_integrable_on M &
   Integral_on(M,B,c(#)f) = R_EAL c * Integral_on(M,B,f);
```

The definitions and theorems above are for integration operations of functions which take values of real numbers including ±∞ (ExtREAL), but in spaces such as L^p, functions which take only finite real number values, not ExtREAL, are used. To conveniently connect them with the constructed library, we formulated a library on integration in MESFUNC6 for functions which take only finite real numbers as values.

3 Conclusion

We reported the current state of library construction concerning Lebesgue integrals. The fundamental areas including the definitions of measurability and integrability as well as the linearity of integration have been completed and the next step will be to use this work to formalize various related theorems as well as other types of function spaces such as L^p space. Also, since the current definition of integration uses measurability as a general σ additive measure, technically we cannot call this Lebesgue integration. For this reason, we must take a careful look at the Lebesgue measure defined in MEASURE7 [3]. Furthermore, we have not yet begun the work on integration of functions on direct product measure spaces and it will be necessary to begin this formalization as soon as possible.

The theory of Lebesgue integrals is considered a classical work and it contains delicate and rich contents. The authors of this paper share a deep interest in this area and would appreciate comments and collaboration in the construction efforts of this portion of the Mizar library.

Finally, the formalization of Lebesgue integration has been pursued thus far in the case where no topology has been introduced on set X. With regard to measures on topological spaces, the definition of Borel sets has been constructed in TOPGEN_4 [4]. Based on this, future work concerning the formalization of integration on topological spaces will become possible as a next step.

References

1. Józef Białas. The σ-additive Measure Theory. *Formalized Mathematics*, 2(**2**), pp.263-270, 1991.
2. Józef Białas. Properties of Caratheodor's Measure. *Formalized Mathematics*, 3(**1**), pp.67-70, 1992.
3. Józef Białas. The One-Dimensional Lebesgue Measure. *Formalized Mathematics*, 5(**2**), pp.253-258, 1996.
4. Adam Grabowski. On the Borel Families of Subsets of Topological Spaces. *Formalized Mathematics*, 13(**4**), pp.453-461, 2005.
5. Noboru Endou, Katsumi Wasaki, and Yasunari Shidama. Basic Properties of Extended Real Numbers. *Formalized Mathematics*, 9(**3**), pp.491-494, 2001.
6. Noboru Endou, Katsumi Wasaki, and Yasunari Shidama. Some Properties of Extended Real Numbers Operations: absolute value, min and max. *Formalized Mathematics*, 9(**3**), pp.511-516, 2001.
7. Noboru Endou, Katsumi Wasaki, and Yasunari Shidama. Definitions and Basic Properties of Measurable Functions. *Formalized Mathematics*, 9(**3**), pp.495-500, 2001.
8. Noboru Endou, Katsumi Wasaki, and Yasunari Shidama. Measurability of Extended Real Valued Functions. *Formalized Mathematics*, 9(**3**), pp.525-529, 2001.
9. Noboru Endou and Yasunari Shidama. Linearity of Lebesgue Integral of Simple Valued Function. *Formalized Mathematics*, 13(**4**), pp.463-466, 2005.
10. Noboru Endou and Yasunari Shidama. Integral of Measurable Function. *Formalized Mathematics*, 14(**2**), pp.53-70, 2006.
11. Yasunari Shidama and Noboru Endou. Lebesgue Integral of Simple Valued Function. *Formalized Mathematics*, 13(**1**), pp.67-72, 2005.
12. Yasunari Shidama and Noboru Endou. Integral of Real-Valued Measurable Function. *Formalized Mathematics*, 14(**4**), pp.143-152, 2006.

Computer-Assisted Reasoning about Algebraic Topology *

Adam Grabowski[i] and Artur Korniłowicz[ii]

[i]Institute of Mathematics
University of Białystok, Poland
adam@math.uwb.edu.pl

[ii]Institute of Computer Science
University of Białystok, Poland
arturk@math.uwb.edu.pl

Abstract. We describe the motivation and main ideas of the formal approach to the algebraic topology which was developed recently. The work provides some machinery allowing the authors of Mizar articles to obtain more advanced and more valuable results, mainly in general topology, but also in algebra. Some aspects of the Mizar language are also discussed based on the examples taken directly from our developments.

1 Introduction

What is algebraic topology? To answer this question briefly, one could say – it is a part of mathematics in which machinery of abstract algebra is used to analyze properties of topological spaces. The idea of algebraic topology is to transform, in the sense of category theory, a topological space into a group (fundamental, homology or cohomology) and make reasoning inside the group. The key point is that homeomorphic spaces are always transformed into isomorphic groups.

This discipline has been successfully applied in proofs of theorems explicitly involving subsets of n-dimensional Euclidean spaces, like:

- the Brouwer fixed point theorem,
- the Borsuk-Ulam theorem about antipodal points,

but also theorems that, at first glance, are not related to topology, such as:

- fundamental theorem of algebra,
- the Nielsen-Schreier theorem about subgroups of a free group.

The rapid development of topology and abstract algebra in the Mizar system could have been observed from the very beginnings of the Mizar Mathematical Library (MML) dated back to 1989 (or even before as we point out in the next section).

* This work has been partially supported by the FP6 IST project no. 510996 *Types for Proofs and Programs* (TYPES).

ISBN 978-83-7431-128-1 ISSN 0860-150X

As the interconnections between topology and abstract algebra are clearly seen and represented in the literature, moreover both disciplines are rather nicely encoded in the MML, the idea arose to benefit from these earlier developments. On the other hand, we wanted to test if the Mizar language itself has enough expressive power to allow the authors for the reasoning not only in separate topics – like the algebra and the topology, as it was done before, but also if it can be used for exploring theories somewhere on their intersection.

So, the question arose, why not to mechanize another broad area of mathematics? Once set-topology was easy choice (remember the foundations of the MML are provided by axioms of Tarski-Grothendieck set theory), the algebraic topology is located somewhat further from the core of the Mizar system.

Incidentally, group theory and category theory were also represented in the MML (as a matter of fact, any of four subdisciplines under consideration was developed by different people than the remaining ones and the cooperation was rather small).

Although the book of Greenberg [6] was chosen as the background of our work, two main tasks were crucial at early stages of this new formalization project. Originally, as for the time [4] has been written, the primary aim was to provide lemmas for the proof of the Jordan Curve Theorem. As the other, and of minor importance, the development of algebraic topology was pointed out. As it soon appeared, due to some technical time-consuming difficulties within the JCT proof sketch, also the Brouwer fix point theorem[1] could be used to complete the proof. So, the situation changed substantially and since some works towards algebraic topology were yet available in the MML, the directions were as follows:

- first, to prove the Brouwer fix point theorem [7],
- then, using the above, to complete JCT [3].

2 From First Experiments Towards Mechanized Topology

General topology was one of the topics chosen for formalization as a starting point when establishing MML. This choice was by no means accidental – since the Mizar system is based on set theory, just set theory and topology seemed to be a promising stage for early experiments.

Shortly thereafter the fundamentals of the theory were translated into the Mizar language, one of the articles of the more advanced type was included in the library. Its MML identifier was `BORSUK_1` [21] to emphasize the influence of the work of the famous Polish mathematician Karol Borsuk[2].

This article, as [16] claims, was written in 1981 to test Mizar-2 experimental release, much before the regular collecting of Mizar articles was started; it was the translation of the paper *On the homotopy types of some decomposition spaces* by Borsuk into a machine-understandable language. Four years before, a similar effort, but using older Mizar-PC, the formalization of the paper by Krasinkiewicz on the homeomorphism of the Sierpiński curve, was also prepared. These experiments had not much in common with the activities extending the real repository of computer-checked mathematical facts – the article had to have the environment section fully axiomatized, i.e. all preliminary lemmas could be stated without proofs.

[1] Authors are greatful to Professor Yasunari Shidama of Shinshu University for proposing to formalize the theorem.

[2] Karol Borsuk was advisor of Andrzej Trybulec's PhD in 1974.

At first sight, algebraic topologists use extensively also the apparatus of category theory; even if it is often the case of very simple properties only, and the theorems which are used are not very advanced (especially diagrams are useful here), also this discipline is one of those chosen once as a primary testbed for the Mizar system.

One can recognize two streams of the category theory reflected in the formalization – one of them, authored mainly by Byliński, even though containing a number of results of the general interest, and especially constructions of various category structures as examples, seems not to be developed nowadays. As `CAT_1` was issued in 1989, from that time the language changed significantly, especially the part devoted to the structure handling. The other approach based on [19], which fully exploits system and language capabilities, i.e. categories without uniqueness of the domain and codomain (with identifiers `ALTCAT` standing for alternative categories) was designed and partially implemented by Andrzej Trybulec.

As the example, Cartesian category based on `ProdCatStr` (`CAT` series) has 10 selectors, which is twice as big as the reasonable limit of this parameter; recently the Library Committee of the Association of Mizar Users works on pushing the average number of selectors down. The authors are usually requested not to put too many type information into structures, rather to carry information via attributes; to be more explicit, the unity in the group should not be defined as a selector (especially because we still need an axiom that the unity behaves well), but rather to assure axiomatically that such unity exists.

3 Basic Notions of Formal Algebraic Topology

As one could see in the previous section, algebraic topology merges different theories: topology, group theory and not necessarily explicitly, but using some auxiliary notions, category theory. Of course, for the moment only a small part of algebraic topology has been formalized in MIZAR, but large enough to require having developed mentioned theories in the MML. Our MIZAR articles containing formalization of the topic are quite clear, in the sense, that we had to prove neither lemmas about pure group theory nor topology.

In spite of the category theory can be explicitly involved in the development of algebraic topology, because for example, fundamental group can be seen as a functor from the category of pointed topological spaces with their homeomorphisms being morphisms in the category into the category of groups with their isomorphisms being morphisms, we decided not to do it.

As we already mentioned, the idea of thorough formalization of algebraic topology evolved from the role of the "side-effect" of JCT project into top priority task. This is reflected in the identifiers of files stored in the MML: while five first belong to the `BORSUK` series (here [4] and [5] will be briefly examined), the remaining, containing more advanced results, form `TOPALG` series, which will be described in the next section.

3.1 Paths

Let us start with the very basic, although important notion, of a continuous function from the unit interval into a given topological space, which satisfies some additional properties:

```
definition let T be TopStruct; let a, b be Point of T;
  assume a, b are_connected;
  mode Path of a, b -> Function of I[01], T means
:: BORSUK_2:def 2
  it is continuous & it.0 = a & it.1 = b;
end;
```

The assumption allowing to show that an object stated in the definiens exists, is of course needed to prove the correctness of the definition. The definition itself is permissive – it states that if the mapping connecting two points, say a and b, exists, we call it a `Path`, otherwise the definiens is not accessible, even if the Mizar analyser accepts the type `Path of a, b` with no errors reported.

```
definition let T be arcwise_connected TopStruct;
  let a, b be Point of T;
  redefine mode Path of a, b means
:: BORSUK_2:def 4
  it is continuous & it.0 = a & it.1 = b;
end;
```

Although the earlier definition is of a slightly more general type (i.e. it assumes the existence of the mapping connecting only two fixed points), every time that the path returns proper values on its limits should be justified via predicate `are_connected` which seems to be too high price for that minor generalization. Hence if the considered space possesses the property of being `arcwise_connected` (which truly speaking should be rather *pathwise connected*), paths behave as expected (and e.g. real Euclidean line or the unit interval has this adjective added automatically to its type) and the reasoning simplifies as the user can forget about the assumption from the original definition of a path.

3.2 Homotopies

Very similar trick to the aforementioned, i.e. first to guarantee that the desired object exists, then to prove the correctness of the appropriate functor definition, was applied to the definition of a homotopy.

```
definition let T be non empty TopStruct;
  let a, b be Point of T;
  let P, Q be Path of a, b;
  pred P, Q are_homotopic means
:: BORSUK_2:def 7
  ex f being Function of [:I[01],I[01]:], T st
    f is continuous &
    for s being Point of I[01] holds f.(s,0) = P.s & f.(s,1) = Q.s &
    for t being Point of I[01] holds f.(0,t) = a & f.(1,t) = b;
  symmetry;
end;
```

Under this assumption the existence of a continuous deformation can be proven; of course not its uniqueness, hence the definition of the `mode`, not of the functor.

```
definition let T be non empty TopSpace;
           let a, b be Point of T;
           let P, Q be Path of a, b;
  assume P, Q are_homotopic;
  mode Homotopy of P, Q -> Function of [:I[01],I[01]:], T means
:: BORSUK_6:def 13
  it is continuous &
  for s being Point of I[01] holds it.(s,0) = P.s & it.(s,1) = Q.s &
  for t being Point of I[01] holds it.(0,t) = a & it.(1,t) = b;
end;
```

As among five arguments of this definition two visible are enough, the definition is pretty readable.

4 Fundamental Groups

The relation `are_homotopic` is reflexive, symmetric and transitive, hence all loops of any pointed topological space can be divided into its equivalence classes. Next, it is possible to define a binary operation on the set of all equivalence classes saying that the result class is generated by the sum of representants of its arguments. Such an operation is well-defined and gives the group with the identity being the class generated by the trivial loop at the basepoint and inverses being classes generated by inverses of representants of the classes. The group is called the fundamental group of the space at the basepoint. The formal definition of the group, its fundamental properties, that is the independence (up to the isomorphism) of the fundamental groups from the choice of basepoints from a path-connected component of the space; isomorphism of fundamental groups of homeomorphic topological spaces are presented in next sections. Moreover, examples of fundamental groups of some spaces are listed.

4.1 Definition

The Mizar functor, which makes a group based on a topological space was created in a typical (for Mizar) way, that is, first fields of a group structure (`HGrStr`, [22]) (its carrier and binary operation) were described.

```
definition
  let X be non empty TopSpace, a be Point of X;
  func FundamentalGroup(X,a) -> strict HGrStr means
:: TOPALG_1:def 3
  the carrier of it = Class EqRel (X,a) &
  for x, y being Element of it ex P, Q being Loop of a st
    x = Class(EqRel(X,a),P) & y = Class(EqRel(X,a),Q) &
    (the mult of it).(x,y) = Class(EqRel(X,a),P+Q);
end;
```

Since in the so-called *paper mathematics* fundamental groups are usually denoted as $\pi_1(X, a)$, an appropriate synonym has been introduced.

```
notation
  let X be non empty TopSpace, a be Point of X;
  synonym pi_1(X,a) for FundamentalGroup(X,a);
end;
```

Then, underlying properties (non-emptiness, associativity, the existence of the unity and inverses) of the carrier and the operation expressed as a cluster of adjectives have been proven.

```
registration
  let X be non empty TopSpace;
  let a be Point of X;
  cluster pi_1(X,a) -> non empty associative Group-like;
end;
```

The above mentioned registration ensures that `pi_1(X,a)` is a group. All details are stored in the MIZAR article [14].

In the next papers in the series fundamental groups of some basic topological spaces have been computed. Examples are listed in Tab. 1.

Table 1. Examples of fundamental groups of some common topological spaces

space S	$\pi_1(S)$
convex spaces	$\{0\}$
circle	$\mathbb{Z}$
torus	$\mathbb{Z} \times \mathbb{Z}$

4.2 Fundamental Groups of Convex Subspaces of Euclidean Spaces

One of the basic examples of fundamental groups are those computed for convex subspaces of n-dimensional Euclidean spaces. Since all paths between two given points of a convex space are homotopic, the group contains exactly one element – the equivalence class represented by a loop, what is expressed in [9] by the functorial registration:

```
registration
  let n be Element of NAT,
      T be non empty convex SubSpace of TOP-REAL n,
      a be Point of T;
  cluster pi_1(T,a) -> trivial;
end;
```

The homotopy is established by the function:

```
definition
  let n be Element of NAT,
      T being non empty convex SubSpace of TOP-REAL n,
      a, b be Point of T,
      P, Q be Path of a,b;
  func ConvexHomotopy(P,Q) -> Function of [:I[01],I[01]:], T means
:: TOPALG_2:def 2
  for s, t being Element of I[01],
      a1, b1 being Point of TOP-REAL n st a1 = P.s & b1 = Q.s holds
   it.(s,t) = (1-t) * a1 + t * b1;
end;
```

4.3 Fundamental Groups of Homeomorphic Spaces

The important property of fundamental groups is the fact that they are invariants of homeomorphic topological spaces. In [11] we proved the following

```
theorem :: TOPALG_3:35
  for S being non empty TopSpace,
      T being non empty arcwise_connected TopSpace,
      s being Point of S, t being Point of T st
  S,T are_homeomorphic holds pi_1(S,s),pi_1(T,t) are_isomorphic;
```

To get rid of the pathwise connectedness the theorem can be weakened to the form: `pi_1(S,s)` and `pi_1(T,h(s))` are isomorphic when `h` is a homeomorphism from `S` onto `T`. An isomorphism can be established by a function from `pi_1(S,s)` to `pi_1(T,h(s))`, which assigns the equivalence class represented by the loop `h*f` at the point `h(s)` of `T`, where `f` is a loop at a given point `s` of `S`.

4.4 Fundamental Groups of Simple Closed Curves

In many papers devoted to algebraic topology the fundamental group of a circle, which is isomorphic to the group of integers, is mentioned. Having results presented in Sec. 4.3 it is easy to generalize the theorem to any simple closed curve, which in fact we did in [12]:

```
theorem :: TOPALG_5:27
  for S being being_simple_closed_curve SubSpace of TOP-REAL 2,
      x being Point of S holds
   INT.Group, pi_1(S,x) are_isomorphic;
```

The proof of the isomorphism for any simple closed curve can be started from an easier case. It is enough to compute the fundamental group of the unit circle centered at the point $(0,0)$ based at the point $(1,0)$, and applying the independence of the fundamental group from the base point pass to the fundamental group of the unit circle centered at the point $(0,0)$ based at any point, and finally using isomorphism of fundamental groups of homeomorphic spaces pass from the unit circle to the general case, that is to simple closed curves.

4.5 Fundamental Groups of Products of Topological Spaces

Another property establishes correspondence between fundamental groups and the product of topological spaces. The main result of [10] states that the fundamental group of the product of two topological spaces is isomorphic to the product of the fundamental groups of the spaces.

```
theorem :: TOPALG_4:32
  for S, T being non empty arcwise_connected TopSpace,
      s1, s2 being Point of S,
      t1, t2 being Point of T holds
  pi_1([:S,T:],[s1,t1]), product <*pi_1(S,s2),pi_1(T,t2)*>
                                  are_isomorphic;
```

The isomorphism is established by the function:

```
definition
  let S, T be non empty TopSpace,
      s be Point of S, t be Point of T;
  func FGPrIso(s,t) ->
    Function of pi_1([:S,T:],[s,t]), product <*pi_1(S,s),pi_1(T,t)*>
      means
:: TOPALG_4:def 2
  for x being Point of pi_1([:S,T:],[s,t])
   ex l being Loop of [s,t] st x = Class(EqRel([:S,T:],[s,t]),l) &
   it.x = <*Class(EqRel(S,s),pr1 l),Class(EqRel(T,t),pr2 l)*>;
end;
```

One trivial consequence of the theorem is that the fundamental group of the torus is isomorphic to $\mathbb{Z} \times \mathbb{Z}$ – the torus can be represented as the product of two circles, which are simple closed curves.

4.6 The Brouwer Fixed Point Theorem

The main result of developing of algebraic topology in MIZAR is the formalization of the Brouwer fixed point theorem for 2-dimensional Euclidean spaces, see [13].

```
theorem :: BROUWER:14
  for r being non negative (real number),
      o being Point of TOP-REAL 2,
      f being continuous Function of Tdisk(o,r), Tdisk(o,r) holds
  f has_a_fixpoint;
```

Its proof (taken directly from [7]) is mainly based on the fact that a circle is not a retract of a disk; and this is the place where machinery of algebraic topology is used. What is needed here is the fact that fundamental group of a circle is non trivial. This is a simple consequence of the fact that fundamental group of a circle, which is a simple closed curve, is isomorphic to the group of integers, which is infinite, that is non-trivial.

5 Conclusions

Although the mechanization of the algebraic topology into Mizar formalism is still at very early stage (some ten articles devoted to the topic out of 960 in the MML), we can estimate the influence of this effort for the system and the library very positively. The development of the fundamentals helped us to speed up the formalization of the proof of the Jordan Curve Theorem in Mizar, and the Brouwer fix point theorem [13] was completed also employing algebraic topology techniques. Another large project of the Mizar team, the translation of the *Compendium of Continuous Lattices*, is still ongoing, although currently at a much slower pace than at the beginning. In this case structures of topological spaces were successfully merged with posets.

There are also articles forming TOPGEN series which cover [15] but this work just started. However, in parallel to broad formalization efforts, within the Mizar Mathematical Library also problems which are more a kind of logical puzzles can be solved – as fourteen Kuratowski's sets or complete formalization of a chosen paper – [8].

Acknowledgements

The authors would like to express their gratitude to Andrzej Trybulec for acquainting them with Mizar and his continuous support since authors‘ students times, as well as for turning our attention to the formalization of topology.

A Mizar Notations Used in the Paper

× `[:X,Y:]` is the Cartesian product of sets X and Y ([1])
× `I[01]` is the interval [0,1] with standard topology ([21])
× `INT.Group` is the additive group of integers ([20])
× `HGrStr` is a structure of a group (equivalent of magma) ([22])
× `TopStruct` is a structure of a topological space ([17])
× `TopSpace` is a topological space ([17])
× `TOP-REAL n` is an `n`-dimensional Euclidean space ([2])
× `EqRel` is the relation `being_homotopic` between loops at a given point ([14])
× `Class` is the set of all equivalence classes of a given equivalence relation ([18])

References

1. Czesław Byliński. Some basic properties of sets. *Formalized Mathematics*, 1(**1**):47–53, 1990. MML Id: `ZFMISC_1`.
2. Agata Darmochwał. The Euclidean space. *Formalized Mathematics*, 2(**4**):599–603, 1991. MML Id: `EUCLID`.
3. David Gauld. Brouwer's Fixed Point Theorem and the Jordan Curve Theorem. `http://aitken.math.auckland.ac.nz/~gauld/750-05/section5.pdf`.
4. Adam Grabowski. Introduction to the homotopy theory. *Formalized Mathematics*, 6(**4**):449–454, 1997. MML Id: `BORSUK_2`.
5. Adam Grabowski and Artur Korniłowicz. Algebraic properties of homotopies. *Formalized Mathematics*, 12(**3**):251–260, 2004. MML Id: `BORSUK_6`.
6. Marvin J. Greenberg. *Lectures on Algebraic Topology.* W. A. Benjamin, Inc., 1973.
7. Allen Hatcher. *Algebraic Topology.* Cambridge University Press, 2002.
8. Yoshinori Isomichi. New concepts in the theory of topological space – supercondensed set, subcondensed set, and condensed set, *Pacific Journal of Mathematics,* 38(**3**):657–668, 1971.
9. Artur Korniłowicz. The fundamental group of convex subspaces of $\mathcal{E}^n_{\mathrm{T}}$. *Formalized Mathematics*, 12(**3**):295–299, 2004. MML Id: `TOPALG_2`.
10. Artur Korniłowicz. On the fundamental groups of products of topological spaces. *Formalized Mathematics*, 12(**3**):421–425, 2004. MML Id: `TOPALG_4`.
11. Artur Korniłowicz. On the isomorphism of fundamental groups. *Formalized Mathematics*, 12(**3**):391–396, 2004. MML Id: `TOPALG_3`.
12. Artur Korniłowicz. The fundamental group of the circle. *Formalized Mathematics*, 13(**2**):325–331, 2005. MML Id: `TOPALG_5`.
13. Artur Korniłowicz and Yasunari Shidama. Brouwer fixed point theorem for disks on the plane. *Formalized Mathematics*, 13(**2**):333–336, 2005. MML Id: `BROUWER`.
14. Artur Korniłowicz, Yasunari Shidama, and Adam Grabowski. The fundamental group. *Formalized Mathematics*, 12(**3**):261–268, 2004. MML Id: `TOPALG_1`.
15. Kazimierz Kuratowski. *Topology*, volume II. PWN – Polish Scientific Publishers, Academic Press, Warsaw, New York and London, 1968.
16. Roman Matuszewski and Piotr Rudnicki. MIZAR: the first 30 years, *Mechanized Mathematics and Its Applications,* 4(**1**):3–24, 2005, `http://mizar.org/people/romat/MatRud2005.pdf`.
17. Beata Padlewska and Agata Darmochwał. Topological spaces and continuous functions. *Formalized Mathematics*, 1(**1**):223–230, 1990. MML Id: `PRE_TOPC`.
18. Konrad Raczkowski and Paweł Sadowski. Equivalence relations and classes of abstraction. *Formalized Mathematics*, 1(**3**):441–444, 1990. MML Id: `EQREL_1`.
19. Zbigniew Semadeni and Antoni Wiweger. *Wstęp do teorii kategorii i funktorów*, volume 45 of *Biblioteka Matematyczna.* PWN, Warszawa, 1978.
20. Dariusz Surowik. Cyclic groups and some of their properties – part I. *Formalized Mathematics*, 2(**5**):623–627, 1991. MML Id: `GR_CY_1`.
21. Andrzej Trybulec. A Borsuk theorem on homotopy types, *Formalized Mathematics,* 2(4):535–545, 1991. MML Id: `BORSUK_1`.
22. Wojciech A. Trybulec. Groups. *Formalized Mathematics*, 1(**5**):821–827, 1990. MML Id: `GROUP_1`.

Evaluating Prospective Built-in Elements of Computer Algebra in MIZAR*

Adam Naumowicz

Institute of Computer Science
University of Białystok, Poland
adamn@mizar.org

Abstract. In this paper we evaluate the benefit of using MIZAR built-in routines devised to automatically deduce some inferences, called `requirements` in the MIZAR terminology. This data is compared with the potential impact of implementing several new `requirements` - almost entirely based on internal procedures already present in the MIZAR source code - namely: `Div`, `Mod`, `Divides`, `GCD`, `LCM`, `Power` and `Factorial`. Some technical aspects of the `requirements` technology are also presented in detail.

1 Introduction

The philosophy behind the MIZAR system has always been to develop a system useful for performing various aspects of mathematical practice in a computer environment (see [2]). This gave rise to a formal language readable for both humans and computers and a system for verifying correctness of reasoning steps. Although the system has been evolving for over 30 years now, there is a lot of space for improvement, especially in the realm of available automation of basic calculations, directed at approaching the very much desirable Poincaré principle (see [10]). The still-too-high value of the so-called De Bruijn factor ([8]) measured for more advanced formal encodings ([3]) indicates the need of further strengthening the power of "obviousness" within MIZAR's CHECKER. In this paper we present ongoing research in this direction by means of implementing internal MIZAR routines which help verify reasoning steps based on frequently used basic concepts - numerical computations, boolean operations on sets, etc.

We assume that the reader is familiar with the theoretical fundamentals of MIZAR's CHECKER as presented in [9] and recognizes the functions of its main modules: PRECHECKER, EQUALIZER and UNIFIER.

In Table 1 there are collected the numbers of calls to each of these modules issued during a complete VERIFIER run on the whole Mizar Mathematical Library (MML). These numbers show the size of the database, and on the other hand, should form a point of reference for the CHECKER's add-ons described in the sequel.

* This work has been partially supported by the FP6 IST project no. 510996 *Types for Proofs and Programs* (TYPES).

ISBN 978-83-7431-128-1 ISSN 0860–150X

Table 1. CHECKER statistics for entire MML (MIZAR ver. 7.8.03, MML ver. 4.76.959)

Module	Number of calls
PRECHECKER	763420
EQUALIZER	958525
UNIFIER	1101031

A few words of explanation are due here: the disjunctive normal form of formulae occurring in the justified inference, which is produced in PRECHECKER, usually consists of one or two disjuncts, so naturally the number of EQUALIZER's calls is greater than that of PRECHECKER's. However, one may ponder why the number of UNIFIER's calls is greater than EQUALIZER's, as there are inferences verified solely by EQUALIZER without running UNIFIER at all. Indeed, out of the total 958525 inferences passed into EQUALIZER, 154902 are verified there in which case UNIFIER is not invoked. But one must know that CHECKER may try to unify several universally quantified formulae, one by one, if more than one such formulae appear in an inference step and verification of previous ones fails. Another point here is that UNIFIER is also called to instantiate variables in formulae within Fraenkel terms in much the same way.

2 Requirements

The most effective technique of strengthening CHECKER's power has proved to be the identification of selected frequently used notions and implementing special internal routines which recognize the notions' constructors and offer extra processing according to their specific properties. Some form of special processing of numerical data was introduced in MIZAR as early as the beginning of 1980s in the implementation of MIZAR-2 ([2]).

More recently, many improvements have been added to automate as much as possible simple reasoning steps concerning operations on numbers and sets. There are now five sets of such internally-identifiable constructors - BOOLE, SUBSET, NUMERALS, ARITHM and REAL - MIZAR users can include these names in the environment directive `requirements` of their articles to benefit from their special processing. For a comprehensive account of these `requirements` directives one should refer to [4] or [5]. The most recent improvement, which these papers do not cover, is the new realization of the `requirements ARITHM` directive thoroughly reimplemented by C. Byliński. In this new implementation all terms equipped with the attribute `complex` are subject to decomposition with regard to basic arithmetic operations - it results in all equality classes in EQUALIZER having a polynomial representation where monomials are formed by "irreducible" `complex` terms - linear algebra methods are then used to find potential equalities between different classes.

Although the `requirements` are not a "regular" language construct, so users cannot create their own ones without reimplementing the MIZAR software, it is highly desirable that users knew more about this technique. Understanding how `requirements` work is important for all those who are interested in using MIZAR for various special purposes - to name just one such case, using MIZAR for educational purposes may require the ability to "turn on and off" selected requirements or to build a small local database from scratch, but be able to use MML `requirements` concerning numbers (such an example of MIZAR usage is presented in [6]). Table 2 shows current built-in `requirements` and their internal numbers (implemented as an enumerated type - if one of the internal numbers is assigned to an appropriate constructor, then the special processing associated with that number is performed for all instances of that constructor).

2.1 The structure of `requirements` files

The `requirements` are imported by users to their articles in the same manner as all other library items. The information is stored in the database as `*.dre` files. Thanks to the work of Josef Urban, all MIZAR library files are currently available in an XML form (see [7]). Their content is easy to parse and grammar is well documented - one can find the documentation of `requirements` files in the 'doc' folder within the MIZAR distribution, e.g. on Unix platforms the file `/usr/local/doc/mizar/xml/Mizar.html#Requirements`. Requirements may be consulted to see that the content of `requirements` files is the following:

```
%Signature;, ( %Requirement; )*
```

where `Signature` is a list of articles from which constructors should be imported and `Requirement` is a description of a constructor specially treated by the system. The description contains its internal number and optionally its name, the article's id (`aid`) and position in the article (`absnr`) - see Table 3 for a complete description.

2.2 Example - `requirements HIDDEN`

Here is the contents of the file `hidden.dre` which is automatically imported by the ACCOMMODATOR to properly associate the most basic MIZAR notions:

```
<?xml version="1.0"?>
<Requirements>
  <Signature>
    <ArticleID name="HIDDEN"/>
  </Signature>
  <Requirement constrkind="M" constrnr="1" nr="1"/>
  <Requirement constrkind="R" constrnr="1" nr="2"/>
  <Requirement constrkind="R" constrnr="2" nr="3"/>
</Requirements>
```

Table 2. Currently implemented `requirements` (MIZAR ver. 7.8.03, MML ver. 4.76.959)

No.	Requirements file	MML definition	Short description
0	-	-	internal variable - not associated with any constructor
1	`hidden.dre`	`HIDDEN(:def 1)`	automatically set to the first mode constructor[a]
2	`hidden.dre`	`HIDDEN(:def 2)`	the absolute equality relation
3	`hidden.dre`	`HIDDEN(:def 3)`	the belonging relation
4	`boole.dre`	`XBOOLE_0:def 5`	the attribute `empty`
5	`boole.dre`	`XBOOLE_0:def 1`	the empty set
6	`subset.dre`	`SUBSET_1:def 2`	element of a set
7	`subset.dre`	`ZFMISC_1:def 1`	the power set
8	`subset.dre`	`TARSKI:def 3`	set-theoretic inclusion
9	`subset.dre`	`SUBSET_1(redef.)`	element of a subset of a set
10	-	-	the set of real numbers[b]
11	-	-	the set of natural numbers[c]
12	`arithm.dre`	`XCMPLX_0:def 4`	the addition of complex numbers
13	`arithm.dre`	`XCMPLX_0:def 5`	the multiplication of complex numbers
14	`real.dre`	`XXREAL_0:def 5`	the <= relation on real numbers
15	`numerals.dre`	`ORDINAL_1:def 1`	the successor of a natural number
16	`boole.dre`	`XBOOLE_0:def 2`	the union of two sets
17	`boole.dre`	`XBOOLE_0:def 3`	the intersection of two sets
18	`boole.dre`	`XBOOLE_0:def 4`	the difference of two sets
19	`boole.dre`	`XBOOLE_0:def 6`	the symmetric difference of two sets
20	`boole.dre`	`XBOOLE_0:def 7`	true if two sets have an empty intersection[d]
21	`arithm.dre`	`XCMPLX_0:def 6`	the negated complex number
22	`arithm.dre`	`XCMPLX_0:def 7`	the inverse complex number
23	`arithm.dre`	`XCMPLX_0:def 8`	the difference of complex numbers
24	`arithm.dre`	`XCMPLX_0:def 9`	the division of complex numbers
25	-	-	the attribute `real`[e]
26	`real.dre`	`XXREAL_0:def 6`	the attribute `positive`
27	`real.dre`	`XXREAL_0:def 7`	the attribute `negative`
28	-	-	the attribute `natural`[f]
29	`arithm.dre`	`XCMPLX_0:def 1`	the imaginary unit
30	`arithm.dre`	`XCMPLX_0:def 2`	the attribute `complex`
31	`numerals.dre`	`ORDINAL_1:def 12`	the set of natural numbers

[a] `set` by default.

[b] Implementation discontinued.

[c] Implementation discontinued, see No. 31.

[d] Implementation postponed.

[e] Implementation discontinued.

[f] Implementation discontinued.

Table 3. The grammar of `*.dre` files (MIZAR ver. 7.8.03)

Attribute	Data type	Use
constrkind	Enumeration: "M" | "L" | "V" | "R" | "K" | "U" | "G"	Obligatory
constrnr	xsd:integer	Obligatory
nr	xsd:integer	Obligatory
reqname	xsd:string	Optional
absnr	xsd:integer	Optional
aid	xsd:string	Optional

2.3 Assessment of `requirements` usage

Several methods could be used to assess the usefulness of specific `requirements` for the authors of new articles and the development of MML. First, the available debug information (from CHECKER's reports) could be used since all relevant occurrences of the `requirements` enumeration type are marked in the MIZAR source code. This, however, may not be enough to state if a given action really resulted in accepting otherwise unacceptable inferences, e.g. joining two equality classes may happen in some "later" part of the EQUALIZER code.

With the access to the MIZAR source code, one could also recompile the ACCOMMODATOR program (`accom`) to disable specific `requirements` by preventing their numbers to be printed into the `*.ere` file which is later used by the VERIFIER. Or similarly, one could post-edit the `*.ere` file produced by the ACCOMMODATOR, which is a sequence of numbers representing constructor numbers associated with subsequent `requirements` numbers - both this solutions are not very "clean".

Much more "elegant" approach seems to be the use of standard MIZAR technique of building local environment - knowing the structure of `requirements` files, one may prepare a local version of a relevant *.dre file with specific lines removed that correspond to a given constructor. This method is easily reproducible by any user and requires no special tools. This way all MML `requirements` can be tested in two steps:

1. one `requirement` disabled each time,
2. full library check with standard `verifier`.

The results of such a test are collected in Table 4. Here is some explanation for the notions measured in this test:

- Possible applications (number of articles that follow the article which introduces `requirements` constructors according to the `mml.lar`[1] ordering, excluding the "documentation" articles[2]

[1] More information to be found in the archives of the MIZAR-Forum mailing list, e.g. http://mizar.uwb.edu.pl/forum/archive/0609/msg00076.html

[2] These are articles accompanying every requirements file, e.g. `boole.miz`, that contain proves of built-in facts - kept in MML only to help preserve the system's consistency.

- Dependent articles (number of articles that indeed rely on specific `requirements`, i.e. there were errors reported in the test)
- Total number of errors
- Average number of errors (calculated for all articles that depend on specific `requirements`)

Table 4. `requirements` usage statistics (MIZAR ver. 7.8.03, MML ver. 4.76.959)

No.	Possible applications	Dependent articles	Total no. of errors	Avg. no. of errors
1	958	958	0[a]	0
2	958	952	343176	360.48
3	958	895	92322	103.15
4	956	879	43080	49.01
5	956	539	3994	7.41
6	951	913	372186	407.65
7	951	857	50745	59.22
8	951	773	20025	25.90
9	951	663	299850	452.26
12	920	564	34051	60.37
13	920	382	14081	36.86
14	917	512	19735	38.54
15	933	20	45	2.25
16	956	169	858	5.07
17	956	112	392	3.50
18	956	86	186	2.16
19	956	1	5	5.00
21	920	315	4770	15.14
22	920	30	187	6.23
23	920	463	16350	35.31
24	920	191	3215	16.83
26	917	377	8463	22.45
27	917	367	7613	20.74
29	920	11	64	5.82
30	920	545	19681	36.11
31	933	668	300488	449.83

[a] In the current implementation, this `requirement` is preset in the code and cannot be turned off without recompiling the ACCOMMODATOR - the MIZAR type system depends on the assumption that the type created with the `mode` constructor number 1 is the "widest" type.

3 New `requirements` proposals

Obviously, the currently implemented set of `requirements` by no means covers the needs of MIZAR users. There are still many operations which users would like to have automatically calculated for them by the system. Below we present a selection of potentially desirable built-in constructors together with assessment of their usefulness based on a rudimentary implementation. Worth noticing is the fact that most of the code required by the implementation has already been present in MIZAR sources as part of `requirements ARITHM` supporting the arithmetic of arbitrarily long integers. Namely, we present the implementation of the following numerical functions: `Div`, `Mod`, `Divides`, `GCD`, `LCM`, `Power` and `Factorial`.

3.1 Usage assessment for new `requirements`

Checking which MML inferences are accepted thanks to new `requirements` requires the whole database to be "clean" with respect to the following utilities reporting unnecessary items in reasoning:

- `relprem` (reports unnecessary premises)
- `relinfer` (reports unnecessary inference steps)
- `reliters` (reports unnecessary inference steps within iterative equalities)

The MIZAR toolbox contains utilities that allow to automatically eliminate all errors reported by `relprem` and `reliters`. At the moment, there is no working program available to do the same with `relinfer`'s errors - in this case the test results have been "simulated" by subtracting the number of errors reported before using the new `requirements` from and number obtained while using them. All tests have been carried out on an Intel(R) Pentium(R) 4 (3.00GHz, 2GB RAM) workstation running 2.6.16.21-0.25-smp i386 GNU/Linux. In such an environment verification of the whole MML with `verifier` currently takes about two hours, while a complete run of `relprem`, `relinfer` and `relites` takes about four, three, and one and a half hours, respectively.

3.2 `requirements NAT_D`

The article `NAT_D` appeared in the MML as a result of a revision aimed at better organization of the various divisibility-oriented notions - it contains the redefinitions for natural numbers, whereas the original constructors are introduced in the article `INT_1`. Table 5 summarizes the constructors used.

The implementation of `Div`, `Mod`, `GCD`, and `LCM` is rather straightforward - the value is calculated by an appropriate function for all instances of terms with a given constructor and arguments having a numerical value in the EQUALIZER - as a result some equality classes may become equal or a contradiction may be found. The `Divides` code is slightly more complicated - in the EQUALIZER we must check all instances of positively and negatively valuated formulae built with the `Divides` constructor and calculate if they agree with the numerical values of

Table 5. requirements NAT_D (MML ver. 4.76.959)

No.	Requirements file	MML definition	Short description
32	nat_d.dre	INT_1:def 7	the integer division
33	nat_d.dre	INT_1:def 8	the remainder of integer division
34	nat_d.dre	NAT_D:def 4	the least common multiple of two natural numbers
35	nat_d.dre	NAT_D:def 5	the greatest common divisor of two natural numbers
36	nat_d.dre	INT_1:def 9	true if and only if one integer divides another

their arguments, but also provide some code in the `Unifier` which tries to pattern-match free variables in appropriate formulae with a contradictory substitution (as described in [9]).
Conforming to the grammar of `*.dre` files presented in Section 2.1 we encoded the file `nat_d.dre` as follows:

```
<?xml version="1.0"?>
<Requirements>
  <Signature>
    <ArticleID name="HIDDEN"/>
    <ArticleID name="INT_1"/>
    <ArticleID name="NAT_D"/>
  </Signature>
  <Requirement constrkind="K" constrnr="5" nr="32"/>
  <Requirement constrkind="K" constrnr="6" nr="33"/>
  <Requirement constrkind="K" constrnr="11" nr="34"/>
  <Requirement constrkind="K" constrnr="13" nr="35"/>
  <Requirement constrkind="R" constrnr="4" nr="36"/>
</Requirements>
```

Below are the results of testing the usage of each of these `requirements`:

Table 6. Number of "irrelevant" items reported by library utilities (MML ver. 4.76.959)

No.	relprem	relinfer	reliters	Total:
32	59 (16 files)	19 (7 files)	8 (5 files)	43
33	224 (15 files)	76 (6 files)	40 (5 files)	340
34	4 (1 file)	0	0	4
35	54 (4 files)	0	45 (1 file)	99
36	201 (9 files)	115 (8 files)	0	316

3.3 `requirements NEWTON`

The current MIZAR code does not contain functions computing the value of `Power` and `Factorial`. However, their implementation is trivial in presence of multiplication for arbitrarily long integers, so can easily be added. Then, the CHECKER implementation of these `requirements` resembles that of e.g. `Div` or `Mod`. The associated constructors are listed in the table below.

Table 7. `requirements NEWTON` (MML ver. 4.76.959)

No.	Requirements file	MML definition	Short description
37	newton.dre	NEWTON:def 1	the exponentiation (complex base, natural exponent)
38	newton.dre	NEWTON:def 2	the factorial of a natural number

A corresponding `newton.dre` file should have the following form:

```
<?xml version="1.0"?>
<Requirements>
  <Signature>
    <ArticleID name="HIDDEN"/>
    <ArticleID name="NEWTON"/>
  </Signature>
  <Requirement constrkind="K" constrnr="2" nr="37"/>
  <Requirement constrkind="K" constrnr="5" nr="38"/>
</Requirements>
```

Finally, Table 8 presents the results of checking the usefulness of implementing special processing for `Power` and `Factorial`:

Table 8. Number of "irrelevant" items reported by library utilities (MML ver. 4.76.959)

No.	relprem	relinfer	reliters	Total:
37	319 (33 files)	68 (19 files)	239 (26 files)	626
38	102 (14 files)	0	38 (8 files)	140

This implementation caused some performance problems - apparently there are inferences in the MML in which quite large exponents are used (e.g. 3^{32768}, i.e. 15635 digits!) - normally the inference gets justified with a universal premise, but with a `requirement` turned on for the `Power` function, the verification takes quite a lot of time. The same may as well concern the `Factorial` implementation.

4 Conclusions

After performing the tests of usability for the prospective `requirements`, it seems that they could be a valuable add-on for the MIZAR VERIFIER in the process of approaching the Poincaré principle. Some of them, however, must be introduced carefully, as there may occur performance issues - in these cases the principle should probably be treated as "a dynamic principle which puts it in the hands of a human user whether to expand an argument in different ways" in the spirit proposed in [1]. The application of optimized libraries or external programs could be another solution to these problems.

References

1. Kerber, M., A Dynamic Poincaré Principle, In J. M. Borwein and W. M. Farmer (Eds.), MKM 2006, LNAI 4108, pp. 44–53, 2006.
2. Matuszewski, R. and P. Rudnicki, MIZAR: the First 30 Years, Mechanized Mathematics and Its Applications, Vol. 4 (1), pp. 3–24, 2005, http://mizar.org/people/romat/MatRud2005.pdf.
3. Naumowicz, A., An Example of Formalizing Recent Mathematical Results in MIZAR. In C. Benzmüller (Ed.), Towards Computer Aided Mathematics, Journal of Applied Logic, 4(4), pp. 396–413, Elsevier, 2006.
4. Naumowicz, A. and C. Byliński, Basic Elements of Computer Algebra in MIZAR, Mechanized Mathematics and Its Applications, Vol. 2, pp. 9–16, 2002.
5. Naumowicz, A. and C. Byliński, Improving Mizar Texts with Properties and Requirements, In A. Asperti et al. (Eds.), MKM 2004, LNCS 3119, pp. 290–301, 2004.
6. Retel, K. and A. Zalewska, Mizar as a Tool for Teaching Mathematics, Mechanized Mathematics and Its Applications, Vol. 4(1), pp. 35–42, 2005.
7. Urban, J., XML-izing Mizar: Making Semantic Processing and Presentation of MML Easy, In M. Kohlhase (Ed.), MKM 2005, LNAI 3863, pp. 346–360, 2006.
8. Wiedijk, F., The De Bruijn Factor, http://www.cs.ru.nl/~freek/factor/factor.pdf .
9. Wiedijk, F., Checker, http://www.cs.ru.nl/~freek/mizar/by.pdf .
10. Wiedijk, F. (ed.), The Seventeen Provers of the World (foreword by Dana S. Scott), LNAI 3600, 2006.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 10 (23) 2007

Proving the Correctness of Functional Programs using Mizar

Yatsuka Nakamura

Shinshu University
Faculty of Engineering
Information Engineering
380-8553 Nagano-city, Japan
ynakamur@cs.shinshu-u.ac.jp

Abstract. A method for proving the correctness of functional programs (e.g., Haskell 98 [1]) using the Mizar system is introduced. The functions in a functional program are translated to Mizar functions and the proofs of existence and uniqueness of Mizar functions are done semi-automatically using schemes in Mizar. Types of variables are also reduced to Mizar types. It is also shown how to design concrete Haskell programs using Mizar. Some problems remain in completing such a method, but they can be overcome and are also discussed here.

1 Introduction

Until now, the designing methods of programs have been a kind of art, not a science. However, as software programs become more and more huge, the work of removing bugs becomes very difficult. Some theoretical method for developing software is necessary.

Recently, people have become aware of the importance of functional programs like Haskell [1]. The reason for this is that we can resolve huge programs into basic and simpler functions with functional programs.

Functions of a functional program are similar to mathematical functions so it may be possible to prove the correctness of each function of a functional program.

If one uses only pencil and paper to try to prove the correctness of a program, it would not be a reliable proof because the level of accuracy is different between computer programs and usual handwritten mathematics.

Fortunately, we now have very strong tools called proof checkers to do mathematics in a rigid way. One of the most popular proof checkers is Mizar, which was originated by Dr. A. Trybulec and his group.

The extent of the library of a proof checker is important when it is used to prove the correctness of programs because the semantics of programs relates to mathematical models in the library.

Fortunately, the Mizar Mathematical Library (MML) is currently the largest [4]. It contains nearly 1,000 articles and approximately 32,000 lemmas and theorems.

ISBN 978-83-7431-128-1 ISSN 0860–150X

In the Mizar library, there are some attempts to prove the correctness of programs. One is the "PRGCOR" series [2] by the author where the concept of "non overwriting" is introduced and the correctness of some C programs written in non overwriting style can be proven.

Most functional programs are essentially "non destructive" in the sense that all values in variables are not destroyed by overwriting. So, the "non destructive" property is the same as our concept of "non overwriting". If we restrict programs to functional ones, the discussion becomes very simple.

The description of functions of 1st order logic in Haskell is very near to the definition of functors in Mizar. For example, the "max" function in Haskell is written as

```
let max a b = if a>=b then a else b
```

On the other hand, the max functor in Mizar is written as

```
definition let a,b;
func max(a,b) equals
::XXREAL_0:def 9
 a if b <= a otherwise b;
```

So, with a little modification of sentences, we can convert the definitions of Mizar functors into Haskell functions.

With the completion of such a correspondence, we have the following possibility: The names of functions which are installed to Haskell are given as

```
    $H_max
```

in Mizar. If the same function is already defined in Mizar, the description of a synonym will be enough, as follows.

```
   notation let x,y be Element of REAL;
    synonym $H_max(x,y) for max(x,y);
end;
```

Basic functions in Haskell (functions in PRELUDE) are defined in a Mizar article. New programs which a programmer wants to write in Haskell form can be written in Mizar. For example, from the following Mizar definition

```
definition let a,b,c;
func $H_max3(a,b,c) equals
:AA: $H_max($H_max(a,b),c);
```

we can derive the corresponding program through some automated method as

```
let max3 a b c = max(max a b) c
```

As a Mizar theorem, we know

```
theorem :: XXREAL_0:34
 max(max(a,b),c) = max(a,max(b,c));
```

Hence we can see that for a new Haskell program

```
max3 a b c = max a (max b c)
```

is valid.

With only a number of theorems about the new function, its correctness is guaranteed. The function can be defined without contradiction in Mizar, but whether or not the definition is intuitively proper is another problem.

More advanced Haskell programs can also be written using $H_ form functions in Mizar. If all variables in the right hand side of a formula in the definition of a new functor are $H_ form functions, then it can be converted to a Haskell function.

When a new Mizar article is verified by the checker, there is a step of "accommodation", where the checker gathers all previous articles used in the new article. By the same method, one can gather all necessary old definitions of $H_ form functors used to define a new function and one can get the necessary library file to run it.

In such a situation, the Mizar library containing $H_ form functors will become a very good manual for describing their use because in the definitions the type of arguments are described and the conditions for using the functor are also described by the clause of "assume". The theorems about such a functor show the conditions for using it as well. For example, the functor to get the roots of an algebraic equation gives the correct answer only if the coefficient of the equation is not zero.

By changing all operations in Haskell to a function form, for example

```
(<=) a b,
```

instead of

```
a <= b,
```

the correspondence of program functions and functors will become very simple.

But, the situation is more difficult because there are additional important capabilities for functional program languages like Haskell which are difficult to correspond to Mizar functors.

One of them is the treatment of recurrent descriptions like

```
let sum0 i = if ((<=)i 0) then 0 else (+) i (sum0 ((-)i 1))
```

There is no corresponding way of making such a description for Mizar functors. One must use Mizar functions in such a case. Definitions of Mizar functions are rather difficult as compared to the case of functors.

There are more important cases where one must use functions.

2 Mizar Functions and Mizar Functors

In the definition of a Mizar function, one must prove the existence and uniqueness of the function itself, not for a value of the function. The following functor "$H_double" is a simple functor introduced as an example.

```
definition let x be Element of REAL;
::PROG. Example.
func $H_double x equals
:BB: $H_(*) (x,2);
correctness;
end;
```

The Mizar verifier accepts the above definition without proof, where we assume the function $H_(*) is already introduced. From the above definition we can automatically get a Haskell program as follows (parts following a :: sign are comments in Mizar. A "PROG." comment means that this definition corresponds to a Haskell program):

```
let double x = (*) x 2
```

We can also define $H_double as a function in Mizar as follows:

```
definition
func $H_double -> Function of REAL,REAL means
:CC: for x being Element of REAL holds
     it.x = $H_(*) (x,2);
existence;
::>     *4
uniqueness;
::>      *4
end;
```

Two *4 comments were imposed on the article when it was checked by the Mizar verifier. These comments indicate incorrect lines of induction. We must prove the existence of the $H_double function, not the existence of the value x*2 which is easier. About 50 lines are necessary to prove the existence and uniqueness of the $H_double function.

The function "double" is the simplest one, but to define it as a function is not easy in Mizar.

Let us return to the recurrent program example shown in the previous section. The Haskell program was:

```
let sum0 i = if ((<=)i 0) then 0 else (+) i (sum0 ((-)i 1))        (1)
```

As stated already, the Haskell function sum0 above cannot be represented by a Mizar functor. As a function of Mizar however this can be defined. To show the definition, we give two definitions of functors as follows:

```
definition let D be set;let x be Element of BOOLEAN,
 z,u be Element of D;
::PROG. Basic
func $H_if_then_else(x,z,u) -> Element of D equals
:AE508d: z if (x=$H_True) otherwise u;
correctness;
end;

definition let n,m be Element of NAT;
::PROG.
func $H_nsub(n,m) -> Element of NAT equals
:AE508e: $H_max($H_(-)(n,m),0);
correctness
proof
 per cases;
 suppose $H_(-)(n,m)<=0;
  hence $H_max($H_(-)(n,m),0) is Element of NAT
   by XXREAL_0:def 9;
 end;
 suppose B0: $H_(-)(n,m)>0;then
   B1: $H_(-)(n,m)=n-m & n-m>0 by AA173;then
   B2: $H_max($H_(-)(n,m),0)=n-m by XXREAL_0:def 9;
   n-m=n-'m by BINARITH:def 3,B1;
  hence $H_max($H_(-)(n,m),0) is Element of NAT by B2;
 end;
end;
end;
```

The second definition of $H_nsub has a proof of correctness. Usually, the proof of correctness of a functor is simple, but sometimes a proof of 10 or 20 lines is necessary.

Using the above two functors, we can give a definition of a $H_sum0 functor as:

```
definition
::PROG. Summing from 0 to l
func $H_sum0 -> Function of NAT, NAT means
:AE508f: for l being Element of NAT holds
 it.l=$H_if_then_else(($H_(<=)(l, 0)), 0, $H_(+)(l,
  it.($H_nsub(l,1))));
existence;
::>     *4
uniqueness;
::>     *4
end;
```

The proof of existence and uniqueness are rather complicated and will be discussed in the next section.

We obtain the Haskell program by extracting the last formula as:

```
let sum0 l = if_then_else ((<=) l 0) 0 ((+)l (sum0 (nsub l 1)))
```

This is slightly different from formula (1) on the page 204, but it is a more theoretical formula.

3 Schemes in Mizar to Prove the Correctness of Functions

This section is a technical discussion on how to prove the correctness of Mizar functions. As stated in Section 1, proving correctness (existence + uniqueness) for functions is more difficult than for functors. It is rather doubtful that the proofs of existence and uniqueness are necessary in actual application levels because existence is clear since the functions are really calculated and uniqueness is no concern for people who use them.

But to prove that the functions in a program are the same as mathematical functions, we must do the proofs of correctness.

The proof of correctness for a function which is essentially a functor is done semi-automatically using a special scheme. A scheme is a special device in Mizar used to prove formulas of higher order logics.

Let us consider the following scheme:

```
scheme Lambdax{X, Y() -> non empty set, F(set)->set}:
 ex f being Function of X(),Y() st (for x being Element of X()
 holds f.x = F(x));
```

The proof of this scheme is not difficult and can be done by using the fundamental scheme in the article of functions (FUNCT_2:sch 1 in MML). Some examples of proofs of existence of functions using the above scheme are shown below:

```
definition
func $H_succ -> Function of REAL,REAL means
:AA509: for n being Element of REAL holds it.n=n+1;
existence
proof
  deffunc F(Element of REAL)=$1+1;
  A1: for x being Element of REAL holds F(x) is Element of REAL;
  consider f being Function of REAL,REAL such that
  A2: (for x being Element of REAL
    holds f.x = F(x)) from Lambdax(A1);
 thus thesis by A2;
end;
uniqueness;
end;
```

This is a definition of the "succ" function in Haskell.

```
definition
func $H_abs -> Function of REAL,REAL means
:AA176: for x being Element of REAL holds it.x= |. x .|;
existence
proof
  deffunc F(Element of REAL)= |. $1 .|;
  A1: for x being Element of REAL
       holds F(x) is Element of REAL;
  consider f being Function of REAL,REAL such that
  A2: (for x being Element of REAL
    holds f.x = F(x)) from Lambdax(A1);
  thus thesis by A2;
end;
uniqueness;
end;
```

By comparing the two proofs above, the lines are almost the same except for the first line of defining the function F. So, the proof is semi–automatic. Similarly, uniqueness is also proved easily using the following scheme.

```
scheme Lambdau{X, Y() -> non empty set, F(set)->set}:
 for f1,f2 being Function of X(),Y() st (for x being
  Element of X()
 holds f1.x = F(x))&(for x being Element of X()
 holds f2.x = F(x)) holds f1=f2;
```

For recursive functions, existence and uniqueness are also proved easily by the following two schemes.

```
scheme Lambdas{Y() -> non empty set,Z() -> Element of Y(),
 G(Element of NAT,Element of Y())->Element of Y()}:
 ex f being Function of NAT,Y() st (for x being Element of NAT
 holds f.x=$H_if_then_else($H_(<=)(x,0),Z(),
  G(x,f.($H_nsub(x,1)))));
```

This scheme is a variation of an old scheme named "LambdaRecExD". The following is used for proofs of recurrent functions.

```
scheme Lambdasu{Y() -> non empty set,Z() -> Element of Y(),
 G(Element of NAT,Element of Y())->Element of Y()}:
 for f1,f2 being Function of NAT,Y() st (for x being
   Element of NAT
 holds f1.x=$H_if_then_else($H_(<=)(x,0),Z(),
   G(x,f1.($H_nsub(x,1)))))&
 (for x being Element of NAT
  holds f2.x=$H_if_then_else($H_(<=)(x,0),Z(),
    G(x,f2.($H_nsub(x,1)))))
  holds f1=f2;
```

4 Types and Classes

There are many classes and types in Haskell. We can translate them to Mizar. There are multiple meanings of "correctness" for programs. One of them is that correctness means that a program runs without overflow (and underflow) errors.

Here, we are concerned with semantic correctness, which means that the program achieves some calculation on a proper mathematical model.

If we are not interested in the size of numbers, the following types of Haskell need not be distinguished:

```
Int,    Integer.
```

These two types correspond to the following type of Mizar:

```
Element of INT
```

Also, the types of Haskell

```
Float and Double
```

correspond to Mizar type

```
Element of REAL,
```

and

```
Bool
```

corresponds to Mizar type

```
Element of BOOLEAN.
```

User defined types in Haskell are also defined in Mizar. In both the cases of basic and user defined types, in Mizar we use the form

```
Element of X,
```

where X is a non empty set. If we restrict the type to such a form, we can define a function of the form

```
Function of X,Y
```

easily.

If we put the definition as

```
definition let D,E be non empty set;
let f be Function of D,E;let x be Element of D;
redefine func f.x -> Element of E;
coherence by FUNCT_2:7;
end;
```

the value of a function from D to E is automatically interpreted as being an Element of E, so that functions work as if they were functors.

5 Functions of More than Two Arguments

Functions of more than two arguments in Haskell correspond to functions of the following:

```
    (<=) x y

definition
func $H_(<=) -> Function of REAL,Funcs(REAL,BOOLEAN) means
:AA200: for x being Element of REAL holds
 for y being Element of REAL holds
  (x<=y implies (it.x).y =$H_True)&
  (x>y implies (it.x).y =$H_True);
existence;
uniqueness;
end;
```

If we fix x to be an element of REAL, `$H_(<=).x` becomes a Function of REAL,BOOLEAN. This relates to the following Haskell program ([1,2,3,4,5] is a list which will be discussed in the next section):

```
Prelude> map ((<=) 2) [1,2,3,4,5]
[False,True,True,True,True]
```

The function "map" claims a function as its first argument. The term ((¡=) 2) represents a function from Num to Bool.

6 Modeling Pairs and Lists

The notion of pairs in Haskell is almost the same as the notion of pairs in Mizar, where a pair of x and y is written as [x,y] (it is written as (x,y) in Haskell). There is no problem in making a correspondence between them. But, the "fst" function and "snd" function give one problem when we want to translate them to Mizar.

What is the type of a pair [x,y]? We can choose any data of any type for x and y. Then, we must consider a set of all sets Ω as the range of "fst", which is impossible because the existence of such a set causes Cantor's paradox. We must choose a set smaller than Ω. As a Mizar functor, "fst" can be defined easily, because the functors z'1 and z'2 already exist in Mizar library. The following is clear:

```
$H_fst z = z'1
```

and

```
$H_snd z=z'2
```

But to define "fst" and "snd" as functions of Mizar, some other device will be necessary. We do not discuss this further here.

There if another concept called "List" in Haskell, like

```
[1,2,3,4,5],
```

which is similar to a finite sequence of D (D is a set) in Mizar. This can be represented as

```
<* 1,2,3,4,5*>,
```

which is a function from {1,2,3,4,5} to D.

We can also use the concept of "XFinSequence of D" in Mizar [3], which is a function from {0,1,2,3,4} to D. With the Haskell function (!!), the n-th location is shown as

```
(!!) 0 [1,2,3,4,5] = 1.
```

So, it seems more natural to use XfinSequence, but the theory for FinSequence is well developed in Mizar.

There is another form of list like:

```
[1..]
```

or

```
[1,3..].
```

Both are infinite sequences which have infinitely many elements. Hence, we must construct a new type "LIST of D" in Mizar.

7 λ–Calculus

There is no corresponding concept of λ–calculus in Mizar. The function λ–calculus in Haskell is as follows:

```
Prelude> (\x -> (+) 2 ((*)x x)) 5
27
```

In the above, the "`(\x -> (+) 2 ((*)x x))`" part acts as a function of one variable (x=5 here. The value is calculated as 2+x*x). If we divide it into two lines, it is not necessary to use λ–calculus.

```
Prelude> let ll2 x=(+)2 ((*)x x)
Prelude> ll2 5
27
```

Here "ll2" is a temporarily introduced function. If we can substitute ll2 in the second line with the first line, then these two lines are mathematically the same as λ–calculus.

So, there is no essential difficulty for returning λ–calculus to Mizar formulas.

References

1. Johns, S.P (ed), *Haskell 98 Language and Libraries: the Revised Report*, Cambridge University Press, (2003).
2. Nakamura, Y., *Correctness of Non Overwriting Programs. Part I*, Formalized Mathematics, Vol. 12, No. 1 (2004), pp. 29–32.
3. Tsunetou, T., Bancerek, G. and Nakamura, Y., *Zero-Based Finite Sequences*, Formalized Mathematics, Vol. 9, No. 4 (2001), pp. 825–829.
4. Wiedijk, F., *Comparing Mathematical Provers*, Mathematical Knowledge Management: Second International Conference, MKM 2003, Bertinoro, Italy, February 16-18, 2003. Proceedings, Springer Berlin / Heidelberg (2004), pp. 188–202.

A Verification for Redundant Signed Digit Adder Circuits

Katsumi Wasaki

Faculty of Engineering
Shinshu University
Nagano, Japan
wasaki@cs.shinshu-u.ac.jp

Abstract. Using calculation models proposed for arithmetic logic units based on many sorted algebras, we continue to verify the structure and design of these circuits using the Mizar proof checking system. The stability of a circuit for a redundant signed digit adder circuit example is proved based on definitions and theorems for logic operations, hardware gates, and signal lines. For this purpose, we use the Mizar proof checking system as a formal verification tool. The motivation for this research is to establish a technique based on formalized mathematics for developing calculation circuits with high reliability.

1 Introduction

When a digital circuit is designed, the most important consideration is the problem of 'Logic' and 'Timing'. In past research on circuit design verification, these two problems are often conflated. The problem becomes highly complex in real circuits when the problems are analyzed at the same time and the range of circuits which can be examined is limited. Consequently, the discussions normally concentrate on small-scale circuits [1][2][3][4][5].

Design verification using the simulation approach has been examined on real-sized circuits [6]; however, difficulties arise when attempting to investigate all of the circuit element values using different delay times (greatest, least and average). Therefore, in the verification method for gate array design, the delay times of circuit elements and wiring are assumed to be zero. The method here simulates the stage of development (zero-delay simulation) above and a technique exists for shortening the time needed for design verification. However, large differences occur when the real gate array is finally laid out. One report [7] states that simulation can easily assume delay information to be attached at the end. Dividing such problems into 'Logic' and 'Timing' categories is thought to be an effective solution. The signal propagation model and a concept of "Timing belts" to solve the 'Timing' problems have already been proposed as techniques for verifying the timing of these circuits [8][9][10].

We now introduce and examine various concepts that have been produced to prove the correctness of design verification and circuit operation.

ISBN 978-83-7431-128-1 ISSN 0860–150X

A many sorted algebra is an *algebra* for handling multiple elements, or *sorts*. We use this to map everything from the operators mapping the state space, to the I/O signals showing all signal points in the circuit, to the operator mapping the signal point, to the input signal point necessary for operation as a structure of the circuit [11][12][13][14].

We are able to treat the definition of circuits in various spaces such as in high impedance signal states and in continuous spaces. Calculation elements with single input operators (one gate elements) are defined [15][16]. With this definition, important properties can be proved such as when stability is met or when computation results are uniquely decided. Next, combined calculation circuits are defined [17][18]. Based on combined circuits defined as conjunctive sets for each functor on many sorted algebraic structures, we are able to prove properties such as the uniqueness and stability of calculation results of combined circuits [20].

To evaluate the method on a real-size calculation circuit, we prove the properties of the adder circuit with carry-saved [22] data flows using Redundant Signed Digit (RSD) [23][24] numeric representation as an example. Basic operations in the simple binary representation are proved [10]. Various concepts for Boolean operations, the gate elements needed to define the digital circuit, and the connections are defined and proved [18]. For gate elements that compose a calculation circuit using many Boolean operations, we have prepared a collection of logic gates [19]. To construct the adder circuit structure for RSD numeric representation, we formalize the definition of the Generalized Full Adder Circuit(GFAC) [22] with three inputs and two outputs [21].

The proof is as follows. The calculation circuit in RSD representation for a 1-bit adder is defined by a structure which combines two generalized full adder circuits. It is possible to define a two-stage pipelined circuit that combines the calculation elements. For combined logic circuits, we prove theorems on properties of their input/output signals. Finally, the signal sets of calculation results are always stable after a finite number of calculation steps (i.e., following) if the state of the input signal points is fixed. Since the 'combined' RSD adder circuit is a composition of two-stage pipelined dataflows, the stability properties of the whole circuit are proved in four steps. To create an extension circuit for the remainder of the calculation to output the carry signal in RSD representation to other circuit units and count with the saved-carry signals, we construct a simple 'AND gate' circuit; thus, the final adder result is obtained. The definitions and theorems in this paper have been verified for correctness using the Mizar proof checking system [25][26][27].

2 Preliminaries of Circuits

This section introduces the mathematical concepts used by Trybulec and Nakamura, et al. [11][12][13][14][15][16][17] for the structure, operations, input/output signals, combinations of logical gates, and states in digital circuits. Bancerek, et al. [18] introduced Boolean operators to represent the interconnections of individual logical gates and gate elements and showed the feasibility of the synthesis of such elements. These definitions and theorems have already been verified with correctness proofs by using the Mizar proof checker.

2.1 Structure of Circuits

A structured-type called *Many Sorted Signature* has been introduced to represent circuit structures [11][12]. This is a quadruple of *carrier, operation symbols, arity, and result sort* . These correspond to the internal states including the operators and input/output signals of the circuit elements and the states of input and output signals when applied to circuit elements (Fig. 1).

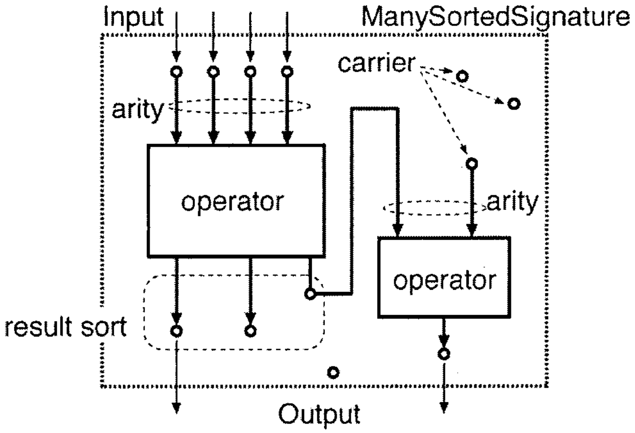


Fig. 1. Circuit structure represented with a *many sorted signature.*

Definition 1. *Many Sorted Signatures*

We introduce many sorted signatures, which are extensions of 1-sorted structures and are systems

$$\ll \textit{carrier, operation symbols, arity, result sort} \gg$$

where the carrier is a set, the operation symbols constitute a set, the arity is a function from the operation symbols into the carrier, and the result sort is a function from the operation symbols into the carrier.*

2.2 Calculation Circuits

A functor called $1GateCircStr(p,f)$ is defined for a circuit with only one operator (or operation symbol) [17]. This is a *non-empty Many Sorted Signature* in the internal state (*carrier*) including an input signal ($rng(p)$), with one tuple of operator (f) and input signal(p). The input state of the operator (*arity*) is equal to $rng(p)$ and the output result sort equals $\langle p, f\rangle$; thus, the output is uniquely decided.

Definition 2. *One gate circuit structure*

Let f be a set and let p be a finite sequence. The functor 1GateCircStr(p, f) yields a non void strict many sorted signature which is defined by the following conditions,

(i) *the carrier of 1GateCircStr(p, f) = rng$(p) \cup \{\langle p, f\rangle\}$,*
(ii) *the operation symbols of 1GateCircStr(p, f) = $\{\langle p, f\rangle\}$,*
(iii) *(the arity of 1GateCircStr(p, f))$(\langle p, f\rangle) = p$, and*
(iv) *(the result sort of 1GateCircStr(p, f))$(\langle p, f\rangle) = \langle p, f\rangle$.*

2.3 Input and Output Signals of Circuits

Functors called *InputVertices* and *InnerVertices* are defined for the input and output signals of the circuit [14]. *InputVertices* takes the difference of the element *rng(result sort of G)* in the output state from the internal state (*carrier*). On the other hand, *InnerVertices* is the element *rng(result sort of G)* in the output state of the operation.

Definition 3. *InputVertices and InnerVertices*

Let G be a non empty many sorted signature. The functor $InputVertices(G)$ yielding a subset of G is defined by:

(i) *$InputVertices(G)$ = (The carrier of G) $\setminus$ rng(the result sort of G).*

The functor $InnerVertices(G)$ yields a subset of G and is defined by:

(ii) *$InnerVertices(G)$ = rng(the result sort of G).*

From these definitions of circuit structure 1*GateCircStr*(p, f) and functors *InputVertices* and *InnerVertices* [17], the following propositions are correct.

Proposition 1. *Input and Output of Calculation Circuits*

Let f be a set and let p be a finite sequence. The functor 1GateCircStr(p, f) yields a nonvoid, strict many sorted signature and is defined by the conditions in Def.2. The following propositions are true:

Let f be a set and let p be a finite sequence. The functor 1GateCircStr(p, f) yields a non void strict many sorted signature and is defined by the conditions in Def.2. The following propositions are true:

(i) *$InputVertices$(1GateCircStr(p, f)) = rng(p), and*
(ii) *$InnerVertices$(1GateCircStr(p, f)) = $\{\langle p, f\rangle\}$.*

2.4 Combined Circuits

The combining of *Many Sorted Signatures* of circuits (combined circuits) can be defined as the conjunctive sets for each element (algebraic operation $+\cdot$) on the structure [17].

Definition 4. *Combining of Circuits*

Let A be a set, let f_1, g_1 be non-empty many sorted sets indexed by A, let B be a set, let f_2, g_2 be non-empty many sorted sets indexed by B, let h_1 be a many sorted function from f_1 into g_1, and let h_2 be a many sorted function from f_2 into g_2. Then $h_1 +\cdot h_2$ is a many sorted function from $f_1 +\cdot f_2$ into $g_1 +\cdot g_2$.

Let us note that the predicate $S_1 \approx S_2$ is reflexive and symmetric. Let S_1, S_2 be non empty many sorted signatures. The functor $S_1 +\cdot S_2$ yielding a strict non empty many sorted signature is defined by:

(i) *the carrier of $S_1 +\cdot S_2$ = (the carrier of S_1) $\cup$ (the carrier of S_2),*
(ii) *the operation symbols of $S_1 +\cdot S_2$*
= (the operation symbols of S_1) $\cup$ (the operation symbols of S_2),
(iii) *the arity of $S_1 +\cdot S_2$*
= (the arity of S_1) $\cup$ (the arity of S_2), and
(iv) *the result sort of $S_1 +\cdot S_2$*
= (the result sort of S_1) $\cup$ (the result sort of S_2).

The following propositions are correct from this definition on the circuit structure 1*GateCircStr*(p, f) and functors *InputVertices* and *InnerVertices* [17].

Proposition 2. *Input and Output of Combined Circuits*

Let S_1 be a non void non empty many sorted signature and let S_2 be a non empty many sorted signature. One can verify that $S_1 +\cdot S_2$ is non void and $S_2 +\cdot S_1$ is non void.

The following propositions are true:

(i) *For all non empty many sorted signatures S_1, S_2 such that $S_1 \approx S_2$ holds InnerVertices$(S_1 +\cdot S_2)$ = InnerVertices(S_1) $\cup$ InnerVertices(S_2), and*
(ii) *For all non empty many sorted signatures S_1, S_2 such that $S_1 \approx S_2$ holds InputVertices$(S_1 +\cdot S_2) \subseteq$ InputVertices(S_1) $\cup$ InputVertices(S_2).*

2.5 Circuit Computations and Stability

The functor *Following* can be defined as the states of a circuit after a single step computation. The stability of the circuit (*stable*) is achieved when no internal state has a value different from its previous value after the circuit computation [16].

Definition 5. *Circuit Computations*

Let I_1 be a circuit-like non void non empty many sorted signature, let S_1 be a non-empty circuit of I_1, and let s be a state of S_1. Let v be a vertex of I_1. Then, the functor Following(s) yields a state of S_1 and is defined by:

(i) *if $v \in$ InputVertices(I_1), then $(Following(s))(v) = s(v)$, and*
(ii) *if $v \in$ InnerVertices(I_1), then $(Following(s))(v)$*
$= (Den(\text{the action at } v,\ S_1))((\text{the action at } v) \text{ depends-on-in } s)$.

For instance, when an operator f is a function from the set $Boolean^2$ to $Boolean$ on the circuit $1GateCircStr(p, f)$, Proposition 3 on stability is proved. This shows that the state of $1GateCircuit$ with $Boolean$ operator f having two inputs is stable within a single step computation as $Following(s)$. Therefore, the stability of a combined circuit acting as a series of n single step circuits is denoted as $Following(s, n)$.

The following propositions are correct from this definition on the circuit structure $1GateCircStr(p, f)$ with two tuples of $Boolean$ [18].

Proposition 3. *Stability of Circuits*

Let x, y be sets and let f be a function from $Boolean^2$ into Boolean, and let the functor $1GateCircuit(\langle x, y\rangle, f)$ yield a Boolean strict circuit of 1GateCircStr $(\langle x, y\rangle, f)$ with denotation held in gates. The following propositions are true:

(i) *For every state s of $1GateCircuit(\langle x, y\rangle, f)$ holds*
$(Following(s))(\langle\langle x, y\rangle, f\rangle) = f(\langle s(x), s(y)\rangle)$ and
$(Following(s))(x) = s(x)$ and $(Following(s))(y) = s(y)$, and
(ii) *For every state s of $1GateCircuit(\langle x, y\rangle, f)$ holds*
$Following(s)$ is stable.

3 Generalized Full Adder Circuit

In this section, we take up the "Generalized Full Adder Circuit" (hereafter, GFAC) [22] as an arithmetic element for RSD expressions and describe its mathematical definition and circuit stability [21]. Using the GFAC, we construct a carry-save high speed adding circuit as a pipeline composition explained in the next section. Here we define an arithmetic circuit using logical operators defined in the Mizar library that have been proved on the basis of various mathematical concepts relating to the circuit explained in Section 2 [19]. The proofs of these definitions and theorems have also been tested using the Mizar proof checker.

3.1 Structure of GFAC

The GFAC is a proposed arithmetic element designed to simultaneously attain both carry-outputs and middle-sum outputs in RSD expression to be explained later. This is done using 3-bit inputs by generalizing the construction of all conventional adder circuits that have the 2-bit input plus 1-bit carry input (Fig. 2) [22].

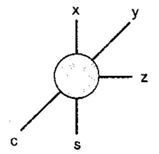
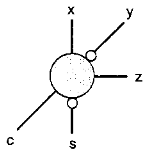
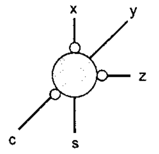
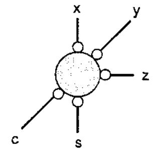
Logic Symbol				
Type	GFA-0	GFA-1	GFA-2	GFA-3
Function	x + y + z = 2c + s	x - y + z = 2c - s	- x + y - z = -2c + s	- x - y - z = -2c - s

Fig. 2. Four types of Generalized Full Adder Circuits. ([22])

The GFAC is a synthesis of two circuits–an adder circuit, which calculates the middle sum for 3-bit inputs, and a carry arithmetic circuit, which calculates carry output. The inputs are x, y, and z, but the input logic is different for four types of the circuit considered. The outputs s and c are also different for each of the four types. In the following, we take up Type 1 (GFA-1) as an example, define its arithmetic circuits, and derive various theorems for the circuit, and then show the stability of the circuit. For the other variant types, definitions are given and various theorems are derived in the same manner.

3.2 Definition of the GFAC Type 1 (GFA-1)

The GFAC Type 1 (GFA-1) is an arithmetic element in which the relationship between the inputs x, y, and z and the outputs s and c satisfy the equation $x - y + z = 2c - s$. The full adder circuit $GFA1AdderStr(x, y, z)$ and the carry arithmetic circuit $GFA1CarryStr(x, y, z)$ are defined as follows. By synthesizing these two circuits, we compose the generalized full adder circuit $BitGFA1Str(x, y, z)$.

Definition 6. *Definition of internal adder circuit: Type 1*

Let x,y,z be sets. The functor $GFA1AdderStr(x, y, z)$ yielding an unsplit non void strict non empty many sorted signature with arity held in gates and Boolean denotation held in gates is defined as follows:

(i) $GFA1AdderStr(x, y, z)$
$= 1GateCircStr(\langle x, y\rangle, xor2c) +\cdot\ 1GateCircStr(\langle[\langle x, y\rangle, xor2c], z\rangle, xor2c)$

where 2-input logical operator xor2c is a function from set $Boolean^2$ to Boolean on the circuit $1GateCircStr(p, f)$ and the result sorts (output) of the inputs $\langle x_1, x_2\rangle$

is defined by $x_1 \oplus not(x_2)$ [21].

The functor $GFA1AdderCirc(x,y,z)$ *yielding a strict Boolean circuit of* $GFA1AdderStr(x,y,z)$ *with denotation held in gates is defined by:*

(ii) $GFA1AdderCirc(x,y,z)$
$= 1GateCircuit(x,y,xor2c) +\cdot\ 1GateCircuit([\langle x,y\rangle, xor2c], z, xor2c)$.

For the result sorts (adderoutput : (s)), the functor $GFA1AdderOutput(x,y,z)$ *yields an element of* $InnerVertices(GFA1AdderStr(x,y,z))$ *and is defined as follows:*

(iii) $GFA1AdderOutput(x,y,z) = [\langle [\langle x,y\rangle, xor2c], z\rangle, xor2c]$.

Definition 7. *Definition of internal majority circuit: Type 1*

Let x,y,z be sets. The functor $GFA1CarryStr(x,y,z)$ *yielding an unsplit non void strict non empty many sorted signature with arity held in gates and Boolean denotation held in gates is defined as follows:*

(i) $GFA1CarryStr(x,y,z)$
$= (1GateCircStr(\langle x,y\rangle, and2c)$
$+\cdot\ 1GateCircStr(\langle y,z\rangle, and2a)$
$+\cdot\ 1GateCircStr(\langle z,x\rangle, and2))$
$+\cdot\ 1GateCircStr(\langle [\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3)$

where 2-input logical operators $and2c, and2a, and2$ *are a function from set* $Boolean^2$ *to Boolean on the circuit* $1GateCircStr(p,f)$*; the result sorts (output) of the inputs* $\langle x_1, x_2\rangle$ *is defined by* $x_1 \wedge not(x_2)$, $not(x_1) \wedge x_2$, *and* $x_1 \wedge x_2$, *respectively* [21][19]*; 3-input logical operator* $or3$ *is a function from set* $Boolean^3$ *to Boolean on the circuit* $1GateCircStr(p,f)$*; and the result sorts (output) of the inputs* $\langle x_1, x_2, x_3\rangle$ *is defined by* $x_1 \vee x_2 \vee x_3$ [19].

The functor $GFA1CarryCirc(x,y,z)$ *yielding a strict Boolean circuit of* $GFA1CarryStr(x,y,z)$ *with denotation held in gates is defined by:*

(ii) $GFA1CarryCirc(x,y,z)$
$= (1GateCircuit(x,y,and2c)$
$+\cdot\ 1GateCircuit(y,z,and2a)$
$+\cdot\ 1GateCircuit(z,x,and2))$
$+\cdot\ 1GateCircuit([\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2], or3)$.

For the result sorts (carryoutput : (c)), the functor $GFA1CarryOutput(x,y,z)$ *yields an element of* $InnerVertices(GFA1CarryStr(x,y,z))$ *and is defined as follows:*

(iii) $GFA1CarryOutput(x,y,z)$
$= [\langle [\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3]$.

Definition 8. *Combining of GFAC Type 1 elements*

Let x,y,z be sets. The functor $BitGFA1Str(x,y,z)$ *yields an unsplit non void strict non empty many sorted signature with arity held in gates and Boolean denotation held in gates and is defined as follows:*

(i) $BitGFA1Str(x,y,z) = GFA1AdderStr(x,y,z) +\cdot\ GFA1CarryStr(x,y,z)$.

The functor $BitGFA1Circ(x,y,z)$ *yielding a strict Boolean circuit of BitGFA1Str* (x,y,z) *with denotation held in gates is defined by:*

(ii) $BitGFA1Circ(x,y,z)$
$= GFA1AdderCirc(x,y,z) +\cdot\ GFA1CarryCirc(x,y,z)$.

3.3 Propositions for Input and Output Signal of Circuits

Propositions for *carrier, InputVertices, InnerVertices* as internal signal states and input and output signals can be proved based on the definitions of the calculation circuits [21].

Proposition 4. *Propositions for the internal adder circuit*

The following propositions of the internal adder circuit are true:

(i) *For all sets x,y,z holds* $InnerVertices(GFA1AdderStr(x,y,z))$ *is a binary relation,*
(ii) *For all sets x,y,z holds the carrier of* $GFA1AdderStr(x,y,z)$
$= \{x,y,z\} \cup \{[\langle x,y\rangle, xor2c], [\langle [\langle x,y\rangle, xor2c], z\rangle, xor2c]\}$,
(iii) *For all sets x,y,z holds* $InnerVertices(GFA1AdderStr(x,y,z))$
$= \{[\langle x,y\rangle, xor2c], [\langle [\langle x,y\rangle, xor2c], z\rangle, xor2c]\}$, *and*
(iv) *For all non pair sets x,y,z holds* $InputVertices(GFA1AdderStr(x,y,z))$
$= \{x,y,z\}$.

Proposition 5. *Propositions for the majority circuit*

The following propositions for the majority circuit are true:

(i) *For all sets x,y,z holds* $InnerVertices(GFA1CarryStr(x,y,z))$ *is a binary relation,*
(ii) *For all sets x,y,z holds the carrier of* $GFA1CarryStr(x,y,z)$
$= \{x,y,z\} \cup \{[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\}$
$\cup \{[\langle [\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3]\}$,
(iii) *For all sets x,y,z holds* $InnerVertices(GFA1CarryStr(x,y,z))$

$= \{[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\}$
$\cup \{[\langle[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3]\}$, *and*

(iv) *For all non pair sets x,y,z holds* $InputVertices(GFA1CarryStr(x,y,z))$
$= \{x,y,z\}$.

Proposition 6. *Propositions for GFAC: Type 1*

The following propositions are true:

(i) *For all sets x,y,z holds* $InnerVertices(BitGFA1StrStr(x,y,z))$ *is a binary relation,*

(ii) *For all sets x,y,z holds the carrier of* $BitGFA1StrStr(x,y,z)$
$=$ *(the carrier of* $GFA1AdderStr(x,y,z)$*)* $\cup$
(the carrier of $GFA1CarryStr(x,y,z)$*),*

(iii) *For all sets x,y,z holds* $InnerVertices(BitGFA1StrStr(x,y,z))$
$= (InnerVertices(GFA1AdderStr(x,y,z))) \cup$
$(InnerVertices(GFA1CarryStr(x,y,z)))$, *and*

(iv) *For all non pair sets x,y,z holds* $InputVertices(BitGFA1StrStr(x,y,z))$
$= \{x,y,z\}$.

3.4 Propositions for the Stability of GFAC

Propositions for circuit stability can be proved based on the definitions of combined circuit structures for GFAC: Type 1, *BitGFA1Str*(x,y,z) [21].

Proposition 7. *Calculation outputs of GFAC: Type 1*

The following proposition of the GFAC type-1 after two steps of computation is true:

(i) *Let x,y,z be sets. Let s be a state of* $BitGFA1Circ(x,y,z)$ *and* a_1, a_2, a_3 *be elements of Boolean. Suppose* $a_1 = s(x)$ *and* $a_2 = s(y)$ *and* $a_3 = s(z)$*. Then,*
$(Following(s,2))(GFA1AdderOutput(x,y,z)) = not(a1 \oplus no(a2) \oplus a3)$, *and*
$(Following(s,2))(GFA1CarryOutput(x,y,z))$
$= (a_1 \wedge not(a_2)) \vee (not(a_2) \wedge a_3) \vee (a_3 \wedge a_1))$.

Finally, we have the main result for the stability of the combined circuit structure for GFAC: Type 1, *BitGFA1Str*(x,y,z), namely that it is stable after two steps of computation [21].

Proposition 8. *Stability of the GFAC: Type 1*

We state the following proposition of the GFAC type-1 after two steps of computation:

(i) *Let x,y,z be sets. Let s be a state of* $BitGFA1Circ(x,y,z)$*. Then,*
$Following(s,2)$ *is stable.*

[Proof] *As a prerequisite, we assume the internal signal points of the circuit to differ from the input signal points x, y, and z. The circuit structure S is assumed to be* $BitGFA1Str(x,y,z)$*. Now, we have to show that* $s_1 = s_2$ *for states of the entire circuit S (*$BitGFA1Circ(x,y,z)$*), where* s_1 *is taken as* $Following(s,2)$ *(state after the operation of two steps) of given s, and* s_2 *is* $Following(Following(s,2))$ *(state after operation of three steps).*

First, [A]: *the carrier of S equals* $dom(s_1)$*, and the carrier of S equals* $dom(s_2)$ *by* [15]*:Th.4.* [B]: *the carrier of S equals InnerVertices(GFA1AdderStr* $(x,y,z)) \cup$ *InnerVertices* $(GFA1CarryStr(x,y,z))$ *by Prop.6.*
Now, assume set a being an element of the carrier of S, then $a \in \{x,y,z\}$ *or* $a \in \{[\langle x,y\rangle, xor2c], [\langle[\langle x,y\rangle, xor2c], z\rangle, xor2c]\}$ *or* $a \in \{[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\}$ *or* $a \in \{[\langle[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3]\}$ *by* [B], [28]*:Th.8.*
Then, [C]: $a = x$ *or* $a = y$ *or* $a = b$ *or* $a = [\langle x,y\rangle, xor2c]$ *or* $a = [\langle[\langle x,y\rangle, xor2c], z\rangle, xor2c]$ *or* $a = [\langle x,y\rangle, and2c]$ *or* $a = [\langle y,z\rangle, and2a]$ *or* $a = [\langle z,x\rangle, and2]$ *or* $a = [\langle[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3]$ *by* [29]*:Th.8.*
Second, [D]: *as the input signals* x,y,z *are not included in the output signals from logical gates, the following are always true* $s_2(x) = s_1(x)$ *and* $s_1(x) = s(x)$ *and* $s_2(y) = s_1(y)$ *and* $s_1(y) = s(y)$ *and* $s_2(c) = s_1(c)$ *and* $s_1(c) = s(c)$.
Next, [E]: $s_1([\langle[\langle x,y\rangle, xor2c], z\rangle, xor2c]) = s_1(GFA1AdderOutput\ (x,y,z))$ *by Def.6, then* $s_1([\langle[\langle x,y\rangle, xor2c], z\rangle, xor2c]) = not(s_1(x) \oplus not(s_1(y)) \oplus s_1(c))$ *by logical equivalency in Prop.4. As the Following computation is repeatedly applied, we have the calculation result of the adder output:* $s_2([\langle[\langle x,y\rangle, xor2], c\rangle, xor2\ ]) = not(s_1(x) \oplus not(s_1(y)) \oplus s_1(c))$.
Similarly, [F]: $s_1([\langle[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3] = s_1(GFA1CarryOutput(x,y,z))$ *by Def.7, then* $s_1([\langle[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\ \rangle, or3] = (s_1(x) \wedge not(s_1(y))) \vee (not(s_1(y)) \wedge s_1(c)) \vee (s_1(c) \wedge s_1(x))$ *by logical equivalency in Prop.5. By continuing to apply the Following computation, we obtain the calculation result of the majority output:*
$s_2([\langle[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3] =$
$(s_1(x) \wedge not(s_1(y))) \vee (not(s_1(y)) \wedge s_1(c)) \vee (s_1(c) \wedge s_1(x))$.
Finally, $s_2(a) = s_1(a)$ *by* [C]~[F]*, thus we prove the equation* $s_1 = s_2$ *by* [A] *and* [30]*:Th.9.* **[end]**

4 Redundant Signed Digit Adder Circuit

In this section, we introduce the carry-saved adder arithmetic circuit [22] using redundant signed digit (RSD) representation [23][24] employed in the interior operation of the Montgomery multiplier as an actual example of arithmetic circuits. We present the circuit definition, various propositions of it, and verify its operation. Furthermore, we employ the Mizar proof checker to testify the correctness of the proofs.

4.1 Structure of RSD Adder Circuits

The redundant signed digit adder (RSDA) is a proposal for an arithmetic circuit with the function of preventing speed degradation that accompanies carry propagation [22]. The design postpones carry-related calculations up to a certain layer at which point a final adding output is done by extending bit-expressions to carry out input and output activities for the adder. The circuit composition is improved– from a single *Boolean*, such as $\{0,1\}$, to two combined pairs of *Boolean* elements, such as $\{\langle 0,0\rangle, \langle 1,0\rangle, \langle 0,1\rangle\}$ – to minimize carries that take place at the time of adding as much as possible (Fig. 3).

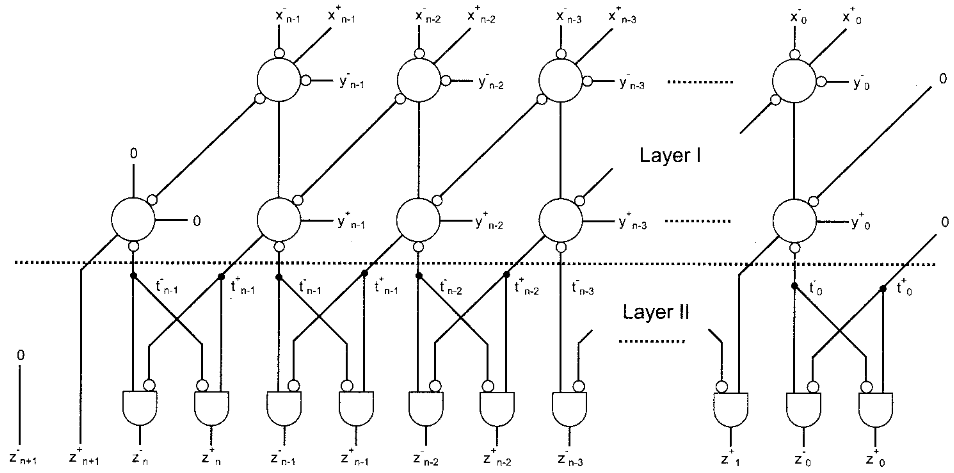


Fig. 3. Circuit configuration of redundant signed digit adder. ([22])

The arithmetic circuit which calculates an intermediate sum using RSD expressions connects by a two-step pipeline at Layer I using the GFAC generalized by 3-inputs/2-outputs [22]. We have shown in Section 3 that the whole GFAC operation stabilizes in two steps. Since the RSD adder has a two-stage pipeline construction after synthesis, the calculation result stabilizes through four steps in total. The pipeline terminal part (Layer II) is constructed with a simple AND circuit (which stabilizes in a single step) to add up carries that have been postponed for succession in the next step with RSD expressions to attain the final result of adding. Therewith, the whole operation stabilizes in five steps total.

4.2 Definition of RSDA Layer I (the intermediate sum)

Layer I (the intermediate sum) of the RSDA carries out RSD adding between c_{in}, the input from the intermediate carry-save from the $k-1$ bit digit and the $k-1$ bit digit inputs $\{x_k^+, x_k^-\}$ and $\{y_k^+, y_k^-\}$ to attain the intermediate sum output as well as the carry pair saved, $\{t_k^+, t_{k+1}^-\}$.

The Layer I adder $BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$, which is a serial composition of GFAC Type 2 ($BitGFA2Str$) and GFAC Type 1 ($BitGFA1Str$) on the sum output line is defined as follows.

Definition 9. *Combining of RSDA Layer I*

Let $x^-, x^+, y^-, y^+, c_{in}$ *be sets. The functor* $BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$ *yields an unsplit non void strict non empty many sorted signature with arity held in gates and Boolean denotation held in gates and is defined as follows:*

(i) $BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$
$= BitGFA2Str(x^-, x^+, y^-)$
$+\cdot\ BitGFA1Str(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)$.

The functor $BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$ *yielding a strict Boolean circuit of BitRSD1Str with denotation held in gates is defined by:*

(ii) $BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$
$= BitGFA2Circ(x^-, x^+, y^-)$
$+\cdot\ BitGFA1Circ(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)$.

4.3 Propositions for Input and Output Signals of RSDA Layer I

Propositions for *carrier*, *InputVertices*, *InnerVertices* as internal signal states and input and output signals can be proved based on the definitions of calculation circuits by Def.9.

Proposition 9. *Propositions for the RSDA Layer I circuit*

The following propositions are true:

(i) *For all sets* $x^-, x^+, y^-, y^+, c_{in}$ *holds*
$InnerVertices(BitRSD1Str(x^-, x^+, y^-, y^+, c_{in}))$ *is a binary relation,*
(ii) *For all sets* $x^-, x^+, y^-, y^+, c_{in}$ *holds*
the carrier of $BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$
$= \{x^-, x^+, y^-, y^+, c_{in}\} \cup$
$\{[\langle x^-, x^+\rangle, xor2c], GFA2AdderOutput(x^-, x^+, y^-)\} \cup$
$\{[\langle x^-, x^+\rangle, and2c], [\langle x^+, y^-\rangle, and2a], [\langle y^-, x^-\rangle, and2],$
$GFA2CarryOutput(x^-, x^+, y^-)\} \cup$
$\{[\langle GFA2AdderOutput(x^-, x^+, y^-), c_{in}\rangle, xor2c],$
$GFA1AdderOutput(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)\} \cup$
$\{[\langle GFA2AdderOutput(x^-, x^+, y^-), c_{in}\rangle, and2c],$
$[\langle c_{in}, y^+\rangle, and2a],$
$[\langle y^+, GFA2AdderOutput(x^-, x^+, y^-)\rangle, and2],$
$GFA1CarryOutput(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)\}$, *and*

(iii) *For all non pair sets* x^-, x^+, y^-, y^+*,set* c_{in} *holds*
$InputVertices(BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})) = \{x^-, x^+, y^-, y^+, c_{in}\}$.

4.4 Propositions for the Stability of RSDA Layer I

Propositions for circuit stability can be proved based on the definitions of combined circuit structures using RSDA Layer I, *BitRSD1Str*.

Proposition 10. *Calculation outputs of RSDA Layer I*

The following proposition of the RSDA Layer I after four steps of computation is true:

(i) *Let* $x^-, x^+, y^-, y^+, c_{in}$ *be sets. Let* s *be a state of* $BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$ *and* a_1,a_2,a_3,a_4,a_5 *be elements of Boolean. Suppose* $a_1 = s(x^-)$, $a_2 = s(x^+)$, $a_3 = s(y^-)$, $a_4 = s(y^+)$ *and* $a_5 = s(c_{in})$. *Then, for an output signal* t^-, *we have*

$$Following(s,4)(BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in})) \\ = not(not(a_1) \oplus a_2 \oplus not(a_3) \oplus a_4 \oplus not(a_5)).$$

We have a similar proposition for output signal t^+
as $BitRSD1Output^+(x^-, x^+, y^-, y^+, c_{in})$.

Now, we have a result for the stability of the combined circuit structure, as RSDA Layer I, *BitRSD1Circ* is stable after four (2 + 2) steps of computation.

Proposition 11. *Stability of RSDA Layer I*

We state the following proposition of RSDA Layer I after four steps of computation:

(i) *Let* $x^-, x^+, y^-, y^+, c_{in}$ *be sets. Let* s *be a state of* $BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$. *Then,* $Following(s, 4)$ *is stable by using Prop.8 for GFAC as Type 1,2 and* [20]*:Th.20 (after 2 + 2 steps of computation).*

4.5 Definition of RSDA Layer I+II

For the circuit construction of RSDA/Layer I+II (the final sum), we merge the intermediate sum from the k bit digit and the intermediate carry saved from the $k-2$ and $k-1$ bit digit inputs and then output the final result (sum) pair $\{z_k^+, z_{k+1}^-\}$.

We define the Layer I+II adder $BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+)$, which is a serial composition between RSDA Layer I(*BitRSD1Str*) and a combined logical AND gates circuit (*1GateCircStr(p, and2c)* $+\cdot$*1GateCircStr(p, and2a)*) with the carry and sum signals as follows.

Definition 10. *Combining of RSDA Layer I+II*

Let $x^-, x^+, y^-, y^+, c_{in}, t^+$ *be sets. The functor* $BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+)$ *yields an unsplit non void strict non empty many sorted signature with arity held in gates and Boolean denotation held in gates and is defined as follows:*

(i) $BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+)$
$= BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$
$+\cdot\ 1GateCircStr(\langle t^+, BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in})\rangle, and2c)$
$+\cdot\ 1GateCircStr(\langle t^+, BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in})\rangle, and2a)$.

The functor $BitRSDCirc(x^-, x^+, y^-, y^+, c_{in}, t^+)$ *yielding a strict Boolean circuit of BitRSDStr with denotation held in gates is defined by:*

(ii) $BitRSDCirc(x^-, x^+, y^-, y^+, c_{in}, t^+)$
$= BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$
$+\cdot\ 1GateCircuit(t^+, BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in}), and2c)$
$+\cdot\ 1GateCircuit(t^+, BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in}), and2a)$.

4.6 Propositions for Input and Output Signals of RSDA Layer I+II

Propositions for *carrier, InputVertices, InnerVertices* as internal signal states and input and output signals can be proved based on the definitions of the calculation circuits by Def.10.

Proposition 12. *Propositions for the RSDA Layer I+II circuit*

The following propositions are true:

(i) *For all sets* $x^-, x^+, y^-, y^+, c_{in}, t^+$ *holds*
$InnerVertices(BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+))$ *is a binary relation,*
(ii) *For all sets* $x^-, x^+, y^-, y^+, c_{in}, t^+$ *holds*
the carrier of $BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+)$
$= \{x^-, x^+, y^-, y^+, c_{in}, t^+\} \cup$
$\{[\langle x^-, x^+\rangle, xor2c], GFA2AdderOutput(x^-, x^+, y^-)\} \cup$
$\{[\langle x^-, x^+\rangle, and2c], [\langle x^+, y^-\rangle, and2a], [\langle y^-, x^-\rangle, and2],$
$GFA2CarryOutput(x^-, x^+, y^-)\} \cup$
$\{[\langle GFA2AdderOutput(x^-, x^+, y^-), c_{in}\rangle, xor2c],$
$GFA1AdderOutput(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)\} \cup$
$\{[\langle GFA2AdderOutput(x^-, x^+, y^-), c_{in}\rangle, and2c],$
$[\langle c_{in}, y^+\rangle, and2a],$
$[\langle y^+, GFA2AdderOutput(x^-, x^+, y^-)\rangle, and2],$
$GFA1CarryOutput(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)\} \cup$
$\{[\langle t^+, BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in})\rangle, and2c],$
$[\langle t^+, BitRSD1Output^-(x^-, x^+, y^-, y^+; c_{in})\rangle, and2a]\}$
, and

(iii) *For all non pair sets* x^-, x^+, y^-, y^+,*set* c_{in}, t^+ *holds*
$InputVertices(BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+))$
$= \{x^-, x^+, y^-, y^+, c_{in}, t^+\}$.

4.7 Propositions for Stability of RSDA Layer I+II

Propositions for circuit stability can be proved based on definitions of the combined circuit structure for RSDA Layer I+II, *BitRSDStr*.

Proposition 13. *Calculation outputs of RSDA Layer I+II*

The following proposition of the RSDA Layer I+II after five steps (4 + 1 steps) of computation is true:

(i) *Let* $x^-, x^+, y^-, y^+, c_{in}, t^+$ *be sets. Let s be a state of* $BitRSDCirc(x^-, x^+, y^-, y^+, c_{in}, t^+)$ *and* a_1,a_2,a_3,a_4,a_5,a_6 *be elements of Boolean. Suppose* $a_1 = s(x^-)$, $a_2 = s(x^+)$, $a_3 = s(y^-)$, $a_4 = s(y^+)$, $a_5 = s(c_{in})$ *and* $a_6 = s(t^+)$. *Then, with respect to output signal* z^+, *we have*
$Following(s, 5)(BitRSDOutput^+(x^-, x^+, y^-, y^+, c_{in}, t^+))$
$= a_6 \wedge (not(a_1) \oplus a_2 \oplus not(a_3) \oplus a_4 \oplus not(a_5)$.

We have a similar proposition for output signal z^- *as* $BitRSDOutput^-(x^-, x^+, y^-, y^+, c_{in}, t^+)$.

Finally, as RSDA Layer I+II, *BitRSDCirc* is stable after five (4 + 1) steps of computation, we have the main result of stability for the combined circuit structure.

Proposition 14. *Stability of RSDA Layer I+II*

We state the following proposition of the RSDA Layer I+II after five steps (4 + 1 steps) of computation:

(i) *Let* $x^-, x^+, y^-, y^+, c_{in}, t^+$ *be sets. Let s be a state of* $BitRSDCirc(x^-, x^+, y^-, y^+, c_{in}, t^+)$. *Then,* $Following(s, 5)$ *is stable by using Prop.11 for RSDA Layer I, Prop.3 and* [20]*:Th.20 (after 4 + 1 steps of computation).*

5 Conclusion

By using mathematical models of an arithmetic logic unit based on many sorted algebraic structures, we successfully proved the important properties of an example of a calculation circuit, the RSD adder. The stability of the RSD adder circuit example is proved based on definitions and theorems about logic operators, hardware gates, and signal line connections using the Mizar proof checking system as a formal verification tool.

In future research on circuit verification by proof checking, we plan to do design verifications of such circuits as Wallace-Tree type multipliers, Montgomery high-speed multipliers, RSA encryption processing units, and various high speed operational circuits necessary for arithmetic logic units.

Acknowledgement

This research was strongly inspired by Professor A. Trybulec, Y. Nakamura, P. Rudnicki and G. Bancerek [11][12][13][14][15][16][17][18][20]. I would like to express my deep thanks to Professor Andrzej Trybulec and the Mizar project teams around the world.

References

1. Ichikawa, S., et al.: A Connection Sleeves Line Architecture (Pipelined MIMD). The 37th Annual Conference of Information Processing Society of Japan, 4, N-4 (1988).
2. Kleeman, L., Cantoni, A.: Can Redundancy and Masking Improve the Performance of Synchronizers? IEEE Trans. Computers, C-35(7) (1986) 643–646.
3. Stoffers, K. E.: Test Sets for Combinational logic – The Edge – Tracing Approach. IEEE Trans. Computers, C-29(8) (1980) 741–746.
4. Strangio, C. E.: DIGITAL ELECTRONICS: Fundamental Concepts and Applications. Prentice-Hall Inc. (1980) 355–385.
5. Unger, S. H., Tan, C. J.: Clocking Schemes for High-Speed Digital Systems. IEEE Trans. Computers, C-35(10) (1986) 880–895.
6. Ishiura, N., Takahashi, M., Yajima, S.: Time-Symbolic Simulation for Accurate Timing Verification of Logic Circuits. Trans. of Information Processing Society of Japan, 31(12) (1990) 1832–1839.
7. Ogura, S.: The Development Methods in CPLD/FPGA. Technical Paper of the 1st Japanese FPGA/PLD Design Conference, 4 (1993) 46–62.
8. Nakamura, Y., Nishiyama, T., Fuwa, Y.: Signal Propagation Timing Check System in Multi-Level Sequential Circuits. Trans. of the Institute of Electronics, Information and Communication Engineers, 75(12) (1992) 1826–1836.
9. Nishiyama, T., Fuwa, Y., Eguchi, M., Nakamura, Y.: A Digital Circuits Timing Verification Scheme based on a Clock Model. Trans. of the Institute of Electronics, Information and Communication Engineers, 77(6) (1994) 860–870.
10. Nishiyama, T., Mizuhara, Y.: Binary Arithmetics. Formalized Mathematics, 4(1) (1993) 83–86.
11. Trybulec, A.: Many-sorted Sets. Formalized Mathematics, 4(1) (1993) 15–22.
12. Trybulec, A.: Many Sorted Algebras. Formalized Mathematics, 5(1) (1996) 37–42.
13. Nakamura, Y., Rudnicki, P., Trybulec, A., Kawamoto, P. N.: Preliminaries to Circuits, I. Formalized Mathematics, 5(2) (1996) 167–172.
14. Nakamura, Y., Rudnicki, P., Trybulec, A., Kawamoto, P. N.: Preliminaries to Circuits, II. Formalized Mathematics, 5(2) (1996) 215–220.
15. Nakamura, Y., Rudnicki, P., Trybulec, A., Kawamoto, P. N.: Introduction to Circuits, I. Formalized Mathematics, 5(2) (1996) 227–232.
16. Nakamura, Y., Rudnicki, P., Trybulec, A., Kawamoto, P. N.: Introduction to Circuits, II. Formalized Mathematics, 5(2) (1996) 273–278.
17. Nakamura, Y., Bancerek, G.: Combining of Circuits. Formalized Mathematics, 5(2) (1996) 283–295.
18. Bancerek, G., Nakamura, Y.: Full Adder Circuit, Part I. Formalized Mathematics, 5(3) (1996) 367–380.
19. Wasaki, K., Kawamoto, P. N.: 2's Complement Circuits, Part I. Formalized Mathematics, 6(2) (1996) 189–197.
20. Bancerek, G., Yamaguchi, S., Shidama, Y.: Combining of Multi Cell Circuits. Formalized Mathematics, 10(1) (2002) 47–64.

21. Yamaguchi, S., Wasaki, K., Shimoi, N.: Generalized Full Adder Circuits (GFAs), Part I. Formalized Mathematics, 13(4) (2005) 549–571.
22. Savas, E: A Carry-Free Architecture for Montgomery Inversion. IEEE Trans. Computers, 54(12) (2005) 1508–1519.
23. Avizienis, A.: Signed-Digit Number Representation for Fast Parallel Arithmetic. IRE Trans. Electronic Computers, 10 (1961) 389–400.
24. Vandemeulebroecke, A., Vanzieleghem, E., Denayer, T., Jespers, P. G. A.: A New Carry-Free Division Algorithm and its Application to a Single-Chip 1024-b RSA Processor. IEEE J. Solid-State Circuits, 25(3) (1990) 748–755.
25. Bonarska, E.: An Introduction to PC Mizar. In Roman Matuszewski, editor, Series of Fondation Philippe le Hodey, Brussels (1990).
26. Trybulec, A., Rudnicki, P.: A Collection of TEXed Mizar Abstracts. University of Alberta, Canada (1989).
27. Mizar Proof Checker: `http://mizar.org/`
28. Trybulec, Z., Świeczkowska, H.: Boolean Properties of Sets. Formalized Mathematics, 1(1) (1990) 17–23.
29. Trybulec, A.: Enumerated Sets. Formalized Mathematics, 1(1) (1990) 25–34.
30. Byliński, C.: Functions and Their Basic Properties. Formalized Mathematics, 1(1) (1990) 55–65.

Computer Supported Formal Work: Towards a Digital Mathematical Assistant

Jörg Siekmann and Serge Autexier

DFKI GmbH & Saarland University
Saarbrücken, Germany
`{siekmann|autexier}@ags.uni-sb.de`
`www.ags.uni-sb.de/~serge`

Abstract. The year 2004 marked the fiftieth birthday of the first computer generated proof of a mathematical theorem: "the sum of two even numbers is again an even number" (with Martin Davis' implementation of Presburger Arithmetic in 1954). While Martin Davis and later the research community of automated deduction used machine oriented calculi to find the proof for a theorem by automatic means, the AUTOMATH project of N.G. de Bruijn[1] – more modest in its aims with respect to automation – showed in the late 1960s and early 70s that a complete mathematical textbook could be coded and proof-checked by a computer. Roughly at the same time in 1973, the MIZAR project[2] started as an attempt to reconstruct mathematics based on computers. Since 1989, the most important activity in the MIZAR project has been the development of a database for mathematics. International cooperation resulted in creating a database which includes more than 7000 definitions of mathematical concepts and more than 42000 theorems.

The work by Milner and others on LCF [28] spawned a research community on tactical theorem proving, which again modest with respect to automation, showed that nevertheless important sequences of deduction could be encapsulated into a tactic and then invoked by the mathematically trained user. Classical automated theorem proving procedures of today are based on ingenious search techniques to find a proof for a given theorem in very large search spaces – often in the range of several billion clauses. But in spite of many successful attempts to prove even open mathematical problems automatically, their use in everyday mathematical practice is still limited. The shift from search based methods to more abstract planning techniques however opened up a new paradigm for mathematical reasoning on a computer and several systems of the new kind now employ a mix of interactive tactic based, search based as well as proof planning based techniques. The ΩMEGA system is at the core of several related and well-integrated research projects of the ΩMEGA research group, whose aim is to develop system support for the working mathematician and formal software engineer, in particular it supports proof development at a human oriented level of abstraction. Sum-

[1] www.win.tue.nl/automath

[2] www.mizar.org

ISBN 978-83-7431-128-1 ISSN 0860-150X

marizing what we have today in terms of technological support, we shall then address the most pressing needs of the future in this paper.

1 Introduction

Computer-based information processing plays an increasingly important role in modern societies and meanwhile touches the very basic organization of our daily life. Our dependency on the well-functioning of this information processing motivates the development of *provably* reliable software and hardware. This both includes human comprehensible specifications and their automatic or interactive verification.

Logical systems were developed originally to provide a foundation for mathematics and their application to the specification and verification of software resulted – since the mid 1950s – in *formal* software development methods. Since then, there is a myriad of formalisms and systems, both for mathematics and formal methods, which are unfortunately not always interoperable. Today even non-trivial mathematical problems can be proved by machines and formal methods have been employed for complex software and hardware developments. In order to do so, increasing parts of mathematics and software have been formalized in system-specific libraries that are mostly distributed over different physical locations.

The vision of computer-supported mathematics and a system which provides integrated support for all work phases of a mathematician (or a software engineer using formal methods) has fascinated researchers in artificial intelligence, particularly in the deduction systems area, and in mathematics for a long time (see Fig. 1). The dream of mechanizing (mathematical) reasoning dates back to Gottfried Wilhelm Leibniz in the 17th century with the touching vision that two philosophers engaged in a dispute would one day simply code their arguments into an appropriate formalism and then *calculate* (Calculemus!) who is right. At the end of the 19th century modern mathematical logic was born with Frege's *Begriffsschrift* and an important milestone in the formalization of mathematics was Hilbert's program and the 20th century Bourbakism.

With the logical formalism for the representation and calculation of mathematical arguments emerging in the first part of the twentieth century it was but a small step to implement these techniques now on a computer as soon as it was widely available. In 1954 Martin Davis' Presburger Arithmetic Program was reported to the US Army Ordnance and the Dartmouth Conference in 1956 is not only known for giving birth to artificial intelligence in general but also more specifically for the demonstration of the first automated reasoning programs for mathematics by Herb Simon and Alan Newell.

However, after the early enthusiasm of the 1960s, in particular the publication of the resolution principle in 1965 [40], and the developments in the 70s a more sober realization of the actual difficulties involved in automating everyday mathematics set in and the field increasingly fragmented into many subareas which all developed their specific techniques and systems[3].

[3] The history of the field is presented in a classical paper by Martin Davis [18] and also in [19] and more generally in his history of the making of the first computers [17]. Another source is Jörg Siekmann [43] and more recently [44].

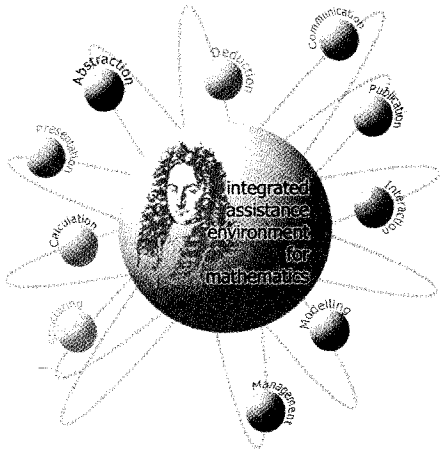


Fig. 1. CALCULEMUS illustration of different challenges for a mathematical assistant system.

Apart from its fragmentation, four major deficiencies of the proof assistant systems are in the way for a more widespread use in mathematics and software engineering. First, the user interfaces of current systems are not convenient enough to attract non-expert users. In particular, the internal representation languages of these systems are still too far away from reaching the flexibility, expressivity and elegance of common mathematical vernacular. Secondly, the formal proof assistant systems have their relative strengths and weaknesses, however they are usually incompatible, both at the conceptual and linguistic levels. Thirdly, according to the Common Criteria [39] the deliverables of a piece of software must also contain documentation or even *security targets* and *protection profiles*. However, current systems do not provide sufficient support for linking such informal or semi-formal documents to formal proof developments. The same applies for a mathematical or engineering publication: How do we link the informal text paragraphs with the formal content? Finally and fourth, the maintenance techniques of existing systems are too immature to cope with the increasing sizes and complexities of their libraries.

It is only very recently that this trend is reversed, with the CALCULEMUS[4] and MKM[5] communities as driving forces of this movement. In CALCULEMUS the viewpoint is bottom-up, starting from existing techniques and tools developed in the community. MKM approaches the goal of computer-based mathematics and formal methods in the new millennium by a complementary top-down approach starting from existing, mainly pen and paper based mathematical practice down to system support.

[4] www.calculemus.org

[5] monet.nag.co.uk/mkm

In both communities the MIZAR project [41, 42] served as an important existence proof, that indeed formalized mathematics is possible: with close to 8000 definitions and more than 42000 theorems represented and proved within the system is now generally viewed as a milestone.

We shall now address two questions: What do we have achieved today in the ΩMEGA project and what do we need in the future. In Sections 2 and 3 we provide an overview and the main developments of the ΩMEGA project and then point to current research and some future goals.

2 What we have: Ωmega

The ΩMEGA project[6] represents one of the major attempts to build an all encompassing assistant tool for the working mathematician. It is a representative of systems in the new paradigm of *proof planning* and combines interactive tactical and automated proof construction for domains with rich and well-structured mathematical knowledge. The inference mechanism at the lowest level of abstraction is an interactive theorem prover based on a higher order natural deduction (ND) variant of a soft-sorted version of Church's simply typed λ-calculus [16]. The logical language, which also supports partial functions, is called $\mathcal{POST}$, for ***p***artial functions and ***o***rder ***s***orted ***t***ype theory. While this represents the "machine code" of the system the user will seldom want to see, the search for a proof is usually conducted at a higher level of abstraction defined by *tactics* and *methods*. Automated proof search at this abstract level is called *proof planning*. Proof construction is also supported by already proved assertions and theorems and by calls to external systems to simplify or solve subproblems.

At the core of ΩMEGA is the *proof plan data structure* $\mathcal{PDS}$ [15], in which proofs and *proof plans* are represented at various levels of granularity and abstraction (see the central part in Fig. 2). The $\mathcal{PDS}$ is a directed acyclic graph, where *open nodes* represent unjustified propositions that still need to be proved and *closed nodes* represent propositions that are already proved. The proof plans are developed and classified with respect to a taxonomy of mathematical theories in the mathematical knowledge base MBASE [26, 31]. The user of ΩMEGA, or the proof planner MULTI [36], or else the suggestion mechanism ΩANTS [11] modify the $\mathcal{PDS}$ during proof development until a complete proof plan has been found. They can also invoke external reasoning systems, whose results are included in the $\mathcal{PDS}$ after appropriate transformation. Once a complete proof plan at an appropriate level of abstraction has been found, this plan must be expanded by sub-methods and sub-tactics into lower levels of abstraction until finally a proof at the level of the logical calculus is established. After expansion of these high-level proofs to the underlying ND calculus, the $\mathcal{PDS}$ can be checked by ΩMEGA's proof checker.

Hence, there are two main tasks supported by this system, namely (i) to find a proof plan, and (ii) to expand this proof plan into a calculus-level proof; and both jobs can be equally difficult and time consuming. Task (ii) employs an LCF-style tactic expansion mechanism, proof search or a combination of both in order to generate a lower-level proof object. It is a design objective of the $\mathcal{PDS}$ that various *proof levels* coexist with their respective relationships being dynamically maintained.

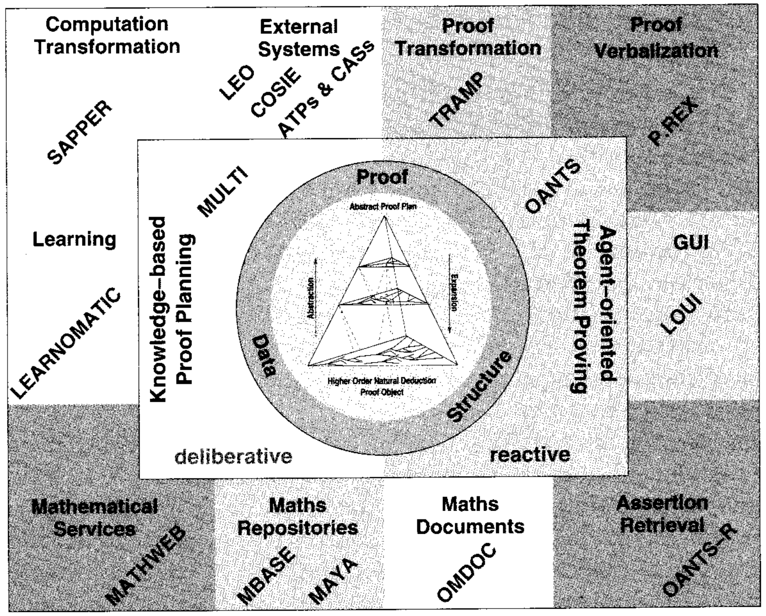


Fig. 2. The vision of an all encompassing mathematical assistant environment: we have now modularized and out-sourced many of the support tools such that they can also be used by other systems via the MATHWEB-SB software bus.

The graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [46] provides both a graphical and a tabular view of the proof under consideration, and the interactive proof explanation system *P.rex* [24, 23, 25] generates a natural-language presentation of the proof.

The previously monolithic system has been split up and separated into several independent modules, which are connected via the mathematical software bus MATHWEB-SB [52]. An important benefit is that MATHWEB-SB modules can be distributed over the Internet and are then remotely accessible by other research groups as well. There is now a very active MATHWEB-SB user community with sometimes several thousand theorems and lemmata being proven per day. Most theorems are generated automatically as (currently non-reusable and non-indexed) subproblems in natural language processing (see the Doris system [21]), proof planning and verification tasks.

For more details about the ΩMEGA system and research activities around the system we refer the interested reader to [45, 3] and to the web-page of the project (www.ags.uni-sb.de/˜omega).

[6] www.ags.uni-sb.de/˜omega

3 What do we need in future?

The vision of a powerful mathematical assistance environment which provides computer-based support for most tasks of a mathematician has stimulated new projects and international research networks across the disciplinary and systems boundaries. Examples are the European Calculemus[7] (Integration of Symbolic Reasoning and Symbolic Computation) and Mkm[8] (Mathematical Knowledge Management, [13]) initiatives, the EU projects Monet[9], OpenMath and Mowgli[10], and the American Qpq[11] repository of deductive software tools. Of course, a milestone in this area has been the Mizar project[12], and there are now numerous national projects in the US and Europe, which cover partial aspects of this vision, such as knowledge representation, deductive system support, user interfaces, mathematical publishing tools, etc.

The longterm goal of the ΩMega project is the all-embracing integration of symbolic reasoning, i.e. computer algebra and deduction systems, into mathematical research, mathematics education, and formal methods in computer science. We anticipate that in the long run these systems will change mathematical practice and they will have a strong societal impact, not least in the sense that a powerful infrastructure for mathematical research and education will become commercially available. Computer supported mathematical reasoning tools and integrated assistance systems will be further specialized to have a strong impact also in many other theoretical fields such as safety and security verification of computer software and hardware, theoretical physics and chemistry and other related subjects.

Our current approach is strictly bottom-up: Starting with existing techniques and tools of our partners for symbolic reasoning (deduction) and symbolic computation (computer algebra), we will step by step improve their interoperability up to the realization of an integrated systems via the mathematical software bus MathWeb-SB. The envisaged system will support the full life-cycle of the evolutionary nature of mathematical research (see Fig. 3) helping an engineer or mathematician who works on a mathematical problem in the improvement, the exploration, the distributed maintenance, the retrieval and the proving and calculation tasks and finally the publication of mathematical theories.

So what does this vision entail in the immediate future? We want to develop methods and tools for a better integration of formal proof systems into everyday mathematical activity and into the development of provably reliable software. Four major aspects are:

- The development of non-proprietary, *semi-formal representation formats* which support a continuous range of proof details in informal mathematical documents.

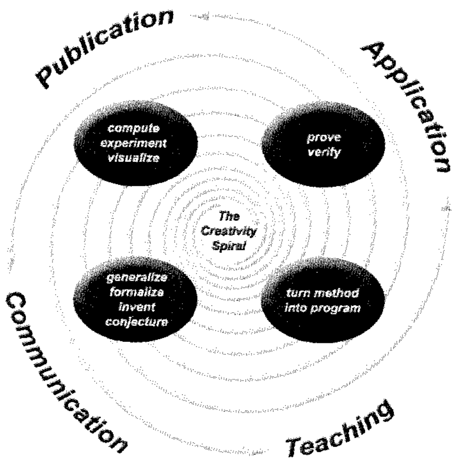


Fig. 3. Mathematical Creativity Spiral; [by Buchberger in 1995]

- Support for the interactive development of semi-formal documents inside an existing open-source scientific WYSIWYG text editor.
- The integration of existing mathematical assistant and verification systems into such an editor.
- The development of sophisticated, semantics-based truth-maintenance techniques for the distributed libraries *inclusive of* their mathematical documents.

Bridging the gap from the informal common mathematical language to the formal proof assistant oriented languages will provide software developers (respectively working mathematicians[13]) access to the reasoning capabilities and the pieces of formalized mathematics for their support.

More specifically, we are working currently towards the following goals:

3.1 Formalization and Proving at a Higher Level of Abstraction

Mathematical reasoning with the ΩMega system is at the comparatively high level of abstraction of the proof planning methods. However, as these methods have to be expanded eventually to the concrete syntax of our higher order ND-calculus, the system still suffers from the effect and influence this logical representation has. In contrast, the proofs developed by a mathematician, say for a mathematical publication, and the proofs developed by a student in a mathematical tutoring system are

[7] www.calculemus.org
[8] monet.nag.co.uk/mkm
[9] monet.nag.co.uk/cocoon/monet
[10] www.mowgli.cs.unibo.it
[11] www.qpq.org
[12] www.mizar.org

[13] We do not expect the high class mathematicians at the forefront of mathematical research to profit from these developments in the foreseeable future. However many mathematical application areas (such as statistics for constructing bridges or buildings, some areas of theoretical physics or mechanical engineering, etc.) are close to formalization anyway, and may well be amendable using computer algebra and automated reasoning tools.

typically developed at an argumentative level. This level has been formally categorized as *proofs at the assertion level* [29] with different types of *under-specification* [2].[14] The CoRe calculus [1, 7] for contextual reasoning has been designed to support reasoning directly on the assertion level and our current goal, which is by large completed, is to completely exchange the natural deduction calculus by the CoRe calculus. The CoRe calculus has been developed for a large class of logics, comprises a uniform notion of a logical context of subformulas as well as replacement rules available in a logical context. Replacement rules operationalize assertion level proof steps and technically are generalized resolution and paramodulation rules, which in turn should suit the implementation of automatic reasoning procedures.

The exchange of the logic layer in ΩMEGA requires the adaptation of all reasoning procedures that are currently tailored to it, including proof planning and the integration of external systems. The first step in that direction was to standardize the basic proof construction mechanism used by both the human user and the automatic reasoning procedures that previously used proprietary mechanisms. This is realized in the so-called *tasklayer* [20] which is an instance of the new proof datastructure (PDS) [4]. The nodes of the PDS are annotated with *tasks*, which are Gentzen-style multi-conclusion sequents augmented by means to define multiple foci of attention on subformulas that are maintained during the proof. Each task is reduced to a possibly empty set of subtasks by one of the following proof construction steps: (1) the introduction of a proof sketch [51][15], (2) deep structural rules for weakening and decomposition of subformulas, (3) the application of a lemma that can be postulated on the fly, (4) the substitution of meta-variables, and (5) the application of a *inferences* inside formulas. *Inferences* are the major proof construction means and unite the concepts of tactics and methods that were separate concepts in the previous versions of ΩMEGA. An inference is a proof step with multiple premises and conclusions augmented by (1) a possibly empty set of hypotheses for each premise, (2) a set of *application conditions* that must be fulfilled upon inference application, (3) a set of *outline functions* that can compute the values of premises and conclusions from values of other premises and conclusions, and (4) an *expansion function* that refines the abstract inference step. Each premise and conclusion consists of a unique name and a formula scheme. Note that we employ the term *inference* in its general meaning; Taken in that sense, an inference can be either valid or invalid in contrast to the formal logic notion of an *inference rule*.

*Ω*ANTS. The *Ω*ANTS-system was originally developed to support a user in an interactive theorem proving environment by distributively searching via agents for possible next proof steps [12, 47] and was later extended to a fully automated system. Conceptually the system consists of two kinds of agents: *command agents* and *argument agents*. Each command agent is associated to a tactic, method or calculus rule, uniformly called rule, and orders suggestions for the associated rule according to some heuristics. These suggestions are generated by a society of argument agents which are assigned to a command agent. These command and argument agents had to be specified manually in the previous version of the ΩMEGA system. In [8] we automated this process by defining a mechanism to automatically create for a given inference a respective command agent together with an optimal set of argument agents.

3.2 Mathematical Knowledge Representation

A mathematical proof assistant relies upon different kinds of knowledge: first, of course, the formalized mathematical domain as organized in structured theories of definitions, lemmata, and theorems. Secondly, there is mathematical knowledge on how to prove a theorem, which is encoded in tactics and methods, in *Ω*ANTS agents, in control knowledge and in strategies. This type of knowledge can be general, theory specific or even problem specific.

The integration of a mathematical proof assistant into the typical and everyday activities of a mathematician requires however other types of knowledge as well. For example, a mathematical tutoring system for students relies upon a database with different samples of proofs and proof plans linked by meta-data in order to advise the student. Another example is the support for mathematical publications: the documents containing both formalized and non-formalized parts need to be related to specific theories, lemmas, theorems, and proofs. This raises the research challenge on how the usual structuring mechanisms for mathematical theories (such as theory hierarchies or the import of theories via renaming or general morphisms) can be extended to tactics and methods as well as to proofs, proof plans and mathematical documents. Furthermore, changing any of these elements requires maintenance support as any change in one part may have consequences in other parts. For example, the validity of a proof needs to be checked again after changing parts of a theory, which in turn may affect the validity of the mathematical documents. This management of change [30, 5, 38] has been implemented in the MAYA system [6], which is now integrated into the ΩMEGA system as well (see Fig. 4). It supports an evolutionary formal development by allowing users to specify and verify developments in a structured manner, incorporates a uniform mechanism for verification in-the-large to exploit the structure of the specification, and maintains the verification work already done when changing the specification. MAYA relies on development graphs [30] as a uniform representation of structured specifications, which enables the use of various (structured) specification languages to formalize the software development. Moreover, MAYA allows the integration of different theorem provers to deal with the actual proof obligations arising from the specification, i.e. to perform verification in-the-small.

Hierarchically structured mathematical knowledge, i.e. an ontology of mathematical theories and assertions has initially been stored in ΩMEGAs hardwired mathematical knowledge base. This mathematical knowledge base was later (end of the 90s) out-sourced and linked to the development of MBASE [27]. We now assume that a mathematical knowledge base also maintains domain specific control rules, strategies, and linguistic knowledge. While this is not directly a subject of

[14] "Under-specification" is a technical term borrowed from research on the semantics of natural language. Illustrating examples and a discussion of our notion can be found in [2, 9].

[15] In the old ΩMEGA system this was realized by using so-called *Island*-methods.

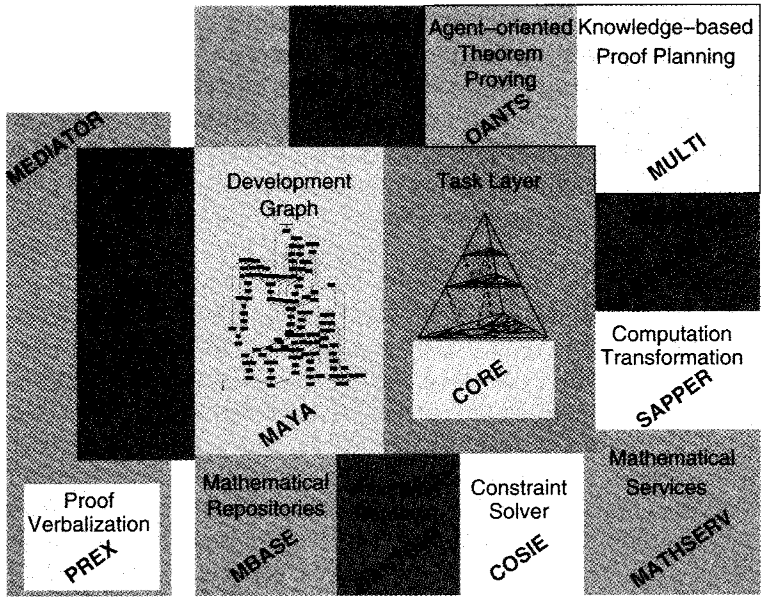


Fig. 4. Architecture of the new version of the mathematical assistance system ΩMEGA.

research in the ΩMEGA project, relying here on other groups of the MKM community and hence on the general development of a worldwide mathematical knowledge base ("the Semantic Web for Mathematicians"), we shall nevertheless concentrate on one aspect, namely how to find the appropriate information.

Semantic Mediators for Mathematical Knowledge Bases. Knowledge acquisition and retrieval in the currently emerging large repositories of formalized mathematical knowledge should not be based purely on syntactic matching, but it needs to be supported by semantic mediators, which suggest applicable theorems and lemmata in a given proof context.

We are working on appropriately limited higher order logic (HOL) reasoning agents for domain- and context-specific retrieval of mathematical knowledge from mathematical knowledge bases. For this we shall adapt a two stage approach as in [10], which combines syntactically oriented pre-filtering with semantic analysis. The pre-filter employ efficiently processable criteria based on meta-data and ontologies that identify sets of candidate theorems from a mathematical knowledge base that are potentially applicable to a focused proof context. The HOL agents act as post-filters to exactly determine the applicable theorems of this set. Exact semantic retrieval includes the following aspects: (i) logical transformations to see the connection between a theorem in a mathematical knowledge base and a focused subgoal. Consider, e.g., a theorem of the form $A \Leftrightarrow B$ in the mathematical knowledge base and a subgoal of the form $(A \Rightarrow B) \wedge (\neg A \Rightarrow \neg B)$; they are not equal in any syntactical sense, but they denote the same assertion. (ii) The variables of a theorem in a mathematical knowledge base may have to be instantiated with terms occurring in a focused subgoal; consider, e.g., a theorem $\forall X \,.\, is{-}square(X \times X)$ and the subgoal $is{-}square(2 \times 2)$. (iii) Free variables (meta-variables) may occur in a focused subgoal and they may have to be instantiated with terms occurring in a theorem of the mathematical knowledge base; consider, e.g., a subgoal $irrational(X)$ with metavariable X and a theorem $irrational(\sqrt{2})$.

We are investigating whether this approach can be successfully coupled with state-of-the-art search engines such as Google.

Synthesizing Inferences. In the old ΩMEGA-system the knowledge contained in the mathematical theories was not automatically available to the proof planner or the ΩANTS-system. In order to make them available for these systems, they had to be specified manually as methods or tactics and agents. In [8] we developed a mechanism that allows the computation of a set of inferences for arbitrary formulas. As example consider the lemma

$$\forall U, V, W : set.U \subseteq V \wedge V \subseteq W \Rightarrow U \subseteq W \tag{1}$$

The inference obtained from that lemma is

$$\frac{P_1 : U \subseteq V \qquad P_2 : V \subseteq W}{C : U \subseteq W} \tag{2}$$

where U, V, W are meta-variables. The benefit of that approach is that proof procedural information used by the automatic reasoning procedures is directly synthesized from declarative representations of mathematical knowledge. Together with the calculus where the application of inferences is the basic proof construction step, this allows interactive and automatic proof development directly at the assertion level and paves the way to offer proof support inside text editors and in a form that is intuitive for a human user.

3.3 MathServe: A Global Web for Mathematical Services

The Internet provides a vast collection of data and computational resources. For example, a travel booking system combines different information sources, such as the search engines, price computation schemes, and the travel information in distributed very large databases, in order to answer complex booking requests. The access to such specialized travel information sources has to be planned, the obtained results combined and, in addition the consistency of time constraints has to be guaranteed.

In [53, 54] this methodology was transferred and applied to mathematical problem solving and developed a system that plans the combination of several mathematical information sources (such as mathematical databases), computer algebra systems, and reasoning processes (such as theorem provers or constraint solvers). Based on the well-developed MATHWEB-SB network of mathematical services [52], the existing client-server architecture has been extended by advanced problem solving capabilities and semantic brokering of mathematical services.

The reasoning systems integrated in MathWeb-SB had to be accessed directly via their API, thus the interface to MathWeb-SB is *system-oriented.* However, these reasoning systems are used also in applications that are not necessarily theorem provers, e.g. for the semantic analysis of natural language, small verification tasks, etc. The main goals of the MathServe framework were therefore:

Problem-Oriented Interface: to develop a more abstract communication level for MathWeb-SB, such that general mathematical problem descriptions can be sent to MathServe which in turn returns a solution to that problem. Essentially, this goal is to move from a *service* oriented interface of MathWeb-SB to a *problem* oriented interface for the MathServe.

Advanced Problem Solving Capabilities: Typically, a given problem cannot be solved by a single service but only by a combination of several services. In order to support the automatic selection and combination of existing services, the key idea is as follows: an ontology is used for the qualitative description of MathWeb-SB services and *these descriptions are then used as AI planning operators*, in analogy to todays proof planning approach. MathServe uses planning techniques [14, 22] to automatically generate a plan that describes how existing services must be combined to solve a given mathematical problem.

3.4 Publishing Tools for Mathematics and Formal Methods

Proof construction is an important but only a small part of a much wider range of mathematical activities an ideal mathematical assistant system should support (see Fig. 1). Therefore the ΩMEGA system is currently extended to support the writing of mathematical publications and documentations of formal methods in software engineering or advising students during proof construction.

The research questions we plan to investigate arise from the following scenario of preparing a mathematical research article with formalized content in a textbook style *and* in professional type-setting quality.

Mathematical Research Article Preparation Scenario: The author starts writing a new mathematical document in a format suitable for publication by using mathematical concepts from different mathematical domains. New mathematical concepts or lemmas introduced in the paper in turn should result in corresponding new formal objects. Furthermore, when writing the document service tools can be used to perform, for instance, intermediate computations in illustrating examples, querying mathematical databases for mathematical publications introducing similar concepts, etc. Proofs of lemmas and theorems contained in the document should be amenable to formal proof checking techniques. Submission of such a document to some journal allows then for instance to check the proofs contained therein to some degree and a long-term goal may be fully automated verification. Publication of such verified mathematical documents then makes the paper and especially its formal contributions accessible to other authors.

The close relationship between the preparation of a mathematical article and the preparation of documents about software is illustrated by the following scenario which consists of the preparation of protection profiles using formal software development techniques.

Preparation of Protection Profiles: The vision underlying this scenario is to support the development cycle of protection profiles and security targets for formal software development as required by the Common Criteria [39]. It starts with the preparation of a protection profile document in publishable format and the three separated parts *Threats*, *Objectives*, and *Requirements.* The first part contains the informal description of the security threats and relates to a formal specification of the threats. The second part formulates the security objectives, which specify the desired system behavior together with a *correspondence demonstration* that this excludes the behaviors leading to threatening situations as specified in the first part. Both the threats and the security objectives are related to formal specifications and the correspondence section to a set of formal proofs. Finally, the security requirement specification further refines the security objectives and also corresponds to a formal specification and a proof that the security requirements indeed refine the security objectives.

The obtained final protection profile is a formal document which can be independently checked by certification authorities (such as in Germany the BSI[16] and the TÜV[17]) in order to obtain a formally certified protection profile. A developer of a specific piece of software (and hardware) can query for existing (certified) protection profiles based on the semantic descriptions of the threats and type of software contained in the documents. The protection profile can be refined to obtain a *security target* document for the developed software, and again can be independently checked by, for instance, the customers.

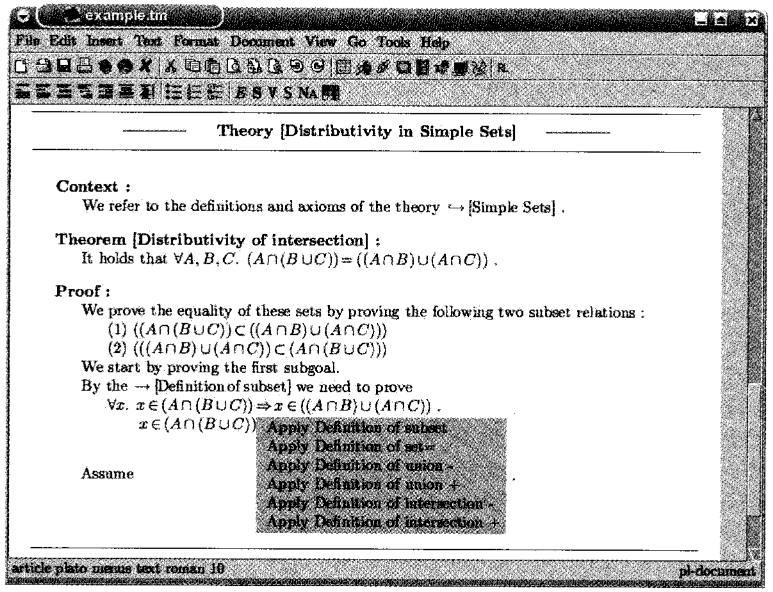
The vision is to enable a document-centric approach to formalizing and verifying mathematics and software, which is a continuation of the approach taken in Mizar[18] [41, 42]. However, unlike the approach taken in Mizar, an author shall be able to prepare a document with a standard text preparation system and without having to follow linguistic or structural restrictions. All concepts, conjectures and (partial) proofs formalized in a document should be processable by the proof assistant of her choice. The author shall be able to access the formal logical parts contained in documents from third parties by a simple citation mechanism. In turn the author can grant other persons access to her document either in the proprietary format of her text preparation system or in a semantically annotated portable print format, e.g. PDF. The proof assistant shall provide authoring support inside the text preparation tool: the type of support ranges from simple type checking, via type reconstruction for underspecified parts, to interactive and automatic proof checking and proof construction support. Any of these supports, including proof construction, shall be in a style that is intuitive even if the author is not an expert in formal logic.

As a result this allows the development of mathematical documents in a publishable style which in addition are formally validated by ΩMEGA, hence obtaining *certified mathematical documents.* A first step in that direction is currently under development by linking the WYSIWYG mathematical editor TeXmacs [49] with the ΩMEGA proof assistant and other proof assistant systems. To this end we developed

[16] www.bsi.de

[17] www.tuev.de

[18] www.mizar.org


example.tm
File Edit Insert Text Format Document View Go Tools Help
article plato menus text roman 10
pl-document
Apply Definition of subset
Apply Definition of set=
Apply Definition of union -
Apply Definition of union +
Apply Definition of intersection -
Apply Definition of intersection +

Theory [Distributivity in Simple Sets]

Context :
We refer to the definitions and axioms of the theory ↪ [Simple Sets] .

Theorem [Distributivity of intersection] :
It holds that $\forall A, B, C.\ (A \cap (B \cup C)) = ((A \cap B) \cup (A \cap C))$.

Proof :
We prove the equality of these sets by proving the following two subset relations :
(1) $((A \cap (B \cup C)) \subset ((A \cap B) \cup (A \cap C)))$
(2) $(((A \cap B) \cup (A \cap C)) \subset (A \cap (B \cup C)))$
We start by proving the first subgoal.
By the → [Definition of subset] we need to prove
$\forall x.\ x \in (A \cap (B \cup C)) \Rightarrow x \in ((A \cap B) \cup (A \cap C))$.
$x \in (A \cap (B \cup C))$
Assume .

Fig. 5. Document in $\TeX_{\text{MACS}}$ with an activated menu provided by ΩMEGA.

the PLATΩ system, that mediates between text-editors and the proof assistants [32, 50] and that maintains the consistency of the representations on either side. Furthermore, it context-sensitively relays service requests from inside the text-editor to the proof assistant and transforms any modifications done by the proof assistant into patches to the document. Fig. 5 presents a document authored in $\TeX_{\text{MACS}}$ together with a context-sensitive menu that is provided by ΩMEGA and displayed inside the text-editor.

Currently, we rely on a semantic markup attached to the different parts of the document, which must be provided manually and contains all information that is relevant to the proof assistant system, e.g. the begin of a theory, the definition of a typed constant, the structure of formulas and the different proof construction steps. The $\TeX_{\text{MACS}}$-system provides LaTeX-like editing and macro-definition features, and we provide the author a set of predefined macros to annotate her document. Only the semantic markup is handled by PLATΩ which ignores the text itself. This provides a clean interface to the proof assistant systems, that can remain fixed while one can gradually include text analysis tools to automatically create the semantic markup. This allows us to translate new textual definitions and lemmas into the formal representation, as well as to translate (partial) textbook proofs into (partial) proof plans. On the other hand, natural language generation tools can be included to present parts of proofs found by the proof assistant system in natural language inside the document. We have started to use parsers for formulas written in standard LaTeX-style syntax in $\TeX_{\text{MACS}}$ and especially to allow the user to define her own notation for the introduced syntax. The user-defined syntax is used for parsing as well as for generation of formulas obtained from the proof assistant.

3.5 E-Learning Mathematics

We are also involved in the DFKI project ACTIVEMATH, which develops an e-learning tool for tutoring students, in particular in advising a student to develop a proof. Thereby the interaction with the student should be conducted via a textual dialog. This scenario is currently under investigation in the DIALOG project [9] and, aside from all linguistic analysis problems, gives rise to the problem of *under-specification* in proofs. An overview of the ACTIVEMATH project gives [37] and more technical reports are [34, 35, 33, 48].

References

1. S. Autexier. *Hierarchical Contextual Reasoning.* PhD thesis, Computer Science Department, Saarland University, Saarbrücken, Germany, 2003.
2. S. Autexier, C. Benzmüller, A. Fiedler, H. Horacek, and Q. Bao Vo. Assertion-level proof representation with under-specification. *Electronic in Theoretical Computer Science*, 93:5–23, 2003.
3. S. Autexier, C. Benzmüller, and J. Siekmann. Computer supported mathematics with ΩMEGA. 2005.
4. S. Autexier, Chr. Benzmüller, D. Dietrich, A. Meier, and C.-P. Wirth. A generic modular data structure for proof attempts alternating on ideas and granularity. In M. Kohlhase, editor, *Proceedings of MKM'05*, LNAI 3863, IUB Bremen, Germany, January 2006. Springer.
5. S. Autexier and D. Hutter. Maintenance of formal software development by stratified verification. In M. Baaz and A. Voronkov, editors, *Proceedings of LPAR'02*, LNCS, Tbilissi, Georgia, September 2002. Springer.
6. S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The development graph manager MAYA. In H. Kirchner and C. Ringeissen, editors, *Proceedings 9th International Conference on Algebraic Methodology And Software Technology (AMAST'02)*, volume 2422 of *LNCS*. Springer, September 2002.
7. Serge Autexier. The CoRE calculus. In Robert Nieuwenhius, editor, *Proceedings of the 20th Conference on Automated Deduction (CADE)*, LNAI, Tallinn, Estonia, July 22–27, 2005. Springer.
8. Serge Autexier and Dominik Dietrich. Synthesizing proof planning methods and oants agents from mathematical knowledge. In Jon Borwein and Bill Farmer, editors, *Proceedings of MKM'06*, volume 4108 of *LNAI*, pages 94–109. Springer, August 2006.
9. C. Benzmüller, A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayova, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Quoc Vo, and M. Wolska. Tutorial dialogs on mathematical proofs. In *Proceedings of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, pages 12–22, Acapulco, Mexico, 2003.
10. C. Benzmüller, A. Meier, and V. Sorge. Bridging theorem proving and mathematical knowledge retrieval. In D. Hutter and W. Stephan, editors, *Festschrift in Honour of Jörg Siekmann's 60s Birthday*, LNAI. Springer, 2004.
11. C. Benzmüller and V. Sorge. Ωants – An open approach at combining Interactive and Automated Theorem Proving. In M. Kerber and M. Kohlhase, editors, *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*. AK Peters, 2000.

12. Chr. Benzmüller and V. Sorge. ΩANTS – an open approach at combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase, editors, *Proceedings of Calculemus-2000*, St. Andrews, UK, 6–7 August 2001. AK Peters.
13. B. Buchberger, G. Gonnet, and M. Hazewinkel. Special issue on mathematical knowledge management. *Annals of Mathematics and Artificial Intelligence*, 38(1-3): 3–232, May 2003.
14. J. Carbonell, J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, Craig. Knoblock, S. Minton, M. A. Pérez, S. Reilly, M. Veloso, and X. Wang. PRODIGY 4.0: The Manual and Tutorial. CMU Technical Report CMU-CS-92-150, Carnegie Mellon University, June 1992.
15. L. Cheikhrouhou and V. Sorge. PDS — A Three-Dimensional Data Structure for Proof Plans. In *Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*, Monastir, Tunisia, 22–24 March 2000.
16. A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.
17. M. Davis, editor. *The Undecidable: Basic Papers on undecidable Propositions, unsolvable Problems and Computable Functions.* Raven Press Hewlett, New York, 1965.
18. M. Davis. The prehistory and early history of automated deduction. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning*, volume 2 Classical Papers on Computational Logic 1967–1970 of *Symbolic Computation.* Springer, 1983.
19. M. Davis. The early history of automated deduction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 1, pages 3–15. Elsevier Science, 2001.
20. D. Dietrich. The task-layer of the ΩMEGA system. Diploma thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2006.
21. The Doris system is available at www.cogsci.ed.ac.uk/~jbos/doris, 2001.
22. K. Erol, J. Hendler, and D. Nau. Semantics for hierarchical task network planning. Technical Report CS-TR-3239, UMIACS-TR-94-31, Computer Science Department, University of Maryland, March 1994.
23. A. Fiedler. Dialog-driven adaptation of explanations of proofs. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1295–1300, Seattle, WA, 2001. Morgan Kaufmann.
24. A. Fiedler. P.rex: An interactive proof explainer. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI. Springer, 2001.
25. A. Fiedler. *User-adaptive proof explanation.* PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.
26. A. Franke and M. Kohlhase. System description: MBase, an open mathematical knowledge base. In D. McAllester, editor, *Proceedings of the 17th Conference on Automated Deduction*, number 1831 in LNAI. Springer, 2000.
27. Andreas Franke and Michael Kohlhase. System description: Mbase, an open mathematical knowledge base. In David McAllester, editor, *17th International Conference on Automated Deduction CADE-17, Pittsburgh, USA, June 17-20, 2000*, volume 1831 of *Lecture notes in computer science.* Springer, 2000.
28. M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation.* Springer Verlag, 1979. LNCS 78.
29. X. Huang. Reconstructing Proofs at the Assertion Level. In A. Bundy, editor, *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI, pages 738–752. Springer, 1994.

30. D. Hutter. Management of change in structured verification. In *Proceedings of Automated Software Engineering, ASE-2000.* IEEE, 2000.
31. M. Kohlhase and A. Franke. MBASE: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation; Special Issue on the Integration of Computer Algebra and Deduction Systems*, 32(4):365–402, September 2001.
32. Christoph Benzmüller, Marc Wagner, Serge Autexier. Plato: A mediator between text-editors and proof assistance systems. In Christoph Benzmüller, Serge Autexier, editor, *7th Workshop on User Interfaces for Theorem Provers (UITP'06)*, ENTCS. Elsevier, August 2006.
33. E. Melis and E. Andrés. Global feedback in ActiveMath. 2003.
34. E. Melis, E. Andrés, J. Bdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. ActiveMath: A generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education*, 12:385–407, 2001.
35. E. Melis, J. Büdenbender E. Andrés, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Knowledge representation and management in ActiveMath. *Annals of Mathematics and Artificial Intelligence, Proceedings of the first Conference on Mathematical Knowledge Management MKM'01*, 38:47–64, 2003.
36. E. Melis and A. Meier. Proof planning with multiple strategies. In J. Loyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagivand, and P. Stuckey, editors, *First International Conference on Computational Logic (CL-2000)*, number 1861 in LNAI, pages 644–659, London, UK, 2000. Springer.
37. E. Melis and J. Siekmann. E-learning logic and mathematics: What we have and what we need. In *Festschrift to Dov Gabbay.* Kings College Publication, 2005.
38. Till Mossakowski, Serge Autexier, and Dieter Hutter. Extending development graphs with hiding. *Journal of Logic and Algebraic Programming, special issue on Algebraic Specification and Development Techniques*, 2005.
39. Common Criteria Project Sponsoring Organisations. Common criteria for information technology security evaluation (CC) version 2.1, 1999. Also ISO/IEC 15408: IT – Security techniques – Evaluation criteria for IT security.
40. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
41. P. Rudnicki. An overview of the MIZAR-project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs.* Chalmers University of Technology, 1992.
42. P. Rudnicki and A. Trybulec. On equivalents of well-foundedness. An experiment in MIZAR. *Journal of Automated Reasoning*, 23:197–234, 1999.
43. J. Siekmann. Geschichte des automatischen beweisens (history of automated deduction). In *Deduktionssysteme, Automatisierung des Logischen Denkens.* R. Oldenbourg Verlag, 2nd edition, 1992. Also in English with Elsewood.
44. J. Siekmann. History of computational logic. In D. Gabbay and J. Woods, editors, *The Handbook of the History of Logic*, volume I-IX. Elsevier, 2005.
45. J. Siekmann and C. Benzmüller. Omega: Computer supported mathematics. In G. Palm S. Biundo, Ṫ. Frhwirth, editor, *KI 2004: Advances in Artificial Intelligence: 27th Annual German Conference on AI*, number 3228 in LNAI, pages 3–28, Ulm, Germany, 2004.
46. J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, and V. Sorge. $\mathcal{LOUI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Aspects of Computing*, 11:326–342, 1999.
47. V. Sorge. *A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning.* PhD thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, November 2001.

48. C. Ullrich, P. Libbrecht, S. Winterstein, and M. Mhlenbrock. A flexible and efficient presentation-architecture for adaptive hypermedia: Description and technical evaluation. pages 21–25. IEEE Computer Society, 2004.
49. J. van der Hoeven. GNU TeXmacs: A free, structured, wysiwyg and technical text editor. In *Actes du congrès Gutenberg*, number 39-40, in Actes du congrès Gutenberg, pages 39–50, Metz, May 2001.
50. Marc Wagner. Mediation between text-editors and proof assistance systems. Diploma thesis, Saarland University, Saarbrücken, Germany, 2006.
51. F. Wiedijk. Formal proof sketches. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs: Third International Workshop, TYPES 2003*, LNCS 3085, pages 378–393, Torino, Italy, 2004. Springer.
52. J. Zimmer and M. Kohlhase. System description: The Mathweb Software Bus for distributed mathematical reasoning. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in LNAI, pages 138–142. Springer, 2002.
53. Jürgen Zimmer. MATHSERVE – *A Framework for Semantic Reasoning Services.* PhD thesis, FR 6.2 Informatik, Universität des Saarlandes, 2007.
54. Jürgen Zimmer and Serge Autexier. The MATHSERVE framework for semantic reasoning web services. In U. Furbach and N. Shankar, editors, *Proceedings of IJCAR'06*, LNAI, pages 140–144, Seattle, USA, August 2006. Springer.

Closing the Gap Between Formal and Digital Libraries of Mathematics

Jeremy Gow and Paul Cairns

UCL Interaction Centre, London
j.gow@ucl.ac.uk
p.cairns@ucl.ac.uk

Abstract. The representational gap between formal mathematics and most users of digital mathematics resources is a challenge for any approach to mathematical knowledge management which aims to combine the benefits of formal and informal mathematics. In this chapter we study this gap in the context of a digital library of mathematics based on the Mizar Mathematical Library and make recommendations for improving such formal systems support for MKM.

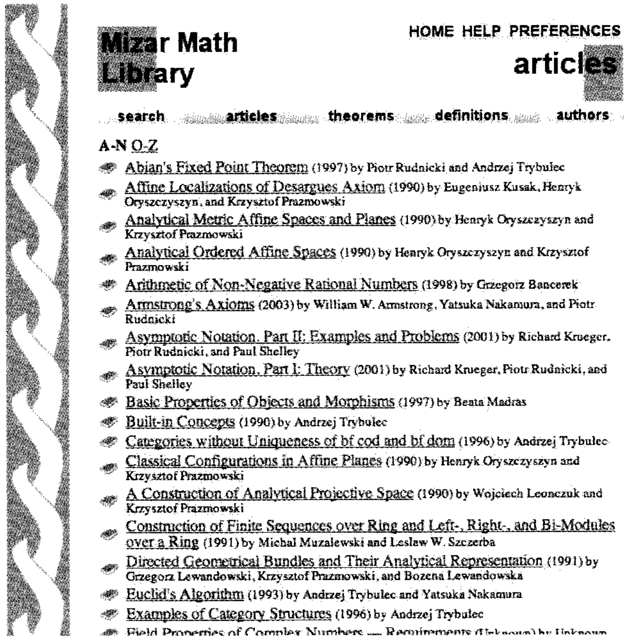
1 Introduction

The aim of mathematical knowledge management (MKM) is to organise a substantial proportion of all mathematical knowledge to make it more easily and widely available [7]. Much of the current effort in this area could be called *formalist*, in that it is based upon formal representations of mathematical content in various logics [6]. The formalist approach has great potential, as mathematical proof can be guaranteed valid and content is far more amenable to automatic processing.

A key problem for formalist MKM is how to account for the *representational gap* between formal and informal mathematics: how can such technology find a role in mainstream mathematics, given that the vast majority of mathematics is carried out in non-logical representations such as specialist natural language, formulae and diagrams? Most users of mathematics are not versed in formal mathematics and, even if they were, it is not yet clear that it could support their activities adequately. To be clear, translation across the representational gap is generally possible *in principle*. In practice, it requires skilled effort for either a person or a machine (e.g. [21, 27]).

In this chapter we suggest that this gap is best closed by a variant of formalist MKM where formal and informal representations are distinct but closely integrated. In order to illustrate the challenges faced by this approach, we present Mizone, a digital library of mathematics based on the Mizar Mathematical Library (MML) [16]. The collection — shown in Figure 1 — demonstrates the expected gap between a formal mathematics library and a digital library system, and we discuss what needs to be done to close the gap in this context.

ISBN 978-83-7431-128-1 ISSN 0860–150X



A-N O-Z

- Abian's Fixed Point Theorem (1997) by Piotr Rudnicki and Andrzej Trybulec
- Affine Localizations of Desargues Axiom (1990) by Eugeniusz Kusak, Henryk Oryszczyszyn, and Krzysztof Prazmowski
- Analytical Metric Affine Spaces and Planes (1990) by Henryk Oryszczyszyn and Krzysztof Prazmowski
- Analytical Ordered Affine Spaces (1990) by Henryk Oryszczyszyn and Krzysztof Prazmowski
- Arithmetic of Non-Negative Rational Numbers (1998) by Grzegorz Bancerek
- Armstrong's Axioms (2003) by William W. Armstrong, Yatsuka Nakamura, and Piotr Rudnicki
- Asymptotic Notation. Part II: Examples and Problems (2001) by Richard Krueger, Piotr Rudnicki, and Paul Shelley
- Asymptotic Notation. Part I: Theory (2001) by Richard Krueger, Piotr Rudnicki, and Paul Shelley
- Basic Properties of Objects and Morphisms (1997) by Beata Madras
- Built-in Concepts (1990) by Andrzej Trybulec
- Categories without Uniqueness of bf cod and bf dom (1996) by Andrzej Trybulec
- Classical Configurations in Affine Planes (1990) by Henryk Oryszczyszyn and Krzysztof Prazmowski
- A Construction of Analytical Projective Space (1990) by Wojciech Leonczuk and Krzysztof Prazmowski
- Construction of Finite Sequences over Ring and Left-, Right-, and Bi-Modules over a Ring (1991) by Michal Muzalewski and Leslaw W. Szczerba
- Directed Geometrical Bundles and Their Analytical Representation (1991) by Grzegorz Lewandowski, Krzysztof Prazmowski, and Bozena Lewandowska
- Euclid's Algorithm (1993) by Andrzej Trybulec and Yatsuka Nakamura
- Examples of Category Structures (1996) by Andrzej Trybulec
- Field Properties of Complex Numbers — Requirements (Unknown) by Unknown

Fig. 1. Browsing the article list in Mizone.

Our main conclusion is that formal material needs to be supplemented by additional information if it is to be made palatable — and we make some specific recommendations for Mizar, the MML and similar formal systems. We also discuss how attitudes to formalisation need to change if the integrated approach to MKM is to succeed.

2 Minding the Gap

Opinions differ as to if and how the MKM's representational gap should be closed, but we distinguish four fundamental stances here. These caricatures are not necessarily mutually exclusive, and in reality different approaches to MKM might mix them to varying degrees. Note also that this kind of labeling comes with a danger of misrepresentation: in much work on MKM the attitude towards the representational gap is either implicit, agnostic or not present.

The first — *strong formalism* — is that mainstream mathematics should adopt formal mathematics. Mathematicians and other users of mathematics can reap the rewards of the formalist approach by adopting it directly into their working practices. At one extreme, all mathematicians would become users of proof assistants and other systems which, while they may be more advanced than existing systems, require the user to work directly with formal content. MKM would simply become a matter of organising this formal material. Many MKM researchers would reject this strong version of formalism: mathematical practice is unlikely to change this significantly, and formal representations are considered too difficult to work with for widespread adoption. For instance, proofs become much longer when formalised [24].

Strong informalism provides MKM solutions using a combination of mainstream mathematical practice and generic knowledge management technology, possibly specialised to the domain but without recourse to formal representations. Currently all mainstream digital mathematical resources fall into this category, such as digital libraries of journal articles. Informalism does not imply that the mathematical content is not rigorous or is unrelated to formal mathematics, but that it has not actually been formalised itself. Again, many MKM researchers would reject the exclusive use of this approach: the widespread adoption of the formalist approach shows that as formal mathematics is viewed as a keystone to MKM. It is both explicit in its content and moreover is provably correct knowledge. Thus, unlike the more informal communications between mathematicians [27], there can be no mistaking the meaning of each theorem.

Both strong formalism and informalism ignores the gap and stay put on one side, and neither currently appeals to the MKM community. We distinguish two further approaches which attempt to bridge the gap: *pluralism* and *integrationism.* Here both formal and informal representations play a role, and it is up to people or A.I. technology to translate between the two. In a pluralist approach formal and informal mathematical activities exist in tandem, but are only weakly connected. For example, a journal article may be formalised and validated by a third-party author. Here MKM involves both formal and informal content, but each is part of separate activities which may involve different people with very different aims. Links between formal and informal content are possible, but generally ad hoc. Following this approach formal mathematics has so far a small impact on MKM via the special cases it has been 'shipped in' — a large scale example of this is the Flyspeck project, which hopes to validate Hales' proof of the Kepler Conjecture using HOL Light [12].

Finally, *integrationism* demands that both formal and informal representations are used and are tightly coupled. In this approach most MKM content has at least one formal and one informal representation which are explicitly related, or at least translation services are available that can provide alternate representations when required. A great deal of MKM research supports or is compatible with this approach. There is work on translation between informal and formal representations [27,21]. Another key idea is the vision that the same services and databases can provide solutions for a wide range of users and an applications, e.g. dynamically presented content into whatever form is most useful for particular users, using the presentational conventions of different disciplines or cultures. Many MKM researchers seem to favour some kind of integrated approach, at least in theory.

Of course, real MKM solutions may not conform to these neat positions. For example, Barendregt and Wiedijk's vision of computer mathematics [5] argues for improvements to proof assistants necessary for their adoption by mathematicians, including a more mathematical style of expression and supporting reasoning with gaps. This could be labeled formalist with informalist influence: a logical representation is made 'more informal' in various ways. Although only a small a representational gap seems to be anticipated, their proposal is also integrationist, in that these representations are bound together within a single system.

We would argue that, in the long term, any approach to MKM needs to decide to what degree their users and systems are formalist or informalist, and if a representational gap remains how strongly integrated the two sides will be. The current trend in MKM is towards some kind of integrationist approach, as opposed to a completely pluralist one.

3 An Integrated Approach in Mizar

From the integrationist MKM stance, the rest of the chapter examines some of the problems with an integrated approach, taking the Mizar Mathematical Library (MML) [16] as a exemplar of the formalist approach. Arguably, the MML offers the largest resource of formally proven mathematics. It covers a wide range of topics in from computation through abstract algebra to topology and set theory. Moreover, Mizar uses a rich language that can strongly resemble the natural usage of mathematicians, so the representational gap can be considered slightly narrower than in other systems. With some further processing, this presentation can even be made more natural — as is done for the Journal of Formalised Mathematics [19].

As a mature system Mizar supports its target user very well — one comfortable (or learning to become so) with the Mizar system, language and library and associated tools such as Mizar Emacs mode and MML Query [4]. Such people are actively engaged with developing there own articles, and quite likely in collaboration or contact with Mizar experts or developers. These users have rightly been the focus of Mizar's developers, who are engaged with advancing the state of computer-supported formal mathematics.

However, the audiences and applications for MKM are considerably more diverse than this. For instance, scientists, engineers, economists and the like who need to find the to tackle their particular problems. Here we focus on the dissemination and archiving of mathematical material for professional mathematicians and mathematics students in higher education — a suitable digital library application for a core group of mathematics users. This is a larger and ill-defined than the Mizar user group, but they share a couple of traits. First, such people are interested in proof-centred mathematical content. Secondly, most have little or no understanding or interest in formal mathematics per se [8]. Such people may be step to a wider group of MKM users.

It is clear that there is a significant representational gap between this MKM user group and the MML. To better understand the issues and possible solutions in this context, we used Greenstone [25], a digital library system, to build a simple digital collection based on the MML. Digital libraries are a standard means of delivering collections of (informal) content electronically. They are used widely in publishing diverse collections and are a proven technology that provides valuable resources to knowledge workers in many disciplines [1, 18, 17]. By forcing a library of formal mathematics into a digital library system we hoped to illustrate some aspects of the current representational gap between the MML and MKM users. As one might expect, the result — the *Mizone* collection — is not a particularly usable or useful resource for mainstream MKM. However, the aim here is to highlight the areas in which further effort and research might close the gap.

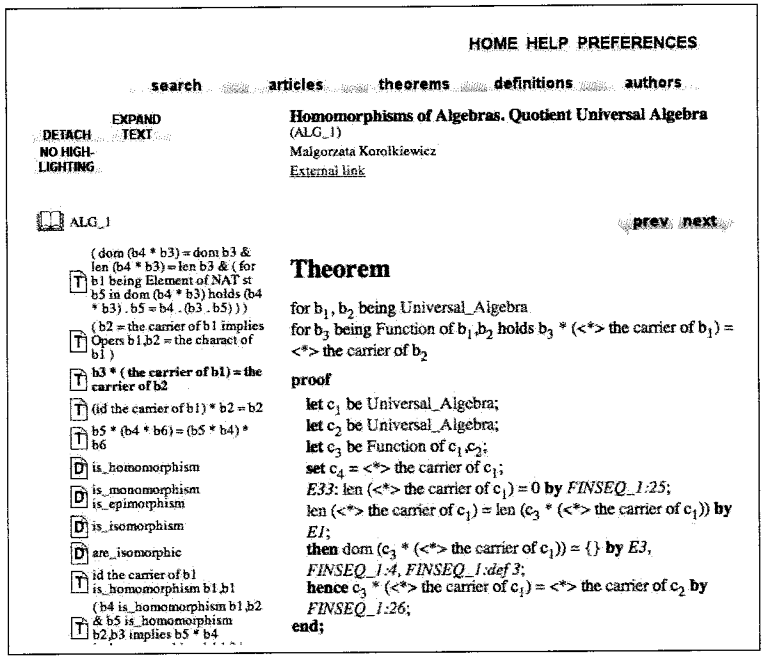
4 Developing the Collection

Greenstone is an open source digital library platform [25, 26]. It was developed as a general research tool for digital library technology and usage, but its flexibility and low entry costs have led to its adoption for numerous real-world projects in a wide range of application areas, such a collections on Maori history, Chopin's music and accounts of the impact of the hurricane Katerina. In 2000, it was adopted by UNESCO as a means to distribute important knowledge on sustainable development to the developing world. This has led to the translation of the Greenstone interface and manuals into forty three different languages. As the MML is a non-standard data set for digital libraries, it made sense to use such a well established and versitile digital library system. The interested reader can find out more about Greenstone, its collections and download the software from `www.greenstone.org`.

The MML consists of a Mizar articles, each made up of a series of items. Each MML article maps to a document within Mizone (see Figure 1), and it was our initial aim to include the entire library. Greenstone allows documents to have a hierarchy of sections and subsections. Each imported Mizar item was mapped to its own section, to allow for theorems and definitions to be individually retrieved and browsed. Not all items were included in Mizone documents. The key concepts in any Mizar article are its definitions and theorems, while we omitted other items which dealt with more the technical details of formalisation and validation within Mizar. The item types which were imported into the Mizone were: `DefinitionBlock`, `NotationBlock`, `JustifiedTheorem`, `Proposition` and `Proof`.

The textual presentation of each item was based on the MML's standard semantic presentation stylesheet [23]. This renders the item as HTML that closely resembles the original Mizar language. It is clear that this presentation could be improved by automatically replacing Mizar keywords with stock natural language phrases, as is done in the Journal of Formalized Mathematics. This is a possible future improvement that will enhance Mizone's readability, but from a theoretical perspective it will probably not tell us much. It was done in a few cases in Mizone as a proof-of-principle, but most of the text is currently in the Mizar style, as shown in Figure 2.

The actual import process automated via scripts, and proceeded as follows: Mizar was run over the MML article, and as a side effect generates an XML representation. A modified version of the semantic presentation stylesheet was then used to generate a HTML version of the article, annotated with sectioning information for Greenstone. In a separate process, HTML indexes of the MML and JFM were used to gather metadata about each article, which was merged into a single Greenstone metadata file. The Mizone was then built from the HTML plus metadata using Greenstone. The work was done on a G5 Mac, but a similar process should be possible under Windows or Linux, as both Mizar and Greenstone are support multiple platforms.



Homomorphisms of Algebras. Quotient Universal Algebra
(ALG_1)
Malgorzata Korolkiewicz
External link

ALG_1

- T (dom (b4 * b3) = dom b3 & len (b4 * b3) = len b3 & (for b1 being Element of NAT st b5 in dom (b4 * b3) holds (b4 * b3) . b5 = b4 . (b3 . b5)))
- T (b2 = the carrier of b1 implies Opers b1,b2 = the charact of b1)
- T **b3 * (the carrier of b1) = the carrier of b2**
- T (id the carrier of b1) * b2 = b2
- T b5 * (b4 * b6) = (b5 * b4) * b6
- D is_homomorphism
- D is_monomorphism is_epimorphism
- D is_isomorphism
- D are_isomorphic
- T id the carrier of b1 is_homomorphism b1,b1
- T (b4 is_homomorphism b1,b2 & b5 is_homomorphism b2,b3 implies b5 * b4

Theorem

for b_1, b_2 being Universal_Algebra
for b_3 being Function of b_1,b_2 holds b_3 * (<*> the carrier of b_1) = <*> the carrier of b_2

proof
  let c_1 **be** Universal_Algebra;
  let c_2 **be** Universal_Algebra;
  let c_3 **be** Function of c_1,c_2;
  set c_4 = <*> the carrier of c_1;
  E33: len (<*> the carrier of c_1) = 0 **by** *FINSEQ_1:25*;
  len (<*> the carrier of c_1) = len (c_3 * (<*> the carrier of c_1)) **by** *E1*;
  then dom (c_3 * (<*> the carrier of c_1)) = {} **by** *E3*, *FINSEQ_1:4*, *FINSEQ_1:def 3*;
  hence c_3 * (<*> the carrier of c_1) = <*> the carrier of c_2 **by** *FINSEQ_1:26*;
end;

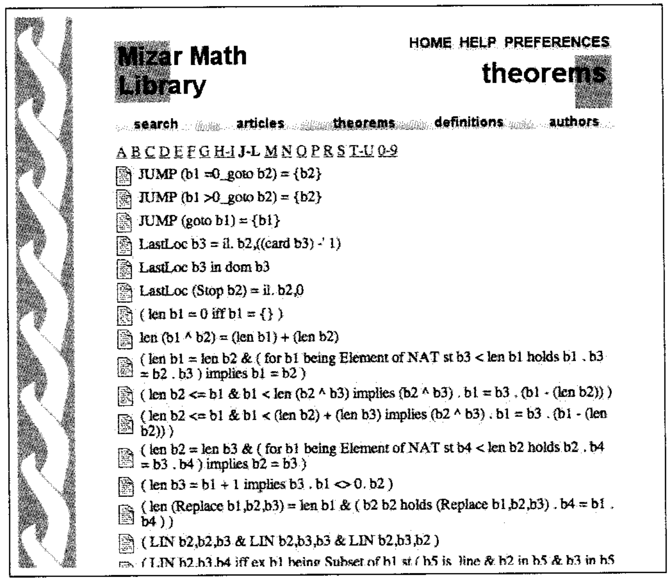
Fig. 2. A theorem in Mizone

Under the default configuration Greenstone is unable to index the entire MML, a problem which can occur with very large collections. This is probably due to the quantity of individual terms to be indexed, rather than the actual size of the collection. The current collection is a substantial subset of the MML, but we hope eventually to include the entire library.

5 Using the Collection

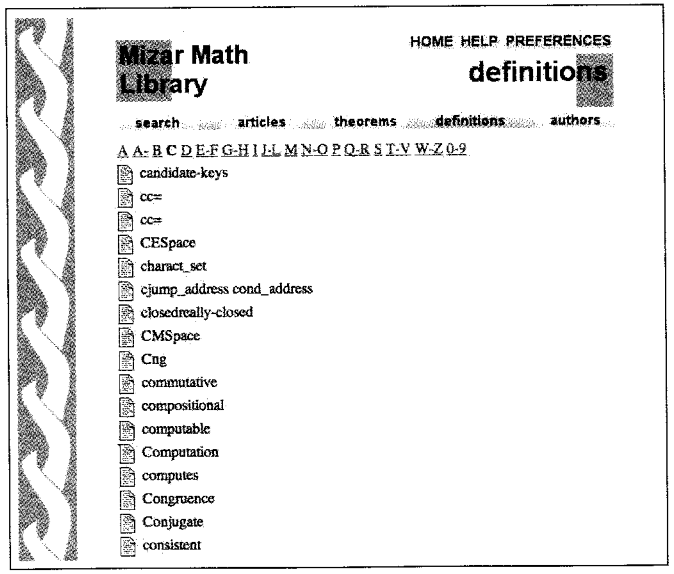
As with all Greenstone collections, Mizone is accessed by a web interface by default, with the initial page consisting of a simple search box and some instructions for using the system. Greenstone supports browsing of documents by *classifiers*, where a list of values is displayed and selecting a value brings up a sublist of related articles. Mizone currently supports browsing by article title, definitions, theorems and article authors. Browsing modes and search are selected via a horizontal classifier/search menu bar that is always visible. Browsing by article title, theorem statement and defined symbol are shown in Figures 1, 4 and 3 respectively. The results of a keyword search are shown in Figure 5.

Selecting an article during browsing or via search results brings up an article page, as shown in Figures 2 and 6. Below the classifier/search bar is a description of the article: title, MML name and authors. Most articles include an external link here to their JFM page. Down the left of the page is a section navigation bar,



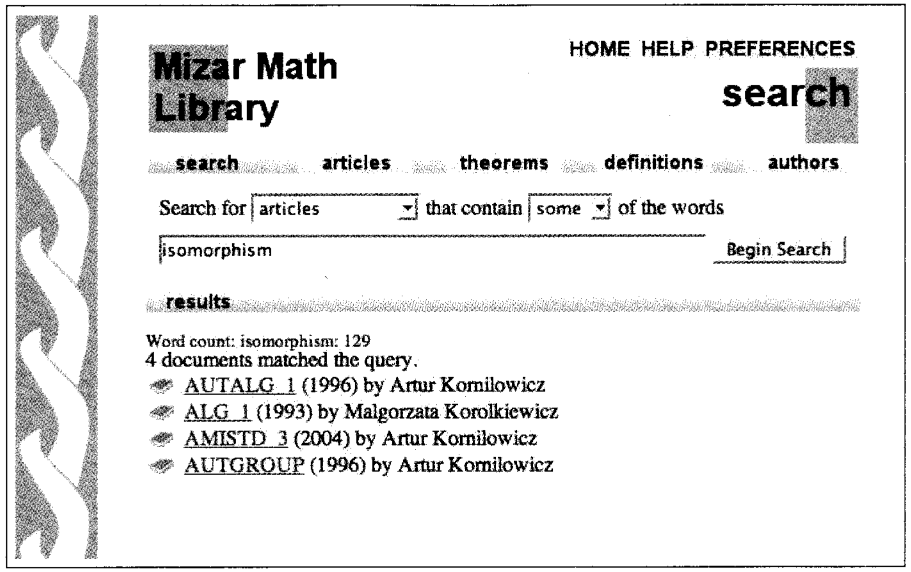
- JUMP (b1 =0_goto b2) = {b2}
- JUMP (b1 >0_goto b2) = {b2}
- JUMP (goto b1) = {b1}
- LastLoc b3 = il. b2,((card b3) -' 1)
- LastLoc b3 in dom b3
- LastLoc (Stop b2) = il. b2,0
- (len b1 = 0 iff b1 = {})
- len (b1 ^ b2) = (len b1) + (len b2)
- (len b1 = len b2 & (for b1 being Element of NAT st b3 < len b1 holds b1 . b3 = b2 . b3) implies b1 = b2)
- (len b2 <= b1 & b1 < len (b2 ^ b3) implies (b2 ^ b3) . b1 = b3 . (b1 - (len b2)))
- (len b2 <= b1 & b1 < (len b2) + (len b3) implies (b2 ^ b3) . b1 = b3 . (b1 - (len b2)))
- (len b2 = len b3 & (for b1 being Element of NAT st b4 < len b2 holds b2 . b4 = b3 . b4) implies b2 = b3)
- (len b3 = b1 + 1 implies b3 . b1 <> 0. b2)
- (len (Replace b1,b2,b3) = len b1 & (b2 b2 holds (Replace b1,b2,b3) . b4 = b1 . b4))
- (LIN b2,b2,b3 & LIN b2,b3,b3 & LIN b2,b3,b2)
- (LIN b2,b3,b4 iff ex b1 being Subset of b1 st (b5 is line & b2 in b5 & b3 in b5

Fig. 3. Browsing the theorem list in the Mizone



- candidate-keys
- cc=
- cc=
- CESpace
- charact_set
- cjump_address cond_address
- closedreally-closed
- CMSpace
- Cng
- commutative
- compositional
- computable
- Computation
- computes
- Congruence
- Conjugate
- consistent

Fig. 4. Browsing the definition list in the Mizone



Search for articles that contain some of the words

isomorphism

Word count: isomorphism: 129
4 documents matched the query.

- AUTALG_1 (1996) by Artur Kornilowicz
- ALG_1 (1993) by Malgorzata Korolkiewicz
- AMISTD_3 (2004) by Artur Kornilowicz
- AUTGROUP (1996) by Artur Kornilowicz

Fig. 5. Searching in Mizone

with rest of the page initially blank and used to display the current section when one is selected. Previous/next arrows are displayed to allow the reader to progress through the sections in order.

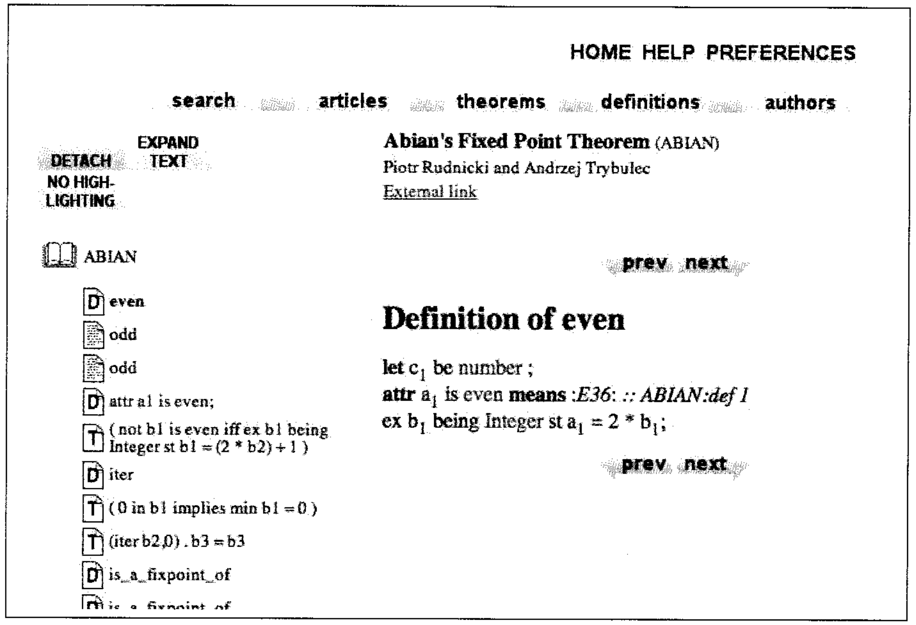
The Greenstone import process preserves HTML links between articles, but unfortunately does not currently preserve links to individual items within articles. Instead, the link takes the reader to the top of the item's article.

6 Critique

We now evaluate Mizone as an information resource, considering the user group of mathematics professionals. Recall that the goal is not to produce a fully usable system that could be presented to such users but rather a functioning prototype that highlights issues with a integrated approach to MKM. This is not to say that the prototype should be unusable and we would hope that those comfortable with the formal mathematics of the MML would find Mizone useful.

The questions arising from Mizone are therefore about the relationship between the formal mathematics and the more general requirements of a mathematical knowledge repository: How does it compare to other mathematical resources, formal and informal? How does it compare to other resources based on the MML? What does this tell us about formal mathematics and digital library design? What role can formal mathematics play in digital mathematics libraries? How can the representational gap in Mizone be closed?

We discuss each of our main areas of concern about Mizone below.



Abian's Fixed Point Theorem (ABIAN)
Piotr Rudnicki and Andrzej Trybulec
External link

ABIAN
- D even
- odd
- odd
- D attr a1 is even;
- T (not b1 is even iff ex b1 being Integer st b1 = (2 * b2) + 1)
- D iter
- T (0 in b1 implies min b1 = 0)
- T (iter b2,0) . b3 = b3
- D is_a_fixpoint_of
- D is_a_fixpoint_of

Definition of even

let c_1 be number ;
attr a_1 is even **means** *:E36: :: ABIAN:def 1*
ex b_1 being Integer st $a_1 = 2 * b_1$;

Fig. 6. A definition in Mizone.

6.1 Presentation of Syntax

As mentioned in §4, article items are presented in a similar style to the Mizar input syntax, reconstructed from the XML output of Mizar. It is already well-understood that readability can be greatly improved by keyword replacement, i.e. replacing formal syntax with stock phrases in natural language, as is done in the JFM [19]. This does not tell us anything new about the representational gap, but the problem is still severe in Mizone. Although the Mizar language is relatively readable, its target audience would find it quite unnatural to use formal syntax throughout, even though they may be familiar with similar notation in their mathematics. We hope to deal with this in a future version of Mizone.

A more serious problem is the verbosity of the syntax [24]— a problem which keyword replacement would only make worse. Rendering formal mathematics in natural language using more sophisticated techniques has already been identified as a research topic by MKM, and this has the potential to generate reasonable length texts. Such an approach is badly needed here.

Another issue, already discussed in [9], is that the original Mizar article is not faithfully reproduced by Mizar's XML output. It loses the original variable names and precise syntactic structure of terms, as they are irrelevant to its function as a proof checker. Although Mizone is built upon a logically identical XML representation, it is the actual language used that can often provides important cues to the reader about what the mathematics is about [27]. Symbols, syntax and terms are important parts of that.

6.2 Informal Commentary

Other than the syntax, the most noticeable feature of Mizone is the lack of any commentary about the mathematics it presents. This kind of material is crucial for our user group, who are used to the more informal 'glue' content that provides examples, motivation and significance that are necessary for the mathematician but not the mathematics. A short-term solution for Mizone would be to import the prose abstracts from the JFM [19], though this would only provide a limited amount of commentary for each article.

Interleaving of such commentary material with formal mathematics is allowed in some systems, such as the Formal Digital Library Navigator [14], but is not practiced in the MML. This is the basic principle behind literate programming [13, 22], which has been applied to formal mathematics in the `maze` system [10]. This approach places reader-oriented content in comments, to be ignored by Mizar but to be interpreted as commands for a presentation system like Greenstone.

Problems arise when referential links need to cross the formal/informal boundary. For instance, commentary on a theorem should be explicitly linked to the formal theorem object in some way, rather than just collocated. Likewise, discussion of mathematical examples would benefit by being linked to formal example objects whose properties could be formally checked as well as being informally discussed. It is interesting that examples are part of the MML formalised mathematics, but not used to explicitly illustrate other material.

6.3 Presentation of Theorems and Definitions

The names of theorems and definitions are used in Mizone as a title for each item's section, which appears as a title on that section's page and in the section menu. These names are currently extracted from the corresponding formal object. In the case of theorems, the basic statement of the theorem without quantification is given. For definitions (and redefinitions), the defined symbol is used.

For theorems, this approach has the disadvantage that the theorem statement may be too lengthy to make a comprehensible and recognizable label. Even better syntactic processing — such as dropping preconditions — will only work in some cases. A problem for both theorems and definitions is that, even when short, the label may still be too obscure for readers not familiar with the formalisation. An obvious fix is to have authors provide additional natural language labels for all formal objects, along with an short version to be used in menus.

We described in §4 how we selected particular kinds of formal items to be included in Mizone, e.g. `DefinitionBlock`. Unfortunately, this meant every one of these items was included, irrespective of its importance to the article, and that other kinds of item were totally inaccessible, preventing the interested reader from finding out the details of the formalisation. It would clearly be preferable if this selection could be made more by the original author, and changed dynamically by the reader.

6.4 Article Structure

Articles however have no explicit internal structure other than that inherited from the Mizar language. Mizone reflects the breakdown of articles into theorems and definitions determined by that language, and this fits well with mathematical practice. Notionally, there are sections in the original Mizar articles, but as with syntactic forms (see §6.1) these are omitted from the XML output and hence not in Mizone. A not untypically lengthy article like *Armstrong's Axioms* [2] show the consequences of this very flat hierarchical structure: the side menu that is meant to aid navigation goes on for several screens. It is not clear which items are key, which are not (e.g. lemmas) and what the logical or thematic grouping is between any of them.

This presents a problem for the reader to trying to form a coherent perception of the whole article rather than any particular constituent. Appropriate thematic sections are common practice in mathematics and would alleviate this Problems such as these are not unique to the MML, and have arisen in other Greenstone collections, but they are particularly acute in a mathematical context where users will often want to navigate between sections for reference. In fact, many MML articles do contain appropriate sections in comments — but as explained in §6.1, these are not accessible in Mizar's XML output.

An issue that does seem unique to mathematics is that there is a huge amount of cross-referencing between sections and articles. This can be definitions building on previous definitions or proofs that can employ existing theorems both within a given article and between articles. In our experience, no other Greenstone collection has required such dense cross-referencing. As mentioned in §4, the system does not currently handle direct navigation into subsections when the user follows a linked cross-reference. As discussed in [8], the presence of many cross-reference links does not help the reader identify which links are important for understanding the mathematics. It would be better if authors could include this information in their original articles.

6.5 Proof Structure

Mizone uses indenting to show the hierarchical structure of proofs. However, it is clear that long proofs — for instance, Abian:8 [20] — with deep hierarchies present a challenge to the reader in terms of understanding the proof as a coherent whole. In more informal mathematics, it is common to split up long proofs into smaller sections or, where that is not possible or reasonable, to provide indications of where in the progression of the proof the reader is. Such structuring information could be added by the author, though it is not clear what form this could take.

Another possibility is interactively represent the hierarchical structure of the proof as in Lamport's method [15]. It would be possible to add such interactivity to Mizone by customising Greenstone's interface with appropriately scripted HTML. This could help manage the complexity of proofs, and allow proof overviews to be expanded on demand. However, in a previous study we found that interactive Lamport proofs may present problems for some mathematical readers [8] — if so, this would be second-best to more explicit attempt to guide the presentation of the proof. It may be that no matter what informal content is included in an article, the structure of a Mizar proof is always going to jar with the expectations of mathematicians just as the Lamport presentational style may. This issue can only be resolved by implementing and developing systems such as Mizone and then evaluating them with mathematicians.

6.6 Search

Perhaps the biggest disadvantage of Mizone is that it has no specialised search tools. Over the past few years, significant development effort has gone into producing tools that are valuable to Mizar author, like MMLQuery [3, 4]. This allows users to search the MML in sophisticated ways such as looking for the close cooccurrence of terms in Mizar articles.

Mizone is currently only using a standard search tool that looks for terms occurring in documents. The collection is configured to allow keyword search over articles, article titles, authors, theorem titles and definition titles. Search over multiple fields is not currently supported by the collection. Keyword search combined with a lack of informal labels for items (see 6.3) means the user has to guess the particular syntax, and that search for terms is extremely limited.

It is not clear what would make an appropriate search tool for mathematicians using Mizone. Keyword search may be more effective in this context if informal commentaries (see §6.2) and similar material were available for indexing. Tools like MMLQuery, though effective for formal content, have a complex syntax of their own and are partly dependent on a knowledge of formal mathematics. More informal yet general methods, such as that used in the Alcor system, present an alternative approach [11]. Overall though, good searching presents issues of how dependent a search tool needs to be on the particular language of the mathematics and more importantly what the goals of the users are. In essence then, resolving this is one of the major goals of MKM generally.

Bespoke search interfaces can be added to Greenstone with some effort, as has been done for a music collection with both a musical keyboard and audio input methods. Thus, it should be possible to have Greenstone versions of existing search tools such as Alcor and MMLQuery. This could form the basis for evaluating the value of such tools with a more general audience.

7 Recommendations

As well as indicating how Mizone could be further developed, our critique contained several insights into the representational gap between the digital collection and its intended readers. We summarise these here as a set of recommendations that will help close this gap between libraries of formal mathematics and users of mathematical digital libraries. Firstly, we have recommendations for systems and libraries that support formal mathematics, including Mizar and the MML. These are not criticisms of Mizar as a system, but of recommendations for formal mathematics within MKM:

- Documents export formats (e.g. XML) should allow the author's original input article to be fully reconstructed. (§6.1, §6.4)
- Informal commentary material should be included for general readers. (§6.2)
- Formal objects that represent examples should be linked to any other material illustrate. (§6.2)
- Clear natural language labels should be assigned to definitions, theorems etc., with additional short labels to be used for menus. (§6.3)
- Indicate which formal objects should be presented to the general reader, and which are trivial or unnecessary details. (§6.3)
- Group items into thematic sections (§6.4)
- Indicate which cross-references between formal objects are significant for understanding, and which are merely book-keeping. (§6.5)

Secondly, we have recommendations for MKM research in general. Some work has already begun to address these questions, but they are significant challenges for MKM:

- How can formal mathematics be presented in natural language to effectively handle verbosity and clarity. (§6.1)
- Can literate proving support more flexible cross-referencing between informal and formal content? (§6.2)
- How can authors structure or annotate proofs so that proof presentations are more readable? (§6.5)
- Are interactive hierarchical proof suitable for the general reader? (§6.5)
- How can formal mathematics search tools be made more accessible to non-specialists? (§6.6)

8 Conclusions

Building and assessing Mizone has been a useful exercise in understanding the representational gap between formal and mainstream mathematics. We argued that this is critical if an integrated approach to MKM is to succeed — that is, one in which formal and informal representations are used and kept tightly coupled. Our critique of Mizone has allowed us to make some concrete recommendations for Mizar-like systems in this context, and to highlight need for more research to understand issues of search and presentation.

The key to the success of these recommendations is the authors of formal mathematics. While automation will always have a role and may improve, we still need authors to annotate, structure and supplement their formal mathematics if it is to find a wider role in an integrated approach to MKM. The problem is similar to the need for programmers to comment their code. However, in MKM we cannot assume the reader will actually understand the code! Rather, they are probably just interested in the ideas it implements. The vision of MKM is that these readers will still be able to interact with the underlying formal objects, and benefit from their formality.

Unfortunately, there is very little motivation for authors to extend their formal mathematics in this way. Some comments may be provided for other specialists, but the process of formalisation is already burdensome and the main goal of the author is that the complete article should be machine checked successfully. The issue here is that articles are written to be checked, not to be widely read. Perhaps if the acceptance process included some element of readability for a mathematical audience then authors would be sufficiently motivated to structure and comment their articles. The success of the JFM is a first step in this direction.

Another solution is to retrospectively add this material to collections like the MML. The expense and time could be prohibitive for large collections, but could perhaps be distributed with a wiki approach, providing a sufficiently motivated group of qualified people existed and quality could be checked before use. However it can be achieved, we cannot avoid the problem of integrating formal and informal mathematical material if we want a integrated approach to MKM to succeed.

Acknowledgments This work was funded through EPSRC grant GR/S73723.

References

1. ACM Digital Library, `http://portal.acm.org/dl.cfm`
2. Armstrong, W., Nakamura, Y., Rudnicki, P. (2003) Armstrong's Axioms. *Journal of Formalized Mathematics*, 11(1), 39–51
3. Bancerek, G., Rudnicki, P. (2003) Information Retrieval in MML. In A. Asperti, B. Buchberger, J. H. Davenport (eds) *Mathematical Knowledge Management, 2nd Int. Conf., MKM 2003.* LNCS 2594, Springer, pp. 119–132.
4. Bancerek, G., Urban, J. (2004) Integrated Semantic Browsing of the Mizar Mathematical Library for Authoring Mizar Articles. In A. Asperti, G. Bancerek, A. Trybulec (eds), *Mathematical Knowledge Management, 3rd Int. Conf., MKM 2004.* LNCS 3119, Springer, pp. 44–57.
5. Barendregt, H., Wiedijk, F. (2005) The Challenge of Computer Mathematics. In A. Bundy, M. Atiyah, A. Macintyre, D. MacKenzie (eds), The nature of mathematical proof, *Philospohical Transactions of The Royal Society A*, 363(1835): 2351–2375.
6. Borwein, J. M., Farmer, W. M. (2006) *Mathematical Knowledge Management, 5th International Conference, MKM 2006.* LNAI 4108, Springer.
7. Buchberger, B. (2001) Mathematical Knowledge Management in Theorema. In B. Buchberger, O. Caprotti (eds), *Proc. of 1st Int. Workshop on Mathematical Knowledge Management*, Linz, Austria
8. Cairns, P., Gow, J., Collins, P. (2003) On dynamically presenting a topology course. *Annals of Mathematics and Artificial Intelligence*, 38, 91–104
9. Cairns, P., Gow, J. (2004) Using and Parsing the Mizar Language. *Electronic Notes in Theoretical Computer Science*, 93, 60–69.
10. Cairns, P., Gow, J. (2006) Literate proving: presenting and documenting formal proofs. In M. Kohlhase (ed.), *4th Int. Conf. on Mathematical Knowledge Management, MKM 2005*, LNCS 3863, Springer, 159–173
11. Cairns, P., Gow, J. (in press) Integrating Searching and Authoring in Mizar. *J. of Automated Reasoning.*
12. Hales, T., *The Flyspeck Project Fact Sheet.* `http://www.math.pitt.edu/t̃hales/flyspeck/`
13. Knuth, D. E. (1984) Literate programming. *The Computer Journal*, 27, 97–111.
14. Kreitz, C. *The FDL Navigator: Browsing and Manipulating Formal Content.* Technical Report, Cornell University, 2003.
15. Lamport, L. (1994) How to write a proof. *American Mathematical Monthly*, 102(7), 600–608.
16. The Mizar Mathematical Library, `http://mizar.org/library`
17. IngentaConnect, `http://www.ingentaconnect.com/`
18. ISI Web of Knowledge, `http://isiwebofknowledge.com`
19. Journal of Formalized Mathematics, `http://mizar.org/fm`
20. Rudnicki, P., Trybulec, A. (1992) Abian's Fixed Point Theorem. *Formalized Mathematics*, 6(3), 335–338
21. Kanahori, T., Sexton, A., Sorge, V., Suzuki, M. (2006) Capturing Abstract Matrices from Paper. In [6], pp. 124–138.
22. Thimbleby, H. (2003) Explaining code for publication. *Software — Practice and Experience* **33**, 975–1001
23. Urban, J. (2005) XML-izing Mizar: Making Semantic Processing and Presentation of MML Easy. In M. Kohlhase (ed.), *4th Int. Conf. on Mathematical Knowledge Management, MKM 2005*, LNCS 3863, Springer, 346–360.
24. Wiedijk, F. (2000) The De Bruijn Factor. Poster at *TPHOL 2000*
25. Witten, I. H., McNab, R. J., Boddie, S. J., Bainbridge, D. (2000) Greenstone: A comprehensive open-source digital library software system. *Proc. ACM Digital Libraries*, San Antonio, TX, pp. 113–121.
26. Witten, I. H., Bainbridge, D. (2003). *How to Build a Digital Library*, Morgan Kaufmann.
27. Zinn, C. (2004) *Understanding Informal Mathematical Discourse.* PhD Thesis, Arbeitsberichter des Instituts für Informatik, Friedrich-Alexander-Universität, 37(4)

Towards a MIZAR Mathematical Library in OMDOC Format

Grzegorz Bancerek[i] and Michael Kohlhase[ii]

[i] Technical University Bialystok, bancerek@mizar.org
[ii] Jacobs University Bremen, m.kohlhase@iu-bremen.de

Abstract. MIZAR is one of largest libraries of formalized mathematics. The language of the library is highly optimized for authoring by humans. Like in natural languages, the meaning of an expression is influenced by its (mathematical) context in a way that is natural to humans, but hard to specify for machine manipulation. From this point of view, it may be considered as locked up in an arcane file format. Indeed, the MIZAR system itself is currently the only system that can reliably operate on the MIZAR library.

This paper presents an experiment of using the MIZAR system to transform the MIZAR library into the OMDOC format (**O**pen **M**athematical **Doc**uments), an XML-based representation format for mathematical knowledge that is geared towards making formula structure and context dependencies explicit.

We expect the result of this experiment: an OMDOC version of the MIZAR library to enhance system support for formal mathematical libraries.

1 Introduction

In the last years we have seen the birth of a new research area: "Mathematical Knowledge Management" (MKM), which is concerned with representation formalisms for mathematical knowledge, such as MATHML [2], OPENMATH [9] or OMDOC [16], mathematical content management systems [12, 1, 5], search engines [24, 23, 19, 15, 10], as well as publication and education systems [18, 20] for mathematics. The perceived interest in the domain of general knowledge management tools applied to mathematics is that mathematics is a very well-structured and well-conceptualized subject. The main focus of the MKM techniques is to recover the content/semantics of mathematical knowledge and exploit it for the application of automated knowledge management techniques, with an emphasis on web-based and distributed access to the knowledge.

Unfortunately, one of the largest resources of formalized mathematics, the MIZAR library is locked up in an arcane file format that is highly optimized for authoring by humans, but not for MKM techniques other than verification of proofs.

ISBN 978-83-7431-128-1 ISSN 0860–150X

1.1 The Mizar Mathematical Library

Mizar is a representation format for mathematics that is close to mathematical vernacular used in publications and a deduction system for verifying proofs in the Mizar language. The continual development of the Mizar system has resulted a centrally maintained library of mathematics (the Mizar mathematical library MML). The MML is a collection of Mizar articles: text-files that contain theorems and definitions, and proofs. Currently the MML (version 4.76.959) contains 959 articles with 43149 theorems and 8185 definitions. Introductory information on Mizar and the MML can be found in [22, 27, 28, 32]. For the rest of this paper, we assume that the reader is at least superficially familiar with these basic texts.

The Mizar language is based on Tarski-Grothendieck set theory [29], it is essentially a first-order logic[1] with an extremely expressive type systems that features dependent types as well as predicate restrictions, [6]. This Mizar language — in particular the type system and the input syntax — are highly optimized for authoring by humans. Consider for instance the following theorem:

```
for A being set holds
  A is finite iff ex f being Function st rng f = A & dom f in omega
```

For a skilled mathematician this can be almost read and understood without Mizar-specific training. Like in natural languages, the meaning of an expression is influenced by its (mathematical) context in a way that is natural to humans, but hard to specify for machine manipulation. For example, Mizar allows the following types:

```
reflexive transitive antisymmetric Relation
normal Subgroup of G
onto Element of MonoSeq(A)
```

As we see, Mizar types consist of 2 parts: a cluster of adjectives (possibly empty) and a radix type. The cluster of adjective for the last type listed above consists of one adjective `onto` and `Element of MonoSeq(A)` is the radix type of it. In internal presentation, the adjective `onto` is recognized as an adjective with two arguments: the set `NAT` of natural numbers and the finite set `A` and the base type of the adjective is `Relation of NAT, A`. This reconstruction is done from the context. The set `MonoSeq(A)` is a subset of `Funcs(NAT, A)` and then the `Element of MonoSeq(A)` is also an element of `Funcs(NAT, A)`. Further, elements of `Funcs(NAT, A)` are `Function of NAT, A` which widens in sequence to `Relation of NAT, A`. The last type fits to the definition of the adjective `onto` and therefore the arguments of the last type are taken as the arguments of the adjective.

The Mizar system itself is currently the only system that can reliably operate on the Mizar library.

This paper presents a series of experiments of using the Mizar system to transform the Mizar library into the OMDoc format (Open Mathematical Documents [16, 26]), an XML-based representation format for mathematical knowledge that is geared towards making formula structure and context dependencies explicit.

[1] Technically, the Fraenkel Operator of Mizar slightly transcends first-order expressivity, but the language is first-order in style.

We expect the result of this experiment: an OMDoc version of the Mizar library to enhance system support for formal mathematical libraries.

1.2 OMDoc in a Nutshell (three levels of modeling)

The OMDoc (Open Mathematical Documents) format is a content markup scheme for (collections of) mathematical documents including articles, textbooks, interactive books, and courses. To achieve content and context markup for mathematical knowledge, OMDoc uses three levels of modeling corresponding to the concerns raised previously. We have visualized this architecture in Figure 1.

Mathematical Formulae At the lowest level of mathematical formulae, OMDoc uses the established standards OpenMath [9] and Content-MathML [2]. These provide content markup for the structure of mathematical formulae and context markup in the form of URI references in the symbol representations

Mathematical Statements OMDoc provides an original markup scheme for making the structure of mathematical statements explicit. Again, we have content and context markup aspects. For instance the definition in the second row of Figure 1 contains an informal description of the definition as a first child and a formal description in the two recursive equations in the second and third children supported by the type attribute, which states that this is a recursive definition. The context markup in this example is simple: it states that this piece of markup pertains to a symbol declaration for the symbol plus in the current theory (presumably the theory arith1).

Mathematical Theories At this level, OMDoc supplies original markup for clustering sets of statements into theories, and for specifying relations between theories by morphisms. By using this scheme, mathematical knowledge can be structured into reusable chunks. Theories also serve as the primary notion of context in OMDoc, they are the natural target for the context aspect of formula and statement markup.

All levels are augmented by markup for various auxiliary information that is present in mathematical documents, e.g. notation declarations, exercises, experimental data, program code, etc.

2 Translating Mizar

The motivations for a translation of the Mizar library are not particular to the OMDoc format, and it is therefore not very surprising that the work reported on in this paper is not the first attempt to translate the Mizar library. In fact there have been many translation experiments:

The Mizar project produces a hyper-linked, pretty-printed version of the assertions of a Mizar article for publication in the journal *Formalized Mathematics* [7] with its electronic counterpart the *Journal of Formalized Mathematics* [14]. The translation generates a human-oriented presentation of Mizar articles, where formulae are presented in mathematical notation LaTeX and parts of the Mizar logical language are verbalized.

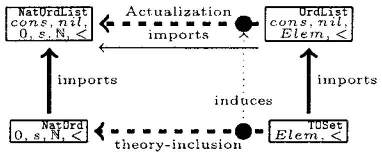
Level of Representation	OMDoc Example
Theory Level: Development Graph – Inheritance via symbol-mapping – Theory inclusion via proof-obligations – Local (one-step) local link vs. global links	 NatOrdList *cons*, *nil*, 0, *s*, ℕ, < Actualization imports OrdList *cons*, *nil*, *Elem*, < imports induces imports NatOrd 0, *s*, ℕ, < theory-inclusion TOSet *Elem*, <
Statement Level: – Axiom, definition, theorem, proof, example,. . . – Structure explicit in statement forms and references	<definition for="#plus" type="recursive"> <CMP>Addition is defined by recursion on the second argument </CMP> <FMP>$X + 0 = 0$</FMP> <FMP>$X + s(Y) = s(X + Y)$</FMP> </definition>
Object Level: OPENMATH/MATHML – Objects as logical formulae – Semantics by pointing to theory level	<OMA> <OMS cd="arith1" name="plus"/> <OMV name="X"/> <OMS cd="nat" name="zero"/> </OMA>

Fig. 1. OMDoc in a Nutshell (the three levels of modeling)

There have been various hand translations of selected MIZAR articles for benchmarking automated theorem provers (see e.g. [11]). Of course this approach does not really scale to the whole MML. In 1997/8 Czeslaw Bylinski and Ingo Dahn translated parts of the MML into a PROLOG syntax by extending the MIZAR system with a PROLOG generator.

Josef Urban has built an extension of the MIZAR parser (now part of the MIZAR) that allows to export the internal, disambiguated form of MIZAR articles used by the MIZAR system to other formats. In the first stage, Urban uses this to generate a first-order version of the MML to test automated theorem provers and independently verify (parts of) the MML. In this translation, the MIZAR type system was relativized to obtain standard first-order formulae; the Fraenkel operator and MIZAR schemas were not treated, since they are second-order constructs. In current stage, Urban's extension reflects the internal form of MIZAR article without relativization and, actually, is incorporated into the MIZAR processor.

The first author uses the MIZAR database and Urban's extension for information retrieval purposes. The MML QUERY system provides a web interface to and a query language the MML QUERY database; see [10, 8] for details. Since this query interface is primarily intended for MIZAR authors and developers, great care has been taken to ensure that no relevant structure of the MIZAR language has been lost. In particular, the MIZAR types have not been relativized away. Currently, the MML QUERY database contains small amount of data about proofs, since this was considered secondary for information retrieval purposes.

2.1 General Issues in the Translation

When processing MIZAR text we have the choice between two levels of language available in the MIZAR (see [30, 10, 8]): The **pattern-level** language is the rich human-oriented input syntax in which MIZAR articles are written, and the **constructor-level**, which is the unique machine-internal representation used in the MIZAR proving engine.

The MIZAR language is a language designed for the practical formalization of mathematics. To enable it MIZAR allows synonyms for an overloading of symbols and patterns, MIZAR allows homonyms, and MIZAR allows also unambiguous tokenization dependent on lexical context (see *three hindrances in text based searching* in [8]). These features were used by different researchers to develop different parts of mathematics in MML. In the result, MML is full of notation conflicts on Pattern-level. To have adequate translation (mirroring the meaning of MIZAR text), we should omitted such conflicts as, in general, the syntactic similarity doesn't implies semantic correlations. Eventually, we decided to use the unambiguous internal representation (constructor-level) in MML QUERY format as a source for the translation.

2.2 The Translation of MML Data to OMDoc

Since version 7.2 the MIZAR system produces quite detailed XML-based semantic description of MIZAR articles at the constructor level [31]. We build on this format for our translation[2], using XSLT style sheets [33]. To understand the process, let us look at a simple example from the article `boole`:

Listing 1.1. A MML theorem

```
theorem
  for X being set holds X \/ {} = X
proof
  let X be set;
  thus X \/ {} c= X
  proof
    let x be set; assume x in X \/ {};
    then x in X or x in {} by XBOOLE_0:def 2;
    hence thesis by XBOOLE_0:def 1;
  end;
  let x be set;
  assume x in X;
  hence thesis by XBOOLE_0:def 2;
end;
```

Note that the identifiers in the MML text above are translated into pointers to internal array at the constructor level. For instance, `set` corresponds to the first constant in this array, witnessed by the attribute `nr="1"` in line four of Listing 1.2. Therefore, the first step is to add absolute MML addresses using Josef Urban's accommodation style sheets `addabsrefs.xsl`, the result of this is given in Listing 1.2

[2] The work reported here extends an earlier translation experiment reported at the "30 years of Mizar" workshop at MKM 2004 in Bialowieza, Poland 2004. That was based on the MML QUERY database and specified the translations as special MML QUERY templates, which filled in the necessary information via MML QUERY queries [10].

below. The style sheet has added the `aid` and `absnr` attributes in line four. This identifies the identifier `set` as the first constant in article `HIDDEN`.

Listing 1.2. The XML representation of Definition 1.1

```
1   <JustifiedTheorem plevel="" aid="BOOLE" kind="T" nr="1">
      <Proposition line="27" col="11" plevel="" propnr="1">
        <For pid="0" vid="3">
          <Typ kind="M" nr="1" pid="1" aid="HIDDEN" absnr="1"><Cluster/><Cluster/></Typ>
          <Pred kind="R" nr="1" pid="7" aid="HIDDEN" absnr="1">
6           <Func kind="K" nr="6" pid="5" aid="XBOOLE_0" absnr="2">
              <Var nr="1"/>
              <Func kind="K" nr="5" pid="4" aid="XBOOLE_0" absnr="1"/>
            </Func>
            <Var nr="1"/>
11        </Pred>
        </For>
      </Proposition>
      <Proof line="28" col="5" plevel="" newlevel="1">...</Proof>
    </JustifiedTheorem>
```

The absolute addresses are needed to translate MML constructors into OpenMath symbols which need a name and a content dictionary for referencing: We interpret Mizar articles as OMDoc content dictionaries, and thus the identifier `set` is translated to the symbol `<OMS cd="HIDDEN" name="c1"/>`; as the surface identifiers like `set` are overloaded, we simply generate symbol names from their numbers.

Note that Mizar language constructs like the `Pred` element on line five of Listing 1.2 are translated into OpenMath applications (via the `OMA` element) where the first child is the applied function and the rest are the arguments. Similarly, Mizar quantifiers are translated into OpenMath bindings (via `OMBIND`) elements with the appropriate binding symbol as the first child. In the case of the `For` element on line three of Listing 1.2, we use the universal quantifier from first-order logic (see Section 3 for a discussion). The bound variable needed as the second child of the is not represented in Mizar-XML, so we invent one, but the type can be translated from the second child of the `For`; it is represented in the OpenMath attribution lines 7-13 below.

Listing 1.3. The OMDoc representation of Theorem 1.1

```
    <assertion xml:id="BOOLE.th1" type="theorem" just-by="BOOLE.pf1">
      <CMP>for X being set holds X \/ {} = X</CMP>
      <FMP logic="mizar">
        <OMOBJ xmlns="http://www.openmath.org/OpenMath">
5         <OMBIND><OMS cd="pl1" name="forall"/>
            <OMBVAR>
              <OMATTR>
                <OMATP>
                  <OMS cd="simpletypes" name="type"/>
10                <OMS cd="HIDDEN" name="c1"/>
                </OMATP>
                <OMV name="X"/>
              </OMATTR>
            </OMBVAR>
15          <OMA><OMS cd="HIDDEN" name="c1"/>
              <OMA><OMS cd="XBOOLE_0" name="c2"/>
                <OMV name="X"/>
                <OMS cd="XBOOLE_0" name="c1"/>
              </OMA>
20            <OMV name="X"/>
            </OMA>
          </OMBIND>
```

```
        </OMOBJ>
      </FMP>
25  </assertion>
```

The discourse-level constructs in Mizar are translated to the appropriate statement-level ones in OMDoc, for instance the `assertion` element in line one above. The `just-by` attribute points to the translated proof, whose OMDoc representation is given in Listing 1.4; as we have already discussed the formula-level translation, we will just gloss them mathematical notation in the boxes.

Listing 1.4. The OMDoc representation of the proof in Listing 1.1

```
    <proof xml:id="BOOLE.pf1" for="BOOLE.thm1">
      <symbol name="X"><type system="mizar">set</type></symbol>
      <derive>
        <FMP>X ∨ {} ⊆ X</FMP>
5       <method xref="by">
          <proof>
            <symbol name="x"><type system="mizar">set</type></symbol>
            <hypothesis><CMP>x ∈ X ∨ {}</CMP></hypothesis>
            <derive>
10            <CMP>x ∈ X ∨ x ∈ {}</CMP>
              <method xref="by"><premise xref="XBOOLE_0.omdoc#XBOOLE_0.def2"/></method>
            </derive>
            <derive type="conclusion">
              <method xref="by"><premise xref="XBOOLE_0.omdoc#XBOOLE_0.def1"/></method>
15          </derive>
          </proof>
        </method>
      </derive>
      <symbol name="x"><type system="mizar">set</type></symbol>
20    <hypothesis><CMP>x ∈ X</CMP></hypothesis>
      <derive type="conclusion">
        <method xref="by"><premise xref="XBOOLE_0.omdoc#XBOOLE_0.def2"/></method>
      </derive>
    </proof>
```

Finally, `environment` declarations in Mizar articles can directly be mapped to `imports` elements in OMDoc, since they have the same function: providing vocabulary and theorems in the current environment (implicitly given by the Mizar article/OMDoc theory).

3 OMDoc Content Dictionaries for Mizar

As the OpenMath formula language underlying the OMDoc format is logic-independent, the "reserved words" of the Mizar language are expressed as OpenMath symbols and need to be documented in content dictionaries. We have developed two OMDoc content dictionaries: `mizar` and `mizar-types`, building on an already existing set of content dictionaries for logical systems presented in [17]; see Figure 2 for details.

The "Mizar-types" theory for the Mizar language defines the following symbols for constructing types, all others can be inherited from the content dictionaries already present.

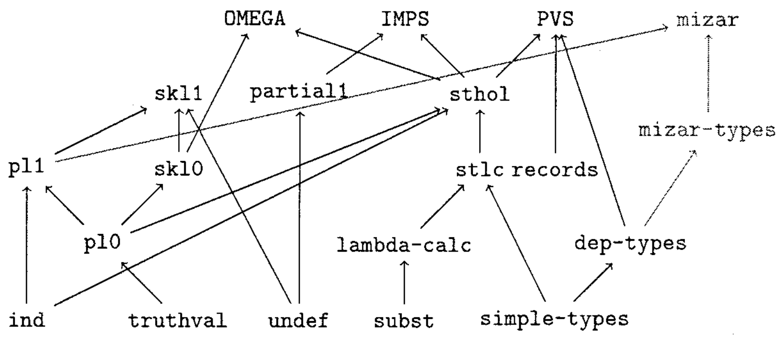


Fig. 2. Integrating the MIZAR language in to a Hierarchy of Logical Systems

type-expression The type expression constructor. The first argument is a radix type and the rest is a list of adjectives that modify it.
adjective-cluster The adjective composition operator. It makes an adjective cluster out of a sequence of simpler ones.
type-expression-nonempty The operator that expresses the fact that a type is legal, since it is non-empty.
adjective-inclusion The operator that expresses the fact that an adjective includes another on a radix type. The first argument is the radix type, the next two are adjective clusters, the first cluster entails the second one on the radix.

For the meanings of these please consult [6]. The term level of the MIZAR language similarly profits from existing content dictionaries, we only need to specify symbols that are not present in regular first-order logic. Thus we only need to provide symbols for dealing with the MIZAR type system and the weakly second-order construct of Fränkel sets. Concretely, we provide the symbols:

is The "is of type" operator in MIZAR. It expresses that its first argument has the type that is expressed as its second argument.
possesses The operator the expresses that a term possesses certain properties. Its first argument is the term and the rest are adjectives that express the properties.
adjective-holds The operator that expresses the fact that an adjective always holds on a radix type. The first argument is the radix type, the second one is an adjective cluster.
fraenkel-bind/set A Fränkel set is a set with typed bound variables in MIZAR, e.g. $\{x^n - 1| \sqrt[n]{x} > 1$where $x \in \mathbb{R}$ and $n \in \mathbb{N}\}$. Given the `mizar` content dictionary, this can be represented in OPENMATH as

```
<OMBIND><OMS cd="mizar" name="fraenkel−bind"/>
  <OMBVAR>$\boxed{x:\mathbb{R}}$$\boxed{n:\mathbb{N}}$</OMBVAR>
  <OMA><OMS cd="mizar" name="fraenkel−set"/>$\boxed{x^n-1}$ $\boxed{\sqrt[n]{x}>1}$</OMA>
</OMBIND>
```

where the **fraenkel-bind** acts as a binding operator binding the variables x and n and the fraenkel-set constructor combines the set expression schema (in these bound variables) and the restriction.

These two content dictionaries act as an explicit context representations for OPENMATH representations of MIZAR terms. As we have seen above, the OMDOC representation uses numbered constant names for representing constants. For presentation to the human user, the original MIZAR identifiers can be regained via the OMDOC presentation language (see [16, chapter 25]). For instance, our translation generates the following `presentation` element in the `HIDDEN` content dictionary.

```
<presentation for="c1"><use format="default">set</format></presentation>
```

Thus presenting the generated OMDOC version of a MIZAR article will lead to a similar user experience (e.g. cross-linked identifiers) as the presentation generated from MIZAR XML via the MIZAR presentation stylesheet in the MIZAR distribution. Similar presentation elements for the MIZAR language symbols allow to extend this presentation mechanism to them as well, linking the generated presentation to the relevant parts of the MIZAR content dictionaries making them an integrated language documentation.

But this is not the only added value we obtain from the translation: other OMDOC-based tools become applicable as well, e.g. the MATHWEBSEARCH engine [21, 15], a highly efficient search engine for mathematical formulae in OPENMATH format.

4 Conclusion

We have presented a structure-preserving, statement-level translation from the MIZAR mathematical library to the OMDOC format. This translation currently produces valid OMDOC documents. Since the OPENMATH formula language underlying the OMDOC format is logic-independent, the "reserved words" of the MIZAR language are also expressed as OPENMATH symbols, which are documented in the OPENMATH content dictionary lstinline[basicstyle=]`mizar.omdoc`, which builds on and is integrated with the logic hierarchy presented at [17].

In the future we plan to experiment with OMDOC-based tools on the transformed MIZAR library, such as the MBASE mathematical knowledge base [12] or the MAYA development graph manager [4], which offers semantics-informed process of change management. Furthermore, we plan to evaluate Immanuel Normann's theory morphism detector [25] to augment the semantic structure of the theory graph of the library, which is currently simply based on the simple inheritance structure of MIZAR articles. We expect to find a theory structure that is based on the "semantic topology" of library content rather than on the historical coincidence that largely governs the structure of the library today.

Finally, we plan to develop an inverse translation from OMDOC to MIZAR to allow round-tripping and checking for completeness of the transformation. If the MIZAR translation of the OMDOC translation of the MML still passes the MIZAR verifier, then we can be reasonably sure that no (essential) information has been lost in the translation processes.

References

1. Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. FDL: A prototype formal digital library – description and draft reference manual. Technical report, Computer Science, Cornell, 2002. `http://www.cs.cornell.edu/Info/Projects/NuPrl/html/FDLProject/02cucs-fdl.pdf`.
2. Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003. Available at http://www.w3.org/TR/MathML2.
3. Andrea Asperti, Bruno Buchberber, and James Harold Davenport, editors. *Mathematical Knowledge Management, MKM'03*, number 2594 in LNCS. Springer Verlag, 2003.
4. Serge Autexier, Dieter Hutter, Till Mossakowski, and Axel Schairer. The development graph manager MAYA (system description). In Hélene Kirchner, editor, *Proceedings of 9th International Conference on Algebraic Methodology And Software Technology (AMAST'02)*. Springer Verlag, 2002.
5. Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, and Irene Schena. HELM and the semantic math-web. In Richard. J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs'01*, volume 2152 of *LNCS*, pages 59–74. Springer Verlag, 2001.
6. Grzegorz Bancerek. On the structure of Mizar types. *Electronic Notes in Theoretical Computer Science*, 85(7), 2003.
7. Grzegorz Bancerek. Automatic translation in Formalized Mathematics. *Mechanized Mathematics and Its Applications*, 5(2):19–31, 2006.
8. Grzegorz Bancerek. Information retrieval and rendering with MML Query. In Jon Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, MKM'06*, number 4108 in LNAI, pages 266–279. Springer Verlag, 2006.
9. Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004. `http://www.openmath.org/standard/om20`.
10. Grzegorz Bancerek and Piotr Rudnicki. Information retrieval in MML. In Asperti et al. [3], pages 119–131.
11. Ingo Dahn and Christoph Wernhard. First order proof problems extracted from an article in the Mizar Mathematical Library. In Ulrich Furbach and Maria Paola Bonacina, editors, *Proceedings of the International Workshop on First order Theorem Proving*, number 97-50 in RISC-Linz Report Series, pages 58–62. Johannes Kepler Universität Linz, 1997.
12. Andreas Franke and Michael Kohlhase. System description: MBase, an open mathematical knowledge base. In David McAllester, editor, *Automated Deduction – CADE-17*, number 1831 in LNAI, pages 455–459. Springer Verlag, 2000.
13. Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors. *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI. Springer Verlag, 2006.
14. Journal of Formalized Mathematics. `http://www.mizar.org/JFM`.
15. Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Ida et al. [13], pages 241–253.
16. Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.
17. Michael Kohlhase. Standardizing context in system interoperability. In OMDoc – *An open markup format for mathematical documents [Version 1.2]* [16].
18. Christoph Lange and Michael Kohlhase. A semantic wiki for mathematical knowledge management. In Max Völkel, Sebastian Schaffert, and Stefan Decker, editors, *Proceedings of the 1st Workshop on Semantic Wikis, European Semantic Web Conference 2006*, volume 206 of *CEUR Workshop Proceedings*, Budva, Montenegro, June 2006.
19. Paul Libbrecht and Erica Melis. Methods for Access and Retrieval of Mathematical Content in ActiveMath. In N. Takayama and A. Iglesias, editors, *Proceedings of ICMS-2006*, number 4151 in LNAI. Springer Verlag, 2006. `http://www.activemath.org/publications/Libbrecht-Melis-Access-and-Retrieval-ActiveMath-ICMS-2006.pdf`.
20. E. Melis, J. Buedenbender E. Andres, Adrian Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. The Activemath Learning Environment. *Artificial Intelligence and Education*, 12(4), 2001.
21. Math web search, web page at `http://kwarc.info/projects/mws/`, seen Jan 2007.
22. Mizar manuals. `http://mizar.org/project/bibliography.html`.
23. Rajesh Munavalli and Robert Miner. Mathfind: a math-aware search engine. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 735–735, New York, NY, USA, 2006. ACM Press.
24. Bruce R. Miller and Abdou Youssef. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):121–136, 2003.
25. Immanuel Normann. Enhanced theorem reuse by partial theory inclusions. In Ida et al. [13].
26. The OMDoc repository, web page at `http://www.mathweb.org/omdoc`.
27. Piotr Rudnicki, Christoph Schwarzweller, and Andrzej Trybulec. Commutative algebra in the Mizar system. *Journal of Symbolic Computation*, 32:143–169, 2001.
28. Andrzej Trybulec and Piotr Rudnicki. On equivalents of well-foundedness. *Journal of Automated Reasoning*, 23(3-4):197–234, 1999.
29. Andrzej Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(1):9–11, 1990.
30. Josef Urban. Translating Mizar for first-order theorem provers. In Asperti et al. [3], pages 203–215.
31. Josef Urban. XML-izing Mizar: making semantic processing and presentation of MML easy. In Michael Kohlhase, editor, *Mathematical Knowledge Management, MKM'05*, number 3863 in LNAI, pages 346 – 360. Springer Verlag, 2005.
32. Freek Wiedijk. Mizar: An impression, 1999. `http://www.cs.kun.nl/~freek/notes`.
33. Xsl transformations (xslt) version 1.0. W3c recommendation, W3C, 1999.

Isabelle/Isar — a Generic Framework for Human-Readable Proof Documents

Makarius Wenzel

Technische Universität München
Institut für Informatik
Boltzmannstraße 3, 85748 Garching, Germany
http://www.in.tum.de/~wenzelm/

Abstract. Isabelle/Isar is a generic framework for human-readable formal proof documents, both like and unlike Mizar. The Isar proof language provides general principles that may be instantiated to particular object-logics and applications. The design of Isar has emerged from careful analysis of some inherent virtues of the existing logical framework of Isabelle/Pure, notably composition of higher-order natural deduction rules, which is a generalization of Gentzen's original calculus. Thus Isar proof texts may be understood as structured compositions of formal entities of the Pure framework, namely propositions, facts, and goals.
This paper provides an extensive overview of the combined Pure + Isar framework, including a full object-logic definition as a working example. Hereby we hope to illustrate the present stage of our particular journey from insight into the logical framework, to proofs written in Isabelle/Isar.

1 Introduction

1.1 Theory documents

Isabelle/Isar [20,21,8,22] is intended as a generic framework for developing formal mathematical documents with full proof checking. Definitions and proofs are organized as theories; an assembly of theory sources may be presented as a printed document (using PDF-LaTeX). The present paper is an example of the result of processing formally checked sources by the Isabelle/Isar document preparation system.

The main objective of Isar is the design of a human-readable structured proof language, which is called the "primary proof format" in Isar terminology. Such a primary proof language is somewhere in the middle between the extremes of primitive proof objects and actual natural language. In this respect, Isar is a bit more formalistic than Mizar [18,16,23,25], using logical symbols for certain reasoning schemes where Mizar would prefer English words; see [26] for further comparisons of these systems.

So Isar challenges the traditional way of recording informal proofs in mathematical prose, as well as the common tendency to see fully formal proofs directly as objects of some logical calculus (e.g. λ-terms in a version of type theory). In fact,

ISBN 978-83-7431-128-1 ISSN 0860-150X

Isar is better understood as an interpreter of a simple block-structured language for describing data flow of local facts and goals, interspersed with occasional invocations of proof methods. Everything is reduced to logical inferences internally, but these steps are somewhat marginal compared to the overall bookkeeping of the interpretation process. Thanks to careful design of the syntax and semantics of Isar language elements, a formal record of Isar instructions may later appear as an intelligible text to the attentive reader.

Isabelle/Isar is based on the existing logical framework of Isabelle/Pure [10,11], which provides a generic environment for higher-order natural deduction. The approach of generic inference systems in Pure is continued by Isar towards actual proof texts. Concrete applications require another intermediate layer: an object-logic. Isabelle/HOL [9] (simply-typed set-theory) is being used most of the time; Isabelle/ZF is less extensively developed, although it would probably fit better for classical mathematics.

After the pioneering approach of Mizar [18] towards mathematical proofs on the machine has become more widely acknowledged around 1995, some existing interactive theorem provers have been extended to support a similar "declarative mode", such as "a Mizar mode for HOL" [5] or [24]. The design of Isar draws both on Mizar and various Mizar modes, but the resulting Isabelle/Isar is a well-integrated system that continues the original Isabelle tradition. Consequently, Isabelle/Isar is neither another version of Mizar, nor a "declarative mode" for Isabelle. The distinctive style of the logical framework is preserved, and Isar works for all of the usual Isabelle object-logics, such as FOL, ZF, HOL, HOLCF.

This paper is dedicated to elements of structured proofs. We shall briefly give a flavor of plain specifications in Isabelle just now, both axiomatic and definitional ones. For example, in Isabelle/HOL three distinct natural numbers can be postulated like this:

axiomatization $a\ b\ c :: nat$ **where** $a \neq b$ **and** $b \neq c$ **and** $a \neq c$

Stating arbitrary axioms is convenient, but also dangerous. Isabelle methodology follows traditional mathematics in preferring proper definitions. For example:

definition $a :: nat$ **where** $a = 1$
definition $b :: nat$ **where** $b = 2$
definition $c :: nat$ **where** $c = 3$

Definitions like this occasionally have the disadvantage of over-specification: we have introduced particular natural numbers instead of three "arbitrary" ones.

Alternatively we may use the locale mechanism of Isabelle [6,1], which combines both specification paradigms conveniently: a given axiomatization is wrapped into a predicate definition, subsequent definitions and proofs depend on the resulting context. Viewed from outside, any consequences become relative to explicit assumptions. E.g.

locale *distinct-nats* = **fixes** $a\ b\ c :: nat$ **assumes** $a \neq b$ **and** $b \neq c$ **and** $a \neq c$

Many interesting questions arise from advanced specifications. We shall occasionally return to such issues, when there is an immediate impact on proof composition.

1.2 Proof texts

The Isar proof language offers common principles that are parameterized by entities of the object-logic and application context. This includes declarations for various classes of rules, such as logical introductions / eliminations, and transitivity / symmetry rules. General reasoning mechanisms refer to such declarations from the context, performing "obvious" reasoning steps in a very elementary sense. Moreover, the system admits arbitrarily complex proof methods to be added later, for example specific support for induction proofs [19]. Automated reasoning tools may be integrated as well [13,14].

In order to illustrate typical natural deduction reasoning in Isar, we assume the background theory and library of Isabelle/HOL [9]. This includes common notions of predicate logic, naive set-theory etc. using fairly standard mathematical notation. From the perspective of generic natural deduction there is nothing special about the logical connectives of HOL ($\wedge$, $\vee$, $\forall$, $\exists$, etc.), only the resulting reasoning principles are relevant to the user. There are similar rules available for set-theory operators ($\cap$, $\cup$, $\bigcap$, $\bigcup$, etc.), or any other theory developed in the library (lattice theory, topology etc.).

Subsequently we briefly review fragments of Isar proof texts corresponding directly to such general natural deduction schemes. The examples shall refer to set-theory.

The following deduction performs $\cap$-introduction, working forwards from assumptions towards the conclusion. We give both the Isar text, and depict the primitive rule involved, as determined by unification of the problem against rules from the context.

assume $x \in A$ **and** $x \in B$
then have $x \in A \cap B$ **..**

$$\frac{x \in A \quad x \in B}{x \in A \cap B}$$

Note that **assume** augments the context, **then** indicates that the current facts shall be used in the next step, and **have** states a local claim. The mysterious "**..**" above is a complete proof of the claim, using the indicated facts and a canonical rule from the context. We could have been more explicit here, writing "**by** (*rule IntI*)" instead.

The format of the $\cap$-introduction rule represents the most basic inference, which proceeds from given premises to a conclusion, without any additional context involved.

The next example performs backwards introduction on $\bigcap \mathcal{A}$, the intersection of all sets within a given set. This requires a nested proof of set membership within a local context of an arbitrary-but-fixed member of the collection:

have $x \in \bigcap \mathcal{A}$
proof
  fix A
  assume $A \in \mathcal{A}$
  show $x \in A$ $\langle proof \rangle$
qed

$$\frac{\begin{array}{c}[A][A \in \mathcal{A}]\\ \vdots \\ x \in A\end{array}}{x \in \bigcap \mathcal{A}}$$

This Isar reasoning pattern again refers to the primitive rule depicted above. The system determines it in the "**proof**" step, which could have been spelt out more explicitly as "**proof** (*rule InterI*)". Note that this rule involves both a local parameter A and an assumption $A \in \mathcal{A}$ in the nested reasoning. This kind of compound rule typically demands a genuine sub-proof in Isar, working backwards rather than forwards as seen before. In the proof body we encounter the **fix**-**assume**-**show** skeleton of nested sub-proofs that is typical for Isar. The final **show** is like **have** followed by an additional refinement of the enclosing claim, using the rule derived from the proof body.

The next example involves $\bigcup \mathcal{A}$, which can be characterized as the set of all x such that $\exists A.\ x \in A \wedge A \in \mathcal{A}$. The elimination rule for $x \in \bigcup \mathcal{A}$ does not mention $\exists$ and $\wedge$ at all, but admits to obtain directly a local A such that $x \in A$ and $A \in \mathcal{A}$ hold. This corresponds to the following Isar proof and inference rule, respectively:

```
assume x ∈ ⋃𝒜
then have C
proof
  fix A
  assume x ∈ A and A ∈ 𝒜
  show C ⟨proof⟩
qed
```

$$\frac{x \in \bigcup\mathcal{A} \qquad \begin{matrix}[A][x \in A,\ A \in \mathcal{A}]\\ \vdots \\ C\end{matrix}}{C}$$

Although the Isar proof follows the natural deduction rule closely, the text reads not as natural as anticipated. There is a double occurrence of an arbitrary conclusion C, which represents the final result, but is irrelevant for now. This issue arises for any elimination rule involving local parameters. Isar provides the derived language element **obtain**, which is able to perform the same elimination proof more conveniently:

```
assume x ∈ ⋃𝒜
then obtain A where x ∈ A and A ∈ 𝒜 ..
```

Here we avoid to mention the final conclusion C and return to plain forward reasoning. The rule involved in the "**..**" proof is the same as before.

1.3 Overview

The rest of the paper is structured as follows: §2 reviews the Pure logical framework of Isabelle, §3 covers the main aspects of the Isar proof language, and §4 demonstrates the combined Pure + Isar framework by first-order predicate logic as object-language.

2 The Pure framework

The Pure logic [10,11] is an intuitionistic fragment of higher-order logic [3]. In type-theoretic parlance, there are three levels of λ-calculus with corresponding arrows: $\Rightarrow$ for syntactic function space (terms depending on terms), $\bigwedge$ for universal quantification (proofs depending on terms), and $\Longrightarrow$ for implication (proofs depending on proofs).

On top of this, Pure implements a generic calculus for nested natural deduction rules, similar to [17]. Here object-logic inferences are internalized as formulae over $\bigwedge$ and $\Longrightarrow$. Combining such rule statements may involve higher-order unification [15].

2.1 Primitive inferences

Term syntax provides explicit notation for abstraction $\lambda x :: \alpha.\ b(x)$ and application $b\ a$, while types are usually implicit thanks to type-inference; terms of type *prop* are called propositions. Logical statements are composed via $\bigwedge x :: \alpha.\ B(x)$ and $A \Longrightarrow B$. Primitive reasoning operates on judgments of the form $\Gamma \vdash \varphi$, with standard introduction and elimination rules for $\bigwedge$ and $\Longrightarrow$ that refer to fixed parameters x_1, …, x_m and hypotheses A_1, …, A_n from the context Γ; the corresponding proof terms are left implicit. The subsequent inference rules define $\Gamma \vdash \varphi$ inductively, relative to a collection of axioms:

$$\frac{(A\ \text{axiom})}{\vdash A} \qquad \frac{}{A \vdash A}$$

$$\frac{\Gamma \vdash B(x) \quad x \notin \Gamma}{\Gamma \vdash \bigwedge x.\ B(x)} \qquad \frac{\Gamma \vdash \bigwedge x.\ B(x)}{\Gamma \vdash B(a)}$$

$$\frac{\Gamma \vdash B}{\Gamma - A \vdash A \Longrightarrow B} \qquad \frac{\Gamma_1 \vdash A \Longrightarrow B \quad \Gamma_2 \vdash A}{\Gamma_1 \cup \Gamma_2 \vdash B}$$

Furthermore, Pure provides a built-in equality $\equiv\ :: \alpha \Rightarrow \alpha \Rightarrow prop$ with axioms for reflexivity, substitution, extensionality, and $\alpha\beta\eta$-conversion on λ-terms.

An object-logic introduces another layer on top of Pure, e.g. with types i for individuals and o for propositions, term constants $Tr :: o \Rightarrow prop$ as (implicit) derivability judgment and connectives like $\wedge :: o \Rightarrow o \Rightarrow o$ or $\forall :: (i \Rightarrow o) \Rightarrow o$, and axioms for object rules such as *conjI*: $A \Longrightarrow B \Longrightarrow A \wedge B$ or *allI*: $(\bigwedge x.\ B\ x) \Longrightarrow \forall x.\ B\ x$. Derived object rules are represented as theorems of Pure.

2.2 Reasoning with rules

Primitive inferences mostly serve foundational purposes. The main reasoning mechanisms of Pure operate on nested natural deduction rules expressed as formulae, using $\bigwedge$ to bind local parameters and $\Longrightarrow$ to express entailment. Multiple parameters and premises are represented by repeating these connectives in a right-associative fashion.

Since $\bigwedge$ and $\Longrightarrow$ commute thanks to $(A \Longrightarrow (\bigwedge x.\ B\ x)) \equiv (\bigwedge x.\ A \Longrightarrow B\ x)$, we may assume w.l.o.g. that rule statements always observe the normal form where quantifiers are pulled in front of implications at each level of nesting. This means that any Pure proposition may be presented as a *Hereditary Harrop Formula* [7] which is of the form $\bigwedge x_1 \ldots x_m.\ H_1 \Longrightarrow \ldots H_n \Longrightarrow A$ for m, $n \geq 0$, and H_1, …, H_n being recursively of the same format, and A atomic. Following the convention that outermost quantifiers are implicit, Horn clauses $A_1 \Longrightarrow \ldots A_n \Longrightarrow A$ are a special case of this.

Goals are also represented as rules: $A_1 \Longrightarrow \ldots A_n \Longrightarrow C$ states that the sub-goals $A_1, \ldots, A_n$ entail the result C; for $n = 0$ the goal is finished. To allow C being a rule statement itself, we introduce the protective marker $\# :: prop \Rightarrow prop$, which is defined as identity and hidden from the user. We initialize and finish goal states as follows:

$$\frac{}{C \Longrightarrow \#C}\ (\mathit{init}) \qquad \frac{\#C}{C}\ (\mathit{finish})$$

Goal states are refined in intermediate proof steps until a finished form is achieved. Here the two main reasoning principles are *resolve*, for back-chaining a rule against a sub-goal (replacing it by zero or more sub-goals), and *close*, for solving a sub-goal (finding a short-circuit with local assumptions). Below $\overline{x}$ stands for $x_1, \ldots, x_n$ $(n \geq 0)$.

$$\frac{\begin{array}{r} \mathit{rule}\colon \overline{A}\ \overline{a} \Longrightarrow B\ \overline{a} \\ \mathit{goal}\colon (\bigwedge\overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow B'\ \overline{x}) \Longrightarrow C \\ \mathit{goal\ unifier}\colon (\lambda\overline{x}.\ B\ (\overline{a}\ \overline{x}))\,\theta = B'\theta \end{array}}{(\bigwedge\overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow \overline{A}\ (\overline{a}\ \overline{x}))\,\theta \Longrightarrow C\,\theta}\ (\mathit{resolve})$$

$$\frac{\begin{array}{r} \mathit{goal}\colon (\bigwedge\overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow A\ \overline{x}) \Longrightarrow C \\ \mathit{assm\ unifier}\colon A\,\theta = H_i\,\theta\ \ \text{(for some } H_i\text{)} \end{array}}{C\,\theta}\ (\mathit{close})$$

The following trace illustrates goal-oriented reasoning in Isabelle/Pure:

$$\begin{array}{rl}
(A \wedge B \Longrightarrow B \wedge A) \Longrightarrow \#(A \wedge B \Longrightarrow B \wedge A) & (\mathit{init}) \\
(A \wedge B \Longrightarrow B) \Longrightarrow (A \wedge B \Longrightarrow A) \Longrightarrow \#\ldots & (\mathit{resolve}\ B \Longrightarrow A \Longrightarrow \\
 & \ B \wedge A) \\
(A \wedge B \Longrightarrow A \wedge B) \Longrightarrow (A \wedge B \Longrightarrow A) \Longrightarrow \#\ldots & (\mathit{resolve}\ A \wedge B \Longrightarrow B) \\
(A \wedge B \Longrightarrow A) \Longrightarrow \#\ldots & (\mathit{close}) \\
(A \wedge B \Longrightarrow B \wedge A) \Longrightarrow \#\ldots & (\mathit{resolve}\ A \wedge B \Longrightarrow A) \\
\#\ldots & (\mathit{close}) \\
A \wedge B \Longrightarrow B \wedge A & (\mathit{finish})
\end{array}$$

Compositions of *close* after *resolve* occurs quite often, typically in elimination steps. Traditional Isabelle tactics accommodate this by a combined *elim-resolve* principle. In contrast, Isar uses a slightly more refined combination, where the assumptions to be closed are marked explicitly, using again the protective marker #:

$$\frac{\begin{array}{r} \mathit{sub\text{-}proof}\colon \overline{G}\ \overline{a} \Longrightarrow B\ \overline{a} \\ \mathit{goal}\colon (\bigwedge\overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow B'\ \overline{x}) \Longrightarrow C \\ \mathit{goal\ unifier}\colon (\lambda\overline{x}.\ B\ (\overline{a}\ \overline{x}))\,\theta = B'\theta \\ \mathit{assm\ unifiers}\colon (\lambda\overline{x}.\ G_j\ (\overline{a}\ \overline{x}))\,\theta = \#H_i\,\theta\ \ \text{(for each marked } G_j \text{ some } \#H_i\text{)} \end{array}}{(\bigwedge\overline{x}.\ \overline{H}\ \overline{x} \Longrightarrow \overline{G}'\ (\overline{a}\ \overline{x}))\,\theta \Longrightarrow C\,\theta}\ (\mathit{refine})$$

Here the *sub-proof* rule stems from the main **fix-assume-show** skeleton of Isar (cf. §3.3): each assumption indicated in the text results in a marked premise G above.

3 The Isar proof language

Structured proofs are presented as high-level expressions for composing entities of Pure (propositions, facts, and goals). The Isar proof language allows to organize reasoning within the underlying rule calculus of Pure, but Isar is not another logical calculus!

Isar is an exercise in sound minimalism. Approximately half of the language is introduced as primitive, the rest defined as derived concepts. The following grammar describes the core language (category *proof*), which is embedded into theory specification elements such as **theorem**; see also §3.2 for the separate category *statement*.

$$\begin{array}{rl}
\mathit{theory\text{-}element} = & \textbf{theorem}\ \mathit{statement\ proof}\ \mid\ \textbf{definition}\ \ldots \mid \ldots \\
\mathit{proof} = & \mathit{prefix}^{*}\ \textbf{proof}\ \mathit{method}^{?}\ \mathit{element}^{*}\ \textbf{qed}\ \mathit{method}^{?} \\
\mathit{prefix} = & \textbf{using}\ \mathit{facts}\ \textbf{and}\ \ldots \\
\mid & \textbf{unfolding}\ \mathit{facts}\ \textbf{and}\ \ldots \\
\mathit{element} = & \{\ \mathit{element}^{*}\ \} \\
\mid & \textbf{next} \\
\mid & \textbf{note}\ \mathit{name} = \mathit{facts}\ \textbf{and}\ \ldots \\
\mid & \textbf{let}\ \mathit{pattern} = \mathit{term}\ \textbf{and}\ \ldots \\
\mid & \textbf{fix}\ \mathit{vars}\ \textbf{and}\ \ldots \\
\mid & \textbf{assm}\ \ll\!\mathit{rule}\!\gg\ \mathit{name}\colon \mathit{props}\ \textbf{and}\ \ldots \\
\mid & \textbf{then}^{?}\ \mathit{claim} \\
\mathit{claim} = & \textbf{have}\ \mathit{name}\colon \mathit{props}\ \textbf{and}\ \ldots\ \mathit{proof} \\
\mid & \textbf{show}\ \mathit{name}\colon \mathit{props}\ \textbf{and}\ \ldots\ \mathit{proof}
\end{array}$$

Here the **and** keyword separates simultaneous elements of the same kind.

The syntax for terms and propositions is inherited from Pure (and the object-logic). A *pattern* is a *term* with schematic variables, to be bound by higher-order matching.

Facts may be referenced by name or proposition. E.g. the result of "**have** a: A $\langle proof \rangle$" becomes available both as a and ‹A›. Moreover, fact expressions may involve attributes that modify either the theorem or the background context. For example, the expression "a [*OF* b]" refers to the composition of two facts according to the *resolve* inference of §2.2, while "a [*intro*]" declares a fact as introduction rule in the context.

The special name "*this*" always refers to the last result, as produced by **note**, **assm**, **have**, or **show**. Since **note** occurs frequently together with **then** we provide some abbreviations: "**from** a" for "**note** a **then**", and "**with** a" for "**from** a **and** *this*".

The *method* category is essentially a parameter and may be populated later. Methods use the facts indicated by **then** or **using**, and then operate on the goal state. Some basic methods are predefined: "–" leaves the goal unchanged, "*this*" applies the facts as rules to the goal, "*rule*" applies the facts to another rule and the result to the goal (both "*this*" and "*rule*" refer to *resolve* of §2.2). The secondary arguments to "*rule*" may be specified explicitly as in "(*rule a*)", or picked from the context. In the latter case, the system first tries rules declared as [*elim*] or [*dest*], followed by those declared as [*intro*].

The default method for **proof** is "*rule*" (arguments picked from the context), for **qed** it is "–". Further abbreviations for terminal proof steps are "**by** $method_1$ $method_2$" for "**proof** $method_1$ **qed** $method_2$", and "**..**" for "**by** *rule*", and "**.**" for "**by** *this*". The **unfolding** element operates directly on the current facts and goal by applying equalities.

Block structure can be indicated explicitly by "**{** ... **}**", although the body of a sub-proof already involves implicit nesting. In any case, **next** jumps into the next section of a block, i.e. it acts like closing an implicit block scope and opening another one.

The remaining elements **fix** and **assm** build up a local context (see §3.1), while **show** refines a pending sub-goal by the rule resulting from a nested sub-proof (see §3.3). Further derived concepts will support calculational reasoning (see §3.4).

3.1 Context elements

In judgments $\Gamma \vdash \varphi$ of the primitive framework, Γ essentially acts like a proof context. Isar elaborates this idea towards a higher-level notion, with separate information for type-inference, term abbreviations, local facts, hypotheses etc.

The element **fix** $x :: \alpha$ declares a local parameter, i.e. an arbitrary-but-fixed entity of a given type; in results exported from the context, x may become anything. The **assm** element provides a general interface to hypotheses: "**assm** «*rule*» A" produces $A \vdash A$ locally, while the included inference rule tells how to discharge A from results $A \vdash B$ later on. There is no user-syntax for «*rule*», i.e. **assm** may only occur in derived elements that provide a suitable inference internally. In particular, "**assume** A" abbreviates "**assm** «*discharge#*» A", and "**def** $x \equiv a$" abbreviates "**fix** x **assm** «*expand*» $x \equiv a$", involving the following inferences:

$$\frac{\Gamma \vdash B}{\Gamma - A \vdash \#A \Longrightarrow B}\;(\mathit{discharge\#}) \qquad \frac{\Gamma \vdash B\ x}{\Gamma - (x \equiv a) \vdash B\ a}\;(\mathit{expand})$$

The most interesting derived element in Isar is **obtain** [21, §5.3], which supports generalized elimination steps in a purely forward manner. This is similar to `consider` in Mizar [18,16,23,25], although the way to get there is quite different, and the result is not tied to particular notions of predicate logic (such as $\wedge$, $\exists$).

The **obtain** element takes a specification of parameters $\overline{x}$ and assumptions $\overline{A}$ to be added to the context, together with a proof of a reduction rule stating that this extension is conservative (i.e. may be removed from closed results later on):

$$\begin{array}{l}
\langle \mathit{facts}\rangle\ \textbf{obtain}\ \overline{x}\ \textbf{where}\ \overline{A}\ \overline{x}\ \langle \mathit{proof}\rangle\ \equiv \\
\quad \textbf{have}\ \mathit{reduction}\text{: } \bigwedge \mathit{thesis}.\ (\bigwedge \overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow \mathit{thesis}) \Longrightarrow \mathit{thesis}\rangle \\
\quad \textbf{proof}\ - \\
\qquad \textbf{fix}\ \mathit{thesis} \\
\qquad \textbf{assume}\ [\mathit{intro}]\text{: } \bigwedge \overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow \mathit{thesis} \\
\qquad \textbf{show}\ \mathit{thesis}\ \textbf{using}\ \langle \mathit{facts}\rangle\ \langle \mathit{proof}\rangle \\
\quad \textbf{qed} \\
\quad \textbf{fix}\ \overline{x}\ \textbf{assm}\ \ll\!\mathit{eliminate\ reduction}\!\gg\ \overline{A}\ \overline{x}
\end{array}$$

$$\frac{\begin{array}{l}\mathit{reduction}\text{: } \Gamma \vdash \bigwedge \mathit{thesis}.\ (\bigwedge \overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow \mathit{thesis}) \Longrightarrow \mathit{thesis} \\ \quad \mathit{result}\text{: } \Gamma \cup \overline{A}\ \overline{y} \vdash B\end{array}}{\Gamma \vdash B}\;(\mathit{eliminate})$$

Here the name "*thesis*" is a specific convention for an arbitrary-but-fixed proposition; in the primitive natural deduction rules shown before we have occasionally used C. The whole statement of "**obtain** x **where** $A\ x$" may be read as a claim that $A\ x$ may be assumed for some arbitrary-but-fixed x. Also note that "**obtain** A **and** B" without parameters is similar to "**have** A **and** B", but the latter involves multiple sub-goals.

The subsequent Isar proof texts explain all context elements introduced above using the formal proof language itself. After finishing a local proof within a block, we indicate the exported result via **note**. This illustrates the meaning of Isar context elements without goals getting in between.

$$\begin{array}{llll}
\{ & \{ & \{ & \{ \\
\quad \textbf{fix}\ x & \quad \textbf{def}\ x \equiv a & \quad \textbf{assume}\ A & \quad \textbf{obtain}\ x \\
\quad \textbf{have}\ B\ x & \quad \textbf{have}\ B\ x & \quad \textbf{have}\ B & \qquad \textbf{where}\ A\ x\ \langle \mathit{proof}\rangle \\
\qquad \langle \mathit{proof}\rangle & \qquad \langle \mathit{proof}\rangle & \qquad \langle \mathit{proof}\rangle & \quad \textbf{have}\ B\ \langle \mathit{proof}\rangle \\
\} & \} & \} & \} \\
\textbf{note}\ \langle \bigwedge x.\ B\ x\rangle & \textbf{note}\ \langle B\ a\rangle & \textbf{note}\ \langle A \Longrightarrow B\rangle & \textbf{note}\ \langle B\rangle
\end{array}$$

3.2 Structured statements

The category *statement* of top-level theorem specifications is defined as follows:

$$\begin{array}{rcl}
\mathit{statement} & \equiv & \mathit{name}\text{: } \mathit{props}\ \textbf{and}\ \ldots \\
 & | & \mathit{context}^*\ \mathit{conclusion} \\
\mathit{context} & \equiv & \textbf{fixes}\ \mathit{vars}\ \textbf{and}\ \ldots \\
 & | & \textbf{assumes}\ \mathit{name}\text{: } \mathit{props}\ \textbf{and}\ \ldots \\
\mathit{conclusion} & \equiv & \textbf{shows}\ \mathit{name}\text{: } \mathit{props}\ \textbf{and}\ \ldots \\
 & | & \textbf{obtains}\ \mathit{vars}\ \textbf{and}\ \ldots\ \textbf{where}\ \mathit{name}\text{: } \mathit{props}\ \textbf{and}\ \ldots\quad \mathbf{|}\quad \ldots
\end{array}$$

A simple *statement* consists of named propositions. The full form admits local context elements followed by the actual conclusions, such as "**fixes** x **assumes** $A\ x$ **shows** $B\ x$". The final result emerges as a Pure rule after discharging the context: $\bigwedge x.\ A\ x \Longrightarrow B\ x$.

The **obtains** variant is another abbreviation defined below; unlike **obtain** (cf. §3.1) there may be several "cases" separated by "**|**", each consisting of several parameters (*vars*) and several premises (*props*). This specifies multi-branch elimination rules.

> **obtains** $\overline{x}$ **where** $\overline{A}\ \overline{x}$ **|** $\ldots\ \equiv$
>
> **fixes** *thesis*
> **assumes** [*intro*]: $\bigwedge\overline{x}.\ \overline{A}\ \overline{x} \Longrightarrow thesis$ **and** $\ldots$
> **shows** *thesis*

Presenting structured statements in such an "open" format usually simplifies the subsequent proof, because the outer structure of the problem is already laid out directly. E.g. consider the following canonical patterns for **shows** and **obtains**, respectively:

theorem
  fixes x **and** y
  assumes $A\ x$ **and** $B\ y$
  shows $C\ x\ y$
proof –
  from ‹$A\ x$› **and** ‹$B\ y$›
  show $C\ x\ y$ ⟨*proof*⟩
qed

theorem
  obtains x **and** y
  where $A\ x$ **and** $B\ y$
proof –
  have $A\ a$ **and** $B\ b$ ⟨*proof*⟩
  then show *thesis* **..**
qed

Here local facts ‹$A\ x$› and ‹$B\ y$› are referenced immediately; there is no need to decompose the logical rule structure again. In the second proof the final "**then show** *thesis* **..**" involves the local reduction rule $\bigwedge x\ y.\ A\ x \Longrightarrow B\ y \Longrightarrow thesis$ for the particular instance of terms a and b produced in the body.

3.3 Structured proof refinement

By breaking up the grammar for the Isar proof language, we may understand a proof text as a linear sequence of individual proof commands. These are interpreted as transitions of the Isar virtual machine (Isar/VM), which operates on a block-structured configuration in single steps. This allows users to write proof texts in an incremental manner, and inspect intermediate configurations for debugging.

The basic idea is analogous to evaluating algebraic expressions on a stack machine: $(a + b) \cdot c$ then corresponds to a sequence of single transitions for each symbol $($, a, $+$, b, $)$, $\cdot$, c. In Isar the algebraic values are facts or goals, and the operations are inferences.

The Isar/VM state maintains a stack of nodes, each node contains the local proof context, the linguistic mode, and a pending goal (optional). The mode determines the type of transition that may be performed next, it essentially alternates between forward and backward reasoning. For example, in *state* mode Isar acts like a mathematical scratch-pad, accepting declarations like **fix**, **assume**, and claims like **have**, **show**. A goal statement changes the mode to *prove*, which means that we may now refine the problem via **unfolding** or **proof**. Then we are again in *state* mode of a proof body, which may issue **show** statements to solve pending sub-goals. A concluding **qed** will return to the original *state* mode one level upwards. The subsequent Isar/VM trace indicates block structure, linguistic mode, goal state, and inferences:

```
  have A →   begin  prove  (A → B) ⟹ #(A → B) (init)
B                   state  (A ⟹ B) ⟹ #(A → B)(resolve (A ⟹ B) ⟹ A →
  proof             state                       B)
   assume A  begin  prove
    show B   end    state  #(A → B)
      ⟨proof⟩ end   state  A → B                (refine #A ⟹ B)
  qed                                           (finish)
```

Here the *refine* inference from §2.2 mediates composition of Isar sub-proofs nicely. Observe that this principle incorporates some degree of freedom in proof composition. In particular, the proof body allows parameters and assumptions to be re-ordered, or commuted according to Hereditary Harrop Form. Moreover, context elements that are not used in a sub-proof may be omitted altogether. For example:

have $\bigwedge x\ y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof –
  fix x **and** y
  assume $A\ x$ **and** $B\ y$
  show $C\ x\ y$ ⟨*proof*⟩
qed

have $\bigwedge x\ y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof –
  fix y **assume** $B\ y$
  fix x **assume** $A\ x$
  show $C\ x\ y$ ⟨*proof*⟩
qed

have $\bigwedge x\ y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof –
  fix x **assume** $A\ x$
  fix y **assume** $B\ y$
  show $C\ x\ y$ ⟨*proof*⟩
qed

have $\bigwedge x\ y.\ A\ x \Longrightarrow B\ y \Longrightarrow C\ x\ y$
proof –
  fix y **assume** $B\ y$
  fix x
  show $C\ x\ y$ ⟨*proof*⟩
qed

Such "peephole optimizations" of Isar texts are practically important to improve readability, by rearranging contexts elements according to the natural flow of reasoning in the body, while still observing the overall scoping rules.

This illustrates the basic idea of structured proof processing in Isar. The main mechanisms are based on natural deduction rule composition within the Pure framework. In particular, there are no direct operations on goal states within the proof body. Moreover, there is no hidden automated reasoning involved, just plain unification.

3.4 Calculational reasoning

The present Isar infrastructure is sufficiently flexible to support calculational reasoning (chains of transitivity steps) as derived concept. The generic proof elements introduced below depend on rules declared as [*trans*] in the context. It is left to the object-logic to provide a suitable rule collection for mixed $=$, $<$, $\leq$, $\subset$, $\subseteq$ etc. Due to the flexibility of rule composition (§2.2), substitution of equals by equals is covered as well, even substitution of inequalities involving monotonicity conditions; see also [21, §6] and [2].

The generic calculational mechanism is based on the observation that rules such as $x = y \Longrightarrow y = z \Longrightarrow x = z$ proceed from the premises towards the conclusion in a deterministic fashion. Thus we may reason in forward mode, feeding intermediate results into rules selected from the context. The course of reasoning is organized by maintaining a secondary fact called "*calculation*", apart from the primary "*this*" already provided by the Isar primitives. In the definitions below, *OF* is *resolve* (§2.2) with multiple rule arguments, and *trans* refers to a suitable rule from the context:

$$\mathbf{also}_0 \equiv \mathbf{note}\ \mathit{calculation} = \mathit{this}$$
$$\mathbf{also}_{n+1} \equiv \mathbf{note}\ \mathit{calculation} = \mathit{trans}\ [\mathit{OF}\ \mathit{calculation}\ \mathit{this}]$$
$$\mathbf{finally} \equiv \mathbf{also\ from}\ \mathit{calculation}$$

The start of a calculation is determined implicitly in the text: here **also** sets *calculation* to the current result; any subsequent occurrence will update *calculation* by combination with the next result and a transitivity rule. The calculational sequence is concluded via **finally**, where the final result is exposed for use in a concluding claim.

Here is a canonical proof pattern, using **have** to establish the intermediate results:

have $a = b$ ⟨*proof*⟩
also have $\ldots = c$ ⟨*proof*⟩
also have $\ldots = d$ ⟨*proof*⟩
finally have $a = d$ **.**

The term "..." above is a special abbreviation provided by the Isabelle/Isar syntax layer: it statically refers to the right-hand side argument of the previous statement given in the text. Thus it happens to coincide with relevant sub-expressions in the calculational chain, but the exact correspondence is dependent on the transitivity rules being involved.

Symmetry rules such as $x = y \Longrightarrow y = x$ are like transitivities with only one premise. Isar maintains a separate rule collection declared via [*sym*], to be used in fact expressions "*a* [*symmetric*]", or single-step proofs "**assume** $x = y$ **then have** $y = x$ **..**".

4 Example: First-order Predicate Logic

In order to commence a new object-logic within Isabelle/Pure we introduce abstract syntactic categories i for individuals and o for object-propositions. The latter is embedded into the language of Pure propositions by means of a separate judgment.

typedecl i
typedecl o

judgment
  $\mathit{Tr} :: o \Rightarrow \mathit{prop}$ (- 5)

Note that the object-logic judgement is implicit in the syntax: writing A produces $\mathit{Tr}\ A$ internally. From the Pure perspective this means "A is derivable in the object-logic".

4.1 Equational reasoning

Equality is axiomatized as a binary predicate on individuals, with reflexivity as introduction, and substitution as elimination principle. Note that the latter is particularly convenient in a framework like Isabelle, because syntactic congruences are implicitly produced by unification of $B\ x$ against expressions containing occurrences of x.

axiomatization
  equal :: $i \Rightarrow i \Rightarrow o$ (**infix** = 50)
where
  refl [*intro*]: $x = x$ **and**
  subst [*elim*]: $x = y \Longrightarrow B\ x \Longrightarrow B\ y$

Substitution is very powerful, but also hard to control in full generality. We derive some common symmetry / transitivity schemes of as particular consequences.

theorem *sym* [*sym*]:
  assumes $x = y$
  shows $y = x$
proof –
  have $x = x$ **..**
  with ‹$x = y$› **show** $y = x$ **..**
qed

theorem *forw-subst* [*trans*]:
  assumes $y = x$ **and** $B\ x$
  shows $B\ y$
proof –
  from ‹$y = x$› **have** $x = y$ **..**
  from *this* **and** ‹$B\ x$› **show** $B\ y$ **..**
qed

theorem *back-subst* [*trans*]:
  assumes $B\ x$ **and** $x = y$
  shows $B\ y$
proof –
  from ‹$x = y$› **and** ‹$B\ x$›
  show $B\ y$ **..**
qed

theorem *trans* [*trans*]:
  assumes $x = y$ **and** $y = z$
  shows $x = z$
proof –
  from ‹$y = z$› **and** ‹$x = y$›
  show $x = z$ **..**
qed

4.2 Group theory

As an example for equational reasoning we consider some bits of group theory. The subsequent locale definition postulates group operations and axioms; we also derive some consequences of this specification.

```
locale $group$ =
  fixes $prod :: i \Rightarrow i \Rightarrow i$ (infix $\circ$ 70)
    and $inv :: i \Rightarrow i$ $((\_^{-1})$ [1000] 999)
    and $unit :: i$ (1)
  assumes $assoc$: $(x \circ y) \circ z = x \circ (y \circ z)$
    and $left\text{-}unit$: $1 \circ x = x$
    and $left\text{-}inv$: $x^{-1} \circ x = 1$
begin

theorem $right\text{-}inv$: $x \circ x^{-1} = 1$
proof –
  have $x \circ x^{-1} = 1 \circ (x \circ x^{-1})$ by (rule left-unit [symmetric])
  also have $\ldots = (1 \circ x) \circ x^{-1}$ by (rule assoc [symmetric])
  also have $1 = (x^{-1})^{-1} \circ x^{-1}$ by (rule left-inv [symmetric])
  also have $\ldots \circ x = (x^{-1})^{-1} \circ (x^{-1} \circ x)$ by (rule assoc)
  also have $x^{-1} \circ x = 1$ by (rule left-inv)
  also have $((x^{-1})^{-1} \circ \ldots) \circ x^{-1} = (x^{-1})^{-1} \circ (1 \circ x^{-1})$ by (rule assoc)
  also have $1 \circ x^{-1} = x^{-1}$ by (rule left-unit)
  also have $(x^{-1})^{-1} \circ \ldots = 1$ by (rule left-inv)
  finally show $x \circ x^{-1} = 1$ .
qed

theorem $right\text{-}unit$: $x \circ 1 = x$
proof –
  have $1 = x^{-1} \circ x$ by (rule left-inv [symmetric])
  also have $x \circ \ldots = (x \circ x^{-1}) \circ x$ by (rule assoc [symmetric])
  also have $x \circ x^{-1} = 1$ by (rule right-inv)
  also have $\ldots \circ x = x$ by (rule left-unit)
  finally show $x \circ 1 = x$ .
qed
```

Reasoning from basic axioms is notoriously tedious. Our proofs work by producing various instances of the given rules (potentially the symmetric form) using the pattern "**have** *eq* **by** (*rule r*)" and composing the chain of results via **also**/**finally**. These steps may involve any of the transitivity rules declared in §4.1, namely *trans* in combining the first two results in *right-inv* and in the final steps of both proofs, *forw-subst* in the first combination of *right-unit*, and *back-subst* in all other calculational steps.

Occasional substitutions in calculations are adequate, but should not be overemphasized. The other extreme is to compose a chain by plain transitivity only, with replacements occurring always in topmost position. For example:

```
  have $x \circ 1 = x \circ (x^{-1} \circ x)$ unfolding left-inv ..
  also have $\ldots = (x \circ x^{-1}) \circ x$ unfolding assoc ..
  also have $\ldots = 1 \circ x$ unfolding right-inv ..
```

```
  also have $\ldots = x$ unfolding left-unit ..
  finally have $x \circ 1 = x$ .
```

Here we have re-used the built-in mechanism for unfolding definitions in order to normalize each equational problem. A more realistic object-logic would include proper setup for the Simplifier, the main automated tool for equational reasoning in Isabelle. Then "**unfolding** *left-inv* **..**" would become "**by** (*simp add*: *left-inv*)" etc.

end

4.3 Propositional logic

We axiomatize basic connectives of propositional logic: implication, disjunction, and conjunction. The associated rules are modeled after Gentzen's natural deduction [4].

```
axiomatization
  $imp :: o \Rightarrow o \Rightarrow o$ (infixr $\longrightarrow$ 25) where
  $impI$ [intro]: $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$ and
  $impD$ [dest]: $(A \longrightarrow B) \Longrightarrow A \Longrightarrow B$

axiomatization
  $disj :: o \Rightarrow o \Rightarrow o$ (infixr $\vee$ 30) where
  $disjE$ [elim]: $A \vee B \Longrightarrow (A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$ and
  $disjI_1$ [intro]: $A \Longrightarrow A \vee B$ and
  $disjI_2$ [intro]: $B \Longrightarrow A \vee B$

axiomatization
  $conj :: o \Rightarrow o \Rightarrow o$ (infixr $\wedge$ 35) where
  $conjI$ [intro]: $A \Longrightarrow B \Longrightarrow A \wedge B$ and
  $conjD_1$ [dest]: $A \wedge B \Longrightarrow A$ and
  $conjD_2$ [dest]: $A \wedge B \Longrightarrow B$
```

The conjunctive destructions have the disadvantage that decomposing $A \wedge B$ involves an immediate decision which component should be projected. The more convenient simultaneous elimination $A \wedge B \Longrightarrow (A \Longrightarrow B \Longrightarrow C) \Longrightarrow C$ can be derived as follows:

```
theorem $conjE$ [elim]:
  assumes $A \wedge B$
  obtains $A$ and $B$
proof
  from ‹$A \wedge B$› show $A$ ..
  from ‹$A \wedge B$› show $B$ ..
qed
```

Here is an example of swapping conjuncts with a single intermediate elimination step:

```
  assume $A \wedge B$
  then obtain $B$ and $A$ ..
  then have B ∧ A ..
```

Note that the analogous elimination for disjunction "**assumes** $A \vee B$ **obtains** $A \mid B$" coincides with the original axiomatization of *disjE*.

We continue propositional logic by introducing absurdity with its characteristic elimination. Plain truth may then be defined as a proposition that is trivially true.

```
axiomatization
  false :: o  (⊥) where
  falseE [elim]: ⊥ ⟹ A

definition
  true :: o  (⊤) where
  ⊤ ≡ ⊥ ⟶ ⊥

theorem trueI [intro]: ⊤
  unfolding true-def ..
```

Now negation represents an implication towards absurdity:

```
definition
  not :: o ⇒ o  (¬ - [40] 40) where
  ¬ A ≡ A ⟶ ⊥

theorem notI [intro]:
  assumes A ⟹ ⊥
  shows ¬ A
unfolding not-def
proof
  assume A
  then show ⊥ by (rule ‹A ⟹ ⊥›)
qed

theorem notE [elim]:
  assumes ¬ A and A
  shows B
proof –
  from ‹¬ A› have A ⟶ ⊥ unfolding not-def .
  from ‹A ⟶ ⊥› and ‹A› have ⊥ ..
  then show B ..
qed
```

4.4 Classical logic

Subsequently we state the principle of classical contradiction as a local assumption. Thus we refrain from forcing the object-logic into the classical perspective. Within that context, we may derive well-known consequences of the classical principle.

```
locale classical =
  assumes classical: (¬ C ⟹ C) ⟹ C
begin
```

```
theorem double-negation:
  assumes ¬ ¬ C
  shows C
proof (rule classical)
  assume ¬ C
  with ‹¬ ¬ C› show C ..
qed

theorem tertium-non-datur: C ∨ ¬ C
proof (rule double-negation)
  show ¬ ¬ (C ∨ ¬ C)
  proof
    assume ¬ (C ∨ ¬ C)
    have ¬ C
    proof
      assume C then have C ∨ ¬ C ..
      with ‹¬ (C ∨ ¬ C)› show ⊥ ..
    qed
    then have C ∨ ¬ C ..
    with ‹¬ (C ∨ ¬ C)› show ⊥ ..
  qed
qed
```

These examples illustrate both classical reasoning and non-trivial propositional proofs in general. All three rules characterize classical logic independently, but the original rule is already the most convenient to use, because it leaves the conclusion unchanged. Note that $(\neg\ C \Longrightarrow C) \Longrightarrow C$ fits again into our format for eliminations, despite the additional twist that the context refers to the main conclusion. So we may write *classical* as the Isar statement "**obtains** ¬ *thesis*". This also explains nicely how classical reasoning really works: whatever the main *thesis* might be, we may always assume its negation!

```
end
```

4.5 Quantifiers

Representing quantifiers is easy, thanks to the higher-order nature of the underlying framework. According to the well-known technique introduced by Church [3], quantifiers are operators on predicates, which are syntactically represented as λ-terms of type $i \Rightarrow o$. Specific binder notation relates *All* $(\lambda x.\ B\ x)$ to $\forall x.\ B\ x$ etc.

```
axiomatization
  All :: (i ⇒ o) ⇒ o  (binder ∀ 10) where
  allI [intro]: (⋀x. B x) ⟹ ∀x. B x and
  allD [dest]: (∀x. B x) ⟹ B a

axiomatization
  Ex :: (i ⇒ o) ⇒ o  (binder ∃ 10) where
  exI [intro]: B a ⟹ (∃x. B x) and
  exE [elim]: (∃x. B x) ⟹ (⋀x. B x ⟹ C) ⟹ C
```

The *exE* rule corresponds to an Isar statement "**assumes** $\exists x.\ B\ x$ **obtains** x **where** $B\ x$". In the following example we illustrate quantifier reasoning with all four rules:

```
theorem
  assumes ∃x. ∀y. R x y
  shows ∀y. ∃x. R x y
proof    — ∀ introduction
  obtain x where ∀y. R x y using ‹∃x. ∀y. R x y› ..    — ∃ elimination
  fix y have R x y using ‹∀y. R x y› ..    — ∀ destruction
  then show ∃x. R x y ..    — ∃ introduction
qed
```

4.6 Further definitions

Real applications routinely work with derived concepts defined in terms of existing principles. This is illustrated below by an artificial notion of *frob* $P\ Q$ for logical predicates P and Q, where *frob* $P\ Q$ holds whenever $P\ x$ and $Q\ y$ hold for some x and y. The most basic definition rephrases this idea in terms of connectives of predicate logic:

```
definition frob P Q ≡ ∃x y. P x ∧ Q y
```

Reasoning with *frob* $P\ Q$ demands characteristic rules, which are derived as follows:

```
theorem frobI [intro]:
  assumes P x and Q y
  shows frob P Q
proof –
  from ‹P x› and ‹Q y› have P x ∧ Q y ..
  then have ∃y. P x ∧ Q y ..
  then have ∃x y. P x ∧ Q y ..
  then show frob P Q unfolding frob-def .
qed

theorem frobE [elim]:
  assumes frob P Q
  obtains x and y where P x and Q y
proof –
  from ‹frob P Q› have ∃x y. P x ∧ Q y unfolding frob-def .
  then obtain x where ∃y. P x ∧ Q y ..
  then obtain y where P x ∧ Q y ..
  then obtain P x and Q y ..
  then show thesis ..
qed
```

Now we can use *frob* in applications, without appealing to the primitive definition again:

```
  assume P a and Q b
```

```
  then have frob P Q ..

  assume frob P Q
  then obtain x y where P x and Q y ..
```

The above transformation of a definition into canonical reasoning patterns works, but is still somewhat cumbersome. This is typically the place to ask for automated reasoning support to deduce *frobI* and *frobE* from *frob-def* in a single invocation each. For example, the following typically appears in applications of Isabelle/HOL [9]:

```
definition frob P Q ≡ ∃x y. P x ∧ Q y

theorem frobI [intro]: P x ⟹ Q y ⟹ frob P Q
  unfolding frob-def by blast

theorem frobE [elim]: frob P Q ⟹ (⋀x y. P x ⟹ Q y ⟹ C) ⟹ C
  unfolding frob-def by blast
```

Here the *blast* method refers to generic Classical Reasoning facilities of Isabelle [14] which may be instantiated to object-logics that provide a rule for classical contradiction. Any realistic object-logic would certainly incorporate such proof tools.

Nevertheless we argue that automated reasoning does not address the above issue adequately. First, there are still three variants of essentially the same logical specification (definition / introduction / elimination). Second, unbounded proof search tends to introduce a factor of uncertainty (it may fail unexpectedly for bigger applications).

Going beyond predicate logic, we propose to use inductive definitions to express the original idea of *frob* $P\ Q$ more directly. The set-theoretic version of inductive definitions in Isabelle/HOL [12] has been recently refined by Stefan Berghofer to work directly with predicates. So we define *frob* $P\ Q$ as a proposition depending on parameters:

```
inductive frob  for P Q  where P x ⟹ Q y ⟹ frob P Q
```

Isabelle/HOL turns this specification into a primitive definition and recovers the introduction rule and its inversion (the elimination rule) as theorems. Since these are declared as [*intro*] and [*elim*], we may reason with *frob* $P\ Q$ immediately as seen before. The **inductive** element uses deterministic proof construction inside, which means it scales up to larger applications, avoiding the dangers of full automated reasoning.

Inductive definitions are far more powerful than illustrated here. More advanced applications involve recursive occurrences of the inductive property, and induction becomes the primary proof principle. Inductive proofs of structured statements raise some additional questions that are addressed by the *induct* method of Isabelle/Isar [19].

5 Conclusion

Isabelle/Isar has been successfully used in much larger applications than the examples presented here. The general concepts for producing theories bit-by-bit are essentially the same, starting from primitive notions and continuing towards complex formalizations. So the common style of Isar proof texts has been represented adequately here.

More elaborate applications will require advanced proof methods (like the generic Simplifier and Classical Reasoner included in Isabelle, specific procedures for fragments of arithmetic etc.), and further derived specification mechanism (like **inductive** shown above). The Isar framework is sufficiently flexible to incorporate such additional components built around the Pure framework.

Apart from gaining this flexibility, the impact of Pure on Isar goes even further: the very idea of composing Isar sub-proofs stems from rule refinement in Pure. In retrospection, structured natural deduction appears to have been present in Pure all the time, we only had make it accessible to the user. Thus the original idea of Isabelle [10,11] to challenge the predominance of classical first-order logic is continued by Isabelle/Isar.

This "puristic" approach to reasoning, without auxiliary logical connectives getting in between, can be observed in the **obtain** / **obtains** element in its extreme. Here the user works directly with collections of local parameters and premises, without having to introduce and eliminate auxiliary propositions involving $\exists$, $\wedge$, $\vee$ first.

On the other hand, *if* such conglomerates of logical connectives need to be accommodated by single steps, Isar will require extra verbosity in repeating some intermediate statements. In this respect, Isar is "declarative" in an extreme sense, where Mizar proofs would admit immediate transformations of the pending problem. Consequently, Mizar usually proceeds quicker in decomposing statements of predicate logic, but Isar does not require any such decomposition, if problems are presented in Pure form. Apart from being theoretically pleasing, the Pure approach also turns out as a genuine practical advantage, e.g. in structured induction proofs involving compound statements [19].

So Isar proofs emphasize statements from the user application, which are composed directly according to natural deduction principles. The need of predicate logic as an auxiliary device is significantly reduced, replacing it by primary proof elements. This is an important insight gained from the experience with Isabelle/Isar in the past few years.

References

1. Clemens Ballarin. Locales and locale expressions in Isabelle/Isar. In Stefano Berardi et al., editors, *Types for Proofs and Programs (TYPES 2003)*, LNCS 3085, 2004.
2. Gertrud Bauer and Markus Wenzel. Calculational reasoning revisited — an Isabelle/Isar experience. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, LNCS 2152, 2001.

3. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 1940.
4. G. Gentzen. Untersuchungen über das logische Schließen. *Math. Zeitschrift*, 1935.
5. J. Harrison. A Mizar mode for HOL. In J. Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: TPHOLs '96*, LNCS 2152, 1996.
6. Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. Locales: A sectioning concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics (TPHOLs '99)*, LNCS 1690, 1999.
7. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4), 1991.
8. Tobias Nipkow. Structured proofs in Isar/HOL. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, LNCS 2646, 2003.
9. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS 2283. 2002.
10. L. C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
11. L. C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.
12. L. C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In Alan Bundy, editor, *Automated Deduction (CADE-12)*, LNAI 814, 1994.
13. L. C. Paulson. Generic automatic proof tools. In R. Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*. MIT Press, 1997.
14. L. C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3), 1999.
15. Lawrence C. Paulson. Natural deduction as higher-order resolution. *Journal of Logic Programming*, 3, 1986.
16. P. Rudnicki. An overview of the Mizar project. In *1992 Workshop on Types for Proofs and Programs*. Chalmers University of Technology, Bastad, 1992.
17. Peter Schroeder-Heister. A natural extension of natural deduction. *Journal of Symbolic Logic*, 49(4), 1984.
18. A. Trybulec. Some features of the Mizar language. Presented at a workshop in Turin, 1993.
19. Makarius Wenzel. Structured induction proofs in Isabelle/Isar. In J. Borwein and W. Farmer, editors, *Mathematical Knowledge Management (MKM 2006)*, LNAI 4108, 2006.
20. Markus Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics: TPHOLs'99*, LNCS 1690, 1999.
21. Markus Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Institut für Informatik, TU München, 2002.
22. Markus Wenzel and Lawrence C. Paulson. Isabelle/Isar. In F. Wiedijk, editor, *The Seventeen Provers of the World*, LNAI 3600. 2006.
23. Freek Wiedijk. Mizar: An impression. Unpublished, 1999.
24. Freek Wiedijk. Mizar light for HOL light. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, LNCS 2152, 2001.
25. Freek Wiedijk. Writing a Mizar article in nine easy steps. Unpublished, 2006.
26. Freek Wiedijk and Markus Wenzel. A comparison of the mathematical proof languages Mizar and Isar. *Journal of Automated Reasoning*, 29(3-4), 2002.

The Curious Inference of Boolos in Mizar and OMEGA*

Christoph E. Benzmüller[1,2] and Chad E. Brown[2]

[1]The University of Cambridge, Cambridge, UK
[2]Universität des Saarlandes, Saarbrücken, Germany
chris|cebrown@ags.uni-sb.de

To Andrzej Trybulec

Abstract. We examine Boolos' curious inference and formalize it in a system based on set theory (Mizar) and a system based on classical higher-order logic (OMEGA). The Boolos example is interesting because while it can in principle be proven using a complete first-order calculus, it is impractical to do so. In our case study we are interested in aspects such as how natural and at what level of granularity Boolos' short second-order proof sketch can be formalized in Mizar and OMEGA.

1 Introduction

In an article in [2, 3], Boolos described a simple theorem of first-order logic which cannot practically be proven by a first-order calculus (even if the calculus includes a cut rule). The idea of the example is as follows:

- Assume we have a constant 1 and a successor function s.
- Axiomatize the definition of a binary function $f(n, x)$ which would grow very rapidly on the natural numbers (analogous to the Ackermann function).
- Assume a predicate D contains 1 and is closed under successor.
- Prove $f(5, 5)$ satisfies the predicate (where 5 is $s(s(s(s(1)))))$.

First-order calculi must essentially compute the value of $f(5, 5)$ in order to prove $D(f(5, 5))$ holds.

Boolos further argues that in a second-order calculus (with comprehension principles), one can prove the theorem very easily. One simply uses the existence of a least set N containing 1 and closed under successor. Since N is the least such set, one obtains an induction principle. Using this induction principle, one can prove that for any n and x in N, $f(n, x)$ must be in $\{x \in N | D(x)\}$. In particular, proving $D(f(5, 5))$ reduces to proving $5 \in N$ – a trivial task.

* This work has been partly funded by the EPSRC project LEO-II (grant EP/D070511/1) at Cambridge University and the SFB 378 project OMEGA at Saarland University.

Since second-order logic is sufficient for performing the short proof, one can clearly also perform the argument in either higher-order logic or first-order set theory. One can perform the argument so long as one works in a strong enough meta-theory to define the set N and the sets used while applying the induction principle. To get a concrete idea of how such an argument looks in modern proof assistant systems, we look at Boolos' curious inference formalized in a system based on set theory (Mizar [1, 11, 7]) and a system based on classical higher-order logic (OMEGA [8, 9]).

Questions that are of interest to us in this case study include

1. How natural can Boolos' proof script be mapped into proof developments in the two systems? How strong is the influence of the logical basis of these systems to the naturalness of these mappings?
2. How much detailed knowledge about each proof assistant is required? How many different commands are needed?
3. Can the logical peculiarities left implicit in Boolos' proof sketch be left implicit in the formal proofs as well? What is the de Bruijn factor of the formal proofs?
4. Are structural changes needed in the formal proofs or can Boolos' proof outline be replayed sequentially step by step? Generally, how far are we away (factoring out the natural language aspects) from a fully automatic verification of Boolos' proof sketch in modern proof assistants?

2 The Curious Inference

We briefly describe the "curious inference" given in detail in [2] and reprinted in Figure 1 with handcrafted proof step annotations.

Boolos constructs a first-order theorem of the form

$$((1) \wedge (2) \wedge (3) \wedge (4) \wedge (5)) \supset (6)$$

where (1)-(6) are given by

(1) $\forall n f(n, 1) = s(1)$
(2) $\forall x f(1, s(x)) = s(s(f(1, x)))$
(3) $\forall n \forall x f(s(n), s(x)) = f(n, f(s(n), x))$
(4) $D(1)$
(5) $\forall x(D(x) \supset D(s(x)))$
(6) $D(f(s(s(s(s(1)))), s(s(s(s(1))))))$

Intuitively, it is clear that (6) follows from (4) and (5) if one can use (1)-(3) to express $f(s(s(s(s(1)))), s(s(s(s(1)))))$ as a term formed by applying s to 1 a finite number of times. However, this "finite number of times" would be an astronomically large number of times. Boolos argues that the number of s's required to represent the simpler term $f(s(s(s(1))), s(s(s(1))))$ (intuitively, $f(4, 4)$) in terms of s and 1 is an exponential stack containing 64K 2s.

Boolos gives the following second-order proof of the first-order theorem. By comprehension (Step 1 in Fig. 1), there is a predicate N such that

$$N(z) \equiv \forall X(X(1) \wedge (\forall y(X(y) \supset X(s(y)))) \supset X(z))$$

$^{\underline{1}}$By the comprehension principle of second order logic, $\exists N \forall z(Nz \leftrightarrow \forall X[X1 \& \forall y(Xy \rightarrow Xsy) \rightarrow Xz])$, $^{\underline{2}}$and then for some N, $\exists E \forall z(Ez \leftrightarrow Nz \& Dz)$.

$^{\underline{3}}$**Lemma 1:** $^{\underline{3.1}}N1$; $^{\underline{3.2}}\forall y(Ny \rightarrow Nsy)$; $^{\underline{3.3}}Nssss1$; $^{\underline{3.4}}E1$; $^{\underline{3.5}}\forall y(Ey \rightarrow Esy)$; $^{\underline{3.6}}Es1$;

$^{\underline{4}}$**Lemma 2:** $\forall n(Nn \rightarrow (\forall x(Nx \rightarrow Efnx)))$
Proof: $^{\underline{4.1}}$By comprehension, $\exists M \forall n(Mn \leftrightarrow \forall x(Nx \Rightarrow Efnx))$. $^{\underline{4.2}}$We want $\forall n(Nn \rightarrow Mn)$. $^{\underline{4.3}}$Enough to show $^{\underline{4.3.1}}M1$ and $^{\underline{4.3.2}}\forall n(Mn \rightarrow Msn)$, for then if $^{\underline{4.4}}Nn$, $^{\underline{4.5}}Mn$.
$^{\underline{4.3.1}}M1$: $^{\underline{4.3.1.1}}$Want $\forall x(Nx \rightarrow Ef1x)$. $^{\underline{4.3.1.2}}$By comprehension, $\exists Q \forall x(Qx \leftrightarrow Ef1x)$. $^{\underline{4.3.1.3}}$Want $\forall x(Nx \rightarrow Qx)$. $^{\underline{4.3.1.4}}$Enough to show $^{\underline{4.3.1.4.1}}Q1$ and $^{\underline{4.3.1.4.2}}\forall x(Qx \rightarrow Qsx)$.
$^{\underline{4.3.1.4.1}}Q1$: $^{\underline{4.3.1.4.1.1}}$Want $Ef11$. $^{\underline{4.3.1.4.1.2}}$But $f11 = s1$ by (1) and $^{\underline{4.3.1.4.1.3}}Es1$ by Lemma 1.
$^{\underline{4.3.1.4.2}}\forall x(Qx \rightarrow Qsx)$: $^{\underline{4.3.1.4.2.1}}$Suppose Qx, $^{\underline{4.3.1.4.2.2}}$i.e. Ef1x. $^{\underline{4.3.1.4.2.3}}$By (2) $f1sx = ssf1x$; $^{\underline{4.3.1.4.2.4}}$by Lemma 1 twice, $Ef1sx$. $^{\underline{4.3.1.4.2.5}}$Thus Qsx and $^{\underline{4.3.1.4.2.6}}M1$.
$^{\underline{4.3.2}}\forall n(Mn \rightarrow Msn)$: $^{\underline{4.3.2.1}}$Suppose Mn, $^{\underline{4.3.2.2}}$i.e. $\forall x(Nx \rightarrow Efnx)$. $^{\underline{4.3.2.3}}$Want Msn, $^{\underline{4.3.2.4}}$i.e. $\forall x(Nx \rightarrow Efsnx)$. $^{\underline{4.3.2.5}}$By comprehension, $\exists P \forall x(Px \leftrightarrow Efsnx)$. $^{\underline{4.3.2.6}}$Want $\forall x(Nx \rightarrow Px)$. $^{\underline{4.3.2.7}}$Enough to show $^{\underline{4.3.2.7.1}}P1$ and $^{\underline{4.3.2.7.2}}\forall x(Px \rightarrow Psx)$.
$^{\underline{4.3.2.7.1}}P1$: $^{\underline{4.3.2.7.1.1}}$Want $Efsn1$. $^{\underline{4.3.2.7.1.2}}$But $fsn1 = s1$ by (1) and $^{\underline{4.3.2.7.1.3}}Es1$ by Lemma 1.
$^{\underline{4.3.2.7.2}}\forall x(Px \rightarrow Psx)$: $^{\underline{4.3.2.7.2.1}}$Suppose Px, $^{\underline{4.3.2.7.2.2}}$i.e. $Efsnx$; $^{\underline{4.3.2.7.2.3}}$thus $Nfsnx$. $^{\underline{4.3.2.7.2.4}}$Want $Efsnsx$. $^{\underline{4.3.2.7.2.5}}$Since $Nfsnx$ and Mn, $Efnfsnx$. $^{\underline{4.3.2.7.2.6}}$But by (3) $fnfsnx = fsnsx$; $^{\underline{4.3.2.7.2.7}}$thus $Efsnsx$.

$^{\underline{5}}$By Lemma 1, $Nssss1$. $^{\underline{6}}$By Lemma 2, $Efssss1ssss1$. $^{\underline{7}}$Thus, $Dfssss1ssss1$, as desired.

Fig. 1. Boolos' original proof sketch from [2, 3]; we have identified single proof steps and annotated them for reference in this paper.

and a predicate E (Step 2) such that

$$E(z) \equiv N(z) \wedge D(z)$$

Boolos states Lemma 1 (Steps 3.1-6) without proof and proves Lemma 2 (Step 4):

LEMMA 1: $N(1)$; $\forall y(N(y) \supset N(s(y)))$; $N(s(s(s(s(1)))))$; $E(1)$; $(\forall y(E(y) \supset E(s(y))))$; $E(s(1))$

LEMMA 2: $\forall n(N(n) \supset \forall x(N(x) \supset E(f(n, x))))$

The proof of LEMMA 2 makes use of the comprehension principle several times (Steps 4.1, 4.3.1.2, and 4.3.2.5) in order to obtain predicates (M, Q, and P) for use in the induction principle implicit in the definition of N. We outline the proof and give the instances of comprehension:

- (Steps 4.1-5) To prove $\forall x(N(x) \supset E(f(n,x)))$ for any n satisfying $N(n)$, Boolos uses a predicate M satisfying
$$M(n) \equiv \forall x(N(x) \supset E(f(n,x)))$$
- (Steps 4.3.1.1-4) To prove $M(1)$, Boolos uses a predicate Q satisfying
$$\forall x((Qx) \equiv E(f(1,x)))$$
and easily proves $Q(1)$ (Steps 4.3.1.4.1.1-3) and closure of Q under s (Steps 4.3.1.4.2.1-6).
- (Steps 4.3.2.1-7) To prove M is closed under successor, suppose n satisfies $N(n)$ and $M(n)$. The goal is now to prove $M(s(n))$. Equivalently, we should prove $\forall x(N(x) \supset E(f(s(n),x)))$. We can prove this by induction on x using a predicate P satisfying
$$\forall x((Px) \equiv E(f(s(n),x)))$$
Boolos argues $P(1)$ (Steps 4.3.2.7.1.1-3) and P is closed under successor (Steps 4.3.2.7.2.1-7), completing the argument. Then he completes the overall proof by using LEMMA 1 and LEMMA 2 (Steps 5-7).

In summary, Boolos uses comprehension to obtain five predicates: N, E, M, Q, and P.

In our set theory version below, we will use a separation principle to obtain sets corresponding to these five predicates. In the higher-order version, we will use λ-abstraction to define corresponding predicate terms (sets) of type $\iota \to o$.

3 Mizar Version

We first describe a version of the proof in Mizar [1, 11, 7]. We begin by reserving A to be a non empty set corresponding to the domain of the first-order problem.

```
reserve A for non empty set;
```

We reserve a to be an element of A, playing the role of 1 in the first order problem. Also, n, x and y will play the role of first-order variables.

```
reserve a,n,x,y for Element of A;
```

We reserve s as a function from A to A and f as a function from $A \times A$ to A, playing the roles of s and f in the first-order problem.

```
reserve s for Function of A,A;
```

```
reserve f for Function of [: A,A :],A;
```

The predicate D in the first-order problem corresponds to a subset of the set A.

```
reserve D for (Subset of A);
```

Second-order predicates will correspond to subsets of A as well. However, we reserve w, X, and z to denote generic sets (without assuming they are subsets of A).

```
reserve w,X,z for set;
```

For variables, we will always use one of the alphabetic characters above. Starting from the variable names reserved above, we can form terms and formulas according to the following grammar:

term ::= *variable* | 's.'*term* | 'f.[' *term* ',' *term* ']' | '$1' | '(' *term* ')'
formula ::= 'thesis' | *term* '=' *term* | *term* 'in' *term* | *predicate* '[' *term* ']'
| *formula* '&' *formula* | *formula* 'implies' *formula*
| 'for' *variable-list* 'holds' *formula* | '(' *formula* ')'

A *predicate* will be an identifier corresponding to a `defpred` reasoning item (described below). A *variable-list* is a list of variables separated by commas.

Warning: This is a simplification of the Mizar syntax for terms and formulas. Terms and formulas as given above are sufficient for the Mizar encoding of the Boolos example.

We can now declare the main theorem in Mizar as follows:

```
theorem BoolosCuriousInference:
  (for n holds ((f.[n,a]) = s.a)) &
  (for x holds ((f.[a,s.x]) = s.(s.(f.[a,x])))) &
  (for n,x holds f.[s.n,s.x] = f.[n,f.[s.n,x]]) &
  (a in D) &
  (for x st (x in D) holds (s.x in D))
  implies
  f.[s.(s.(s.(s.a))),s.(s.(s.(s.a)))] in D
```

In Mizar, such a theorem in an article should be followed by a *Justification*. We want this justification to correspond closely to Boolos' proof sketch. Boolos outlines a proof of this theorem by applying comprehension to obtain N and E, stating Lemma 1 (without proof), proving Lemma 2, and finally concluding the main result. In order to understand how to translate this proof sketch to a Mizar justification, we describe a subset of the full Mizar syntax sufficient for this example.

- A *Justification* can either be
  - empty,
  - a simple justification of the form 'by' *References*,
  - a simple justification of the form 'from' *Scheme-Reference*, or
  - a *Proof*.
- A *Proof* is a list of *Reasoning Items* between the keywords 'proof' and 'end'.
- A *Reasoning Item* is of one of the following forms:
  - 'assume' *Conditions* ';'
  - 'let' *Identifier* 'such' *Conditions* ';'
  - 'defpred' *Identifier* '[set]' 'means' *formula* ';'
  - 'consider' *Identifier* 'such' *Conditions* 'from' *Scheme-Reference* ';'
  - ['then' | 'hence' | 'thus'] [*Identifier* ':'] *formula Justification* ';'
- *Conditions* is always of the form
'that' [*Identifier* ':'] *formula* { 'and' [*Identifier* ':'] *formula* }
- Every *Identifier* we use will be a string of alphanumeric characters starting with an alphabetic character.

- *References* is a list of separated by commas. Each member of this list is either an *Identifier* (a local reference to something labeled in the article) or a reference to the a theorem or definition in the Mizar Mathematical Library (MML). The only references to the MML we will use are 'ZFMISC_1:106', 'XBOOLE_0:def 1', and 'FUNCT_2:7'.
- The only *Scheme-Reference* we will use is 'XBOOLE_0:sch 1' (see below).

Warning: The description above can be used to create valid Mizar syntax, but is a simplification of the actual Mizar grammar. For the full Mizar grammar, see [11] or the grammar given on the Mizar web site.

We followed Boolos' outline to write a Mizar version of the proof (see Appendix A.1). The resulting file contains 177 non-comment lines and consisted of 78 reasoning items. This corresponds closely to the 60 steps we identified in Boolos' proof in Figure 1. We now briefly consider some particular aspects of the Mizar version.

The first two uses of comprehension in Boolos' proof are to obtain N and E. In Mizar, we can obtain the existence of these sets using a form of separation, referenced as a scheme from the MML article XBOOLE_0 [5]. In order to use this scheme, we define local predicates Np and Ep. Obtaining references to such schemes from the vast MML can be a barrier to writing proofs in Mizar, though some recent work by Urban [10] should help in this respect. In our case, we used grep in the MML directory to locate examples of set separation. Once we know such a set exists, we can use consider to give it the appropriate name. (Double colons indicate comments.)

```
:: 1. existence of N
   defpred Np[set] means (for X holds
   (((a in X) &
     (for y holds (y in X) implies (s.y in X)))
    implies ($1 in X)));
   consider N such that
   Ndef:for z holds z in N iff z in A & Np[z]
   from XBOOLE_0:sch 1;
:: 2. existence of E
   defpred Ep[set] means (($1 in N) & ($1 in D));
   consider E such that
   Edef:for z holds z in E iff z in A & Ep[z]
   from XBOOLE_0:sch 1;
```

Lemma 1 is actually six short lemmas. We split Lemma 1 into six lines in the main proof, each given without justification. Mizar does not accept these inferences. Instead, Mizar reports that the inferences are not accepted and continues processing the file. This is a very useful feature of Mizar since it allows us to complete the outline of the proof and then fill in any missing justifications afterwards.

```
:: 3. Lemma 1
:: 3.1
LEMMA1a: (a in N);
:: 3.2
```

```
LEMMA1b: (for y holds (y in N) implies (s.y in N));
:: 3.3
LEMMA1c: s.(s.(s.(s.a))) in N;
:: 3.4
LEMMA1d: a in E;
:: 3.5
LEMMA1e: (for y holds (y in E) implies (s.y in E));
:: 3.6
LEMMA1f: s.a in E;
```

The main part of the proof is the proof of Lemma 2. We include the proof in the Mizar article at the same level of detail Boolos gives. The only significant complication involves steps such as 4.2 (see Figure 1) in which Boolos states "We want..." or 4.3 in which Boolos states "Enough to show..." In each of these steps, Boolos is implicitly asserting that instead of showing the current goal (the "thesis" in Mizar), we can reduce the current goal to the new, explicitly given, goal or goals. One can simulate such steps in Mizar as follows. Suppose we must show G, but we want to show A. The justification that we can reduce G to A is a justification of $A \supset G$. Using a proof of A and $A \supset G$, we can complete the proof of G. The following outlines how to perform such a simulation of "enough to show" in Mizar:

```
 enoughtoshow: A implies thesis;
 A
 proof
 ...
 end;
 hence thesis by enoughtoshow;
```

The fragment above corresponds to a single "subgoal reduction" step. For a concrete example, consider proof Step 4.2:

```
:: 4.2 Begin We want...
      wewant42: (for n holds (n in N) implies (n in M))
                                           implies thesis;
      (for n holds (n in N) implies (n in M))
      proof
        :: 4.3 Begin
        ...
        :: 4.3 End
      end;
      hence thesis by wewant42;
:: 4.2 End
```

Mizar could not verify the file with the proof outline since some justifications were not explicit in Boolos' proof. In particular, there were 25 inferences Mizar did not accept. However, starting with the Mizar article containing the outline, it was not difficult to fill in the remaining inferences to obtain a proof Mizar accepts. Sometimes, this was simply a matter of making justifications explicit. For example, the final part of Lemma 1 could be justified simply by referring to the previous parts:

```
    LEMMA1f: s.a in E by LEMMA1d,LEMMA1e;
```

For most other parts of Lemma 1, we simply gave an explicit proof.

In a few cases, we needed auxiliary lemmas. For example, Boolos often reduces showing every member of N is a member of another set X to showing the base case and the induction case. In the proof outline, the first example is Step 4.3:

```
  ets43: a in M & (for n holds (n in M) implies (s.n in M))
                                             implies thesis;
```

where the thesis is that every n in N is also in M. Mizar considers this step to be unjustified. It is not difficult justify such statements using the definition of N. However, we did not want to add too many steps to the outline in order to complete the proof. Instead, we included (and proved) a lemma Nindprinc:

```
theorem Nindprinc:
  (for z holds z in N iff z in A &
  (for X holds (((a in X) &
  (for y holds (y in X) implies (s.y in X))) implies (z in X))))
  &
  (a in X)
  &
  (for y st (y in X) holds (s.y in X))
  implies
  (for n holds (n in N) implies (n in X))
```

Using this lemma, Step 4.3 can be justified as follows:

```
  ets43: a in M & (for n holds (n in M) implies (s.n in M))
                                  implies thesis by Nindprinc,Ndef;
```

Another of the lemmas concludes that terms of the form f.[x,y] are elements of A. To prove this particular lemma, three references to the MML were made. These three references plus the set separation scheme are the only references to the MML in the article.

In total, we stated and proved 7 such lemmas.

The Mizar article with the complete proof contains 313 non-comment lines (see Appendix A.2). The main proof contains 104 reasoning items, and the proofs of the lemmas contain an additional 35 reasoning items. The file is not only short, but also quite readable.

4 OMEGA Version

In this section we discuss a solution of Boolos' problem in OMEGA [8, 9], a proof assistant based on classical higher-order logic (Church's simple type theory [4]).

In order to construct the proof in OMEGA the user essentially only needs to know 7 quite general proof commands, they are:

- call-otter-on-node (proof line justification is 'Otter'; see Appendix B.1): calls the first order theorem prover otter in order to close a subgoal; these calls are performed whenever Boolos leaves out the logical details in his sketch. In each call to OTTER the problem is implicitly mapped to first-order logic using an applicational transformation; using OMEGA's TRAMP subsystem [6] these proofs can be, upon request, translated to OMEGA's natural natural deduction calculus, integrated in OMEGA's proof object, and verified.
- support (no extra proof line justification needed) this command restricts the available hypotheses for an open proof line; we here use support exclusively in connection with call-otter-on-node in order to precisely specify before each call the hypotheses OTTER may use in its proof attempt. Both commands could easily be (and in fact should be) combined into one single command.
- local-def-intro (proof line justification is 'Local-Def'): introduces a local definition; each time Boolos uses the comprehension principle to introduce a new definition, we use 'local-def-intro' command to introduce the corresponding definition in OMEGA.
- defn-contract-local-def and defn-expand-local-def (proof line justifications are 'CDef' and 'EDef'): these commands allow to expand (unfold) and contract (fold) definitions; note that the clever use of definitions is an essential aspect in Boolos' proof.
- impi (proof line justification is 'IMPI'): the natural deduction rule for implication introduction.
- foralli (proof line justification is 'FORALLI'): the natural deduction command for introduction of a universal quantification.
- ande (proof line justification is 'ANDE'): the natural deduction rule for conjunction elimination.

Furthermore, the commands load-problems and prove for loading and initializing the problem are needed. The command show-pds (for displaying the current partial proof in OMEGA's emacs interface and the graphical interface LOUI (see Appendix B.4)) is useful but not mandatory.

The problem is initially defined in OMEGA's Post syntax as follows:

```
(th~defproblem boolos-curious-inference
  (in boolos)
  (constants (s (i i)) (f ((i i) i)) (one i) (D (o i)))
  (assumption a1 (forall (lam (n i) (= (f n one) (s one)))))
  (assumption a2
    (forall (lam (x i) (= (f one (s x)) (s (s (f one x)))))))
  (assumption a3
    (forall (lam (n i)
      (forall (lam (x i) (= (f (s n) (s x)) (f n (f (s n) x))))))))
  (assumption a4 (D one))
  (assumption a5 (forall (lam (x i) (implies (D x) (D (s x))))))
  (conclusion conc (D (f (s (s (s (s one)))) (s (s (s (s one))))))))
```

This problem specification is then added to OMEGA's theory library.

The complete interactive session is given in Appendix B.5, the proof script obtained from this interactive session in Appendix B.2, and the final proof object in ASCII, LaTeX, and GUI representation in Appendices B.3, B.1, and B.4. As for Mizar, we here discuss only some specific fragments of the formal development of Boolos' proof sketch in OMEGA.

After initialization of the OMEGA system with the above proof problem, we proceed step by step along Boolos' proof sketch. In Step 1 Boolos defines property N which introduces, as mentioned before, an induction principle wrt. the constructors

1 and s. Step 2 introduces E which connects N and D. In OMEGA these definition introduction steps (as well as all further definition introduction steps) are handled with `local-def-intro` which takes as single argument the definiens of the new definition. For instance, the set N is defined by the definiens (λ-term of type $\iota \leftarrow o$):

$$\lambda z_\iota[(\forall X_{\iota \leftarrow o} X(One) \land (\forall y_\iota X(y) \Rightarrow X(s(y)))) \Rightarrow X(z)]$$

The name (definiendum) of each local definition is automatically chosen by OMEGA, so that N becomes $LD1$ and E becomes $LD2$. Here is the protocol excerpt of the first steps of the proof in OMEGA:

```
OMEGA: prove boolos-curious-inference
 Changing to proof plan BOOLOS-CURIOUS-INFERENCE-1
OMEGA: LOCAL-DEF-INTRO (LAM (Z I) (FORALL (LAM (X (O I)) (IMPLIES (AND (X ONE)
 (FORALL (LAM (Y I) (IMPLIES (X Y) (X (S Y)))))) (X Z)))))
OMEGA:  LOCAL-DEF-INTRO (LAM (Z I) (AND (LD1 Z) (D Z)))
OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                  HYP
A2   (A2)        ! (FORALL [X:I]                                         HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))
A3   (A3)        ! (FORALL [N:I,X:I]                                     HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))
A4   (A4)        ! (D ONE)                                               HYP
A5   (A5)        ! (FORALL [X:I]                                         HYP
                    (IMPLIES (D X) (D (S X))))
LD1  (LD1)       ! (=DEF                                           LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))
LD2  (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))            LOCAL-DEF
                ...
CONC (A1 A2 A3  ! (D                                                    OPEN
      A4 A5)      (F
                   (S (S (S (S ONE))))
                   (S (S (S (S ONE))))))
```

In Steps 3.1-6 Boolos introduces the six statements of Lemma 1. As in Mizar we instead introduce six individual lemmas for the final goal line 'conc'. Then we subsequently prove them with the help of OTTER after appropriately choosing the support nodes and unfolding the definitions. We illustrate only the Step 3.1 and omit the others since they are analogous (see also the lines l1-l12 in the final proof object in the Appendix B.1).

```
OMEGA: LEMMA CONC (LD1 ONE)
OMEGA: DEFN-CONTRACT-LOCAL-DEF L1 () LD1 (0)
OMEGA: SUPPORT L2 ()
                 ...
L2 (A1 A2 A3  ! (FORALL [DC-13:(O I)]                              OPEN
    A4 A5)      (IMPLIES
                 (AND
                  (DC-13 ONE)
```

```
                  (FORALL [DC-17:I]
                   (IMPLIES
                    (DC-13 DC-17)
                    (DC-13 (S DC-17)))))
                 (DC-13 ONE)))
OMEGA: CALL-OTTER-ON-NODE L2 ...
...
-------- PROOF --------
...
OMEGA: show-pds
...
L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                      OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                   (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)            DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
...
```

Next, we illustrate how the Steps 4.3.2.1-7 are treated; an interesting aspect here is that the definition of P has to refer to the locally introduced Eigenvariable n (named $N1$ in the OMEGA proof). We start with Step 4.3.2.1 is (note that OMEGA has previously chosen the name $LD3$ for predicate M):

```
                  ...
L16  (A1 A2 A3  ! (FORALL [N:I]                                          OPEN
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))
```

We extract hypothesis and conclusion and unfold the definition of $LD3$.

```
OMEGA: FORALLI L16 n1 ()
OMEGA: IMPI L26
OMEGA: DEFN-EXPAND-LOCAL-DEF () L27 LD3 (0)
OMEGA: DEFN-CONTRACT-LOCAL-DEF L28 () LD3 (0)
OMEGA: show-pds
LD3  (LD3)       ! (=DEF                                            LOCAL-DEF
                    LD3
                    ([N].
                     (FORALL [X:I]
                      (IMPLIES
                       (LD1 X)
                       (LD2 (F N X))))))
L27  (L27)       ! (LD3 N1)                                                HYP
L29  (L27)       ! (FORALL [DC-217:I]       DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                    (IMPLIES
                     (LD1 DC-217)
                     (LD2 (F N1 DC-217))))
L30  (L27 A1     ! (FORALL ·[DC-225:I]                                    OPEN
      A2 A3 A4      (IMPLIES
      A5)            (LD1 DC-225)
                     (LD2 (F (S N1) DC-225))))
L28  (L27 A1     ! (LD3 (S N1))           DEFN-CONTRACT-LOCAL-DEF: ((0)) (L30 LD3)
      A2 A3 A4
      A5)

L26  (A1 A2 A3   ! (IMPLIES (LD3 N1) (LD3 (S N1)))                 IMPI: (L28)
      A4 A5)
L16  (A1 A2 A3   ! (FORALL [N:I]                             FORALLI: (N1) (L26)
      A4 A5)        (IMPLIES (LD3 N) (LD3 (S N))))
```

Now the predicate P (here automatically named $LD5$) is introduced and the problem is then reduced to showing that P holds for One and is closed under s.

```
OMEGA: LOCAL-DEF-INTRO (LAM (X I) (LD2 (F (S N1) X)))
OMEGA: LEMMA L30 (FORALL (LAM (X I) (IMPLIES (LD1 X) (LD5 X))))
OMEGA: DEFN-EXPAND-LOCAL-DEF L30 L31 LD5 (1 0 2 0)
OMEGA: LEMMA L31 (LD5 ONE)
OMEGA: LEMMA L31 (FORALL (LAM (X I) (IMPLIES (LD5 X) (LD5 (S X)))))
OMEGA: DEFN-CONTRACT-LOCAL-DEF L31 () LD1 (1 0 1 0)
OMEGA: show-pds
...
LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                           LOCAL-DEF
L30  (L27 A1    ! (FORALL [DC-225:I]           DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L31 LD5)
      A2 A3 A4     (IMPLIES
      A5)           (LD1 DC-225)
                    (LD2 (F (S N1) DC-225))))
L31  (L27 A1    ! (FORALL [X:I]              DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L34 LD1)
      A2 A3 A4     (IMPLIES (LD1 X) (LD5 X)))
      A5)
L32  (L27 A1    ! (LD5 ONE)                                                           OPEN
      A2 A3 A4
      A5)
L33  (L27 A1    ! (FORALL [X:I]                                                       OPEN
      A2 A3 A4     (IMPLIES (LD5 X) (LD5 (S X))))
      A5)
L34  (L27 A1    ! (FORALL [DC-239:I]                                                  OPEN
      A2 A3 A4     (IMPLIES
      A5)           (FORALL [DC-250:(O I)]
                     (IMPLIES
                      (AND
                       (DC-250 ONE)
                       (FORALL [DC-254:I]
                        (IMPLIES
                         (DC-250 DC-254)
                         (DC-250 (S DC-254)))))
                      (DC-250 DC-239)))
                    (LD5 DC-239)))
```

OTTER can be employed to verify the reduction of lines $L34$ to lines $L32$ and $L33$. Thus, as for lemmas 3.1-6, OTTER can be employed to check the logical details not mentioned explicitly in Boolos proof sketch and to treat these steps implicit in OMEGA as well.

```
OMEGA: SUPPORT L34 (L32 L33)
OMEGA: CALL-OTTER-ON-NODE L34 ...
...
-------- PROOF --------
...
OMEGA:
```

In fact, the idea to keep logical details implicit with the help of OTTER allows to further shorten the proof sketch of Boolos. We exemplarily demonstrate this for proof Steps 4.3.1.4.2.1-6 of Boolos original proof sketch. Here we avoid the explicit extraction of hypothesis and conclusion (with `FORALLI` and `IMPI`) and only unfold the definition of $LD4$ (resp. Q) before calling OTTER (cf. the proof lines $L21$, $L24$, and $L25$ in Appendix B.1).

In OMEGA there are altogether 83 proof steps needed (see the uncommented lines in OMEGA's proof script in Appendix B.2; it consists of 83 commands to OMEGA) to replay Boolos' proof idea as we sketched above. This number is quite close to the 60 annotations used to identify single proof steps in Boolos' original proof sketch in Figure 1. Note that each call to OTTER (there are 16 altogether) comes with a previous call of `support` in order to specify which hypotheses OTTER shall use. By simply integrating both commands we would get $83 - 16 = 67$ steps which comes actually very close to Boolos' original number of steps. The number in OMEGA can possibly further reduced by calling OTTER (or other external systems available to OMEGA) to larger subproblems. But this was not the idea of the exercise, since we aimed at replaying Boolos' proof as close as possible. The detailed proof-object is displayed in Appendix B.3; it has 357 lines and 2949 characters. The quite readable LaTeX conversion of this proof-object is presented in Appendix B.1. Note, that the justification 'OTTER' can be further expanded in which case the subproofs obtained from OTTER are transformed into natural deduction proof objects in OMEGA to be verified. We have not done this here. By doing so the proof object will clearly grow.

5 Evaluation

Boolos' famous curious inference has been formalized in two modern proof assistants: Mizar and OMEGA. We compare and discuss these formalizations according to the following aspects:

1. **How natural can Boolos' proof script be mapped into proof developments in the systems? How strong is the influence of logical basis of these systems to the 'naturalness' of these mappings?**
   Boolos' original proof outline with its 60 steps can be very closely replayed in both Mizar and OMEGA. The different logical bases of the systems does not appear to play a significant role.
   In Mizar, one can use the proof outline to create a Mizar article with 78 proof steps (reasoning items). Though this article cannot be verified by Mizar, the file can be completed by adding lemmas, justifications and further proof steps in order to obtain a verified Mizar article. The completed article contains 139 proof steps (reasoning items). The Mizar proof is quite readable.
   In OMEGA, there are 83 proof steps needed (and this number could easily be further reduced to 67 by a natural merge of the `call-otter-on-node` and `support` commands). The formalization in POST is admittedly not very readable. In the GUI representation (see Appendix B.4) the readability is significantly better. OMEGA experts, however, seldom use OMEGA's LOUI interface in practice. The machine-oriented proof script of OMEGA is not readable/informative to human users at all. By replaying it step by step in OMEGA and by investigating the development of the partial proof in OMEGA's emacs or LOUI interface, however, a good understanding of the proof script can be obtained.
2. **How much detailed knowledge about the proof assistant is needed? How many different commands are needed?**
   Anyone can write a proof in Mizar after perusing [11] or [7]. There are a few essential ingredients:

- One needs to know the concrete syntax for quantifiers and logical connectives.
- One needs to know the syntax for different kinds of reasoning items.
- One needs to know how to give simple justifications.

There are other issues as well (e.g., vocabularies and definitions) which one can learn as needed. Sometimes giving a justification can involve finding an appropriate definition, theorem or scheme in the MML (Mizar Mathematical Library). Finding such references can be challenging for a novice.

In order to construct the proof in OMEGA the user essentially only needs to have a basic understanding of the POST syntax and to know the 7(+2) rather general proof commands as mentioned before.

3. **Can the logical peculiarities left implicit in Boolos' argument be left implicit in these proof developments as well? What is the de Bruijn factor of the formal proofs?**

One can leave most of the logical details implicit in Mizar, but one sacrifices full verification. In order to obtain full verification, some of these logical details must be made explicit (though many logical rules are built into the Mizar verifier). This is, in fact, a natural way to write a Mizar article: one first writes an article in which some justifications are not accepted, then one fills in the logical details.

The steps that are left implicit by Boolos can be treated implicit in OMEGA as well. This is done by calling the first order theorem prover OTTER[1]. In order to automatically verify the proofs delivered by OTTER within OMEGA the TRAMP system can be employed. Generally we can say that Boolos' proof sketch is detailed enough such that all remaining logical peculiarities can already be automatically dealt with in OMEGA.

To obtain the de Bruijn factor for both formalizations, we consider the (relative) sizes of the following compressed (using gzip) files:

- A LaTeX file containing Figure 1 (Boolos' proof sketch) without annotations: 637 bytes.
- The completed Mizar article (without comments): 2310 bytes; de Bruijn factor 3.6.
- There are two files we may consider in OMEGA: (A) the proof script (see Appendix B.2) that can be employed to automatically reconstruct the final proof object in OMEGA, and (B) the ASCII representation of the final proof object itself (see Appendix B.3). For (A) we have 838 bytes and de Bruijn factor 1.3 and for (B) we obtain 2602 bytes and de Bruijn factor 4, 1. (Note, that by merging the commands `call-otter-on-node` and `support` commands in OMEGA we would reach de Bruijn factor close to 1.1 for file (A) – a very impressive figure.)

4. **Are structural changes needed in the formal proofs or can Boolos' proof outline be replayed sequentially step by step? Generally, how far are we away, from a fully automatic verification of Boolos' rather detailed proof script in modern proof assistants?**

[1] There is no particular reason why we have chosen OTTER; other first-order ATPs should be capable of proving all subgoals as well.

The only significant structural changes necessary to code the proof into Mizar resulted from Boolos' use of subgoals in the proof. One can imagine a program translating the LaTeX version of Boolos' proof sketch into the first Mizar article and then another program (using an automated theorem prover) filling in the remaining gaps to form the second Mizar article.

In OMEGA none (or only very minor) structural modifications of Boolos' original proof sketch are needed.[2] The proof can in fact be replayed very naturally step by step with a rather small amount of expertise about OMEGA. The main challenge for full automatic verification seems to be the bridge between the mixed use of natural language and mathematical formulas in Boolos' proof sketch and the respective proof commands in OMEGA.

6 Conclusion

We have shown that the (second-order) proof sketch of Boolos for his curious inference can be very naturally formalized and verified in Mizar and OMEGA. The fact that Mizar is based on first-order set theory and OMEGA on classical higher-order logic is of minor influence to the the naturalness of proof development. We have shown in particular that Boolos proof sketch can be replayed (in particular in OMEGA) at the same level of granularity (many details that Boolos leaves out can be omitted in the formal developments as well) and step by step, i.e., without need for major structural proof reorganisation. Assuming sufficient progress in the area of semantic analysis of natural language (in fact, mixed mathematical and natural language), this gives hope that we will one day be able to fully automatically analyze and verify human proof sketches such as the one by Boolos.

Furthermore, as we have already indicated in the paper, it should be possible to further raise the level of granularity. The stronger the automated subsystems (such as the ATPs in OMEGA) will get, the fewer details have to be provided to verify the proof. The full automation of Boolos curious inference seems not to be in reach and it will be a challenge problem to automated theorem proving for a long time to come. The key steps, namely the invention of the predicates $N - P$, the invention of the appropriate lemmata, and the clever unfolding of definitions are not yet sufficiently supported in todays theorem provers.

References

1. Grzegorz Bancerek and Piotr Rudnicki. A Compendium of Continuous Lattices in MIZAR. *J. Autom. Reason.*, 29(3-4):189–224, 2002.
2. George Boolos. A curious inference. *Journal of Philosophical Logic*, 16:1–12, 1987.
3. George Boolos. *Logic, logic, and logic.* Harvard University Press, 1998.

[2] Note that we have slightly modified (shortened) Boolos' argument in Steps 4.3.1.4.2.1 – 6 for demonstration purposes only. As Steps 4.3.2.1 – 7 show, we could well have followed Boolos' original proof sketch more closely by explicitly extracting hypothesis and conclusion.

4. Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.
5. Library Committee. Boolean properties of sets — definitions. *Journal of Formalized Mathematics*, EMM, 2002. http://mizar.org/JFM/EMM/xboole_0.html.
6. Andreas Meier. System description: TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level. In David McAllester, editor, *Automated Deduction - CADE-17*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 460–464, Pittsburgh, PA, USA, 2000. Springer-Verlag.
7. Michal Muzalewski. *An Outline of PC Mizar.* In R.Matuszewski, editor, Series of the Fondation Ph. le Hodey. Brussels, 1993.
8. Jörg Siekmann, Christoph Benzmüller, and Serge Autexier. Computer supported mathematics with omega. *Journal of Applied Logic*, 4(4):533–559, 2006.
9. Jörg Siekmann, Christoph Benzmüller, Armin Fiedler, Andreas Meier, Immanuel Normann, and Martin Pollet. *Proof Development in OMEGA: The Irrationality of Square Root of 2*, pages 271–314. Kluwer Applied Logic series (28). Kluwer Academic Publishers, 2003. ISBN 1-4020-1656-5.
10. Josef Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *International Journal on Artificial Intelligence Tools*, 2005. forthcoming, available online at http://ktiml.mff.cuni.cz/~urban/MoMM/momm.ps.
11. F. Wiedijk. Mizar: An impression, 1999.

A Mizar Version

By Chad E. Brown.

In the following subsections we give the file corresponding to the outline of the proof (with 25 unaccepted inference) and the file in which all inferences are accepted. Both of these Mizar articles begin by declaring the following environment.

```
   environ

 vocabularies FUNCT_1, FINSEQ_1, RELAT_1, FINSET_1, CARD_1,
   BOOLE, PARTFUN1, ARYTM_1, INT_1, RLSUB_2, TARSKI, FINSEQ_4,
   ORDINAL2, ARYTM, ZFMISC_1;
 notations TARSKI, XBOOLE_0, ENUMSET1, SUBSET_1, ORDINAL1,
   ORDINAL2, RELAT_1, NUMBERS, XCMPLX_0, FINSEQ_1, FUNCT_1,
   PARTFUN1, CARD_1, FUNCT_2, FINSET_1, INT_1, NAT_1, FINSEQ_3,
   XXREAL_0, ZFMISC_1, RELSET_1;
 constructors TARSKI, XBOOLE_0, ENUMSET1, WELLORD2, FUNCT_2,
   XXREAL_0, NAT_1, INT_1, MEMBERED, FINSEQ_3, ORDINAL2,
   ZFMISC_1;
 registrations XBOOLE_0, FINSEQ_1, FUNCT_1, INT_1, FINSET_1,
   RELSET_1, MEMBERED, ARYTM_3;
 requirements REAL, NUMERALS, SUBSET, BOOLE, ARITHM;
 theorems XBOOLE_0, TARSKI, FUNCT_1, FUNCT_2, ZFMISC_1;
 schemes XBOOLE_0;
```

A.1 Proof Outline in Mizar

Below we give the Mizar article which closely corresponds to the outline given by Boolos. Mizar does not accept 25 of the inferences in this file.

```
   begin

reserve A for non empty set;
reserve a,n,x,y for Element of A;
reserve s for Function of A,A;
reserve f for Function of [: A,A :],A;
reserve D for (Subset of A);
reserve X,z,N,E,M,P,Q for set;

theorem BoolosCuriousInference:
  (for n holds ((f.[n,a]) = s.a)) &
  (for x holds ((f.[a,s.x]) = s.(s.(f.[a,x])))) &
  (for n,x holds f.[s.n,s.x] = f.[n,f.[s.n,x]]) &
  (a in D) &
  (for x st (x in D) holds (s.x in D))
  implies
  f.[s.(s.(s.(s.a))),s.(s.(s.(s.a)))] in D
  proof
  assume that
  H1: (for n holds ((f.[n,a]) = s.a)) and
  H2: (for x holds ((f.[a,s.x]) = s.(s.(f.[a,x])))) and
  H3: (for n,x holds f.[s.n,s.x] = f.[n,f.[s.n,x]]) and
  H4: (a in D) and
  H5: (for x st (x in D) holds (s.x in D));
:: 1. existence of N
  defpred Np[set] means (for X holds
  (((a in X) &
    (for y holds (y in X) implies (s.y in X)))
   implies ($1 in X)));
  consider N such that
  Ndef:for z holds z in N iff z in A & Np[z]
  from XBOOLE_0:sch 1;
:: 2. existence of E
  defpred Ep[set] means (($1 in N) & ($1 in D));
  consider E such that
  Edef:for z holds z in E iff z in A & Ep[z]
  from XBOOLE_0:sch 1;
:: 3. Lemma 1
:: 3.1
  LEMMA1a: (a in N);
:: 3.2
  LEMMA1b: (for y holds (y in N) implies (s.y in N));
:: 3.3
  LEMMA1c: s.(s.(s.(s.a))) in N;
:: 3.4
  LEMMA1d: a in E;
:: 3.5
  LEMMA1e: (for y holds (y in E) implies (s.y in E));
:: 3.6
  LEMMA1f: s.a in E;
:: 4
  LEMMA2: (for n holds (n in N) implies
           (for x holds (x in N) implies (f.[n,x] in E)))
  proof
    :: 4.1 (existence of M)
    defpred Mp[set] means (for x holds (x in N)
                            implies f.[$1,x] in E);
    consider M such that
    Mdef:(for z holds (z in M) iff z in A & Mp[z])
    from XBOOLE_0:sch 1;
    :: 4.2 Begin We want...
    wewant42: (for n holds (n in N) implies (n in M))
                                          implies thesis;
    (for n holds (n in N) implies (n in M))
```

```
proof
  :: 4.3 Begin Enough to show...
  ets43: a in M & (for n holds (n in M) implies (s.n in M))
                                     implies thesis;
  :: 4.3.1 Begin
  aM: (a in M)
  proof
    :: 4.3.1.1 Begin Want...
    want4311:(for x holds (x in N) implies f.[a,x] in E)
                                      implies thesis;
    (for x holds (x in N) implies f.[a,x] in E)
    proof
      :: 4.3.1.2 Q exists
      defpred Qp[set] means f.[a,$1] in E;
      consider Q such that
      Qdef:(for z holds (z in Q) iff z in A & Qp[z])
      from XBOOLE_0:sch 1;
      :: 4.3.1.2 End
      :: 4.3.1.3 Begin Want...
      want4313: (for x holds (x in N) implies (x in Q))
                                          implies thesis;
      (for x holds (x in N) implies (x in Q))
      proof
        :: 4.3.1.4 Begin Enough to show...
        ets4314: (a in Q) &
                 (for x holds (x in Q) implies (s.x in Q))
                                           implies thesis;
        :: 4.3.1.4.1 Begin
        aQ: (a in Q)
        proof
          :: 4.3.1.4.1.1 Begin Want
          want431411:(f.[a,a] in E) implies thesis;
          f.[a,a] in E
          proof
          :: 4.3.1.4.1.2
            fact431412: f.[a,a] = s.a by H1;
          :: 4.3.1.4.1.3
            s.a in E by LEMMA1f;
            hence thesis by fact431412;
          end;
          hence thesis by want431411;
          :: 4.3.1.4.1.1 End
        end;
        :: 4.3.1.4.1 End
        :: 4.3.1.4.2 Begin
        sQ: (for x holds (x in Q) implies (s.x in Q))
      proof
      :: 4.3.1.4.2.1
        let x such that xQ: x in Q;
      :: 4.3.1.4.2.2
        fact431422: f.[a,x] in E;
      :: 4.3.1.4.2.3
        fact431423: f.[a,s.x] = s.(s.(f.[a,x])) by H2;
      :: 4.3.1.4.2.4 Begin "By Lemma 1 twice"
        f.[a,s.x] in E;
      :: 4.3.1.4.2.4 End
      :: 4.3.1.4.2.5
        hence s.x in Q;
      end;
      :: 4.3.1.4.2 End
      thus thesis by ets4314,aQ,sQ;
      :: 4.3.1.4 End
    end;
    hence thesis by want4313;
    :: 4.3.1.3 End
  end;
  hence thesis by want4311; :: 4.3.1.4.2.6
  :: 4.3.1.1 End
end;
:: 4.3.1 End
:: 4.3.2 Begin
sM: (for n holds (n in M) implies (s.n in M))
proof
:: 4.3.2.1
  let n such that nM: (n in M);
:: 4.3.2.2
  (for x holds (x in N) implies f.[n,x] in E);
:: 4.3.2.3 Begin Want
  want4323: (s.n in M) implies thesis;
  (s.n in M)
  proof
:: 4.3.2.4 Begin IE
    ie4324:(for x holds (x in N) implies f.[s.n,x] in E)
                                         implies thesis;
    (for x holds (x in N) implies f.[s.n,x] in E)
    proof
    :: 4.3.2.5 P exists
      defpred Pp[set] means f.[s.n,$1] in E;
      consider P such that
      Pdef:(for z holds (z in P) iff z in A & Pp[z])
      from XBOOLE_0:sch 1;
    :: 4.3.2.5 End
```

```
:: 4.3.2.6 Begin Want
  want4326: (for x holds (x in N) implies (x in P))
                                    implies thesis;
  (for x holds (x in N) implies (x in P))
  proof
:: 4.3.2.7 Begin Enough to Show...
    ets4327: (a in P) &
          (for x holds (x in P) implies (s.x in P))
                                    implies thesis;
:: 4.3.2.7.1
    aP: (a in P)
    proof
:: 4.3.2.7.1.1 Begin Want
      want432711: f.[s.n,a] in E implies thesis;
      f.[s.n,a] in E
      proof
:: 4.3.2.7.1.2
        eh1: f.[s.n,a] = s.a by H1;
:: 4.3.2.7.1.3
        s.a in E by LEMMA1f;
        hence thesis by eh1;
      end;
      hence thesis by want432711;
:: 4.3.2.7.1.1 End
  end;
:: 4.3.2.7.2
  sP: (for x holds (x in P) implies (s.x in P))
  proof
:: 4.3.2.7.2.1
    let x such that xP: x in P;
:: 4.3.2.7.2.2
    f.[s.n,x] in E;
:: 4.3.2.7.2.3
    then fact432723: f.[s.n,x] in N;
:: 4.3.2.7.2.4 Begin Want...
    want432724: f.[s.n,s.x] in E implies thesis;
    f.[s.n,s.x] in E
    proof
:: 4.3.2.7.2.5
      fact432725: f.[n,f.[s.n,x]] in E
      by fact432723,nM;
:: 4.3.2.7.2.6
      fact432726: f.[n,f.[s.n,x]] = f.[s.n,s.x]
      by H3;
:: 4.3.2.7.2.7
      thus f.[s.n,s.x] in E;
```

```
        end;
        hence thesis by want432724;
      :: 4.3.2.7.2.4 End
       end;
       thus thesis by ets4327,aP,sP;
      :: 4.3.2.7 End
       end;
       hence thesis by want4326;
      :: 4.3.2.6 End
      end;
      hence thesis by ie4324;
    :: 4.3.2.4 End
     end;
     hence thesis by want4323;
    :: 4.3.2.3 End
    end;
    :: 4.3.2 End
    hence thesis by ets43,aM,sM;
    :: 4.3 End
   end;
   hence thesis by wewant42;
:: 4.2 End
  end;
 :: 5
  s.(s.(s.(s.a))) in N by LEMMA1c;
 :: 6
  then f.[s.(s.(s.(s.a))),s.(s.(s.(s.a)))] in E
  by LEMMA2,H1,H2,H3,H4,H5;
 :: 7
  hence thesis by Edef;
 end;
```

A.2 Complete Proof in Mizar

The following is the body of the file which Mizar accepts. The file is the proof outline with additional lemmas and justifications.

```
  begin

reserve A for non empty set;
reserve a,n,x,y for Element of A;
reserve s for Function of A,A;
reserve f for Function of [: A,A :],A;
reserve D for (Subset of A);
reserve X,z,N,E,M,P,Q for set;

:: Typing lemma for s
theorem stype:
```

```
 for x holds s.x is Element of A;

:: Typing lemma for f
theorem ftype:
 for x,y holds f.[x,y] is Element of A
 proof
  let x,y;
  xA: x in A;
  yA: y in A;
  A1: [x,y] in [:A,A:] by ZFMISC_1:106,xA,yA;
  then A2: [:A,A:] <> {} by XBOOLE_0:def 1;
  f.[x,y] in A by FUNCT_2:7,A1,A2;
  hence thesis;
 end;

theorem inference42:
 (for z holds (z in M) iff z in A & (for x holds (x in N)
                              implies f.[z,x] in E))
 &
 (for n holds (n in N) implies (n in M))
 implies
 (for n holds (n in N) implies
 (for x holds (x in N) implies (f.[n,x] in E)))
 proof
  assume Mdef: (for z holds (z in M) iff
      z in A & (for x holds (x in N) implies f.[z,x] in E));
  assume NM: (for n holds (n in N) implies (n in M));
  let n such that nN: (n in N);
  let x such that xN: (x in N);
  (n in M) by nN,NM;
  hence thesis by Mdef,xN;
 end;

theorem inference4313:
 (for z holds (z in Q) iff z in A & f.[a,z] in E)
 &
 (for x holds (x in N) implies (x in Q))
 implies
 (for x holds (x in N) implies f.[a,x] in E)
 proof
  assume Qdef:(for z holds (z in Q) iff z in A & f.[a,z] in E);
  assume NQ: (for x holds (x in N) implies (x in Q));
  let x such that xN: x in N;
  x in Q by xN,NQ;
  hence thesis by Qdef;
 end;
```

```
theorem inference4326:
  (for z holds (z in P) iff z in A & f.[s.n,z] in E)
  &
  (for x holds (x in N) implies (x in P))
  implies
  (for x holds (x in N) implies f.[s.n,x] in E)
  proof
    assume Pdef: (for z holds (z in P) iff
                      z in A & f.[s.n,z] in E);
    assume NP: (for x holds (x in N) implies (x in P));
    let x such that xN: x in N;
    x in P by xN,NP;
    hence thesis by Pdef;
  end;

theorem inference432725:
  f.[s.n,x] in N
  &
  n in M
  &
  (for z holds (z in M) iff z in A & (for x holds (x in N)
                                       implies f.[z,x] in E))
  implies
  f.[n,f.[s.n,x]] in E
  proof
    assume fsnxN: f.[s.n,x] in N;
    assume nM: n in M;
    assume Mdef: (for z holds (z in M) iff
         z in A & (for x holds (x in N) implies f.[z,x] in E));
    ftA: f.[s.n,x] is Element of A by ftype;
    (for x holds (x in N) implies f.[n,x] in E) by Mdef,nM;
    hence thesis by fsnxN,ftA;
  end;

theorem Nindprinc:
  (for z holds z in N iff z in A &
  (for X holds (((a in X) &
  (for y holds (y in X) implies (s.y in X))) implies (z in X))))
  &
  (a in X)
  &
  (for y st (y in X) holds (s.y in X))
  implies
  (for n holds (n in N) implies (n in X))
  proof
```

```
    assume Ndef: (for z holds z in N iff z in A &
    (for X holds (((a in X) &
    (for y holds (y in X) implies (s.y in X)))
    implies (z in X))));
    assume aX: a in X;
    assume sX: (for y st (y in X) holds (s.y in X));
    let n such that nN: n in N;
    (for X holds
    (((a in X) &
    (for y holds (y in X) implies (s.y in X))) implies (n in X)))
    by nN,Ndef;
    hence thesis by aX,sX;
  end;

theorem BoolosCuriousInference:
  (for n holds ((f.[n,a]) = s.a)) &
  (for x holds ((f.[a,s.x]) = s.(s.(f.[a,x])))) &
  (for n,x holds f.[s.n,s.x] = f.[n,f.[s.n,x]]) &
  (a in D) &
  (for x st (x in D) holds (s.x in D))
  implies
  f.[s.(s.(s.(s.a))),s.(s.(s.(s.a)))] in D
  proof
    assume that
    H1: (for n holds ((f.[n,a]) = s.a)) and
    H2: (for x holds ((f.[a,s.x]) = s.(s.(f.[a,x])))) and
    H3: (for n,x holds f.[s.n,s.x] = f.[n,f.[s.n,x]]) and
    H4: (a in D) and
    H5: (for x st (x in D) holds (s.x in D));
  :: 1. existence of N
    defpred Np[set] means (for X holds
    (((a in X) &
      (for y holds (y in X) implies (s.y in X)))
     implies ($1 in X)));
    consider N such that
    Ndef:for z holds z in N iff z in A & Np[z]
    from XBOOLE_0:sch 1;
  :: 2. existence of E
    defpred Ep[set] means (($1 in N) & ($1 in D));
    consider E such that
    Edef:for z holds z in E iff z in A & Ep[z]
    from XBOOLE_0:sch 1;
  :: 3. Lemma 1
  :: 3.1
    LEMMA1a: (a in N)
    proof
    Np[a];
    hence thesis by Ndef;
  end;
:: 3.2
  LEMMA1b: (for y holds (y in N) implies (s.y in N))
  proof
    let y such that yN: (y in N);
    Npy: Np[y] by Ndef,yN;
    Np[s.y]
    proof
      let X such that aX: a in X and
      sX: (for y st (y in X) holds (s.y in X));
      y in X by aX,sX,Npy;
      hence thesis by sX;
    end;
    hence thesis by Ndef;
  end;
:: 3.3
  LEMMA1c: s.(s.(s.(s.a))) in N
  proof
    a in N by LEMMA1a;
    then s.a in N by LEMMA1b;
    then s.(s.a) in N by LEMMA1b;
    then s.(s.(s.a)) in N by LEMMA1b;
    hence thesis by LEMMA1b;
  end;
:: 3.4
  LEMMA1d: a in E by LEMMA1a,H4,Edef;
:: 3.5
  LEMMA1e: (for y holds (y in E) implies (s.y in E))
  proof
    let y such that yE: y in E;
    y in N by Edef,yE;
    then syN: s.y in N by LEMMA1b;
    y in D by Edef,yE;
    then s.y in D by H5;
    hence thesis by syN,Edef;
  end;
:: 3.6
  LEMMA1f: s.a in E by LEMMA1d,LEMMA1e;
:: 4
  LEMMA2: (for n holds (n in N) implies
           (for x holds (x in N) implies (f.[n,x] in E)))
  proof
    :: 4.1 (existence of M)
    defpred Mp[set] means (for x holds (x in N)
```

```
                        implies f.[$1,x] in E);
consider M such that
Mdef:(for z holds (z in M) iff z in A & Mp[z])
from XBOOLE_0:sch 1;
:: 4.2 Begin We want...
wewant42: (for n holds (n in N) implies (n in M))
                     implies thesis by inference42,Mdef;
(for n holds (n in N) implies (n in M))
proof
  :: 4.3 Begin Enough to show...
  ets43: a in M & (for n holds (n in M) implies (s.n in M))
                          implies thesis by Nindprinc,Ndef;
  :: 4.3.1 Begin
  aM: (a in M)
  proof
    :: 4.3.1.1 Begin Want...
    want4311:(for x holds (x in N) implies f.[a,x] in E)
                                   implies thesis by Mdef;
    (for x holds (x in N) implies f.[a,x] in E)
    proof
      :: 4.3.1.2 Q exists
      defpred Qp[set] means f.[a,$1] in E;
      consider Q such that
      Qdef:(for z holds (z in Q) iff z in A & Qp[z])
      from XBOOLE_0:sch 1;
      :: 4.3.1.2 End
      :: 4.3.1.3 Begin Want...
      want4313: (for x holds (x in N) implies (x in Q))
                     implies thesis by inference4313,Qdef;
      (for x holds (x in N) implies (x in Q))
      proof
        :: 4.3.1.4 Begin Enough to show...
        ets4314: (a in Q) &
                 (for x holds (x in Q) implies (s.x in Q))
                         implies thesis by Nindprinc,Ndef;
        :: 4.3.1.4.1 Begin
        aQ: (a in Q)
        proof
          :: 4.3.1.4.1.1 Begin Want
          want431411:(f.[a,a] in E) implies thesis by Qdef;
          f.[a,a] in E
          proof
          :: 4.3.1.4.1.2
            fact431412: f.[a,a] = s.a by H1;
          :: 4.3.1.4.1.3
            s.a in E by LEMMA1f;
```

```
     hence thesis by fact431412;
    end;
    hence thesis by want431411;
    :: 4.3.1.4.1.1 End
   end;
   :: 4.3.1.4.1 End
   :: 4.3.1.4.2 Begin
   sQ: (for x holds (x in Q) implies (s.x in Q))
   proof
   :: 4.3.1.4.2.1
    let x such that xQ: x in Q;
   :: 4.3.1.4.2.2
    fact431422: f.[a,x] in E by xQ,Qdef;
   :: 4.3.1.4.2.3
    fact431423: f.[a,s.x] = s.(s.(f.[a,x])) by H2;
   :: 4.3.1.4.2.4 Begin "By Lemma 1 twice"
    faxA: f.[a,x] is Element of A by ftype;
    then sfaxA: s.(f.[a,x]) is Element of A by stype;
    s.(f.[a,x]) in E by fact431422,LEMMA1e,faxA;
    then s.(s.(f.[a,x])) in E by LEMMA1e,sfaxA;
    then f.[a,s.x] in E by fact431423;
   :: 4.3.1.4.2.4 End
   :: 4.3.1.4.2.5
    hence s.x in Q by Qdef;
   end;
   :: 4.3.1.4.2 End
   thus thesis by ets4314,aQ,sQ;
   :: 4.3.1.4 End
  end;
  hence thesis by want4313;
  :: 4.3.1.3 End
 end;
 hence thesis by want4311; :: 4.3.1.4.2.6
 :: 4.3.1.1 End
end;
:: 4.3.1 End
:: 4.3.2 Begin
sM: (for n holds (n in M) implies (s.n in M))
proof
:: 4.3.2.1
 let n such that nM: (n in M);
:: 4.3.2.2
 (for x holds (x in N) implies f.[n,x] in E) by nM,Mdef;
:: 4.3.2.3 Begin Want
 want4323: (s.n in M) implies thesis;
 (s.n in M)
```

```
 proof
:: 4.3.2.4 Begin IE
  ie4324:(for x holds (x in N) implies f.[s.n,x] in E)
                implies thesis by Mdef;
  (for x holds (x in N) implies f.[s.n,x] in E)
  proof
  :: 4.3.2.5 P exists
   defpred Pp[set] means f.[s.n,$1] in E;
   consider P such that
   Pdef:(for z holds (z in P) iff z in A & Pp[z])
   from XBOOLE_0:sch 1;
  :: 4.3.2.5 End
  :: 4.3.2.6 Begin Want
   want4326: (for x holds (x in N) implies (x in P))
          implies thesis by inference4326,Pdef;
   (for x holds (x in N) implies (x in P))
   proof
  :: 4.3.2.7 Begin Enough to Show...
    ets4327: (a in P) &
       (for x holds (x in P) implies (s.x in P))
            implies thesis by Nindprinc,Ndef;
  :: 4.3.2.7.1
    aP: (a in P)
    proof
  :: 4.3.2.7.1.1 Begin Want
     want432711: f.[s.n,a] in E implies thesis
     by Pdef;
     f.[s.n,a] in E
     proof
  :: 4.3.2.7.1.2
      eh1: f.[s.n,a] = s.a by H1;
  :: 4.3.2.7.1.3
      s.a in E by LEMMA1f;
      hence thesis by eh1;
     end;
     hence thesis by want432711;
  :: 4.3.2.7.1.1 End
   end;
  :: 4.3.2.7.2
   sP: (for x holds (x in P) implies (s.x in P))
   proof
  :: 4.3.2.7.2.1
    let x such that xP: x in P;
  :: 4.3.2.7.2.2
    f.[s.n,x] in E by xP,Pdef;
  :: 4.3.2.7.2.3
```

```
                then fact432723: f.[s.n,x] in N by Edef;
            :: 4.3.2.7.2.4 Begin Want...
                want432724: f.[s.n,s.x] in E implies thesis
                by Pdef;
                f.[s.n,s.x] in E
                proof
            :: 4.3.2.7.2.5
                  fact432725: f.[n,f.[s.n,x]] in E
                  by fact432723,nM,inference432725,Mdef;
            :: 4.3.2.7.2.6
                  fact432726: f.[n,f.[s.n,x]] = f.[s.n,s.x]
                  by H3;
            :: 4.3.2.7.2.7
                  thus f.[s.n,s.x] in E by fact432725,fact432726;
                end;
                hence thesis by want432724;
            :: 4.3.2.7.2.4 End
              end;
              thus thesis by ets4327,aP,sP;
            :: 4.3.2.7 End
              end;
              hence thesis by want4326;
            :: 4.3.2.6 End
            end;
            hence thesis by ie4324;
        :: 4.3.2.4 End
          end;
          hence thesis by want4323;
        :: 4.3.2.3 End
        end;
        :: 4.3.2 End
        hence thesis by ets43,aM,sM;
        :: 4.3 End
      end;
      hence thesis by wewant42;
:: 4.2 End
    end;
  :: 5
    s.(s.(s.(s.a))) in N by LEMMA1c;
  :: 6
    then f.[s.(s.(s.(s.a))),s.(s.(s.(s.a)))] in E
    by LEMMA2,H1,H2,H3,H4,H5;
  :: 7
    hence thesis by Edef;
  end;
```

B OMEGA Version

By Christoph Benzmüller.

B.1 OMEGA's final proof object in LaTeX

OMEGA's final proof object (see Appendix B.3 can be automatically transformed in a human readable LaTeX representation. In order to illustrate the very close correspondence to Boolos' original proof the author has annotated the beginning of each line with the corresponding proof label of Boolos' original proof as given in Figure 1.

A1.	A1	$\vdash \forall N_{[\iota]} \bullet F_{[(\iota,\iota)\to\iota]}(N, One_{[\iota]}) = S_{[\iota\to\iota]}(One)$	(Hyp)
A2.	A2	$\vdash \forall X_{[\iota]} \bullet F(One, S(X)) = S(S(F(One, X)))$	(Hyp)
A3.	A3	$\vdash \forall N_{[\iota]}, X_{[\iota]} \bullet F(S(N), S(X)) = F(N, F(S(N), X))$	(Hyp)
A4.	A4	$\vdash D_{[\iota\to o]}(One)$	(Hyp)
A5.	A5	$\vdash \forall X_{[\iota]} \bullet [D(X) \Rightarrow D(S(X))]$	(Hyp)
$^{\underline{1}}$Ld1.	Ld1	$\vdash = Def_{[(\iota\to o,\iota\to o)\to o]}(Ld1_{[\iota\to o]}, \lambda Z_{[\iota]} \bullet \forall X_{[\iota\to o]} \bullet [[X(One) \wedge \forall Y_{[\iota]} \bullet [X(Y) \Rightarrow X(S(Y))]] \Rightarrow X(Z)])$	(Local-Def)
$^{\underline{2}}$Ld2.	Ld2	$\vdash Ld_{2[\iota\to o]} := \lambda Z_{[\iota]} \bullet [Ld_1(Z) \wedge D(Z)]$	(Local-Def)
$^{\underline{3.1}}$L2.	$\mathcal{H}_1$	$\vdash \forall Z_{13[\iota\to o]} \bullet [[Z_{13}(One) \wedge \forall Z_{17[\iota]} \bullet [Z_{13}(Z_{17}) \Rightarrow Z_{13}(S(Z_{17}))]] \Rightarrow Z_{13}(One)]$	(Otter)
$^{\underline{3.1}}$L1.	$\mathcal{H}_1$	$\vdash Ld_1(One)$	(CDef L2,Ld1)
$^{\underline{3.2}}$L5.	$\mathcal{H}_1$	$\vdash \forall Z_{48[\iota]} \bullet [\forall Z_{59[\iota\to o]} \bullet [[Z_{59}(One) \wedge \forall Z_{63[\iota]} \bullet [Z_{59}(Z_{63}) \Rightarrow Z_{59}(S(Z_{63}))]] \Rightarrow Z_{59}(Z_{48})] \Rightarrow \forall Z_{68[\iota\to o]} \bullet [[Z_{68}(One) \wedge \forall Z_{72[\iota]} \bullet [Z_{68}(Z_{72}) \Rightarrow Z_{68}(S(Z_{72}))]] \Rightarrow Z_{68}(S(Z_{48}))]]$	(Otter)
$^{\underline{3.2}}$L4.	$\mathcal{H}_1$	$\vdash \forall Z_{26[\iota]} \bullet [\forall Z_{37[\iota\to o]} \bullet [[Z_{37}(One) \wedge \forall Z_{41[\iota]} \bullet [Z_{37}(Z_{41}) \Rightarrow Z_{37}(S(Z_{41}))]] \Rightarrow Z_{37}(Z_{26})] \Rightarrow Ld_1(S(Z_{26}))]$	(CDef L5,Ld1)
$^{\underline{3.2}}$L3.	$\mathcal{H}_1$	$\vdash \forall Y_{[\iota]} \bullet [Ld_1(Y) \Rightarrow Ld_1(S(Y))]$	(CDef L4,Ld1)
$^{\underline{3.3/5}}$L6.	$\mathcal{H}_1$	$\vdash Ld_1(S(S(S(S(One)))))$	(Otter L1,L3)
$^{\underline{3.4}}$L8.	$\mathcal{H}_1$	$\vdash [Ld_1(One) \wedge D(One)]$	(Otter L1,A4)
$^{\underline{3.4}}$L7.	$\mathcal{H}_1$	$\vdash Ld_2(One)$	(CDef L8,Ld2)
$^{\underline{3.5}}$L11.	$\mathcal{H}_1$	$\vdash \forall Z_{93[\iota]} \bullet [[Ld_1(Z_{93}) \wedge D(Z_{93})] \Rightarrow [Ld_1(S(Z_{93})) \wedge D(S(Z_{93}))]]$	(Otter L3,A5)
$^{\underline{3.5}}$L10.	$\mathcal{H}_1$	$\vdash \forall Z_{86[\iota]} \bullet [[Ld_1(Z_{86}) \wedge D(Z_{86})] \Rightarrow Ld_2(S(Z_{86}))]$	(CDef L11,Ld2)
$^{\underline{3.5}}$L9.	$\mathcal{H}_1$	$\vdash \forall Y_{[\iota]} \bullet [Ld_2(Y) \Rightarrow Ld_2(S(Y))]$	(CDef L10,Ld2)
$^{\underline{3.6}}$L12.	$\mathcal{H}_1$	$\vdash Ld_2(S(One))$	(Otter L7,L9)
$^{\underline{4}}$L13.	$\mathcal{H}_1$	$\vdash \forall N_{[\iota]} \bullet [Ld_1(N) \Rightarrow \forall X_{[\iota]} \bullet [Ld_1(X) \Rightarrow Ld_2(F(N, X))]]$	(EDef L14,Ld3)
$^{\underline{4.1}}$Ld3.	Ld3	$\vdash Ld_{3[\iota\to o]} := \lambda N_{[\iota]} \bullet \forall X_{[\iota]} \bullet [Ld_1(X) \Rightarrow Ld_2(F(N, X))]$	(Local-Def)
$^{\underline{4.3.1.1}}$L18.	$\mathcal{H}_1$	$\vdash \forall Z_{151[\iota]} \bullet [Ld_1(Z_{151}) \Rightarrow Ld_2(F(One, Z_{151}))]$	(EDef L19,Ld4)
$^{\underline{4.3.1}}$L15.	$\mathcal{H}_1$	$\vdash Ld_3(One)$	(CDef L18,Ld3)
$^{\underline{4.3.2.4}}$L30.	$\mathcal{H}_2$	$\vdash \forall Z_{225[\iota]} \bullet [Ld_1(Z_{225}) \Rightarrow Ld_2(F(S(N_{1[\iota]}), Z_{225}))]$	(EDef L31,Ld5)
$^{\underline{4.3.2.3}}$L28.	$\mathcal{H}_2$	$\vdash Ld_3(S(N_1))$	(CDef L30,Ld3)
$^{\underline{4.3.2.1}}$L26.	$\mathcal{H}_1$	$\vdash [Ld_3(N_1) \Rightarrow Ld_3(S(N_1))]$	($\Rightarrow I$ L28)
$^{\underline{4.3.2}}$L16.	$\mathcal{H}_1$	$\vdash \forall N_{[\iota]} \bullet [Ld_3(N) \Rightarrow Ld_3(S(N))]$	($\forall I$ L26)
$^{\underline{4.4-5}}$L17.	$\mathcal{H}_1$	$\vdash \forall Z_{123[\iota]} \bullet [\forall Z_{134[\iota\to o]} \bullet [[Z_{134}(One) \wedge \forall Z_{138[\iota]} \bullet [Z_{134}(Z_{138}) \Rightarrow Z_{134}(S(Z_{138}))]] \Rightarrow Z_{134}(Z_{123})] \Rightarrow Ld_3(Z_{123})]$	(Otter L15,L16)
$^{\underline{4.2}}$L14.	$\mathcal{H}_1$	$\vdash \forall N_{[\iota]} \bullet [Ld_1(N) \Rightarrow Ld_3(N)]$	(CDef L17,Ld1)

$^{\underline{4.3.1.2}}$Ld4.	Ld4	$\vdash Ld_{4[\iota\rightarrow o]} := \lambda X_{[\iota]}\centerdot Ld_2(F(One, X))$	(Local-Def)
$^{\underline{4.3.2.1}}$L27.	L27	$\vdash Ld_3(N_1)$	(Hyp)
$^{\underline{4.3.2.2}}$L29.	$\mathcal{H}_3$	$\vdash \forall Z_{217[\iota]}\centerdot [Ld_1(Z_{217}) \Rightarrow Ld_2(F(N_1, Z_{217}))]$	(EDef L27,Ld3)
$^{\underline{4.3.1.4.1.1-3}}$L23.	$\mathcal{H}_1$	$\vdash Ld_2(F(One, One))$	(Otter L12,A1)
$^{\underline{4.3.1.4.1(.1)}}$L20.	$\mathcal{H}_1$	$\vdash Ld_4(One)$	(CDef L23,Ld4)
$^{\underline{4.3.1.4.2.1-6}}$L25.	$\mathcal{H}_1$	$\vdash \forall Z_{201[\iota]}\centerdot [Ld_2(F(One, Z_{201})) \Rightarrow Ld_2(F(One, S(Z_{201})))]$	(Otter L9,A2)
$^{\underline{4.3.1.4.2.1-6}}$L24.	$\mathcal{H}_1$	$\vdash \forall Z_{194[\iota]}\centerdot [Ld_2(F(One, Z_{194})) \Rightarrow Ld_4(S(Z_{194}))]$	(CDef L25,Ld4)
$^{\underline{4.3.1.4.2(.1)}}$L21.	$\mathcal{H}_1$	$\vdash \forall X_{[\iota]}\centerdot [Ld_4(X) \Rightarrow Ld_4(S(X))]$	(CDef L24,Ld4)
$^{\underline{4.3.1.4}}$L22.	$\mathcal{H}_1$	$\vdash \forall Z_{165[\iota]}\centerdot [\forall Z_{176[\iota\rightarrow o]}\centerdot [[Z_{176}(One) \wedge \forall Z_{180[\iota]}\centerdot [Z_{176}(Z_{180}) \Rightarrow Z_{176}(S(Z_{180}))]] \Rightarrow Z_{176}(Z_{165})] \Rightarrow Ld_4(Z_{165})]$	(Otter L20,L21)
$^{\underline{4.3.1.3}}$L19.	$\mathcal{H}_1$	$\vdash \forall X_{[\iota]}\centerdot [Ld_1(X) \Rightarrow Ld_4(X)]$	(CDef L22,Ld1)
$^{\underline{4.3.2.5}}$Ld5.	Ld5	$\vdash Ld_{5[\iota\rightarrow o]} := \lambda X_{[\iota]}\centerdot Ld_2(F(S(N_1), X))$	(Local-Def)
$^{\underline{4.3.2.7.2.1}}$L37	L37	$\vdash Ld_5(X_{1[\iota]})$	(Hyp)
$^{\underline{4.3.2.7.2.2}}$L39	Ld5, L37	$\vdash Ld_2(F(S(N_1), X_1))$	(EDef L37,Ld5)
$^{\underline{4.3.2.7.2.3}}$L40	$\mathcal{H}_4$	$\vdash [Ld_1(F(S(N_1), X_1)) \wedge D(F(S(N_1), X_1))]$	(EDef L39,Ld2)
$^{\underline{6}}$L45.	$\mathcal{H}_1$	$\vdash Ld_2(F(S(S(S(S(One)))), S(S(S(S(One))))))$	(Otter L6,L13)
$^{\underline{7}}$L46.	$\mathcal{H}_1$	$\vdash [Ld_1(F(S(S(S(S(One)))), S(S(S(S(One)))))) \wedge D(F(S(S(S(S(One)))), S(S(S(S(One))))))]$	(EDef L45,Ld2)
$^{\underline{4.3.2.7.2.3}}$L42	$\mathcal{H}_4$	$\vdash D(F(S(N_1), X_1))$	($\wedge E$ L40)
$^{\underline{4.3.2.7.2.3}}$L41	$\mathcal{H}_4$	$\vdash Ld_1(F(S(N_1), X_1))$	($\wedge E$ L40)
$^{\underline{4.3.2.7.1.1-3}}$L35.	$\mathcal{H}_2$	$\vdash Ld_2(F(S(N_1), One))$	(Otter L12,A1)
$^{\underline{4.3.2.7.1}}$L32.	$\mathcal{H}_2$	$\vdash Ld_5(One)$	(CDef L35,Ld5)
$^{\underline{4.3.2.7.2.5}}$L44	$\mathcal{H}_5$	$\vdash Ld_2(F(N_1, F(S(N_1), X_1)))$	(Otter L29,L41)
$^{\underline{4.3.2.7.2.4,6,7}}$L43.	$\mathcal{H}_5$	$\vdash Ld_2(F(S(N_1), S(X_1)))$	(Otter A3,L44)
$^{\underline{4.3.2.7.2.4}}$L38	$\mathcal{H}_5$	$\vdash Ld_5(S(X_1))$	(CDef L43,Ld5)
$^{\underline{4.3.2.7.2.1}}$L36	$\mathcal{H}_2$	$\vdash [Ld_5(X_1) \Rightarrow Ld_5(S(X_1))]$	($\Rightarrow I$ L38)
$^{\underline{4.3.2.7.2}}$L33.	$\mathcal{H}_2$	$\vdash \forall X_{[\iota]}\centerdot [Ld_5(X) \Rightarrow Ld_5(S(X))]$	($\forall I$ L36)
$^{\underline{4.3.2.7}}$L34.	$\mathcal{H}_2$	$\vdash \forall Z_{239[\iota]}\centerdot [\forall Z_{250[\iota\rightarrow o]}\centerdot [[Z_{250}(One) \wedge \forall Z_{254[\iota]}\centerdot [Z_{250}(Z_{254}) \Rightarrow Z_{250}(S(Z_{254}))]] \Rightarrow Z_{250}(Z_{239})] \Rightarrow Ld_5(Z_{239})]$	(Otter L32,L33)
$^{\underline{4.3.2.6}}$L31.	$\mathcal{H}_2$	$\vdash \forall X_{[\iota]}\centerdot [Ld_1(X) \Rightarrow Ld_5(X)]$	(CDef L34,Ld1)
$^{\underline{0/7}}$Conc.	$\mathcal{H}_1$	$\vdash D(F(S(S(S(S(One)))), S(S(S(S(One))))))$	(Otter L46)

$\mathcal{H}_1$ = A1, A2, A3, A4, A5, Ld1, Ld2, Ld3, Ld4, Ld5
$\mathcal{H}_2$ = A1, A2, A3, A4, A5, Ld1, Ld2, Ld3, Ld4, Ld5, L27
$\mathcal{H}_3$ = Ld3, Ld5, L27
$\mathcal{H}_4$ = Ld2, Ld5, L37
$\mathcal{H}_5$ = A1, A2, A3, A4, A5, Ld1, Ld2, Ld3, Ld4, Ld5, L27, L37
CDef = Defn-Contract-Local-Def
EDef = Defn-Expand-Local-Def
LD1 = N
LD2 = E
LD3 = M
LD4 = Q
LD5 = P

B.2 The final proof script in OMEGA

The following (proof step annotated) OMEGA proof script has been obtained by automatically storing the commands of the interactive session presented in Appendix B.5 in a file. Replaying this script reproduces the final proof project presented in Appendix B.3.

```
 ;;; step 1
OMEGA-BASIC LOCAL-DEF-INTRO ((LAM (Z I)
                              (FORALL
                               (LAM
                                (X (O I))
                                (IMPLIES
                                 (AND
                                  (X ONE)
                                  (FORALL
                                   (LAM
                                    (Y I)
                                    (IMPLIES (X Y) (X (S Y))))))
                                 (X Z))))))
 ;;; step 2
OMEGA-BASIC LOCAL-DEF-INTRO ((LAM (Z I) (AND (LD1 Z) (D Z))))
 ;;; step 3.1
OMEGA-BASIC LEMMA (CONC) ((LD1 ONE))
 ;;; step 3.1
RULES DEFN-CONTRACT-LOCAL-DEF (L1) (NIL) (LD1) ((0))
 ;;; step 3.1
OMEGA-BASIC SUPPORT (L2) (NIL)
 ;;; step 3.1
EXTERN CALL-OTTER-ON-NODE (L2) default default (TEST) default default default default default default default default default default
                               default
 ;;; step 3.2
OMEGA-BASIC LEMMA (CONC) ((FORALL
                           (LAM (Y I) (IMPLIES (LD1 Y) (LD1 (S Y))))))
 ;;; step 3.2
RULES DEFN-CONTRACT-LOCAL-DEF (L3) (NIL) (LD1) ((1 0 1 0))
 ;;; step 3.2
RULES DEFN-CONTRACT-LOCAL-DEF (L4) (NIL) (LD1) ((1 0 2 0))
 ;;; step 3.2
OMEGA-BASIC SUPPORT (L5) (NIL)
 ;;; step 3.2
EXTERN CALL-OTTER-ON-NODE (L5) default default (TEST) default default default default default default default default default default
                               default
 ;;; step 3.3
OMEGA-BASIC LEMMA (CONC) ((LD1 (S (S (S (S ONE))))))
 ;;; step 3.3
OMEGA-BASIC SUPPORT (L6) ((L1 L3))
 ;;; step 3.3
EXTERN CALL-OTTER-ON-NODE (L6) default default (TEST) default default default default default default default default default default
                               default
 ;;; step 3.4
OMEGA-BASIC LEMMA (CONC) ((LD2 ONE))
 ;;; step 3.4
RULES DEFN-CONTRACT-LOCAL-DEF (L7) (NIL) (LD2) ((0))
 ;;; step 3.4
OMEGA-BASIC SUPPORT (L8) ((A4 L1))
 ;;; step 3.4
EXTERN CALL-OTTER-ON-NODE (L8) default default (TEST) default default default default default default default default default default
                               default
 ;;; step 3.5
OMEGA-BASIC LEMMA (CONC) ((FORALL
                           (LAM (Y I) (IMPLIES (LD2 Y) (LD2 (S Y))))))
 ;;; step 3.5
RULES DEFN-CONTRACT-LOCAL-DEF (L9) (NIL) (LD2) ((1 0 1 0))
 ;;; step 3.5
RULES DEFN-CONTRACT-LOCAL-DEF (L10) (NIL) (LD2) ((1 0 2 0))
 ;;; step 3.5
OMEGA-BASIC SUPPORT (L11) ((A5 L3))
 ;;; step 3.5
EXTERN CALL-OTTER-ON-NODE (L11) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 3.6
OMEGA-BASIC LEMMA (CONC) ((LD2 (S ONE)))
 ;;; step 3.6
OMEGA-BASIC SUPPORT (L12) ((L7 L9))
 ;;; step 3.6
EXTERN CALL-OTTER-ON-NODE (L12) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4
OMEGA-BASIC LEMMA (CONC) ((FORALL
                           (LAM (N I)
                            (IMPLIES (LD1 N)
                             (FORALL
                              (LAM (X I)
                               (IMPLIES (LD1 X) (LD2 (F N X)))))))))
 ;;; step 4.1
OMEGA-BASIC LOCAL-DEF-INTRO ((LAM (N I)
                              (FORALL
                               (LAM
                                (X I)
                                (IMPLIES (LD1 X) (LD2 (F N X)))))))
 ;;; step 4.2
OMEGA-BASIC LEMMA (L13) ((FORALL (LAM (N I) (IMPLIES (LD1 N) (LD3 N)))))
 ;;; step 4.2
```

```
RULES DEFN-EXPAND-LOCAL-DEF (L13) (L14) (LD3) ((1 0 2 0))
 ;;; step 4.3.1
OMEGA-BASIC LEMMA (L14) ((LD3 ONE))
 ;;; step 4.3.2
OMEGA-BASIC LEMMA (L14) ((FORALL
                          (LAM (N I) (IMPLIES (LD3 N) (LD3 (S N))))))
 ;;; step 4.3 -- Enough to show ...
RULES DEFN-CONTRACT-LOCAL-DEF (L14) (NIL) (LD1) ((1 0 1 0))
 ;;; step 4.3
OMEGA-BASIC SUPPORT (L17) ((L15 L16))
 ;;; step 4.3
EXTERN CALL-OTTER-ON-NODE (L17) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4.3.1.1
RULES DEFN-CONTRACT-LOCAL-DEF (L15) (NIL) (LD3) ((0))
 ;;; step 4.3.1.2
OMEGA-BASIC LOCAL-DEF-INTRO ((LAM (X I) (LD2 (F ONE X))))
 ;;; step 4.3.1.3
OMEGA-BASIC LEMMA (L18) ((FORALL (LAM (X I) (IMPLIES (LD1 X) (LD4 X)))))
 ;;; step 4.3.1.3
RULES DEFN-EXPAND-LOCAL-DEF (L18) (L19) (LD4) ((1 0 2 0))
 ;;; step 4.3.1.4.1
OMEGA-BASIC LEMMA (L19) ((LD4 ONE))
 ;;; step 4.3.1.4.2
OMEGA-BASIC LEMMA (L19) ((FORALL
                          (LAM (X I) (IMPLIES (LD4 X) (LD4 (S X))))))
 ;;; step 4.3.1.4 -- Enough to show
RULES DEFN-CONTRACT-LOCAL-DEF (L19) (NIL) (LD1) ((1 0 1 0))
 ;;; step 4.3.1.4
OMEGA-BASIC SUPPORT (L22) ((L20 L21))
 ;;; step 4.3.1.4
EXTERN CALL-OTTER-ON-NODE (L22) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4.3.1.4.1.1
RULES DEFN-CONTRACT-LOCAL-DEF (L20) (NIL) (LD4) ((0))
 ;;; step 4.3.1.4.1.2-3
OMEGA-BASIC SUPPORT (L23) ((A1 L12))
 ;;; step 4.3.1.4.1.2-3
EXTERN CALL-OTTER-ON-NODE (L23) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4.3.1.4.2.1-6
RULES DEFN-CONTRACT-LOCAL-DEF (L21) (NIL) (LD4) ((1 0 1 0))
 ;;; step 4.3.1.4.2.1-6
RULES DEFN-CONTRACT-LOCAL-DEF (L24) (NIL) (LD4) ((1 0 2 0))
 ;;; step 4.3.1.4.2.1-6
OMEGA-BASIC SUPPORT (L25) ((A2 L9))
 ;;; step 4.3.1.4.2.1-6
EXTERN CALL-OTTER-ON-NODE (L25) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4.3.2.1
RULES FORALLI default default default
 ;;; step 4.3.2.1,3
RULES IMPI default
 ;;; step 4.3.2.2
RULES DEFN-EXPAND-LOCAL-DEF (NIL) (L27) (LD3) ((0))
 ;;; step 4.3.2.4
RULES DEFN-CONTRACT-LOCAL-DEF (L28) (NIL) (LD3) ((0))
 ;;; step 4.3.2.5
OMEGA-BASIC LOCAL-DEF-INTRO ((LAM (X I) (LD2 (F (S N1) X))))
 ;;; step 4.3.2.6
OMEGA-BASIC LEMMA default ((FORALL
                            (LAM (X I) (IMPLIES (LD1 X) (LD5 X)))))
 ;;; step 4.3.2.6
RULES DEFN-EXPAND-LOCAL-DEF (L30) (L31) (LD5) ((1 0 2 0))
 ;;; step 4.3.2.7.1
OMEGA-BASIC LEMMA (L31) ((LD5 ONE))
 ;;; step 4.3.2.7.2
OMEGA-BASIC LEMMA (L31) ((FORALL
                          (LAM (X I) (IMPLIES (LD5 X) (LD5 (S X))))))
 ;;; step 4.3.2.7 -- Enough to ...
RULES DEFN-CONTRACT-LOCAL-DEF (L31) (NIL) (LD1) ((1 0 1 0))
 ;;; step 4.3.2.7
OMEGA-BASIC SUPPORT (L34) ((L32 L33))
 ;;; step 4.3.2.7
EXTERN CALL-OTTER-ON-NODE (L34) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4.3.2.7.1
RULES DEFN-CONTRACT-LOCAL-DEF (L32) (NIL) (LD5) ((0))
 ;;; step 4.3.2.7.2-3
OMEGA-BASIC SUPPORT (L35) ((A1 L12))
 ;;; step 4.3.2.7.2-3
EXTERN CALL-OTTER-ON-NODE (L35) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4.3.2.7.2.1
RULES FORALLI default default default
 ;;; step 4.3.2.7.2.1
RULES IMPI default
 ;;; step 4.3.2.7.2.2
RULES DEFN-EXPAND-LOCAL-DEF (NIL) (L37) (LD5) ((0))
 ;;; step 4.3.2.7.2.3
RULES DEFN-EXPAND-LOCAL-DEF (NIL) (L39) (LD2) ((0))
```

```
 ;;; step 4.3.2.7.2.3
TACTICS ANDE (L40) default default
 ;;; step 4.3.2.7.2.4
RULES DEFN-CONTRACT-LOCAL-DEF (L38) (NIL) (LD5) ((0))
 ;;; step 4.3.2.7.2.5
OMEGA-BASIC LEMMA (L43) ((LD2 (F N1 (F (S N1) X1))))
 ;;; step 4.3.2.7.2.5
OMEGA-BASIC SUPPORT (L44) ((L41 L29))
 ;;; step 4.3.2.7.2.5
EXTERN CALL-OTTER-ON-NODE (L44) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 4.3.2.7.2.6
OMEGA-BASIC SUPPORT (L43) ((L44 A3))
 ;;; step 4.3.2.7.2.7
EXTERN CALL-OTTER-ON-NODE (L43) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 6
OMEGA-BASIC LEMMA (CONC) ((ld2 (F (S (S (S (S ONE)))) (S (S (S (S ONE)))))))
 ;;; step 6
OMEGA-BASIC SUPPORT (L45) ((L6 L13))
 ;;; step 6
EXTERN CALL-OTTER-ON-NODE (L45) default default (TEST) default default default default default default default default default default
                                default
 ;;; step 7
RULES DEFN-EXPAND-LOCAL-DEF (NIL) (L45) (LD2) ((0))
 ;;; step 7
OMEGA-BASIC SUPPORT (CONC) ((L46))
 ;;; step 7
EXTERN CALL-OTTER-ON-NODE (CONC) default default (TEST) default default default default default default default default default default
                                 default
```

B.3 OMEGA's final proof object in ASCII format

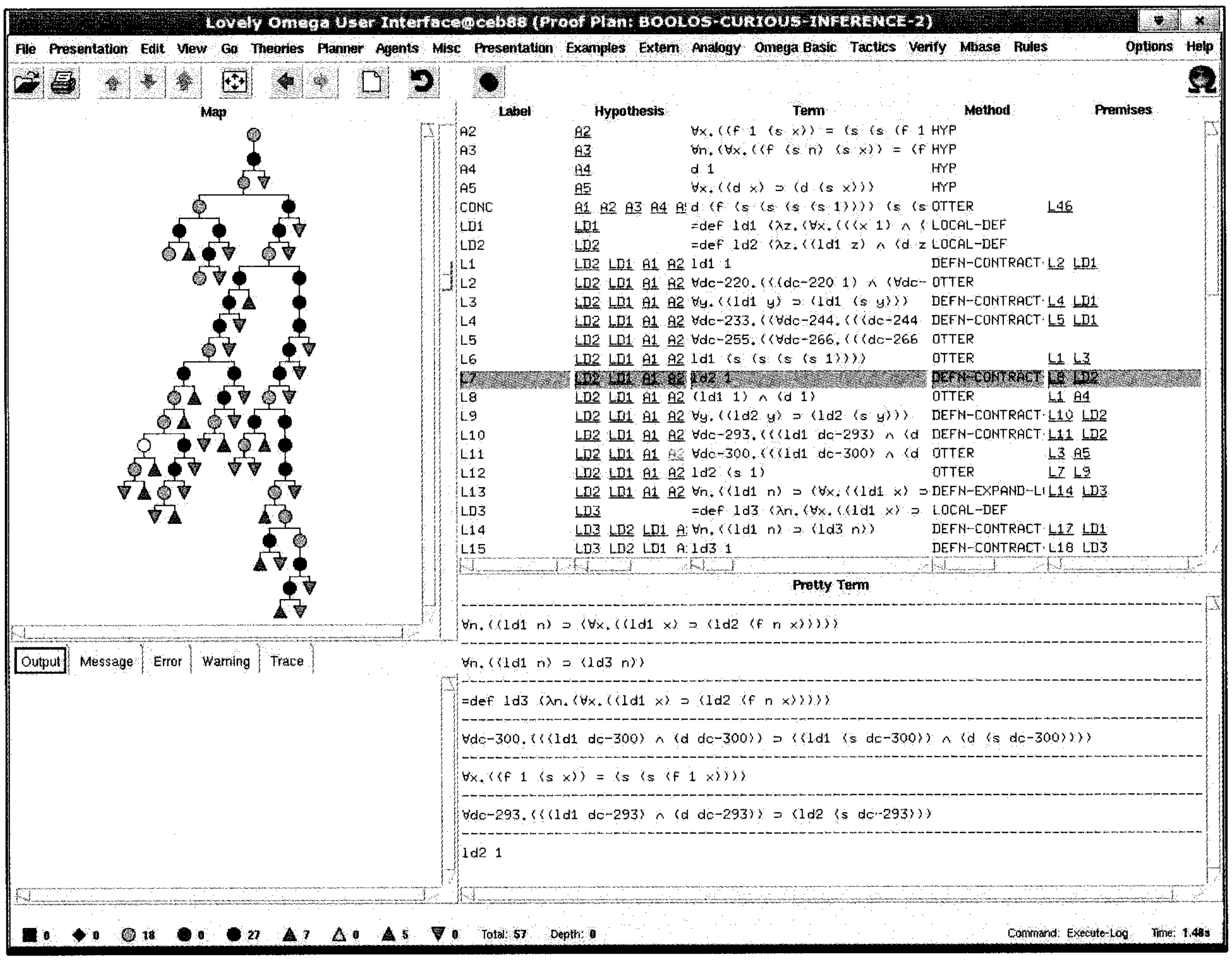
```
(PDS (problem BOOLOS-CURIOUS-INFERENCE)
  (in BOOLOS)
 (declarations (type-variables )(type-constants )
    (constants  (X1 I)
       (LD5 (O I))
       (N1 I)
       (LD4 (O I))
       (LD3 (O I))
       (LD2 (O I))
       (LD1 (O I)))
    (meta-variables )(variables ))
   (conclusion CONC)
   (assumptions A1 A2 A3 A4 A5)
   (open-nodes)
   (support-nodes LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5)
   (nodes
     (A1 (A1) (FORALL (lam (VAR0 I) (= (F VAR0 ONE) (S ONE))))
      (0 ("HYP" () () "grounded" () ()))
      )
     (A2 (A2) (FORALL (lam (VAR3 I) (= (F ONE (S VAR3)) (S (S (F ONE VAR3))))))
      (0 ("HYP" () () "grounded" () ()))
      )
     (A3 (A3) (FORALL (lam (VAR4 I) (FORALL (lam (VAR5 I) (= (F (S VAR4) (S VAR5)) (F VAR4 (F (S VAR4) VAR5)))))))
      (0 ("HYP" () () "grounded" () ()))
      )
     (A4 (A4) (D ONE)
      (0 ("HYP" () () "grounded" () ()))
      )
     (A5 (A5) (FORALL (lam (VAR6 I) (IMPLIES (D VAR6) (D (S VAR6)))))
      (0 ("HYP" () () "grounded" () ()))
      )
     (LD1 (LD1) (=DEF LD1 (lam (VAR7 I) (FORALL (lam (VAR8 (O I)) (IMPLIES (AND (VAR8 ONE) (FORALL (lam (VAR9 I)
(IMPLIES (VAR8 VAR9) (VAR8 (S VAR9)))))) (VAR8 VAR7))))))
      (0 ("LOCAL-DEF" () () "grounded" () ()))
      )
     (LD2 (LD2) (=DEF LD2 (lam (VAR10 I) (AND (LD1 VAR10) (D VAR10))))
      (0 ("LOCAL-DEF" () () "grounded" () ()))
      )
     (L2 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR11 (O I)) (IMPLIES (AND (VAR11 ONE)
(FORALL (lam (VAR12 I) (IMPLIES (VAR11 VAR12) (VAR11 (S VAR12)))))) (VAR11 ONE))))
      (1 ("OTTER" ((:pds-nil)) () "expanded" () ("EXISTENT"))
       ("OTTER" ((:pds-nil)) () "untested" () ()))
      )
     (L1 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD1 ONE)
      (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 0))) (L2 LD1) "grounded"
          () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
      )
     (L5 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR13 I) (IMPLIES (FORALL (lam (VAR14 (O I))
(IMPLIES (AND (VAR14 ONE) (FORALL (lam (VAR15 I) (IMPLIES (VAR14 VAR15) (VAR14 (S VAR15)))))) (VAR14 VAR13))))
(FORALL (lam (VAR16 (O I)) (IMPLIES (AND (VAR16 ONE) (FORALL (lam (VAR17 I) (IMPLIES (VAR16 VAR17)
(VAR16 (S VAR17)))))) (VAR16 (S VAR13))))))))
      (1 ("OTTER" ((:pds-nil)) () "expanded" () ("EXISTENT"))
       ("OTTER" ((:pds-nil)) () "untested" () ()))
      )
     (L4 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR18 I) (IMPLIES (FORALL (lam (VAR19 (O I))
(IMPLIES (AND (VAR19 ONE) (FORALL (lam (VAR20 I) (IMPLIES (VAR19 VAR20) (VAR19 (S VAR20)))))) (VAR19 VAR18))))
(LD1 (S VAR18)))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 2 0))) (L5 LD1) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L3 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR21 I) (IMPLIES (LD1 VAR21) (LD1 (S VAR21)))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 1 0))) (L4 LD1) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L6 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD1 (S (S (S (S ONE))))))
       (1 ("OTTER" ((:pds-nil)) (L3 L1) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L1 L3) "untested" () ()))
       )
      (L8 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (AND (LD1 ONE) (D ONE))
       (1 ("OTTER" ((:pds-nil)) (A4 L1) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L1 A4) "untested" () ()))
       )
      (L7 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD2 ONE)
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 0))) (L8 LD2) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L11 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR22 I) (IMPLIES (AND (LD1 VAR22) (D VAR22))
(AND (LD1 (S VAR22)) (D (S VAR22))))))
       (1 ("OTTER" ((:pds-nil)) (A5 L3) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L3 A5) "untested" () ()))
       )
      (L10 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR23 I) (IMPLIES (AND (LD1 VAR23) (D VAR23))
(LD2 (S VAR23)))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 2 0))) (L11 LD2) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L9 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR24 I) (IMPLIES (LD2 VAR24) (LD2 (S VAR24)))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 1 0))) (L10 LD2) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L12 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD2 (S ONE))
       (1 ("OTTER" ((:pds-nil)) (L9 L7) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L7 L9) "untested" () ()))
       )
      (L13 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR25 I) (IMPLIES (LD1 VAR25) (FORALL (lam (VAR26 I)
(IMPLIES (LD1 VAR26) (LD2 (F VAR25 VAR26))))))))
       (0 ("DEFN-EXPAND-LOCAL-DEF" ((:pds-post-obj (position 1 0 2 0))) (L14 LD3) "grounded"
           () ("EXISTENT" "EXISTENT" "EXISTENT")))
       )
      (LD3 (LD3) (=DEF LD3 (lam (VAR27 I) (FORALL (lam (VAR28 I) (IMPLIES (LD1 VAR28) (LD2 (F VAR27 VAR28)))))))
       (0 ("LOCAL-DEF" () () "grounded" () ()))
       )
      (L18 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR29 I) (IMPLIES (LD1 VAR29) (LD2 (F ONE VAR29)))))
       (0 ("DEFN-EXPAND-LOCAL-DEF" ((:pds-post-obj (position 1 0 2 0))) (L19 LD4) "grounded"
           () ("EXISTENT" "EXISTENT" "EXISTENT")))
       )
      (L15 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD3 ONE)
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 0))) (L18 LD3) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L30 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR30 I) (IMPLIES (LD1 VAR30)
(LD2 (F (S N1) VAR30)))))
       (0 ("DEFN-EXPAND-LOCAL-DEF" ((:pds-post-obj (position 1 0 2 0))) (L31 LD5) "grounded"
           () ("EXISTENT" "EXISTENT" "EXISTENT")))
       )
      (L28 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD3 (S N1))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 0))) (L30 LD3) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L26 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (IMPLIES (LD3 N1) (LD3 (S N1)))
       (0 ("IMPI" () (L28) "grounded" () ("EXISTENT" "NONEXISTENT")))
       )
      (L16 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR31 I) (IMPLIES (LD3 VAR31) (LD3 (S VAR31)))))
       (0 ("FORALLI" ((:pds-term N1)) (L26) "grounded" ()
           ("EXISTENT" "NONEXISTENT")))
       )
      (L17 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR32 I) (IMPLIES (FORALL (lam (VAR33 (O I))
(IMPLIES (AND (VAR33 ONE) (FORALL (lam (VAR34 I) (IMPLIES (VAR33 VAR34) (VAR33 (S VAR34)))))) (VAR33 VAR32))))
(LD3 VAR32))))
       (1 ("OTTER" ((:pds-nil)) (L16 L15) "expanded" ()
           ("EXISTENT" "EXISTENT" "EXISTENT"))
        ("OTTER" ((:pds-nil)) (L15 L16) "untested" () ()))
       )
      (L14 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR35 I) (IMPLIES (LD1 VAR35) (LD3 VAR35))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 1 0))) (L17 LD1) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (LD4 (LD4) (=DEF LD4 (lam (VAR36 I) (LD2 (F ONE VAR36))))
       (0 ("LOCAL-DEF" () () "grounded" () ()))
       )
      (L27 (L27) (LD3 N1)
       (0 ("HYP" () () "grounded" () ()))
```

```
       )
      (L29 (LD5 LD3 L27) (FORALL (lam (VAR37 I) (IMPLIES (LD1 VAR37) (LD2 (F N1 VAR37)))))
       (0 ("DEFN-EXPAND-LOCAL-DEF" ((:pds-post-obj (position 0))) (L27 LD3) "grounded"
           () ("NONEXISTENT" "EXISTENT" "EXISTENT")))
       )
      (L23 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD2 (F ONE ONE))
       (1 ("OTTER" ((:pds-nil)) (A1 L12) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L12 A1) "untested" () ()))
       )
      (L20 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD4 ONE)
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 0))) (L23 LD4) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L25 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR38 I) (IMPLIES (LD2 (F ONE VAR38))
(LD2 (F ONE (S VAR38))))))
       (1 ("OTTER" ((:pds-nil)) (A2 L9) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L9 A2) "untested" () ()))
       )
      (L24 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR39 I) (IMPLIES (LD2 (F ONE VAR39))
(LD4 (S VAR39)))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 2 0))) (L25 LD4) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L21 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR40 I) (IMPLIES (LD4 VAR40) (LD4 (S VAR40)))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 1 0))) (L24 LD4) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L22 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR41 I) (IMPLIES (FORALL (lam (VAR42 (O I))
(IMPLIES (AND (VAR42 ONE) (FORALL (lam (VAR43 I) (IMPLIES (VAR42 VAR43) (VAR42 (S VAR43)))))) (VAR42 VAR41))))
(LD4 VAR41))))
       (1 ("OTTER" ((:pds-nil)) (L21 L20) "expanded" ()
           ("EXISTENT" "EXISTENT" "EXISTENT"))
        ("OTTER" ((:pds-nil)) (L20 L21) "untested" () ()))
       )
      (L19 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR44 I) (IMPLIES (LD1 VAR44) (LD4 VAR44))))
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 1 0))) (L22 LD1) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (LD5 (LD5) (=DEF LD5 (lam (VAR45 I) (LD2 (F (S N1) VAR45))))
       (0 ("LOCAL-DEF" () () "grounded" () ()))
       )
      (L37 (L37) (LD5 X1)
       (0 ("HYP" () () "grounded" () ()))
       )
      (L39 (LD5 L37) (LD2 (F (S N1) X1))
       (0 ("DEFN-EXPAND-LOCAL-DEF" ((:pds-post-obj (position 0))) (L37 LD5) "grounded"
           () ("NONEXISTENT" "EXISTENT" "EXISTENT")))
       )
      (L40 (LD2 LD5 L37) (AND (LD1 (F (S N1) X1)) (D (F (S N1) X1)))
       (0 ("DEFN-EXPAND-LOCAL-DEF" ((:pds-post-obj (position 0))) (L39 LD2) "grounded"
           () ("NONEXISTENT" "EXISTENT" "EXISTENT")))
       )
      (L45 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD2 (F (S (S (S (S ONE)))) (S (S (S (S ONE)))))))
       (1 ("OTTER" ((:pds-nil)) (L13 L6) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L6 L13) "untested" () ()))
       )
      (L46 (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (AND (LD1 (F (S (S (S (S ONE)))) (S (S (S (S ONE))))))
(D (F (S (S (S (S ONE)))) (S (S (S (S ONE))))))))
       (0 ("DEFN-EXPAND-LOCAL-DEF" ((:pds-post-obj (position 0))) (L45 LD2) "grounded"
           () ("NONEXISTENT" "EXISTENT" "EXISTENT")))
       )
      (L42 (LD2 LD5 L37) (D (F (S N1) X1))
       (0 ("ANDE" () (L40) "unexpanded" ()
           ("L41" "NONEXISTENT" "EXISTENT")))
       )
      (L41 (LD2 LD5 L37) (LD1 (F (S N1) X1))
       (0 ("ANDE" () (L40) "unexpanded" ()
           ("NONEXISTENT" "L42" "EXISTENT")))
       )
      (L35 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD2 (F (S N1) ONE))
       (1 ("OTTER" ((:pds-nil)) (A1 L12) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L12 A1) "untested" () ()))
       )
      (L32 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD5 ONE)
       (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 0))) (L35 LD5) "grounded"
           () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
       )
      (L44 (L37 LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD2 (F N1 (F (S N1) X1)))
       (1 ("OTTER" ((:pds-nil)) (L41 L29) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (L29 L41) "untested" () ()))
       )
      (L43 (L37 LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD2 (F (S N1) (S X1)))
       (1 ("OTTER" ((:pds-nil)) (L44 A3) "expanded" ()
           ("EXISTENT" "CLOSED" "CLOSED"))
        ("OTTER" ((:pds-nil)) (A3 L44) "untested" () ()))
       )
```

```
    (L38 (L37 LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (LD5 (S X1))
     (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 0))) (L43 LD5) "grounded"
        () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
     )
    (L36 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (IMPLIES (LD5 X1) (LD5 (S X1)))
     (0 ("IMPI" () (L38) "grounded" () ("EXISTENT" "NONEXISTENT")))
     )
    (L33 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR46 I) (IMPLIES (LD5 VAR46) (LD5 (S VAR46)))))
     (0 ("FORALLI" ((:pds-term X1)) (L36) "grounded" ()
        ("EXISTENT" "NONEXISTENT")))
     )
    (L34 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR47 I) (IMPLIES (FORALL (lam (VAR48 (O I))
(IMPLIES (AND (VAR48 ONE) (FORALL (lam (VAR49 I) (IMPLIES (VAR48 VAR49) (VAR48 (S VAR49)))))) (VAR48 VAR47))))
(LD5 VAR47))))
     (1 ("OTTER" ((:pds-nil)) (L33 L32) "expanded" ()
        ("EXISTENT" "EXISTENT" "EXISTENT"))
      ("OTTER" ((:pds-nil)) (L32 L33) "untested" () ()))
     )
    (L31 (LD5 L27 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (FORALL (lam (VAR50 I) (IMPLIES (LD1 VAR50) (LD5 VAR50))))
     (0 ("DEFN-CONTRACT-LOCAL-DEF" ((:pds-post-obj (position 1 0 1 0))) (L34 LD1) "grounded"
        () ("EXISTENT" "NONEXISTENT" "EXISTENT")))
     )
    (CONC (LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5) (D (F (S (S (S (S ONE)))) (S (S (S (S ONE))))))
     (1 ("OTTER" ((:pds-nil)) (L46) "expanded" ()
        ("EXISTENT" "CLOSED"))
      ("OTTER" ((:pds-nil)) (L46) "untested" () ()))
     ))
  (lemmata)
  (agenda)
  (controls
    (A1 (() () () ()))
    (A2 (() () () ()))
    (A3 (() () () ()))
    (A4 (() () () ()))
    (A5 (() () () ()))
    (LD1 (() () () ()))
    (LD2 (() () () ()))
    (L2 (() (LD2 LD1 A1 A2 A3 A4 A5) () ()) ())
    (L1 (() () () ()))
    (L5 ((L1) (L1 LD2 LD1 A1 A2 A3 A4 A5) () ()) ())
    (L4 ((L1) () () ()))
    (L3 ((L1) () () ()))
    (L6 ((L3 L1) (A5 A4 A3 A2 A1 LD1 LD2) () ()) (() () () ()))
    (L8 ((L6 L3 L1) (A5 A3 A2 A1 LD1 LD2 L6 L3) () ()) ())
    (L7 ((L6 L3 L1) () () ()))
    (L11 ((L7 L6 L3 L1) (A4 A3 A2 A1 LD1 LD2 L7 L6 L1) () ()) ())
    (L10 ((L7 L6 L3 L1) () () ()))
    (L9 ((L7 L6 L3 L1) () () ()))
    (L12 ((L9 L7 L6 L3 L1) (A5 A4 A3 A2 A1 LD1 LD2 L6 L3 L1) () ())
         (() () () ()))
    (L13 ((L14 L12 L9 L7 L6 L3 L1) () () ()))
    (LD3 (() () () ()))
    (L18 ((L19 L12 L9 L7 L6 L3 L1) () () ()))
    (L15 ((L12 L9 L7 L6 L3 L1) () () ()))
    (L30 ((L31 L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L28 ((L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L26 ((L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L16 ((L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L17 ((L16 L15 L12 L9 L7 L6 L3 L1) (A5 A4 A3 A2 A1 LD1 LD2 LD3 L12 L9 L7 L6 L3 L1) () ())
         ())
    (L14 ((L16 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (LD4 (() () () ()))
    (L27 (() () () ()))
    (L29 (() () () ()))
    (L23 ((L12 L9 L7 L6 L3 L1) (A5 A4 A3 A2 LD1 LD2 LD3 LD4 L9 L7 L6 L3 L1) () ())
         ())
    (L20 ((L12 L9 L7 L6 L3 L1) () () ()))
    (L25 ((L20 L12 L9 L7 L6 L3 L1) (A5 A4 A3 A1 LD1 LD2 LD3 LD4 L20 L12 L7 L6 L3 L1) () ())
         ())
    (L24 ((L20 L12 L9 L7 L6 L3 L1) () () ()))
    (L21 ((L20 L12 L9 L7 L6 L3 L1) () () ()))
    (L22 ((L21 L20 L12 L9 L7 L6 L3 L1) (A5 A4 A3 A2 A1 LD1 LD2 LD3 LD4 L12 L9 L7 L6 L3 L1) () ())
         ())
    (L19 ((L21 L20 L12 L9 L7 L6 L3 L1) () () ()))
    (LD5 (() () () ()))
    (L37 (() () () ()))
    (L39 (() () () ()))
    (L40 ((L13 L37 L32 L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L45 ((L40 L13 L12 L9 L7 L6 L3 L1) (A5 A4 A3 A2 A1 LD1 LD2 LD3 LD4 LD5 L40 L12 L9 L7 L3 L1) () ())
         (() () () ()))
    (L46 (() () () ()))
    (L42 (() () () ()))
    (L41 (() () () ()))
    (L35 ((L27 L15 L12 L9 L7 L6 L3 L1) (A5 A4 A3 A2 LD1 LD2 LD3 LD4 LD5 L27 L15 L9 L7 L6 L3 L1) () ())
         ())
    (L32 ((L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L44 ((L29 L41 L1 L3 L6 L7 L9 L12 L15 L27 L32 L37 L40) (A5 A4 A3 A2 A1 LD1 LD2 LD3 LD4 LD5 L1 L3 L6 L7 L9 L12
L15 L27 L32 L37 L40) () ())
         (() () () ()))
    (L43 ((L44 L40 L37 L32 L27 L15 L12 L9 L7 L6 L3 L1) (L1 L3 L6 L7 L9 L12 L15 L27 L32 L37 L40 LD5 LD4 LD3 LD2 LD1
A1 A2 A4 A5) () ())
```

```
         ())
    (L38 ((L40 L37 L32 L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L36 ((L32 L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L33 ((L32 L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (L34 ((L33 L32 L27 L15 L12 L9 L7 L6 L3 L1) (A5 A4 A3 A2 A1 LD1 LD2 LD3 LD4 LD5 L27 L15 L12 L9 L7 L6 L3 L1)
() ())
         ())
    (L31 ((L33 L32 L27 L15 L12 L9 L7 L6 L3 L1) () () ()))
    (CONC ((L46 L45 L40 L13 L12 L9 L7 L6 L3 L1) (L1 L3 L6 L7 L9 L12 L13 L40 L45 LD5 LD4 LD3 LD2 LD1 A1 A2 A3 A4 A5)
 () ())
          ()))
  (plan-steps (L1 0 L2 0 LD1 0) (L2 0) (L3 0 L4 0 LD1 0)
    (L4 0 L5 0 LD1 0) (L5 0) (L6 0 L3 0 L1 0) (L7 0 L8 0 LD2 0)
    (L8 0 A4 0 L1 0) (L9 0 L10 0 LD2 0) (L10 0 L11 0 LD2 0)
    (L11 0 A5 0 L3 0) (L12 0 L9 0 L7 0) (L13 0 L14 0 LD3 0)
    (L14 0 L17 0 LD1 0) (L17 0 L16 0 L15 0) (L15 0 L18 0 LD3 0)
    (L18 0 L19 0 LD4 0) (L19 0 L22 0 LD1 0) (L22 0 L21 0 L20 0)
    (L20 0 L23 0 LD4 0) (L23 0 A1 0 L12 1) (L21 0 L24 0 LD4 0)
    (L24 0 L25 0 LD4 0) (L25 0 A2 0 L9 0) (L16 0 L26 0)
    (L26 0 L27 0 L28 0) (L29 0 L27 0 LD3 0) (L28 0 L30 0 LD3 0)
    (L30 0 L31 0 LD5 0) (L31 0 L34 0 LD1 0) (L34 0 L33 0 L32 0)
    (L32 0 L35 0 LD5 0) (L35 0 A1 0 L12 1) (L33 0 L36 0)
    (L36 0 L37 0 L38 0) (L39 0 L37 0 LD5 0) (L40 0 L39 0 LD2 0)
    (L41 0 L40 0) (L42 0 L40 0) (L38 0 L43 0 LD5 0)
    (L44 0 L41 0 L29 0) (L43 0 L44 1 A3 0) (L45 0 L13 0 L6 1)
    (L46 0 L45 1 LD2 0) (CONC 0 L46 0) ))
```

B.4 The final proof in OMEGA's graphical user interface LOUI



Lovely Omega User Interface@ceb88 (Proof Plan: BOOLOS-CURIOUS-INFERENCE-2)
File Presentation Edit View Go Theories Planner Agents Misc Presentation Examples Extern Analogy Omega Basic Tactics Verify Mbase Rules Options Help
Map
Output Message Error Warning Trace

Label	Hypothesis	Term	Method	Premises
A2	A2	∀x.((f 1 (s x)) = (s (s (f 1	HYP	
A3	A3	∀n.(∀x.((f (s n) (s x)) = (f	HYP	
A4	A4	d 1	HYP	
A5	A5	∀x.((d x) ⊃ (d (s x)))	HYP	
CONC	A1 A2 A3 A4 A!	d (f (s (s (s (s 1)))) (s (s	OTTER	L46
LD1	LD1	=def ld1 (λz.(∀x.(((x 1) ∧ (	LOCAL-DEF	
LD2	LD2	=def ld2 (λz.((ld1 z) ∧ (d z	LOCAL-DEF	
L1	LD2 LD1 A1 A2	ld1 1	DEFN-CONTRACT	L2 LD1
L2	LD2 LD1 A1 A2	∀dc-220.(((dc-220 1) ∧ (∀dc-	OTTER	
L3	LD2 LD1 A1 A2	∀y.((ld1 y) ⊃ (ld1 (s y)))	DEFN-CONTRACT	L4 LD1
L4	LD2 LD1 A1 A2	∀dc-233.((∀dc-244.(((dc-244	DEFN-CONTRACT	L5 LD1
L5	LD2 LD1 A1 A2	∀dc-255.((∀dc-266.(((dc-266	OTTER	
L6	LD2 LD1 A1 A2	ld1 (s (s (s (s 1))))	OTTER	L1 L3
L7	LD2 LD1 A1 A2	ld2 1	DEFN-CONTRACT	L8 LD2
L8	LD2 LD1 A1 A2	(ld1 1) ∧ (d 1)	OTTER	L1 A4
L9	LD2 LD1 A1 A2	∀y.((ld2 y) ⊃ (ld2 (s y)))	DEFN-CONTRACT	L10 LD2
L10	LD2 LD1 A1 A2	∀dc-293.(((ld1 dc-293) ∧ (d	DEFN-CONTRACT	L11 LD2
L11	LD2 LD1 A1 A2	∀dc-300.(((ld1 dc-300) ∧ (d	OTTER	L3 A5
L12	LD2 LD1 A1 A2	ld2 (s 1)	OTTER	L7 L9
L13	LD2 LD1 A1 A2	∀n.((ld1 n) ⊃ (∀x.((ld1 x) ⊃	DEFN-EXPAND-L	L14 LD3
LD3	LD3	=def ld3 (λn.(∀x.((ld1 x) ⊃	LOCAL-DEF	
L14	LD3 LD2 LD1 A:	∀n.((ld1 n) ⊃ (ld3 n))	DEFN-CONTRACT	L17 LD1
L15	LD3 LD2 LD1 A:	ld3 1	DEFN-CONTRACT	L18 LD3

Pretty Term

```
∀n.((ld1 n) ⊃ (∀x.((ld1 x) ⊃ (ld2 (f n x)))))
----------------------------------------------------------------------
∀n.((ld1 n) ⊃ (ld3 n))
----------------------------------------------------------------------
=def ld3 (λn.(∀x.((ld1 x) ⊃ (ld2 (f n x)))))
----------------------------------------------------------------------
∀dc-300.(((ld1 dc-300) ∧ (d dc-300)) ⊃ ((ld1 (s dc-300)) ∧ (d (s dc-300))))
----------------------------------------------------------------------
∀x.((f 1 (s x)) = (s (s (f 1 x))))
----------------------------------------------------------------------
∀dc-293.(((ld1 dc-293) ∧ (d dc-293)) ⊃ (ld2 (s dc-293)))
----------------------------------------------------------------------
ld2 1
```

■ 0 ◆ 0 ● 18 ● 0 ● 27 ▲ 7 △ 0 ▲ 5 ▼ 0 Total: 57 Depth: 0
Command: Execute-Log Time: 1.48s

B.5 Protocol of the interactive session in OMEGA

We present the complete protocol of the interactive session with OMEGA. All proof relevant commands are stored in a proof script (see Appendix B.2). This proof script contains all the information to replay this interactive proof. Note also that interactive proof development is supported by the graphical user interface LOUI (see Appendix B.4).

```
OMEGA: show-pds
OMEGA: prove boolos-curious-inference
Changing to proof plan BOOLOS-CURIOUS-INFERENCE-1

 ;;; step 1
OMEGA: LOCAL-DEF-INTRO

TERM (TERM) a term that should be used as definiens: (LAM (Z I)
                              (FORALL
                               (LAM
                                (X (O I))
                                (IMPLIES
                                 (AND
                                  (X ONE)
                                  (FORALL
                                   (LAM
                                    (Y I)
                                    (IMPLIES (X Y) (X (S Y))))))
                                 (X Z)))))

OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                    HYP

A2   (A2)        ! (FORALL [X:I]                                           HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))

A3   (A3)        ! (FORALL [N:I,X:I]                                       HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                 HYP

A5   (A5)        ! (FORALL [X:I]                                           HYP
                    (IMPLIES (D X) (D (S X))))

LD1  (LD1)       ! (=DEF                                             LOCAL-DEF
                    LD1
                    ([Z].
                     (FORALL [X:(O I)]
                      (IMPLIES
                       (AND
                        (X ONE)
                        (FORALL [Y:I]
                         (IMPLIES (X Y) (X (S Y)))))
                       (X Z)))))
                ...

CONC (A1 A2 A3  ! (D                                                      OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 2
OMEGA:  LOCAL-DEF-INTRO (LAM (Z I) (AND (LD1 Z) (D Z)))

 ;;; step 3.1
OMEGA: LEMMA CONC (LD1 ONE)

OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (≐ (F N ONE) (S ONE)))                    HYP

A2   (A2)        ! (FORALL [X:I]                                           HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))

A3   (A3)        ! (FORALL [N:I,X:I]                                       HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                 HYP

A5   (A5)        ! (FORALL [X:I]                                           HYP
                    (IMPLIES (D X) (D (S X))))
```

```
LD1  (LD1)      ! (=DEF                                                   LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                    LOCAL-DEF
               ...

L1   (A1 A2 A3  ! (LD1 ONE)                                                    OPEN
      A4 A5)
               ...

CONC (A1 A2 A3  ! (D                                                           OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 3.1
OMEGA: DEFN-CONTRACT-LOCAL-DEF L1 () LD1 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                            HYP

A2   (A2)       ! (FORALL [X:I]                                                                   HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                               HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                         HYP

A5   (A5)       ! (FORALL [X:I]                                                                   HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                     LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                      LOCAL-DEF
               ...

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                          OPEN
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                   DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)
               ...

CONC (A1 A2 A3  ! (D                                                                             OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 3.1
OMEGA: SUPPORT L2 ()

               ...

L2 (A1 A2 A3  ! (FORALL [DC-13:(O I)]                         OPEN
    A4 A5)       (IMPLIES
                  (AND
                   (DC-13 ONE)
                   (FORALL [DC-17:I]
                    (IMPLIES
```

```
                     (DC-13 DC-17)
                     (DC-13 (S DC-17)))))
                  (DC-13 ONE)))

 ;;; step 3.1
OMEGA: CALL-OTTER-ON-NODE L2 ...
...
-------- PROOF --------
...

 ;;; step 3.2
OMEGA:  LEMMA CONC (FORALL
                         (LAM (Y I) (IMPLIES (LD1 Y) (LD1 (S Y)))))

 ;;; step 3.2
OMEGA: DEFN-CONTRACT-LOCAL-DEF L3 () LD1 (1 0 1 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 3.2
OMEGA: DEFN-CONTRACT-LOCAL-DEF L4 () LD1 (1 0 2 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 3.2
OMEGA: support L5 ()

               ...

L5 (A1 A2 A3  ! (FORALL [DC-48:I]                                OPEN
    A4 A5)       (IMPLIES
                  (FORALL [DC-59:(O I)]
                   (IMPLIES
                    (AND
                     (DC-59 ONE)
                     (FORALL [DC-63:I]
                      (IMPLIES
                       (DC-59 DC-63)
                       (DC-59 (S DC-63)))))
                    (DC-59 DC-48)))
                  (FORALL [DC-68:(O I)]
                   (IMPLIES
                    (AND
                     (DC-68 ONE)
                     (FORALL [DC-72:I]
                      (IMPLIES
                       (DC-68 DC-72)
                       (DC-68 (S DC-72)))))
                    (DC-68 (S DC-48))))))

 ;;; step 3.2
OMEGA: CALL-OTTER-ON-NODE L5 ...
...
-------- PROOF --------
...

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                            HYP

A2   (A2)       ! (FORALL [X:I]                                                                   HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                               HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                         HYP

A5   (A5)       ! (FORALL [X:I]                                                                   HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                     LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                      LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                   OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                        (DC-13 DC-17)
                        (DC-13 (S DC-17)))))
                      (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                            OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                            DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                                DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))
               ...

CONC (A1 A2 A3  ! (D                                                                                   OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 3.3
OMEGA:  LEMMA CONC (LD1 (S (S (S (S ONE)))))

 ;;; step 3.3
OMEGA: SUPPORT L6 (L1 L3)


L3 (A1 A2 A3  ! (FORALL [Y:I]                                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
    A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L1 (A1 A2 A3  ! (LD1 ONE)                                           DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
    A4 A5)
             ...

L6 (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                                            OPEN
    A4 A5)

 ;;; step 3.3
OMEGA: CALL-OTTER-ON-NODE L6 ...
...
-------- PROOF --------
...

 ;;; step 3.4
OMEGA:  LEMMA CONC (LD2 ONE)

 ;;; step 3.4
OMEGA:  DEFN-CONTRACT-LOCAL-DEF L7 () LD2 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 3.4
OMEGA: SUPPORT L8 (A4 L1)


A4 (A4)       ! (D ONE)                                                                               HYP

L1 (A1 A2 A3  ! (LD1 ONE)                                           DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
    A4 A5)
             ...

L8 (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                                              OPEN
```

```
    A4 A5)

 ;;; step 3.4
OMEGA:  CALL-OTTER-ON-NODE L8 ...
...
-------- PROOF --------
...

 ;;; step 3.5
OMEGA:  LEMMA CONC (FORALL
                         (LAM (Y I) (IMPLIES (LD2 Y) (LD2 (S Y)))))

 ;;; step 3.5
OMEGA: DEFN-CONTRACT-LOCAL-DEF L9 () LD2 (1 0 1 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 3.5
OMEGA: DEFN-CONTRACT-LOCAL-DEF L10 () LD2 (1 0 2 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 3.5
OMEGA:  SUPPORT L11 (A5 L3)


A5  (A5)       ! (FORALL [X:I]                                                                       HYP
                  (IMPLIES (D X) (D (S X))))

L3  (A1 A2 A3  ! (FORALL [Y:I]                              DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
     A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))
               ...

L11 (A1 A2 A3  ! (FORALL [DC-93:I]                                                                  OPEN
     A4 A5)       (IMPLIES
                   (AND (LD1 DC-93) (D DC-93))
                   (AND
                    (LD1 (S DC-93))
                    (D (S DC-93)))))

 ;;; step 3.5
OMEGA: CALL-OTTER-ON-NODE L11 ...
...
-------- PROOF --------
...

OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                               HYP

A2   (A2)        ! (FORALL [X:I]                                                                      HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))

A3   (A3)        ! (FORALL [N:I,X:I]                                                                  HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                                            HYP

A5   (A5)        ! (FORALL [X:I]                                                                      HYP
                    (IMPLIES (D X) (D (S X))))

LD1  (LD1)       ! (=DEF                                                                        LOCAL-DEF
                    LD1
                    ([Z].
                     (FORALL [X:(O I)]
                      (IMPLIES
                       (AND
                        (X ONE)
                        (FORALL [Y:I]
                         (IMPLIES (X Y) (X (S Y)))))
                       (X Z)))))

LD2  (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                         LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                      OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                      DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                          OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                             DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                           OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                             OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                                   OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                            DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))
                ...

CONC (A1 A2 A3  ! (D                                                                                OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 3.6
OMEGA: LEMMA CONC (LD2 (S ONE))

 ;;; step 3.6
OMEGA: SUPPORT L12 (L7 L9)


L9  (A1 A2 A3  ! (FORALL [Y:I]                            DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
     A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L7  (A1 A2 A3  ! (LD2 ONE)                                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
     A4 A5)
               ...

L12 (A1 A2 A3  ! (LD2 (S ONE))                                                                     OPEN
     A4 A5)

 ;;; step 3.6
OMEGA: CALL-OTTER-ON-NODE L12 ...
...
-------- PROOF --------
...

OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                              HYP

A2   (A2)        ! (FORALL [X:I]                                                                     HYP
```

```
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                                 HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                           HYP

A5   (A5)       ! (FORALL [X:I]                                                                     HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                       LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                        LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                     OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                       DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                         OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                             DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                           OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                             OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                                   OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                           DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                                      OTTER: (NIL) (L7 L9)
      A4 A5)
               ...

CONC (A1 A2 A3  ! (D                                                                                OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4
OMEGA: LEMMA CONC (FORALL
                           (LAM (N I)
                            (IMPLIES (LD1 N)
                             (FORALL
                              (LAM (X I)
                               (IMPLIES (LD1 X) (LD2 (F N X))))))))
 ;;; step 4.1
OMEGA:  LOCAL-DEF-INTRO (LAM (N I)
                             (FORALL
                              (LAM
                               (X I)
                               (IMPLIES (LD1 X) (LD2 (F N X))))))

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                               HYP

A2   (A2)       ! (FORALL [X:I]                                                                      HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                                  HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                            HYP

A5   (A5)       ! (FORALL [X:I]                                                                      HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                        LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                         LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                     OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                         OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
```

```
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                            DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                           OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                             OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                                     DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                                   OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                           DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                                       OTTER: (NIL) (L7 L9)
      A4 A5)
               ...

L13  (A1 A2 A3  ! (FORALL [N:I]                                                                      OPEN
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                                         LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))
               ...

CONC (A1 A2 A3  ! (D                                                                                 OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.2
OMEGA: LEMMA L13 (FORALL (LAM (N I) (IMPLIES (LD1 N) (LD3 N))))

 ;;; step 4.2
OMEGA:  DEFN-EXPAND-LOCAL-DEF L13 L14 LD3 (1 0 2 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                                HYP

A2   (A2)       ! (FORALL [X:I]                                                                       HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                                   HYP
                   (=
                    (F (S N) (S X))
```

```
                  (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                        HYP

A5   (A5)        ! (FORALL [X:I]                                                  HYP
                    (IMPLIES (D X) (D (S X))))

LD1  (LD1)       ! (=DEF                                                    LOCAL-DEF
                    LD1
                    ([Z].
                     (FORALL [X:(O I)]
                      (IMPLIES
                       (AND
                        (X ONE)
                        (FORALL [Y:I]
                         (IMPLIES (X Y) (X (S Y)))))
                       (X Z)))))

LD2  (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                     LOCAL-DEF

L2   (A1 A2 A3   ! (FORALL [DC-13:(O I)]                                  OTTER: (NIL)
      A4 A5)        (IMPLIES
                     (AND
                      (DC-13 ONE)
                      (FORALL [DC-17:I]
                       (IMPLIES
                        (DC-13 DC-17)
                        (DC-13 (S DC-17)))))
                     (DC-13 ONE)))

L1   (A1 A2 A3   ! (LD1 ONE)                     DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3   ! (FORALL [DC-48:I]                                      OTTER: (NIL)
      A4 A5)        (IMPLIES
                     (FORALL [DC-59:(O I)]
                      (IMPLIES
                       (AND
                        (DC-59 ONE)
                        (FORALL [DC-63:I]
                         (IMPLIES
                          (DC-59 DC-63)
                          (DC-59 (S DC-63)))))
                       (DC-59 DC-48)))
                     (FORALL [DC-68:(O I)]
                      (IMPLIES
                       (AND
                        (DC-68 ONE)
                        (FORALL [DC-72:I]
                         (IMPLIES
                          (DC-68 DC-72)
                          (DC-68 (S DC-72)))))
                       (DC-68 (S DC-48))))))

L4   (A1 A2 A3   ! (FORALL [DC-26:I]             DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)        (IMPLIES
                     (FORALL [DC-37:(O I)]
                      (IMPLIES
                       (AND
                        (DC-37 ONE)
                        (FORALL [DC-41:I]
                         (IMPLIES
                          (DC-37 DC-41)
                          (DC-37 (S DC-41)))))
                       (DC-37 DC-26)))
                     (LD1 (S DC-26))))

L3   (A1 A2 A3   ! (FORALL [Y:I]                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)        (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3   ! (LD1 (S (S (S (S ONE)))))                        OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3   ! (AND (LD1 ONE) (D ONE))                          OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3   ! (LD2 ONE)                     DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3   ! (FORALL [DC-93:I]                                OTTER: (NIL) (L3 A5)
      A4 A5)        (IMPLIES
                     (AND (LD1 DC-93) (D DC-93))
                     (AND
                      (LD1 (S DC-93))
                      (D (S DC-93)))))

L10  (A1 A2 A3   ! (FORALL [DC-86:I]             DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)        (IMPLIES
                     (AND (LD1 DC-86) (D DC-86))
                     (LD2 (S DC-86))))
```

```
L9   (A1 A2 A3   ! (FORALL [Y:I]                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)        (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3   ! (LD2 (S ONE))                                    OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3   ! (FORALL [N:I]                 DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)        (IMPLIES
                     (LD1 N)
                     (FORALL [X:I]
                      (IMPLIES
                       (LD1 X)
                       (LD2 (F N X))))))

LD3  (LD3)       ! (=DEF                                                    LOCAL-DEF
                    LD3
                    ([N].
                     (FORALL [X:I]
                      (IMPLIES
                       (LD1 X)
                       (LD2 (F N X))))))
                 ...

L14  (A1 A2 A3   ! (FORALL [N:I]                                                 OPEN
      A4 A5)        (IMPLIES (LD1 N) (LD3 N)))
                 ...

CONC (A1 A2 A3   ! (D                                                            OPEN
      A4 A5)        (F
                     (S (S (S (S ONE))))
                     (S (S (S (S ONE))))))

 ;;; step 4.3.1
OMEGA: LEMMA L14 (LD3 ONE)

 ;;; step 4.3.2
OMEGA:  LEMMA L14 (FORALL
                       (LAM (N I) (IMPLIES (LD3 N) (LD3 (S N)))))

OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                           HYP

A2   (A2)        ! (FORALL [X:I]                                                  HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))

A3   (A3)        ! (FORALL [N:I,X:I]                                              HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                        HYP

A5   (A5)        ! (FORALL [X:I]                                                  HYP
                    (IMPLIES (D X) (D (S X))))

LD1  (LD1)       ! (=DEF                                                    LOCAL-DEF
                    LD1
                    ([Z].
                     (FORALL [X:(O I)]
                      (IMPLIES
                       (AND
                        (X ONE)
                        (FORALL [Y:I]
                         (IMPLIES (X Y) (X (S Y)))))
                       (X Z)))))

LD2  (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                     LOCAL-DEF

L2   (A1 A2 A3   ! (FORALL [DC-13:(O I)]                                  OTTER: (NIL)
      A4 A5)        (IMPLIES
                     (AND
                      (DC-13 ONE)
                      (FORALL [DC-17:I]
                       (IMPLIES
                        (DC-13 DC-17)
                        (DC-13 (S DC-17)))))
                     (DC-13 ONE)))

L1   (A1 A2 A3   ! (LD1 ONE)                     DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3   ! (FORALL [DC-48:I]                                      OTTER: (NIL)
      A4 A5)        (IMPLIES
                     (FORALL [DC-59:(O I)]
                      (IMPLIES
                       (AND
                        (DC-59 ONE)
                        (FORALL [DC-63:I]
                         (IMPLIES
                        (DC-59 DC-63)
                        (DC-59 (S DC-63)))))
                     (DC-59 DC-48)))
                   (FORALL [DC-68:(O I)]
                    (IMPLIES
                     (AND
                      (DC-68 ONE)
                      (FORALL [DC-72:I]
                       (IMPLIES
                        (DC-68 DC-72)
                        (DC-68 (S DC-72)))))
                     (DC-68 (S DC-48)))))

L4   (A1 A2 A3 ! (FORALL [DC-26:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)      (IMPLIES
                   (FORALL [DC-37:(O I)]
                    (IMPLIES
                     (AND
                      (DC-37 ONE)
                      (FORALL [DC-41:I]
                       (IMPLIES
                        (DC-37 DC-41)
                        (DC-37 (S DC-41)))))
                     (DC-37 DC-26)))
                   (LD1 (S DC-26))))

L3   (A1 A2 A3 ! (FORALL [Y:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)      (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3 ! (LD1 (S (S (S (S ONE)))))                                     OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3 ! (AND (LD1 ONE) (D ONE))                                        OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3 ! (LD2 ONE)                                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3 ! (FORALL [DC-93:I]                                              OTTER: (NIL) (L3 A5)
      A4 A5)      (IMPLIES
                   (AND (LD1 DC-93) (D DC-93))
                   (AND
                    (LD1 (S DC-93))
                    (D (S DC-93)))))

L10  (A1 A2 A3 ! (FORALL [DC-86:I]                    DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)      (IMPLIES
                   (AND (LD1 DC-86) (D DC-86))
                   (LD2 (S DC-86))))

L9   (A1 A2 A3 ! (FORALL [Y:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)      (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3 ! (LD2 (S ONE))                                                  OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3 ! (FORALL [N:I]                          DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)      (IMPLIES
                   (LD1 N)
                   (FORALL [X:I]
                    (IMPLIES
                     (LD1 X)
                     (LD2 (F N X))))))

LD3  (LD3)     ! (=DEF                                                                     LOCAL-DEF
                  LD3
                  ([N].
                   (FORALL [X:I]
                    (IMPLIES
                     (LD1 X)
                     (LD2 (F N X))))))
                ...

L15  (A1 A2 A3 ! (LD3 ONE)                                                                      OPEN
      A4 A5)
                ...

L16  (A1 A2 A3 ! (FORALL [N:I]                                                                  OPEN
      A4 A5)      (IMPLIES (LD3 N) (LD3 (S N))))
                ...

L14  (A1 A2 A3 ! (FORALL [N:I]                                                                  OPEN
      A4 A5)      (IMPLIES (LD1 N) (LD3 N)))
                ...

CONC (A1 A2 A3 ! (D                                                                             OPEN
      A4 A5)      (F
                   (S (S (S (S ONE))))
                   (S (S (S (S ONE))))))

OMEGA: show-subproblem
```

```
PLAN (NDPLANLINE) Plan line to show: l14


L1   (A1 A2 A3 ! (LD1 ONE)                                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L3   (A1 A2 A3 ! (FORALL [Y:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)      (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3 ! (LD1 (S (S (S (S ONE)))))                                     OTTER: (NIL) (L1 L3)
      A4 A5)

L7   (A1 A2 A3 ! (LD2 ONE)                                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L9   (A1 A2 A3 ! (FORALL [Y:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)      (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3 ! (LD2 (S ONE))                                                  OTTER: (NIL) (L7 L9)
      A4 A5)
                ...

L15  (A1 A2 A3 ! (LD3 ONE)                                                                      OPEN
      A4 A5)
                ...

L16  (A1 A2 A3 ! (FORALL [N:I]                                                                  OPEN
      A4 A5)      (IMPLIES (LD3 N) (LD3 (S N))))

LD3  (LD3)     ! (=DEF                                                                     LOCAL-DEF
                  LD3
                  ([N].
                   (FORALL [X:I]
                    (IMPLIES
                     (LD1 X)
                     (LD2 (F N X))))))

LD2  (LD2)     ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                      LOCAL-DEF

LD1  (LD1)     ! (=DEF                                                                     LOCAL-DEF
                  LD1
                  ([Z].
                   (FORALL [X:(O I)]
                    (IMPLIES
                     (AND
                      (X ONE)
                      (FORALL [Y:I]
                       (IMPLIES (X Y) (X (S Y)))))
                     (X Z)))))

A1   (A1)      ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                            HYP

A2   (A2)      ! (FORALL [X:I]                                                                   HYP
                  (=
                   (F ONE (S X))
                   (S (S (F ONE X)))))

A3   (A3)      ! (FORALL [N:I,X:I]                                                               HYP
                  (=
                   (F (S N) (S X))
                   (F N (F (S N) X))))

A4   (A4)      ! (D ONE)                                                                         HYP

A5   (A5)      ! (FORALL [X:I]                                                                   HYP
                  (IMPLIES (D X) (D (S X))))
                ...

L14  (A1 A2 A3 ! (FORALL [N:I]                                                                  OPEN
      A4 A5)      (IMPLIES (LD1 N) (LD3 N)))

 ;;; step 4.3 -- Enough to show ...
OMEGA:  DEFN-CONTRACT-LOCAL-DEF L14 () LD1 (1 0 1 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3
OMEGA: SUPPORT L17 (L15 L16)

                ...

L16  (A1 A2 A3 ! (FORALL [N:I]                                 OPEN
      A4 A5)      (IMPLIES (LD3 N) (LD3 (S N))))
                ...

L15  (A1 A2 A3 ! (LD3 ONE)                                     OPEN
      A4 A5)
                ...

L17  (A1 A2 A3 ! (FORALL [DC-123:I]                            OPEN
      A4 A5)      (IMPLIES
```

```
                    (FORALL [DC-134:(O I)}
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

 ;;; step 4.3 -- Enough to show ...
OMEGA: CALL-OTTER-ON-NODE L17 ...
...
-------- PROOF --------
...

 ;;; step 4.3.1.1
OMEGA: DEFN-CONTRACT-LOCAL-DEF L15 () LD3 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.1.2
OMEGA: LOCAL-DEF-INTRO (LAM (X I) (LD2 (F ONE X)))

OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                     HYP

A2   (A2)        ! (FORALL [X:I]                                                            HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))

A3   (A3)        ! (FORALL [N:I,X:I]                                                        HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                                  HYP

A5   (A5)        ! (FORALL [X:I]                                                            HYP
                    (IMPLIES (D X) (D (S X))))

LD1  (LD1)       ! (=DEF                                                              LOCAL-DEF
                    LD1
                    ([Z].
                     (FORALL [X:(O I)]
                      (IMPLIES
                       (AND
                        (X ONE)
                        (FORALL [Y:I]
                         (IMPLIES (X Y) (X (S Y)))))
                       (X Z)))))

LD2  (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                               LOCAL-DEF

L2   (A1 A2 A3   ! (FORALL [DC-13:(O I)]                                            OTTER: (NIL)
      A4 A5)        (IMPLIES
                     (AND
                      (DC-13 ONE)
                      (FORALL [DC-17:I]
                       (IMPLIES
                        (DC-13 DC-17)
                        (DC-13 (S DC-17)))))
                     (DC-13 ONE)))

L1   (A1 A2 A3   ! (LD1 ONE)                                DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3   ! (FORALL [DC-48:I]                                                OTTER: (NIL)
      A4 A5)        (IMPLIES
                     (FORALL [DC-59:(O I)]
                      (IMPLIES
                       (AND
                        (DC-59 ONE)
                        (FORALL [DC-63:I]
                         (IMPLIES
                          (DC-59 DC-63)
                          (DC-59 (S DC-63)))))
                       (DC-59 DC-48)))
                     (FORALL [DC-68:(O I)]
                      (IMPLIES
                       (AND
                        (DC-68 ONE)
                        (FORALL [DC-72:I]
                         (IMPLIES
                          (DC-68 DC-72)
                          (DC-68 (S DC-72)))))
                       (DC-68 (S DC-48))))))

L4   (A1 A2 A3   ! (FORALL [DC-26:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)        (IMPLIES
```

```
                     (FORALL [DC-37:(O I)]
                      (IMPLIES
                       (AND
                        (DC-37 ONE)
                        (FORALL [DC-41:I]
                         (IMPLIES
                          (DC-37 DC-41)
                          (DC-37 (S DC-41)))))
                       (DC-37 DC-26)))
                     (LD1 (S DC-26))))

L3   (A1 A2 A3   ! (FORALL [Y:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)        (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3   ! (LD1 (S (S (S (S ONE)))))                                   OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3   ! (AND (LD1 ONE) (D ONE))                                     OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3   ! (LD2 ONE)                                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3   ! (FORALL [DC-93:I]                                           OTTER: (NIL) (L3 A5)
      A4 A5)        (IMPLIES
                     (AND (LD1 DC-93) (D DC-93))
                     (AND
                      (LD1 (S DC-93))
                      (D (S DC-93)))))

L10  (A1 A2 A3   ! (FORALL [DC-86:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)        (IMPLIES
                     (AND (LD1 DC-86) (D DC-86))
                     (LD2 (S DC-86))))

L9   (A1 A2 A3   ! (FORALL [Y:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)        (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3   ! (LD2 (S ONE))                                               OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3   ! (FORALL [N:I]                           DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)        (IMPLIES
                     (LD1 N)
                     (FORALL [X:I]
                      (IMPLIES
                       (LD1 X)
                       (LD2 (F N X))))))

LD3  (LD3)       ! (=DEF                                                                  LOCAL-DEF
                    LD3
                    ([N].
                     (FORALL [X:I]
                      (IMPLIES
                       (LD1 X)
                       (LD2 (F N X))))))
                 ...

L18  (A1 A2 A3   ! (FORALL [DC-151:I]                                                          OPEN
      A4 A5)        (IMPLIES
                     (LD1 DC-151)
                     (LD2 (F ONE DC-151))))

L15  (A1 A2 A3   ! (LD3 ONE)                                   DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)
                 ...

L16  (A1 A2 A3   ! (FORALL [N:I]                                                               OPEN
      A4 A5)        (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3   ! (FORALL [DC-123:I]                                         OTTER: (NIL) (L15 L16)
      A4 A5)        (IMPLIES
                     (FORALL [DC-134:(O I)]
                      (IMPLIES
                       (AND
                        (DC-134 ONE)
                        (FORALL [DC-138:I]
                         (IMPLIES
                          (DC-134 DC-138)
                          (DC-134 (S DC-138)))))
                       (DC-134 DC-123)))
                     (LD3 DC-123)))

L14  (A1 A2 A3   ! (FORALL [N:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)        (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)       ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                       LOCAL-DEF
                 ...

CONC (A1 A2 A3   ! (D                                                                          OPEN
      A4 A5)        (F
```

```
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.1.3
OMEGA: LEMMA L18 (FORALL (LAM (X I) (IMPLIES (LD1 X) (LD4 X))))

 ;;; step 4.3.1.3
OMEGA: DEFN-EXPAND-LOCAL-DEF L18 L19 LD4 (1 0 2 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1    (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                       HYP

A2    (A2)        ! (FORALL [X:I]                                                               HYP
                     (=
                      (F ONE (S X))
                      (S (S (F ONE X)))))

A3    (A3)        ! (FORALL [N:I,X:I]                                                           HYP
                     (=
                      (F (S N) (S X))
                      (F N (F (S N) X))))

A4    (A4)        ! (D ONE)                                                                     HYP

A5    (A5)        ! (FORALL [X:I]                                                               HYP
                     (IMPLIES (D X) (D (S X))))

LD1   (LD1)       ! (=DEF                                                                 LOCAL-DEF
                     LD1
                     ([Z].
                      (FORALL [X:(O I)]
                       (IMPLIES
                        (AND
                         (X ONE)
                         (FORALL [Y:I]
                          (IMPLIES (X Y) (X (S Y)))))
                        (X Z)))))

LD2   (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                  LOCAL-DEF

L2    (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                OTTER: (NIL)
       A4 A5)       (IMPLIES
                     (AND
                      (DC-13 ONE)
                      (FORALL [DC-17:I]
                       (IMPLIES
                        (DC-13 DC-17)
                        (DC-13 (S DC-17)))))
                     (DC-13 ONE)))

L1    (A1 A2 A3  ! (LD1 ONE)                                 DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
       A4 A5)

L5    (A1 A2 A3  ! (FORALL [DC-48:I]                                                    OTTER: (NIL)
       A4 A5)       (IMPLIES
                     (FORALL [DC-59:(O I)]
                      (IMPLIES
                       (AND
                        (DC-59 ONE)
                        (FORALL [DC-63:I]
                         (IMPLIES
                          (DC-59 DC-63)
                          (DC-59 (S DC-63)))))
                       (DC-59 DC-48)))
                     (FORALL [DC-68:(O I)]
                      (IMPLIES
                       (AND
                        (DC-68 ONE)
                        (FORALL [DC-72:I]
                         (IMPLIES
                          (DC-68 DC-72)
                          (DC-68 (S DC-72)))))
                       (DC-68 (S DC-48))))))

L4    (A1 A2 A3  ! (FORALL [DC-26:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
       A4 A5)       (IMPLIES
                     (FORALL [DC-37:(O I)]
                      (IMPLIES
                       (AND
                        (DC-37 ONE)
                        (FORALL [DC-41:I]
                         (IMPLIES
                          (DC-37 DC-41)
                          (DC-37 (S DC-41)))))
                       (DC-37 DC-26)))
                     (LD1 (S DC-26))))

L3    (A1 A2 A3  ! (FORALL [Y:I]                            DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
       A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))
```

```
L6    (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                      OTTER: (NIL) (L1 L3)
       A4 A5)

L8    (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                        OTTER: (NIL) (L1 A4)
       A4 A5)

L7    (A1 A2 A3  ! (LD2 ONE)                                 DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
       A4 A5)

L11   (A1 A2 A3  ! (FORALL [DC-93:I]                                              OTTER: (NIL) (L3 A5)
       A4 A5)       (IMPLIES
                     (AND (LD1 DC-93) (D DC-93))
                     (AND
                      (LD1 (S DC-93))
                      (D (S DC-93)))))

L10   (A1 A2 A3  ! (FORALL [DC-86:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
       A4 A5)       (IMPLIES
                     (AND (LD1 DC-86) (D DC-86))
                     (LD2 (S DC-86))))

L9    (A1 A2 A3  ! (FORALL [Y:I]                           DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
       A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12   (A1 A2 A3  ! (LD2 (S ONE))                                                  OTTER: (NIL) (L7 L9)
       A4 A5)

L13   (A1 A2 A3  ! (FORALL [N:I]                             DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
       A4 A5)       (IMPLIES
                     (LD1 N)
                     (FORALL [X:I]
                      (IMPLIES
                       (LD1 X)
                       (LD2 (F N X))))))

LD3   (LD3)       ! (=DEF                                                                  LOCAL-DEF
                     LD3
                     ([N].
                      (FORALL [X:I]
                       (IMPLIES
                        (LD1 X)
                        (LD2 (F N X))))))

L18   (A1 A2 A3  ! (FORALL [DC-151:I]                        DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
       A4 A5)       (IMPLIES
                     (LD1 DC-151)
                     (LD2 (F ONE DC-151))))

L15   (A1 A2 A3  ! (LD3 ONE)                                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
       A4 A5)
                 ...

L16   (A1 A2 A3  ! (FORALL [N:I]                                                                  OPEN
       A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17   (A1 A2 A3  ! (FORALL [DC-123:I]                                            OTTER: (NIL) (L15 L16)
       A4 A5)       (IMPLIES
                     (FORALL [DC-134:(O I)]
                      (IMPLIES
                       (AND
                        (DC-134 ONE)
                        (FORALL [DC-138:I]
                         (IMPLIES
                          (DC-134 DC-138)
                          (DC-134 (S DC-138)))))
                       (DC-134 DC-123)))
                     (LD3 DC-123)))

L14   (A1 A2 A3  ! (FORALL [N:I]                           DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
       A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4   (LD4)       ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                      LOCAL-DEF
                 ...

L19   (A1 A2 A3  ! (FORALL [X:I]                                                                  OPEN
       A4 A5)       (IMPLIES (LD1 X) (LD4 X)))
                 ...

CONC  (A1 A2 A3  ! (D                                                                             OPEN
       A4 A5)       (F
                     (S (S (S (S ONE))))
                     (S (S (S (S ONE))))))

 ;;; step 4.3.1.4.1
OMEGA: LEMMA L19 (LD4 ONE)

;;; step 4.3.1.4.2
OMEGA:  LEMMA L19 (FORALL
                            (LAM (X I) (IMPLIES (LD4 X) (LD4 (S X)))))
 ;;; step 4.3.1.4 -- Enough to show
OMEGA: DEFN-CONTRACT-LOCAL-DEF L19 () LD1 (1 0 1 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1    (A1)         ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                        HYP

A2    (A2)         ! (FORALL [X:I]                                                               HYP
                      (=
                       (F ONE (S X))
                       (S (S (F ONE X)))))

A3    (A3)         ! (FORALL [N:I,X:I]                                                           HYP
                      (=
                       (F (S N) (S X))
                       (F N (F (S N) X))))

A4    (A4)         ! (D ONE)                                                                     HYP

A5    (A5)         ! (FORALL [X:I]                                                               HYP
                      (IMPLIES (D X) (D (S X))))

LD1   (LD1)        ! (=DEF                                                                 LOCAL-DEF
                      LD1
                      ([Z].
                       (FORALL [X:(O I)]
                        (IMPLIES
                         (AND
                          (X ONE)
                          (FORALL [Y:I]
                           (IMPLIES (X Y) (X (S Y)))))
                         (X Z)))))

LD2   (LD2)        ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                  LOCAL-DEF

L2    (A1 A2 A3    ! (FORALL [DC-13:(O I)]                                               OTTER: (NIL)
       A4 A5)         (IMPLIES
                       (AND
                        (DC-13 ONE)
                        (FORALL [DC-17:I]
                         (IMPLIES
                          (DC-13 DC-17)
                          (DC-13 (S DC-17)))))
                       (DC-13 ONE)))

L1    (A1 A2 A3    ! (LD1 ONE)                                   DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
       A4 A5)

L5    (A1 A2 A3    ! (FORALL [DC-48:I]                                                   OTTER: (NIL)
       A4 A5)         (IMPLIES
                       (FORALL [DC-59:(O I)]
                        (IMPLIES
                         (AND
                          (DC-59 ONE)
                          (FORALL [DC-63:I]
                           (IMPLIES
                            (DC-59 DC-63)
                            (DC-59 (S DC-63)))))
                         (DC-59 DC-48)))
                       (FORALL [DC-68:(O I)]
                        (IMPLIES
                         (AND
                          (DC-68 ONE)
                          (FORALL [DC-72:I]
                           (IMPLIES
                            (DC-68 DC-72)
                            (DC-68 (S DC-72)))))
                         (DC-68 (S DC-48))))))

L4    (A1 A2 A3    ! (FORALL [DC-26:I]                   DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
       A4 A5)         (IMPLIES
                       (FORALL [DC-37:(O I)]
                        (IMPLIES
                         (AND
                          (DC-37 ONE)
                          (FORALL [DC-41:I]
                           (IMPLIES
                            (DC-37 DC-41)
                            (DC-37 (S DC-41)))))
                         (DC-37 DC-26)))
                       (LD1 (S DC-26))))

L3    (A1 A2 A3    ! (FORALL [Y:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
       A4 A5)         (IMPLIES (LD1 Y) (LD1 (S Y))))

L6    (A1 A2 A3    ! (LD1 (S (S (S (S ONE)))))                                   OTTER: (NIL) (L1 L3)
       A4 A5)

L8    (A1 A2 A3    ! (AND (LD1 ONE) (D ONE))                                     OTTER: (NIL) (L1 A4)
       A4 A5)
```

```
L7    (A1 A2 A3    ! (LD2 ONE)                                   DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
       A4 A5)

L11   (A1 A2 A3    ! (FORALL [DC-93:I]                                           OTTER: (NIL) (L3 A5)
       A4 A5)         (IMPLIES
                       (AND (LD1 DC-93) (D DC-93))
                       (AND
                        (LD1 (S DC-93))
                        (D (S DC-93)))))

L10   (A1 A2 A3    ! (FORALL [DC-86:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
       A4 A5)         (IMPLIES
                       (AND (LD1 DC-86) (D DC-86))
                       (LD2 (S DC-86))))

L9    (A1 A2 A3    ! (FORALL [Y:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
       A4 A5)         (IMPLIES (LD2 Y) (LD2 (S Y))))

L12   (A1 A2 A3    ! (LD2 (S ONE))                                               OTTER: (NIL) (L7 L9)
       A4 A5)

L13   (A1 A2 A3    ! (FORALL [N:I]                        DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
       A4 A5)         (IMPLIES
                       (LD1 N)
                       (FORALL [X:I]
                        (IMPLIES
                         (LD1 X)
                         (LD2 (F N X))))))

LD3   (LD3)        ! (=DEF                                                                 LOCAL-DEF
                      LD3
                      ([N].
                       (FORALL [X:I]
                        (IMPLIES
                         (LD1 X)
                         (LD2 (F N X))))))

L18   (A1 A2 A3    ! (FORALL [DC-151:I]                   DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
       A4 A5)         (IMPLIES
                       (LD1 DC-151)
                       (LD2 (F ONE DC-151))))

L15   (A1 A2 A3    ! (LD3 ONE)                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
       A4 A5)
                 ...

L16   (A1 A2 A3    ! (FORALL [N:I]                                                              OPEN
       A4 A5)         (IMPLIES (LD3 N) (LD3 (S N))))

L17   (A1 A2 A3    ! (FORALL [DC-123:I]                                        OTTER: (NIL) (L15 L16)
       A4 A5)         (IMPLIES
                       (FORALL [DC-134:(O I)]
                        (IMPLIES
                         (AND
                          (DC-134 ONE)
                          (FORALL [DC-138:I]
                           (IMPLIES
                            (DC-134 DC-138)
                            (DC-134 (S DC-138)))))
                         (DC-134 DC-123)))
                       (LD3 DC-123)))

L14   (A1 A2 A3    ! (FORALL [N:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
       A4 A5)         (IMPLIES (LD1 N) (LD3 N)))

LD4   (LD4)        ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                      LOCAL-DEF
                 ...

L20   (A1 A2 A3    ! (LD4 ONE)                                                                  OPEN
       A4 A5)
                 ...

L21   (A1 A2 A3    ! (FORALL [X:I]                                                              OPEN
       A4 A5)         (IMPLIES (LD4 X) (LD4 (S X))))
                 ...

L22   (A1 A2 A3    ! (FORALL [DC-165:I]                                                         OPEN
       A4 A5)         (IMPLIES
                       (FORALL [DC-176:(O I)]
                        (IMPLIES
                         (AND
                          (DC-176 ONE)
                          (FORALL [DC-180:I]
                           (IMPLIES
                            (DC-176 DC-180)
                            (DC-176 (S DC-180)))))
                         (DC-176 DC-165)))
                       (LD4 DC-165)))

L19   (A1 A2 A3    ! (FORALL [X:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))
               ...

CONC (A1 A2 A3  ! (D                                                      OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.1.4
OMEGA: SUPPORT L22 (L20 L21)

               ...

L21 (A1 A2 A3  ! (FORALL [X:I]                            OPEN
     A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))
               ...

L20 (A1 A2 A3  ! (LD4 ONE)                                OPEN
     A4 A5)
               ...

L22 (A1 A2 A3  ! (FORALL [DC-165:I]                       OPEN
     A4 A5)       (IMPLIES
                   (FORALL [DC-176:(O I)]
                    (IMPLIES
                     (AND
                      (DC-176 ONE)
                      (FORALL [DC-180:I]
                       (IMPLIES
                        (DC-176 DC-180)
                        (DC-176 (S DC-180)))))
                     (DC-176 DC-165)))
                   (LD4 DC-165)))

 ;;; step 4.3.1.4
OMEGA:  CALL-OTTER-ON-NODE L22 ...
...
-------- PROOF --------
...

;;; step 4.3.1.4.1.1
OMEGA: DEFN-CONTRACT-LOCAL-DEF L20 () LD4 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                    HYP

A2   (A2)        ! (FORALL [X:I]                                           HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))

A3   (A3)        ! (FORALL [N:I,X:I]                                       HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                 HYP

A5   (A5)        ! (FORALL [X:I]                                           HYP
                    (IMPLIES (D X) (D (S X))))

LD1  (LD1)       ! (=DEF                                             LOCAL-DEF
                    LD1
                    ([Z].
                     (FORALL [X:(O I)]
                      (IMPLIES
                       (AND
                        (X ONE)
                        (FORALL [Y:I]
                         (IMPLIES (X Y) (X (S Y)))))
                       (X Z)))))

LD2  (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))              LOCAL-DEF

L2   (A1 A2 A3   ! (FORALL [DC-13:(O I)]                          OTTER: (NIL)
      A4 A5)        (IMPLIES
                     (AND
                      (DC-13 ONE)
                      (FORALL [DC-17:I]
                       (IMPLIES
                        (DC-13 DC-17)
                        (DC-13 (S DC-17)))))
                     (DC-13 ONE)))

L1   (A1 A2 A3   ! (LD1 ONE)                DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3   ! (FORALL [DC-48:I]                              OTTER: (NIL)
      A4 A5)        (IMPLIES
```

```
                     (FORALL [DC-59:(O I)]
                      (IMPLIES
                       (AND
                        (DC-59 ONE)
                        (FORALL [DC-63:I]
                         (IMPLIES
                          (DC-59 DC-63)
                          (DC-59 (S DC-63)))))
                       (DC-59 DC-48)))
                     (FORALL [DC-68:(O I)]
                      (IMPLIES
                       (AND
                        (DC-68 ONE)
                        (FORALL [DC-72:I]
                         (IMPLIES
                          (DC-68 DC-72)
                          (DC-68 (S DC-72)))))
                       (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                  OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                               DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                        OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                    DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                            OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                      DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                               LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]                 DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                            DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)
              ...

L16  (A1 A2 A3  ! (FORALL [N:I]                                                            OPEN
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                     OTTER: (NIL) (L15 L16)
     A4 A5)       (IMPLIES
                   (FORALL [DC-134:(O I)]
                    (IMPLIES
                     (AND
                      (DC-134 ONE)
                      (FORALL [DC-138:I]
                       (IMPLIES
                        (DC-134 DC-138)
                        (DC-134 (S DC-138)))))
                     (DC-134 DC-123)))
                   (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
     A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                               LOCAL-DEF
                ...

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                                                   OPEN
     A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
     A4 A5)
                ...

L21  (A1 A2 A3  ! (FORALL [X:I]                                                                       OPEN
     A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                             OTTER: (NIL) (L20 L21)
     A4 A5)       (IMPLIES
                   (FORALL [DC-176:(O I)]
                    (IMPLIES
                     (AND
                      (DC-176 ONE)
                      (FORALL [DC-180:I]
                       (IMPLIES
                        (DC-176 DC-180)
                        (DC-176 (S DC-180)))))
                     (DC-176 DC-165)))
                   (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
     A4 A5)       (IMPLIES (LD1 X) (LD4 X)))
                ...

CONC (A1 A2 A3  ! (D                                                                                  OPEN
     A4 A5)       (F
                   (S (S (S (S ONE))))
                   (S (S (S (S ONE))))))

 ;;; step 4.3.1.4.1.2-3
OMEGA: SUPPORT L23 (A1 L12)


A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                      HYP

L12 (A1 A2 A3   ! (LD2 (S ONE))                              OTTER: (NIL) (L7 L9)
     A4 A5)
                ...

L23 (A1 A2 A3   ! (LD2 (F ONE ONE))                                         OPEN
     A4 A5)

 ;;; step 4.3.1.4.1.2-3
OMEGA: CALL-OTTER-ON-NODE L23 ...
...
-------- PROOF --------
...

 ;;; step 4.3.1.4.2.1-6
OMEGA: DEFN-CONTRACT-LOCAL-DEF L21 () LD4 (1 0 1 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.1.4.2.1-6
OMEGA: DEFN-CONTRACT-LOCAL-DEF L24 () LD4 (1 0 2 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.1.4.2.1-6
OMEGA:  SUPPORT L25 (A2 L9)


A2   (A2)       ! (FORALL [X:I]                                                                      HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

L9  (A1 A2 A3   ! (FORALL [Y:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
     A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))
                ...

L25 (A1 A2 A3   ! (FORALL [DC-201:I]                                                                 OPEN
```

```
     A4 A5)       (IMPLIES
                   (LD2 (F ONE DC-201))
                   (LD2 (F ONE (S DC-201)))))

 ;;; step 4.3.1.4.2.1-6
OMEGA:  CALL-OTTER-ON-NODE L25 ...
...
-------- PROOF --------
...

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                               HYP

A2   (A2)       ! (FORALL [X:I]                                                                      HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                                  HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                            HYP

A5   (A5)       ! (FORALL [X:I]                                                                      HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                        LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                         LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                     OTTER: (NIL)
      A4 A5)      (IMPLIES
                   (AND
                    (DC-13 ONE)
                    (FORALL [DC-17:I]
                     (IMPLIES
                      (DC-13 DC-17)
                      (DC-13 (S DC-17)))))
                   (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                         OTTER: (NIL)
      A4 A5)      (IMPLIES
                   (FORALL [DC-59:(O I)]
                    (IMPLIES
                     (AND
                      (DC-59 ONE)
                      (FORALL [DC-63:I]
                       (IMPLIES
                        (DC-59 DC-63)
                        (DC-59 (S DC-63)))))
                     (DC-59 DC-48)))
                   (FORALL [DC-68:(O I)]
                    (IMPLIES
                     (AND
                      (DC-68 ONE)
                      (FORALL [DC-72:I]
                       (IMPLIES
                        (DC-68 DC-72)
                        (DC-68 (S DC-72)))))
                     (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)      (IMPLIES
                   (FORALL [DC-37:(O I)]
                    (IMPLIES
                     (AND
                      (DC-37 ONE)
                      (FORALL [DC-41:I]
                       (IMPLIES
                        (DC-37 DC-41)
                        (DC-37 (S DC-41)))))
                     (DC-37 DC-26)))
                   (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                           DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)      (IMPLIES (LD1 Y) (LD1 (S Y))))
```

```
L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                            OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                              OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                          DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                    OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                        OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                       DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                            LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]                  DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                          DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)
               ...

L16  (A1 A2 A3  ! (FORALL [N:I]                                                         OPEN
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                  OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                 LOCAL-DEF

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                   OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                          DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                   OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]                DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))
```

```
L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                 OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))
               ...

CONC (A1 A2 A3  ! (D                                                                 OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.1
OMEGA:  FORALLI L16 n1 ()

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                               HYP

A2   (A2)       ! (FORALL [X:I]                                                      HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                  HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                            HYP

A5   (A5)       ! (FORALL [X:I]                                                      HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                        LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                         LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                     OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                         OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))
```

```
L4   (A1 A2 A3  ! (FORALL [DC-26:I]                    DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                    OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                      OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                                 DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                            OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                   DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                                OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                         DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                                  LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]                    DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                                DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)
               ...

L26  (A1 A2 A3  ! (IMPLIES (LD3 N1) (LD3 (S N1)))                                             OPEN
      A4 A5)

L16  (A1 A2 A3  ! (FORALL [N:I]                                               FORALLI: (N1) (L26)
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                        OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                       LOCAL-DEF
```

```
L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                           OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                               DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                           OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                        OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))
               ...

CONC (A1 A2 A3  ! (D                                                                          OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.1,3
OMEGA: IMPI

IMPLICATION (NDLINE) Implication to justify: [L26]L26
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                         HYP

A2   (A2)       ! (FORALL [X:I]                                                                HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                            HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                      HYP

A5   (A5)       ! (FORALL [X:I]                                                                HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                  LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                   LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                               OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                 DEFN-CONTRACT-LOCAL-DEF: .((0)) (L2 LD1)
     A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                   OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                    DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                   OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                     OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                           OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                   DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                               OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                         DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)        ! (=DEF                                                                LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]                    DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                                 DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)
                ...
```

```
L28  (L27 A1    ! (LD3 (S N1))                                                                OPEN
      A2 A3 A4
      A5)

L26  (A1 A2 A3  ! (IMPLIES (LD3 N1) (LD3 (S N1)))                                       IMPI: (L28)

L16  (A1 A2 A3  ! (FORALL [N:I]                                                 FORALLI: (N1) (L26)
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                        OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)        ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                     LOCAL-DEF

L27  (L27)        ! (LD3 N1)                                                                   HYP

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                           OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                                 DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                           OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                        OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                       DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))
                ...

CONC (A1 A2 A3  ! (D                                                                          OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.2
OMEGA: DEFN-EXPAND-LOCAL-DEF () L27 LD3 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.2.4
OMEGA:  DEFN-CONTRACT-LOCAL-DEF L28 () LD3 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1   (A1)         ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                       HYP

A2   (A2)         ! (FORALL [X:I]                                                              HYP
                     (=
                      (F ONE (S X))
                      (S (S (F ONE X)))))

A3   (A3)         ! (FORALL [N:I,X:I]                                                          HYP
                     (=
                      (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                         HYP

A5   (A5)       ! (FORALL [X:I]                                                   HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                     LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                      LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                   OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                       OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]           DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                        OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                          OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]          DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))
```

```
L9   (A1 A2 A3  ! (FORALL [Y:I]              DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                    OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                     LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]           DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                       DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)
                ...

L30  (L27 A1    ! (FORALL [DC-225:I]                                             OPEN
      A2 A3 A4     (IMPLIES
      A5)           (LD1 DC-225)
                    (LD2 (F (S N1) DC-225))))

L28  (L27 A1    ! (LD3 (S N1))                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L30 LD3)
      A2 A3 A4
      A5)

L26  (A1 A2 A3  ! (IMPLIES (LD3 N1) (LD3 (S N1)))                           IMPI: (L28)
      A4 A5)

L16  (A1 A2 A3  ! (FORALL [N:I]                                      FORALLI: (N1) (L26)
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                              OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]              DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                          LOCAL-DEF

L27  (L27)      ! (LD3 N1)                                                        HYP

L29  (L27)      ! (FORALL [DC-217:I]                DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                   (IMPLIES
                    (LD1 DC-217)
                    (LD2 (F N1 DC-217))))

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                       DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]         DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]              DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3  ! (FORALL [DC-165:I]                              OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                                    DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))
                ...

CONC (A1 A2 A3  ! (D                                                                                          OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.5
OMEGA: LOCAL-DEF-INTRO (LAM (X I) (LD2 (F (S N1) X)))

 ;;; step 4.3.2.6
OMEGA: LEMMA L30       (FORALL
                              (LAM (X I) (IMPLIES (LD1 X) (LD5 X))))

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                                         HYP

A2   (A2)       ! (FORALL [X:I]                                                                                HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                                            HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                                      HYP

A5   (A5)       ! (FORALL [X:I]                                                                                HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                                  LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                                   LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                                               OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                                   OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
```

```
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                                   OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                                     OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                                                DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                                           OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                                               OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                                       DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                                                  LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]                                  DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                                               DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)
                ...

L30  (L27 A1    ! (FORALL [DC-225:I]                                                                          OPEN
      A2 A3 A4     (IMPLIES
      A5)           (LD1 DC-225)
                    (LD2 (F (S N1) DC-225))))

L28  (L27 A1    ! (LD3 (S N1))                                            DEFN-CONTRACT-LOCAL-DEF: ((0)) (L30 LD3)
      A2 A3 A4
      A5)

L26  (A1 A2 A3  ! (IMPLIES (LD3 N1) (LD3 (S N1)))                                                      IMPI: (L28)
      A4 A5)

L16  (A1 A2 A3  ! (FORALL [N:I]                                                                  FORALLI: (N1) (L26)
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                                        OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                     (FORALL [DC-138:I]
                      (IMPLIES
                       (DC-134 DC-138)
                       (DC-134 (S DC-138)))))
                    (DC-134 DC-123)))
                  (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                      LOCAL-DEF

L27  (L27)      ! (LD3 N1)                                                                    HYP

L29  (L27)      ! (FORALL [DC-217:I]                DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                   (IMPLIES
                    (LD1 DC-217)
                    (LD2 (F N1 DC-217))))

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                    OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                       DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                    OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]             DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                   OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))

LD5  (LD5)      ! (=DEF LD5 ([X].(LD2 (F (S N1) X))))                                   LOCAL-DEF
               ...

L31  (L27 A1    ! (FORALL [X:I]                                                          OPEN
      A2 A3 A4     (IMPLIES (LD1 X) (LD5 X)))
      A5)
               ...

CONC (A1 A2 A3  ! (D                                                                     OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.6
OMEGA: DEFN-EXPAND-LOCAL-DEF L30 L31 LD5 (1 0 2 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.2.7.1
OMEGA: LEMMA L31 (LD5 ONE)

 ;;; step 4.3.2.7.2
OMEGA: LEMMA L31 (FORALL
                       (LAM (X I) (IMPLIES (LD5 X) (LD5 (S X)))))

 ;;; step 4.3.2.7 -- Enough to ...
OMEGA: DEFN-CONTRACT-LOCAL-DEF L31 () LD1 (1 0 1 0)
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                    HYP

A2   (A2)       ! (FORALL [X:I]                                                           HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))
```

```
A3   (A3)       ! (FORALL [N:I,X:I]                                                       HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                 HYP

A5   (A5)       ! (FORALL [X:I]                                                           HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                             LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                                  LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                          OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                              OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                   DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                  OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                        OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
```

```
                  (AND (LD1 DC-86) (D DC-86))
                  (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                             OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                        DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                                LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]                   DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                               DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)

L30  (L27 A1    ! (FORALL [DC-225:I]                   DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L31 LD5)
      A2 A3 A4     (IMPLIES
      A5)           (LD1 DC-225)
                    (LD2 (F (S N1) DC-225))))

L28  (L27 A1    ! (LD3 (S N1))                            DEFN-CONTRACT-LOCAL-DEF: ((0)) (L30 LD3)
      A2 A3 A4
      A5)

L26  (A1 A2 A3  ! (IMPLIES (LD3 N1) (LD3 (S N1)))                                     IMPI: (L28)
      A4 A5)

L16  (A1 A2 A3  ! (FORALL [N:I]                                                 FORALLI: (N1) (L26)
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                           OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                     LOCAL-DEF

L27  (L27)      ! (LD3 N1)                                                                   HYP

L29  (L27)      ! (FORALL [DC-217:I]                         DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                   (IMPLIES
                    (LD1 DC-217)
                    (LD2 (F N1 DC-217))))

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                         OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                               DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                         OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))
```

```
L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                       OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))

LD5  (LD5)      ! (=DEF LD5 ([X].(LD2 (F (S N1) X))))                                  LOCAL-DEF
               ...

L32  (L27 A1    ! (LD5 ONE)                                                                 OPEN
      A2 A3 A4
      A5)
               ...

L33  (L27 A1    ! (FORALL [X:I]                                                             OPEN
      A2 A3 A4     (IMPLIES (LD5 X) (LD5 (S X))))
      A5)
               ...

L34  (L27 A1    ! (FORALL [DC-239:I]                                                        OPEN
      A2 A3 A4     (IMPLIES
      A5)           (FORALL [DC-250:(O I)]
                     (IMPLIES
                      (AND
                       (DC-250 ONE)
                       (FORALL [DC-254:I]
                        (IMPLIES
                         (DC-250 DC-254)
                         (DC-250 (S DC-254)))))
                      (DC-250 DC-239)))
                    (LD5 DC-239)))

L31  (L27 A1    ! (FORALL [X:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L34 LD1)
      A2 A3 A4     (IMPLIES (LD1 X) (LD5 X)))
      A5)
               ...

CONC (A1 A2 A3  ! (D                                                                        OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.7
OMEGA: SUPPORT L34 (L32 L33)

               ...

L33 (L27 A1    ! (FORALL [X:I]                                          OPEN
     A2 A3 A4     (IMPLIES (LD5 X) (LD5 (S X))))
     A5)
               ...

L32 (L27 A1    ! (LD5 ONE)                                              OPEN
     A2 A3 A4
     A5)
               ...

L34 (L27 A1    ! (FORALL [DC-239:I]                                     OPEN
     A2 A3 A4     (IMPLIES
     A5)           (FORALL [DC-250:(O I)]
                    (IMPLIES
                     (AND
                      (DC-250 ONE)
                      (FORALL [DC-254:I]
                       (IMPLIES
                        (DC-250 DC-254)
                        (DC-250 (S DC-254)))))
                     (DC-250 DC-239)))
                   (LD5 DC-239)))

 ;;; step 4.3.2.7
OMEGA: CALL-OTTER-ON-NODE L34 ...
...
-------- PROOF --------
...

 ;;; step 4.3.2.7.1
OMEGA: DEFN-CONTRACT-LOCAL-DEF L32 () LD5 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed
```

```
 ;;; step 4.3.2.7.2-3
OMEGA: SUPPORT L35 (A1 L12)

A1  (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                        HYP

L12 (A1 A2 A3   ! (LD2 (S ONE))                          OTTER: (NIL) (L7 L9)
     A4 A5)
                ...

L35 (L27 A1     ! (LD2 (F (S N1) ONE))                                       OPEN
     A2 A3 A4
     A5)

 ;;; step 4.3.2.7.2-3
OMEGA:  CALL-OTTER-ON-NODE L35 ...
...
-------- PROOF --------
...

 ;;; step 4.3.2.7.2.1
OMEGA:  FORALLI

UNIV-LINE (NDLINE) A Universal line to prove: [L33]

PARAMETER (TERMSYM) New parameter: [x1]

LINE (NDLINE) A line: [()]
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.2.7.2.1
OMEGA: IMPI

IMPLICATION (NDLINE) Implication to justify: [L36]
;;;CSM Arbitrary [2]: 0 provers have to be killed


OMEGA: show-pds
A1   (A1)        ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                HYP

A2   (A2)        ! (FORALL [X:I]                                                       HYP
                    (=
                     (F ONE (S X))
                     (S (S (F ONE X)))))

A3   (A3)        ! (FORALL [N:I,X:I]                                                   HYP
                    (=
                     (F (S N) (S X))
                     (F N (F (S N) X))))

A4   (A4)        ! (D ONE)                                                             HYP

A5   (A5)        ! (FORALL [X:I]                                                       HYP
                    (IMPLIES (D X) (D (S X))))

LD1  (LD1)       ! (=DEF                                                         LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)       ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                          LOCAL-DEF

L2   (A1 A2 A3   ! (FORALL [DC-13:(O I)]                                      OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                     (DC-13 ONE)
                     (FORALL [DC-17:I]
                      (IMPLIES
                       (DC-13 DC-17)
                       (DC-13 (S DC-17)))))
                    (DC-13 ONE)))

L1   (A1 A2 A3   ! (LD1 ONE)                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3   ! (FORALL [DC-48:I]                                          OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
```

```
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3   ! (FORALL [DC-26:I]                  DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3   ! (FORALL [Y:I]                      DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3   ! (LD1 (S (S (S (S ONE)))))                                OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3   ! (AND (LD1 ONE) (D ONE))                                  OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3   ! (LD2 ONE)                                 DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3   ! (FORALL [DC-93:I]                                        OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3   ! (FORALL [DC-86:I]                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3   ! (FORALL [Y:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3   ! (LD2 (S ONE))                                            OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3   ! (FORALL [N:I]                       DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)       ! (=DEF                                                               LOCAL-DEF
                  LD3
                  ([N].
                   (FORALL [X:I]
                    (IMPLIES
                     (LD1 X)
                     (LD2 (F N X))))))

L18  (A1 A2 A3   ! (FORALL [DC-151:I]                  DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3   ! (LD3 ONE)                                DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)

L30  (L27 A1     ! (FORALL [DC-225:I]                  DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L31 LD5)
      A2 A3 A4     (IMPLIES
      A5)           (LD1 DC-225)
                    (LD2 (F (S N1) DC-225))))

L28  (L27 A1     ! (LD3 (S N1))                             DEFN-CONTRACT-LOCAL-DEF: ((0)) (L30 LD3)
      A2 A3 A4
      A5)

L26  (A1 A2 A3   ! (IMPLIES (LD3 N1) (LD3 (S N1)))                                    IMPI: (L28)
      A4 A5)

L16  (A1 A2 A3   ! (FORALL [N:I]                                                FORALLI: (N1) (L26)
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))
```

```
L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                          OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                          LOCAL-DEF

L27  (L27)      ! (LD3 N1)                                                                        HYP

L29  (L27)      ! (FORALL [DC-217:I]                           DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                   (IMPLIES
                    (LD1 DC-217)
                    (LD2 (F N1 DC-217))))

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                            OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                            OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                          OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))

LD5  (LD5)      ! (=DEF LD5 ([X].(LD2 (F (S N1) X))))                                       LOCAL-DEF

L37  (L37)      ! (LD5 X1)                                                                        HYP

L35  (L27 A1    ! (LD2 (F (S N1) ONE))                                         OTTER: (NIL) (L12 A1)
      A2 A3 A4
      A5)

L32  (L27 A1    ! (LD5 ONE)                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L35 LD5)
      A2 A3 A4
      A5)
                ...

L38  (L37 L27   ! (LD5 (S X1))                                                                   OPEN
      A1 A2 A3
      A4 A5)

L36  (L27 A1    ! (IMPLIES (LD5 X1) (LD5 (S X1)))                                          IMPI: (L38)
      A2 A3 A4
      A5)

L33  (L27 A1    ! (FORALL [X:I]                                                   FORALLI: (X1) (L36)
      A2 A3 A4     (IMPLIES (LD5 X) (LD5 (S X))))
      A5)

L34  (L27 A1    ! (FORALL [DC-239:I]                                          OTTER: (NIL) (L32 L33)
      A2 A3 A4     (IMPLIES
      A5)           (FORALL [DC-250:(O I)]
                     (IMPLIES
                      (AND
```

```
                       (DC-250 ONE)
                       (FORALL [DC-254:I]
                        (IMPLIES
                         (DC-250 DC-254)
                         (DC-250 (S DC-254)))))
                      (DC-250 DC-239)))
                    (LD5 DC-239)))

L31  (L27 A1    ! (FORALL [X:I]                          DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L34 LD1)
      A2 A3 A4     (IMPLIES (LD1 X) (LD5 X)))
      A5)
                ...

CONC (A1 A2 A3  ! (D                                                                             OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.7.2.2
OMEGA: DEFN-EXPAND-LOCAL-DEF () L37 LD5 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.2.7.2.3
OMEGA: DEFN-EXPAND-LOCAL-DEF () L39 LD2 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.2.7.2.3
OMEGA: ANDE

CONJUNCTION (NDLINE) Conjunction to split: [L40]

LCONJ (NDLINE) Left conjunct: [()]

RCONJ (NDLINE) Right conjunct: [()]
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.2.7.2.4
OMEGA: DEFN-CONTRACT-LOCAL-DEF L38 () LD5 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 4.3.2.7.2.5
OMEGA:   LEMMA L43 (LD2 (F N1 (F (S N1) X1)))

 ;;; step 4.3.2.7.2.5
OMEGA: SUPPORT L44 (L41 L29)


L41 (L37)      ! (LD1 (F (S N1) X1))                                                      ANDE: (L40)

L29 (L27)      ! (FORALL [DC-217:I]                           DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                  (IMPLIES
                   (LD1 DC-217)
                   (LD2 (F N1 DC-217))))
               ...

L44 (L37 L27   ! (LD2 (F N1 (F (S N1) X1)))                                                      OPEN
     A1 A2 A3
     A4 A5)

 ;;; step 4.3.2.7.2.5
OMEGA: CALL-OTTER-ON-NODE L44 ...
...
-------- PROOF --------
...

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                                            HYP

A2   (A2)       ! (FORALL [X:I]                                                                   HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                               HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                                         HYP

A5   (A5)       ! (FORALL [X:I]                                                                   HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                                     LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)        ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                            LOCAL-DEF

L2   (A1 A2 A3    ! (FORALL [DC-13:(0 I)]                                        OTTER: (NIL)
      A4 A5)         (IMPLIES
                      (AND
                       (DC-13 ONE)
                       (FORALL [DC-17:I]
                        (IMPLIES
                         (DC-13 DC-17)
                         (DC-13 (S DC-17)))))
                      (DC-13 ONE)))

L1   (A1 A2 A3    ! (LD1 ONE)                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3    ! (FORALL [DC-48:I]                                            OTTER: (NIL)
      A4 A5)         (IMPLIES
                      (FORALL [DC-59:(0 I)]
                       (IMPLIES
                        (AND
                         (DC-59 ONE)
                         (FORALL [DC-63:I]
                          (IMPLIES
                           (DC-59 DC-63)
                           (DC-59 (S DC-63)))))
                        (DC-59 DC-48)))
                      (FORALL [DC-68:(0 I)]
                       (IMPLIES
                        (AND
                         (DC-68 ONE)
                         (FORALL [DC-72:I]
                          (IMPLIES
                           (DC-68 DC-72)
                           (DC-68 (S DC-72)))))
                        (DC-68 (S DC-48))))))

L4   (A1 A2 A3    ! (FORALL [DC-26:I]            DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)         (IMPLIES
                      (FORALL [DC-37:(0 I)]
                       (IMPLIES
                        (AND
                         (DC-37 ONE)
                         (FORALL [DC-41:I]
                          (IMPLIES
                           (DC-37 DC-41)
                           (DC-37 (S DC-41)))))
                        (DC-37 DC-26)))
                      (LD1 (S DC-26))))

L3   (A1 A2 A3    ! (FORALL [Y:I]                DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)         (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3    ! (LD1 (S (S (S (S ONE)))))                             OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3    ! (AND (LD1 ONE) (D ONE))                               OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3    ! (LD2 ONE)                        DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3    ! (FORALL [DC-93:I]                                     OTTER: (NIL) (L3 A5)
      A4 A5)         (IMPLIES
                      (AND (LD1 DC-93) (D DC-93))
                      (AND
                       (LD1 (S DC-93))
                       (D (S DC-93)))))

L10  (A1 A2 A3    ! (FORALL [DC-86:I]           DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)         (IMPLIES
                      (AND (LD1 DC-86) (D DC-86))
                      (LD2 (S DC-86))))

L9   (A1 A2 A3    ! (FORALL [Y:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)         (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3    ! (LD2 (S ONE))                                         OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3    ! (FORALL [N:I]                 DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)         (IMPLIES
                      (LD1 N)
                      (FORALL [X:I]
                       (IMPLIES
                        (LD1 X)
                        (LD2 (F N X))))))

LD3  (LD3)        ! (=DEF                                                           LOCAL-DEF
```

```
                     LD3
                     ([N].
                      (FORALL [X:I]
                       (IMPLIES
                        (LD1 X)
                        (LD2 (F N X))))))

L18  (A1 A2 A3    ! (FORALL [DC-151:I]            DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
      A4 A5)         (IMPLIES
                      (LD1 DC-151)
                      (LD2 (F ONE DC-151))))

L15  (A1 A2 A3    ! (LD3 ONE)                       DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)

L30  (L27 A1      ! (FORALL [DC-225:I]            DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L31 LD5)
      A2 A3 A4       (IMPLIES
      A5)             (LD1 DC-225)
                      (LD2 (F (S N1) DC-225))))

L28  (L27 A1      ! (LD3 (S N1))                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L30 LD3)
      A2 A3 A4
      A5)

L26  (A1 A2 A3    ! (IMPLIES (LD3 N1) (LD3 (S N1)))                                IMPI: (L28)
      A4 A5)

L16  (A1 A2 A3    ! (FORALL [N:I]                                           FORALLI: (N1) (L26)
      A4 A5)         (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3    ! (FORALL [DC-123:I]                                  OTTER: (NIL) (L15 L16)
      A4 A5)         (IMPLIES
                      (FORALL [DC-134:(0 I)]
                       (IMPLIES
                        (AND
                         (DC-134 ONE)
                         (FORALL [DC-138:I]
                          (IMPLIES
                           (DC-134 DC-138)
                           (DC-134 (S DC-138)))))
                        (DC-134 DC-123)))
                      (LD3 DC-123)))

L14  (A1 A2 A3    ! (FORALL [N:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)         (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)        ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                LOCAL-DEF

L27  (L27)        ! (LD3 N1)                                                              HYP

L29  (L27)        ! (FORALL [DC-217:I]                DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                     (IMPLIES
                      (LD1 DC-217)
                      (LD2 (F N1 DC-217))))

L23  (A1 A2 A3    ! (LD2 (F ONE ONE))                                    OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3    ! (LD4 ONE)                       DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3    ! (FORALL [DC-201:I]                                    OTTER: (NIL) (L9 A2)
      A4 A5)         (IMPLIES
                      (LD2 (F ONE DC-201))
                      (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3    ! (FORALL [DC-194:I]          DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)         (IMPLIES
                      (LD2 (F ONE DC-194))
                      (LD4 (S DC-194))))

L21  (A1 A2 A3    ! (FORALL [X:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)         (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3    ! (FORALL [DC-165:I]                                  OTTER: (NIL) (L20 L21)
      A4 A5)         (IMPLIES
                      (FORALL [DC-176:(0 I)]
                       (IMPLIES
                        (AND
                         (DC-176 ONE)
                         (FORALL [DC-180:I]
                          (IMPLIES
                           (DC-176 DC-180)
                           (DC-176 (S DC-180)))))
                        (DC-176 DC-165)))
                      (LD4 DC-165)))

L19  (A1 A2 A3    ! (FORALL [X:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)         (IMPLIES (LD1 X) (LD4 X)))

LD5  (LD5)        ! (=DEF LD5 ([X].(LD2 (F (S N1) X))))                             LOCAL-DEF
```

```
L37  (L37)      ! (LD5 X1)                                                         HYP

L39  (L37)      ! (LD2 (F (S N1) X1))                  DEFN-EXPAND-LOCAL-DEF: ((0)) (L37 LD5)

L40  (L37)      ! (AND                                 DEFN-EXPAND-LOCAL-DEF: ((0)) (L39 LD2)
                   (LD1 (F (S N1) X1))
                   (D (F (S N1) X1)))

L42  (L37)      ! (D (F (S N1) X1))                                          ANDE: (L40)

L41  (L37)      ! (LD1 (F (S N1) X1))                                        ANDE: (L40)

L35  (L27 A1    ! (LD2 (F (S N1) ONE))                              OTTER: (NIL) (L12 A1)
      A2 A3 A4
      A5)

L32  (L27 A1    ! (LD5 ONE)                         DEFN-CONTRACT-LOCAL-DEF: ((0)) (L35 LD5)
      A2 A3 A4
      A5)

L44  (L37 L27   ! (LD2 (F N1 (F (S N1) X1)))                       OTTER: (NIL) (L29 L41)
      A1 A2 A3
      A4 A5)
                ...

L43  (L37 L27   ! (LD2 (F (S N1) (S X1)))                                         OPEN
      A1 A2 A3
      A4 A5)

L38  (L37 L27   ! (LD5 (S X1))                      DEFN-CONTRACT-LOCAL-DEF: ((0)) (L43 LD5)
      A1 A2 A3
      A4 A5)

L36  (L27 A1    ! (IMPLIES (LD5 X1) (LD5 (S X1)))                            IMPI: (L38)
      A2 A3 A4
      A5)

L33  (L27 A1    ! (FORALL [X:I]                                      FORALLI: (X1) (L36)
      A2 A3 A4     (IMPLIES (LD5 X) (LD5 (S X))))
      A5)

L34  (L27 A1    ! (FORALL [DC-239:I]                               OTTER: (NIL) (L32 L33)
      A2 A3 A4     (IMPLIES
      A5)           (FORALL [DC-250:(O I)]
                     (IMPLIES
                      (AND
                       (DC-250 ONE)
                       (FORALL [DC-254:I]
                        (IMPLIES
                         (DC-250 DC-254)
                         (DC-250 (S DC-254)))))
                      (DC-250 DC-239)))
                    (LD5 DC-239)))

L31  (L27 A1    ! (FORALL [X:I]                 DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L34 LD1)
      A2 A3 A4     (IMPLIES (LD1 X) (LD5 X)))
      A5)
                ...

CONC (A1 A2 A3  ! (D                                                              OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 4.3.2.7.2.6
OMEGA:  SUPPORT L43 (L44 A3)


L44 (L37 L27   ! (LD2 (F N1 (F (S N1) X1)))              OTTER: (NIL) (L29 L41)
     A1 A2 A3
     A4 A5)

A3  (A3)       ! (FORALL [N:I,X:I]                                        HYP
                  (=
                   (F (S N) (S X))
                   (F N (F (S N) X))))
               ...

L43 (L37 L27   ! (LD2 (F (S N1) (S X1)))                                 OPEN
     A1 A2 A3
     A4 A5)

 ;;; step 4.3.2.7.2.7
OMEGA: CALL-OTTER-ON-NODE L43 ...
...
-------- PROOF --------
....

 ;;; step 6
OMEGA: LEMMA CONC (ld2 (F (S (S (S (S ONE)))) (S (S (S (S ONE))))))
```

```
 ;;; step 6
OMEGA: SUPPORT L45 (L6 L13)


L13 (A1 A2 A3  ! (FORALL [N:I]                  DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
     A4 A5)       (IMPLIES
                   (LD1 N)
                   (FORALL [X:I]
                    (IMPLIES
                     (LD1 X)
                     (LD2 (F N X))))))

L6  (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                           OTTER: (NIL) (L1 L3)
     A4 A5)
               ...

L45 (A1 A2 A3  ! (LD2                                                               OPEN
     A4 A5)       (F
                   (S (S (S (S ONE))))
                   (S (S (S (S ONE))))))

OMEGA:  CALL-OTTER-ON-NODE L45 ...
...
-------- PROOF --------
...

 ;;; step 6
OMEGA:  DEFN-EXPAND-LOCAL-DEF () L45 LD2 (0)
;;;CSM Arbitrary [2]: 0 provers have to be killed

 ;;; step 7
OMEGA:  SUPPORT CONC (L46)


L46  (A1 A2 A3  ! (AND                               DEFN-EXPAND-LOCAL-DEF: ((0)) (L45 LD2)
      A4 A5)       (LD1
                    (F
                     (S (S (S (S ONE))))
                     (S (S (S (S ONE))))))
                   (D
                    (F
                     (S (S (S (S ONE))))
                     (S (S (S (S ONE)))))))
                ...

CONC (A1 A2 A3  ! (D                                                                OPEN
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

 ;;; step 7
OMEGA: CALL-OTTER-ON-NODE ...
...
-------- PROOF --------
...

OMEGA: show-pds
A1   (A1)       ! (FORALL [N:I] (= (F N ONE) (S ONE)))                               HYP

A2   (A2)       ! (FORALL [X:I]                                                      HYP
                   (=
                    (F ONE (S X))
                    (S (S (F ONE X)))))

A3   (A3)       ! (FORALL [N:I,X:I]                                                  HYP
                   (=
                    (F (S N) (S X))
                    (F N (F (S N) X))))

A4   (A4)       ! (D ONE)                                                            HYP

A5   (A5)       ! (FORALL [X:I]                                                      HYP
                   (IMPLIES (D X) (D (S X))))

LD1  (LD1)      ! (=DEF                                                        LOCAL-DEF
                   LD1
                   ([Z].
                    (FORALL [X:(O I)]
                     (IMPLIES
                      (AND
                       (X ONE)
                       (FORALL [Y:I]
                        (IMPLIES (X Y) (X (S Y)))))
                      (X Z)))))

LD2  (LD2)      ! (=DEF LD2 ([Z].(AND (LD1 Z) (D Z))))                         LOCAL-DEF

L2   (A1 A2 A3  ! (FORALL [DC-13:(O I)]                                     OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (AND
                    (DC-13 ONE)
                    (FORALL [DC-17:I]
                     (IMPLIES
                      (DC-13 DC-17)
                      (DC-13 (S DC-17)))))
                   (DC-13 ONE)))

L1   (A1 A2 A3  ! (LD1 ONE)                                   DEFN-CONTRACT-LOCAL-DEF: ((0)) (L2 LD1)
      A4 A5)

L5   (A1 A2 A3  ! (FORALL [DC-48:I]                                                     OTTER: (NIL)
      A4 A5)       (IMPLIES
                    (FORALL [DC-59:(O I)]
                     (IMPLIES
                      (AND
                       (DC-59 ONE)
                       (FORALL [DC-63:I]
                        (IMPLIES
                         (DC-59 DC-63)
                         (DC-59 (S DC-63)))))
                      (DC-59 DC-48)))
                    (FORALL [DC-68:(O I)]
                     (IMPLIES
                      (AND
                       (DC-68 ONE)
                       (FORALL [DC-72:I]
                        (IMPLIES
                         (DC-68 DC-72)
                         (DC-68 (S DC-72)))))
                      (DC-68 (S DC-48))))))

L4   (A1 A2 A3  ! (FORALL [DC-26:I]                     DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L5 LD1)
      A4 A5)       (IMPLIES
                    (FORALL [DC-37:(O I)]
                     (IMPLIES
                      (AND
                       (DC-37 ONE)
                       (FORALL [DC-41:I]
                        (IMPLIES
                         (DC-37 DC-41)
                         (DC-37 (S DC-41)))))
                      (DC-37 DC-26)))
                    (LD1 (S DC-26))))

L3   (A1 A2 A3  ! (FORALL [Y:I]                         DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L4 LD1)
      A4 A5)       (IMPLIES (LD1 Y) (LD1 (S Y))))

L6   (A1 A2 A3  ! (LD1 (S (S (S (S ONE)))))                                     OTTER: (NIL) (L1 L3)
      A4 A5)

L8   (A1 A2 A3  ! (AND (LD1 ONE) (D ONE))                                       OTTER: (NIL) (L1 A4)
      A4 A5)

L7   (A1 A2 A3  ! (LD2 ONE)                                   DEFN-CONTRACT-LOCAL-DEF: ((0)) (L8 LD2)
      A4 A5)

L11  (A1 A2 A3  ! (FORALL [DC-93:I]                                             OTTER: (NIL) (L3 A5)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-93) (D DC-93))
                    (AND
                     (LD1 (S DC-93))
                     (D (S DC-93)))))

L10  (A1 A2 A3  ! (FORALL [DC-86:I]                    DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L11 LD2)
      A4 A5)       (IMPLIES
                    (AND (LD1 DC-86) (D DC-86))
                    (LD2 (S DC-86))))

L9   (A1 A2 A3  ! (FORALL [Y:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L10 LD2)
      A4 A5)       (IMPLIES (LD2 Y) (LD2 (S Y))))

L12  (A1 A2 A3  ! (LD2 (S ONE))                                                 OTTER: (NIL) (L7 L9)
      A4 A5)

L13  (A1 A2 A3  ! (FORALL [N:I]                          DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L14 LD3)
      A4 A5)       (IMPLIES
                    (LD1 N)
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

LD3  (LD3)      ! (=DEF                                                                    LOCAL-DEF
                   LD3
                   ([N].
                    (FORALL [X:I]
                     (IMPLIES
                      (LD1 X)
                      (LD2 (F N X))))))

L18  (A1 A2 A3  ! (FORALL [DC-151:I]                     DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L19 LD4)
```

```
      A4 A5)       (IMPLIES
                    (LD1 DC-151)
                    (LD2 (F ONE DC-151))))

L15  (A1 A2 A3  ! (LD3 ONE)                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L18 LD3)
      A4 A5)

L30  (L27 A1    ! (FORALL [DC-225:I]                     DEFN-EXPAND-LOCAL-DEF: ((1 0 2 0)) (L31 LD5)
      A2 A3 A4     (IMPLIES
      A5)           (LD1 DC-225)
                    (LD2 (F (S N1) DC-225))))

L28  (L27 A1    ! (LD3 (S N1))                               DEFN-CONTRACT-LOCAL-DEF: ((0)) (L30 LD3)
      A2 A3 A4
      A5)

L26  (A1 A2 A3  ! (IMPLIES (LD3 N1) (LD3 (S N1)))                                        IMPI: (L28)
      A4 A5)

L16  (A1 A2 A3  ! (FORALL [N:I]                                                 FORALLI: (N1) (L26)
      A4 A5)       (IMPLIES (LD3 N) (LD3 (S N))))

L17  (A1 A2 A3  ! (FORALL [DC-123:I]                                          OTTER: (NIL) (L15 L16)
      A4 A5)       (IMPLIES
                    (FORALL [DC-134:(O I)]
                     (IMPLIES
                      (AND
                       (DC-134 ONE)
                       (FORALL [DC-138:I]
                        (IMPLIES
                         (DC-134 DC-138)
                         (DC-134 (S DC-138)))))
                      (DC-134 DC-123)))
                    (LD3 DC-123)))

L14  (A1 A2 A3  ! (FORALL [N:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L17 LD1)
      A4 A5)       (IMPLIES (LD1 N) (LD3 N)))

LD4  (LD4)      ! (=DEF LD4 ([X].(LD2 (F ONE X))))                                         LOCAL-DEF

L27  (L27)      ! (LD3 N1)                                                                       HYP

L29  (L27)      ! (FORALL [DC-217:I]                           DEFN-EXPAND-LOCAL-DEF: ((0)) (L27 LD3)
                   (IMPLIES
                    (LD1 DC-217)
                    (LD2 (F N1 DC-217))))

L23  (A1 A2 A3  ! (LD2 (F ONE ONE))                                            OTTER: (NIL) (L12 A1)
      A4 A5)

L20  (A1 A2 A3  ! (LD4 ONE)                                  DEFN-CONTRACT-LOCAL-DEF: ((0)) (L23 LD4)
      A4 A5)

L25  (A1 A2 A3  ! (FORALL [DC-201:I]                                            OTTER: (NIL) (L9 A2)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-201))
                    (LD2 (F ONE (S DC-201)))))

L24  (A1 A2 A3  ! (FORALL [DC-194:I]                   DEFN-CONTRACT-LOCAL-DEF: ((1 0 2 0)) (L25 LD4)
      A4 A5)       (IMPLIES
                    (LD2 (F ONE DC-194))
                    (LD4 (S DC-194))))

L21  (A1 A2 A3  ! (FORALL [X:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L24 LD4)
      A4 A5)       (IMPLIES (LD4 X) (LD4 (S X))))

L22  (A1 A2 A3  ! (FORALL [DC-165:I]                                          OTTER: (NIL) (L20 L21)
      A4 A5)       (IMPLIES
                    (FORALL [DC-176:(O I)]
                     (IMPLIES
                      (AND
                       (DC-176 ONE)
                       (FORALL [DC-180:I]
                        (IMPLIES
                         (DC-176 DC-180)
                         (DC-176 (S DC-180)))))
                      (DC-176 DC-165)))
                    (LD4 DC-165)))

L19  (A1 A2 A3  ! (FORALL [X:I]                        DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L22 LD1)
      A4 A5)       (IMPLIES (LD1 X) (LD4 X)))

LD5  (LD5)      ! (=DEF LD5 ([X].(LD2 (F (S N1) X))))                                      LOCAL-DEF

L37  (L37)      ! (LD5 X1)                                                                       HYP

L39  (L37)      ! (LD2 (F (S N1) X1))                          DEFN-EXPAND-LOCAL-DEF: ((0)) (L37 LD5)

L40  (L37)      ! (AND                                         DEFN-EXPAND-LOCAL-DEF: ((0)) (L39 LD2)
                   (LD1 (F (S N1) X1))
                   (D (F (S N1) X1)))
```

```
L45  (A1 A2 A3  ! (LD2                                                       OTTER: (NIL) (L6 L13)
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

L46  (A1 A2 A3  ! (AND                              DEFN-EXPAND-LOCAL-DEF: ((0)) (L45 LD2)
      A4 A5)       (LD1
                    (F
                     (S (S (S (S ONE))))
                     (S (S (S (S ONE))))))
                   (D
                    (F
                     (S (S (S (S ONE))))
                     (S (S (S (S ONE)))))))

L42  (L37)      ! (D (F (S N1) X1))                                                ANDE: (L40)

L41  (L37)      ! (LD1 (F (S N1) X1))                                              ANDE: (L40)

L35  (L27 A1    ! (LD2 (F (S N1) ONE))                                   OTTER: (NIL) (L12 A1)
      A2 A3 A4
      A5)

L32  (L27 A1    ! (LD5 ONE)                       DEFN-CONTRACT-LOCAL-DEF: ((0)) (L35 LD5)
      A2 A3 A4
      A5)

L44  (L37 L27   ! (LD2 (F N1 (F (S N1) X1)))                            OTTER: (NIL) (L29 L41)
      A1 A2 A3
      A4 A5)

L43  (L37 L27   ! (LD2 (F (S N1) (S X1)))                                OTTER: (NIL) (A3 L44)
      A1 A2 A3
      A4 A5)

L38  (L37 L27   ! (LD5 (S X1))                    DEFN-CONTRACT-LOCAL-DEF: ((0)) (L43 LD5)
      A1 A2 A3
      A4 A5)

L36  (L27 A1    ! (IMPLIES (LD5 X1) (LD5 (S X1)))                                  IMPI: (L38)
      A2 A3 A4
      A5)

L33  (L27 A1    ! (FORALL [X:I]                                             FORALLI: (X1) (L36)
      A2 A3 A4     (IMPLIES (LD5 X) (LD5 (S X))))
      A5)

L34  (L27 A1    ! (FORALL [DC-239:I]                                    OTTER: (NIL) (L32 L33)
      A2 A3 A4     (IMPLIES
      A5)           (FORALL [DC-250:(O I)]
                     (IMPLIES
                      (AND
                       (DC-250 ONE)
                       (FORALL [DC-254:I]
                        (IMPLIES
                         (DC-250 DC-254)
                         (DC-250 (S DC-254)))))
                      (DC-250 DC-239)))
                    (LD5 DC-239)))

L31  (L27 A1    ! (FORALL [X:I]               DEFN-CONTRACT-LOCAL-DEF: ((1 0 1 0)) (L34 LD1)
      A2 A3 A4     (IMPLIES (LD1 X) (LD5 X)))
      A5)

CONC (A1 A2 A3  ! (D                                                         OTTER: (NIL) (L46)
      A4 A5)       (F
                    (S (S (S (S ONE))))
                    (S (S (S (S ONE))))))

OMEGA:
```

Mizar Attributes: A Technique to Encode Mathematical Knowledge into Type Systems

Christoph Schwarzweller

Department of Computer Science
University of Gdańsk
ul. Wita Stwosza 57, 80-952 Gdańsk, Poland
schwarzw@math.univ.gda.pl

Abstract. At first glance Mizar attributes look like unary predicates over mathematical objects enabling a more natural writing and reading. Attributes in Mizar, however, serve additional, more important purposes concerning typing of mathematical objects: Using attributes not only new (sub)types can be introduced, but also the user can characterize further relations between types and in this way make available existing notations for new objects. Thereby it should be stressed that these type relations can stand for elaborated mathematical theorems.

This paper describes the properties and benefits of Mizar attributes from a user's perspective. We comprehend the development of Mizar attributes, and give examples highlighting their use — essentially in the area of algebra. Concluding we discuss their impact on building mathematical repositories.

1 Introduction

After more than 30 years the Mizar system [6] is still widely-used and under active development. This is rather seldom for a mechanized reasoning system. The first presentation of Mizar in 1973 and the first write-up of 1975, that was published in 1977 [14], however, did not even mention proof checking as the main goal. At this stage Mizar basically was a language supposed to assist authors in the mathematical editing process. Therefore, though the language was meant to be understood and processed by a computer, great emphasis was placed on the linguistical component of the language. The point of reference here was the question how mathematicians write and present results in their papers. It is this fact, that from the very beginning an appealing formal language for mathematics was taken into consideration, that distinguishes Mizar from other systems in the area.

Later, of course, when proof checking for Mizar texts was developed, other items became equally important and influenced the further design of the Mizar language. The linguistical roots are still visible anyway and from our point of view provide the basis of Mizar's resistance against the course of time. This was again approved

ISBN 978-83-7431-128-1 ISSN 0860–150X

by the success in the last couple of years, in which Mizar has evolved to one of the leading systems in the new area of mathematical knowledge management.

In this paper we appreciate the above mentioned development by focusing on Mizar attributes. In fact Mizar attributes have changed over the time from a poorly syntactical device to a powerful technique for defining and manipulating mathematical types in a user-friendly way — a topic that gains more and more importance, especially in mathematical knowledge management. We first review the development of Mizar attributes from the beginning and describe their present role in defining Mizar types. Then we illustrate in detail, how using attributes types can be introduced, refined and extended. The main point here is — in the spirit of mathematical knowledge management — the generation of a user-friendly environment, in which the provided knowledge can be reused, even in modified situations. In doing so, we concentrate on examples from the area of algebra. Finally, we discuss the impact of Mizar attributes on building large mathematical repositories.

2 Some History: Predicates, Attributes and Types

In the present version of the Mizar language predicates describe properties of mathematical objects, and so do attributes. The only difference is that while predicates may have any number of arguments, attributes must have exactly one. So, for example that a number x is even can be described by an attribute (or by a predicate), while that x is a divisor of y cannot. This was not always so. Although types were present from the very beginning, the first versions of Mizar employed predicates only. An attributive format for writing predicates was introduced in 1981 for the language Mizar-2 [7]. The reason was, as already indicated, a linguistical one. The new format enabled authors to write predicates in a more familiar form, such as

```
x is even
x is divisor of y.
```

Note, that the second predicate has two arguments, thus is not an attribute in today's sense, though here written in its attributive form.

This idea was further developed and in 1983 a number of extensions was realized in Mizar-HPF [7]. So, for example, more than one argument could be placed in front of the predicate's name. Also the keyword `is` could be replaced by `are`, if this seemed appropriate. In addition a more natural format for the negation of predicates using the keyword `not` was implemented. This more general attributive format lead to natural formulations such as

```
x, y are associated
x is not even.
```

Note, however, that all these phrases are just dealing with the natural writing of predicates and do not involve any kind of typing.

Mizar-4 [7] in 1986 saw the introduction of a special language construct, the so-called mode definition, for defining types (both with or without parameters),

that is still used today. Note that in this setting the second predicate from above should be rather defined as a mode `divisor` with one parameter. Also the concept of redefinitions was implemented, which allowed for redefining the type of a functor in case its arguments are more specific than in the original definition. For example the union of two finite sets could be redefined to be a finite set again. Note, however, that the type of finite sets at this stage was not constructed from the type set with the help of an additional adjective finite.

The development that started in Mizar-4 has been carried on and the present version of Mizar provides a very flexible type system in which attributes, or more precisely the adjectives constructed by attribute definitions, play a major role. Attribute definitions were separated from predicate definitions by the keyword `attr`, so that for example the definition of `even` now looks like

```
definition
let x be Integer;
attr x is even means 2 divides x;
end;
```

The negation of the defined attribute, now written using the keyword `non`, is automatically generated, so that the above definition introduces both adjectives `even` and `non even`. The negation of predicates in general, however, has not survived. The actual power of attributes for typing mathematical objects, however, rests on the rather obvious observation, that an attribute basically defines a subset of the type given in the definition. This allowed for the following extension of Mizar types. Radix types, the basic types given by mode definitions, can be equipped with adjectives in this way establishing a new subtype of the original radix type. Of course the adjective considered must be defined for this radix type. So, for example the adjective `even` generated by the above attribute definition together with the radix type `Integer` can be used to define the new type `even Integer` describing all objects of type `Integer`, that in addition fulfill the attribute `even`, in other words all even integers. In general (see [1] or [16]) Mizar types T have the form

$$T \;=\; \alpha_1 \ldots \alpha_n \; R \text{ of } t_1, \ldots t_k$$

where R is a radix type, the α_i are adjectives of R, the t_i are terms describing the parameters of type R and both n and k may be 0. Note that a type T_1 is a subtype of a type T_2, if T_1's radix type is a subtype of T_2's radix type and T_2's adjectives are a subset of T_1's adjectives.

This is a very powerful scheme to describe types of mathematical objects. On the other hand in this setting it is very easy to define empty types such as for example the type

```
empty infinite set.
```

To prevent Mizar users from wasting time with non-existing mathematical objects, they are obliged to prove the existence of an object with the newly defined type. As a consequence empty types are not allowed in Mizar. The power of typing with adjectives, however, stems from the fact that adjectives can be also employed after the "main" definition of a type. More adjectives can be added to a type later and even relations between set of adjectives can be stated (and again must be proven), in the sense that one set of adjectives logically implies another one. This effects Mizar's type system in that new subtype relations are introduced not comprised by the original definition. The consequence is that definitions, notations and theorems already elaborated for a mathematical type are now inherited to the new established subtype. We shall illustrate these techniques in detail in the next section with examples from the area of algebra.

3 Attributes and Registrations

In the following we show how Mizar attributes can be used to define algebraic types and subtypes (see also [12]). The main goal is to illustrate how in this context using attributes enables the reuse of notations and theorems already defined and proven for other types: By formulating (and proving) registrations the user can show that a mathematical object fulfills more adjectives than given in its definition, that is the object indeed have a more elaborated type than the one originally given. Even more, if the type is parameterized, registrations can be used to show that a more specific parameter type implies a more specific resulting type. In either case the registration "enriches" the Mizar checker in the sense that the checker automatically infers the new type given by the registration, which leads to the automated reuse of notations and theorems mentioned above. Note that using such registrations no original definition has to be modified in order to change or refine an object's type.

3.1 Existential Registrations

In defining algebraic types the radix type describes the carriers and operations of the algebraic structure. This is carried out in a so-called structure definition. The result is a new type describing mathematical objects with (at least) the components given in the definition. Thus, for example [4,15], the backbone structure for groups is given by the following structure definition LoopStr providing a carrier, a binary operation over the carrier and a distinguished element of the carrier. Note, that at this stage no further properties of the operation such as associativity or commutativity are required.

```
definition
struct (ZeroStr) LoopStr
  (# carrier -> set,
     add -> BinOp of the carrier,
     Zero -> Element of the carrier #);
end;
```

For a structure type Mizar automatically provides selectors allowing to access the individual components of the object, for example `the add of L` or `the Zero of L`, if `L` has type `LoopStr`. It is straightforward in Mizar to introduce the usual mathematical notation for algebraic operations, in the case of groups `x + y` for `(the add of L).(x,y)` and `0.L` for `the Zero of L`. Note that, in contrast to `0.L`, the binary operation `x + y` need not explicitly address the structure parameter `L`. The reason is that the parameter can be inferred from the type of the arguments `x` and `y`, which is `Element of the carrier of L` or shorter `Element of L`. Using these notations we can now formulate the usual properties of groups in a straightforward manner as attribute definitions. Note that the following attributes describe unary predicates over a LoopStr L.

```
definition
let L be non empty LoopStr;
attr L is add-associative means
  for x,y,z being Element of L holds (x + y) + z = x + (y + z);
attr L is right_zeroed means
  for x being Element of L holds x + 0.L = x;
attr L is right_complementable means
  for x being Element of L ex y being Element of L st x + y = 0.L;
end;
```

A group is now obviously a `LoopStr L`, that fulfills all three just defined attributes. This of course could have been also be defined using ordinary predicates such as `is_add-associative` etc. However, though describing group properties — and thus enabling to prove theorems for groups — predicates cannot be used to introduce the Mizar type `Group`. With the attributes from above we can define the attributed type `Group` in an existential registration as follows.

```
registration
cluster add-associative right_zeroed right_complementable
        (non empty LoopStr);
end;

definition
mode Group is add-associative right_zeroed right_complementable
              (non empty LoopStr);
end;
```

In an existential registration we summarize the attributes necessary to define a mathematical object. Note that the radix type `non empty LoopStr` to be refined must be explicitly given in the cluster. Because Mizar does not allow for empty types such a registration demands an existence proof, in which it has to be shown that there exists a mathematical object of radix type `non empty LoopStr` fulfilling the three mentioned adjectives. After that the type `Group` can be introduced as a synonym for the registered object. This, however, is not really necessary, because `add-associative right_zeroed right_complementable (non empty LoopStr)` is already a type for itself, so in fact `Group` is no more than an abbreviation.

For objects of type `Group` we can now formulate and prove theorems, that is for proving properties we can use exactly the definitions of the attributes constituting the type, and nothing more. Consequently, theorems proven this way are valid in all groups. For example, we can show that in groups the neutral element is unique:

```
theorem G1:
for G being Group, x,y being Element of G
holds x + y = x implies y = 0.G;
```

Now, groups are just the most general case. There are numerous more specific groups such as for example Abelian or solvable groups, which in a manner of speaking are refinements of the original group definition. The use of adjectives supports not only such refinements of mathematical types, but also guarantuees that theorems for general groups like the example theorem `G1` are automatically applicable to the refined types.

For example, to define Abelian groups, we simply introduce another attribute describing the additional property of commutativity and show — again in an existential registration — that there exists a group that fulfills this property. Afterwards the Mizar type `AbGroup` describing Abelian groups can be easily defined.

```
definition
let L be non empty LoopStr;
attr L is Abelian means
  for x,y being Element of L holds x + y = y + x;
end;

registration
cluster Abelian Group;
end;

definition
mode AbGroup is Abelian Group;
end;
```

The key point here is that `AbGroup` becomes a subtype of `Group`. This is due to the fact that both types share the same radix type and the adjectives of `Group` are a subset of the adjectives of `AbGroup`, or the other way round the type `AbGroup` widens to the type `Group`. As a consequence theorem `G1` from above works also for the newly defined type `AbGroup`:

```
for G being AbGroup, x,y being Element of G
holds x + y = x implies y = 0.G by G1;
```

Note that for introducing Abelian groups only one attribute definition (and an existence proof) were necessary, although afterwards Mizar provides the user of Abelian groups with a vast number of theorems for this type — all theorems proven for general groups.

There is, however, more to it than that [4]: We can also define new subtypes by extending the underlying structure, that is by refining the radix type. Again the mechanism of adjectives enables the natural reuse of already proven theorems. Fields, for example, are based on a structure that extends the one of groups by another binary operation and another element of the carrier. This can be directly expressed in Mizar by the following structure definition.

```
definition
struct (LoopStr, multLoopStr_0) doubleLoopStr
  (# carrier -> set,
     add, mult -> BinOp of the carrier,
     unity, Zero -> Element of the carrier #);
end;
```

The structure type `doubleLoopStr` in fact is a merge of two other ones: `LoopStr` and `multLoopStr` as can be seen in the definition. `doubleLoopStr` provides the components of both of them — further components can be introduced, but are not necessary in the case of fields. The main effect of this definition — besides the introduction of a field structure — is that the type `doubleLoopStr` becomes a subtype of both mentioned structure types `LoopStr` and `multLoopStr`. The algebraic type `Field` is now defined analogously to groups: Attributes describing field properties are introduced and after the existential registration we can introduce the corresponding Mizar type.

```
definition
mode Field is add-associative right_zeroed right_complementable
     Abelian commutative associative left_unital distributive
     Field-like non degenerated (non empty doubleLoopStr);
end;
```

Note, that the three attributes `add-associative` `right_complementable` and `right_zeroed` defined for `LoopStr` need not to be introduced for fields, that is for the type `doubleLoopStr`, again: Because the radix type `doubleLoopStr` widens to the type `LoopStr` they are available and can be reused.

Now every field is in particular a group with respect to addition, thus group theorems apply to fields also. As a result of the Mizar type system — and the definitions of groups and fields based on attributes — this is directly mirrored and feasible: The radix type of `Field` is a subtype of the radix type of `Group` and the adjectives of `Group` are a subset of the adjectives of `Field`. This implies that `Field` is a subtype of `Group`, and hence theorems for type `Group` are valid for type `Field` also. For our example theorem `G1` from above we get

```
for F being Field, x,y being Element of F
holds x + y = x implies y = 0.F by G1;
```

3.2 Functional Registrations

When defining algebraic structures functional registrations of attributes are used to introduce properties of concrete structures, such as integers or polynomials. Of course these properties can also be shown in theorems, proving for example that the integers constitute a ring. The main effect of such a functional registration, however, is again that the type of the mathematical object is extended. In this way not only theorems already proven for the refined type become available, but also definitions and notations introduced for mathematical objects of this type.

Consider the integers as an example [13]. In Mizar the integers are introduced as a doubleLoopStr providing the set of integers, addition and multiplication of integers as well as the numbers 0 and 1. The operations addint and mulint are defined as binary operations over the set of integers INT and then identified with the corresponding components of doubleLoopStr in a function definition. Note that the function In changes the type of 0 and 1 to Element of INT — the proper type according to the definition of doubleLoopStr.

```
definition
func INT.Ring -> non empty doubleLoopStr equals
  doubleLoopStr(#INT,addint,multint,In(1,INT),In(0,INT)#);
end;
```

With this definition the following functional registration shows not only that INT.Ring fulfills the three attributes constituting a group, but also that INT.Ring is of type Group — remember that the type doubleLoopStr widens to the type LoopStr. Of course a (coherence) proof, that the three attributes are fulfilled by the object INT.Ring, is necessary.

```
registration
cluster INT.Ring -> add-associative right_zeroed right_complementable;
end;
```

One consequence of changing the type of INT.Ring is that our theorem G1 from section 3.1 now applies to INT.Ring also. Note that the "group" parameter G from theorem G1 has vanished, that is the following is a pure integer version of the general group theorem G1.

```
for x,y being Element of INT.Ring
holds x + y = x implies y = 0.(INT.Ring) by G1;
```

Another effect is that notations defined for the type Group are now available for INT.Ring, that is for integers. For example, the unary and binary operations - are defined for general groups the usual way: -x is the element of a group G such that x + -x = 0.G and x - y equals x + (-y). Now INT.Ring having type Group there is no need to define these operations again for integers, they are just inherited from Group.

Of course on could argue that this is all obvious and that it could already be stated in the definition of INT.Ring that this object constitutes a ring. So there is no need to employ adjectives here. This is right, however, ignores the fact that later there can occur situations in which additional properties of an object such as INT.Ring are necessary to go on working. With the rather naive solution from above this would imply changing the definition in order to adopt the object's type according to the additional properties — rather unpleasant, especially in a system with a large number of users. Using attributes, in contrast, we can easily include the additional properties into Mizar's type system — without changing the original definition.

A typical example are Euclidean domains in which greatest common divisors can be easily computed using degree functions. The property of being Euclidian is straightforwardly introduced as usual in an attribute definition. For objects of type doubleLoopStr fulfilling this attribute a degree function can be defined as follows. Note that Mizar expects an existence proof in such a definition, therefore the attribute Euclidian in the definition is essential.

```
definition
let E be Euclidian (non empty doubleLoopStr);
mode DegreeFunction of E -> Function of the carrier of E,NAT means
  (for a,b being Element of E st b <> 0.E holds
    (ex q,r being Element of E
     st (a = q * b + r & (r = 0.E or it.r < it.b))));
end;
```

Consequently DegreeFunctions exist only for objects of type Euclidian (non empty doubleLoopStr). The type of INT.Ring — using the above registrations — unfortunately does not widen to this type, although the radix types are identical. If we want to work with the integers as a Euclidean domain, we can, however, easily repair this "defect" — without changing the original definition of INT.Ring: Stating an additional registration for INT.Ring, in which we prove that INT.Ring fulfills the definition of Euclidian.

```
registration
cluster INT.Ring -> Euclidian;
end;
```

In this way, the fact that the ring of integers is Euclidean is not only proven, but also encoded into Mizar's type system with the consequence that results for abstract Euclidean domains can be easily reused for integers, among others that there exists a DegreeFunction of INT.Ring.

The same can be done for algebraic structures with parameters — even if the additional property such as Euclidian is only valid if the parameter has additional properties also. Let us consider polynomial rings as an example [10]. Polynomial rings are defined analogously to the ring of integers as a function Polynom-Ring(n,L) with result type doubleLoopStr. Here the parameter n gives the number of indeterminates, the parameter L is the ring of coefficients. Then a number of functional registrations shows that Polynom-Ring(n,L) is of type Ring. Of course, if L is commutative, so is the resulting polynomial ring. Changing in this case the type of Polynom-Ring(n,L) to be in addition commutative is straightforward using a functional registration.

```
registration
let n be Ordinal,
    L be commutative Ring;
cluster Polynom-Ring(n,L) -> commutative;
end;
```

After this registration the type of the object Polynom-Ring(n,L) remains Ring if the parameter ring L is not commutative. If, however, L is commutative, that is if L comes with the type `commutative Ring`, then the type of `Polynom-Ring(n,L)` inferred by Mizar is `commutative Ring` also.

We close this section with an example concerning again Euclidean rings. Polynomial rings are Euclidean, if the coefficient ring is in fact a field and the number of indeterminates is 1, so that in this case there exist degree functions for polynomials. The type `DegreeFunction of E`, however, only exists if E fulfills the attribute `Euclidian`. This can be easily expressed by the following registration.

```
registration
let F be Field;
cluster Polynom-Ring(1,F) -> Euclidian;
end;
```

Again we have two results: The mathematical statement is formally proven and the type of `Polynom-Ring(1,F)` for fields F is changed to `Euclidian Ring`, so that in this case the type `DegreeFunction of Polynom-Ring(1,F)` is available.

3.3 Conditional Registrations

The last kind of registration is used to deal with the types of classes of algebraic structures. This is of particular interest, if the adjectives stated in an object's definition imply other adjectives. In this case, conditional registrations allow to enrich the original type, so that the implied adjectives are automatically inferred. Of course again a (coherence) proof of the implication is necessary in Mizar.

For example it is obvious that in a commutative ring both left ideals and right ideals are in fact two-sided ideals. Proving these statements as theorems does not enrich the object's type: Notations defined for two-sided ideals are not available for left or right ideals in commutative rings — even after proving the theorem showing that they are (theoretically) available. Stating the proofs in the following conditional registration the first cluster changes the type `left-ideal (non empty Subset of R)` to `left-ideal right-ideal (non empty Subset of R)` if R is commutative. The second cluster does the analogous for right ideals.

```
registration
let R be commutative Ring;
cluster left-ideal -> right-ideal (non empty Subset of R);
cluster right-ideal -> left-ideal (non empty Subset of R);
end;
```

Thus after this registration in commutative rings objects of type `left-` or `right-ideal` automatically have type (two-sided) `ideal` also. Note that to establish a subset of a commutative ring as an ideal it is then sufficient to show that the subset is a left (or a right) ideal.

For a more involved example we return to groups as described in section 3.1. There we defined groups G using the attribute `right_zeroed` describing the axiom `x + 0.G = x`. Consequently the following statement is automatically accepted by the Mizar checker.

```
for G being Group holds G is right_zeroed;
```

Now, in a group G we also have `0.G + x = x`, that is G is `left_zeroed` also. The corresponding statement, however, is not automatically accepted by the Mizar checker, because the type `Group` does not include the corresponding attribute `left_zeroed`. Of course, one could add this property to the group definition. But this would imply, that in the definition is more stated than necessary, rather unaesthetic from a mathematical point of view. The solution again is a conditional registration showing that the attributes used to define the type `Group` imply the attribute `left_zeroed`. Note that the antecedent of the cluster is empty, because all necessary attributes are given by the type `Group`. After the registration the Mizar checker automatically accepts the fact that an object of type `Group` fulfills the attribute `left_zeroed`.

```
registration
cluster -> left_zeroed Group;
end;
```

```
for G being Group holds G is left_zeroed;
```

The same technique can again be applied to parameterized types. We illustrate this by introducing subgroups. A subgroup is a subset of a group that is itself a group. This can be easily described by the following mode definition. Note that it is necessary to explicitly express that the addition and the neutral element are inherited from the given group G.

```
definition let G be Group;
mode Subgroup of G -> Group means
  the carrier of it c= the carrier of G &
  the add of it = (the add of G)||the carrier of it &
  the Zero of it = the Zero of G;
end;
```

Now, as the above definition states, a subgroup of a group is in particular of type `Group`. Hence, because — according to section 3.1 — Abelian groups are in particular groups, the type `Subgroup of G` where G is an Abelian group exists and the following statement about Abelian groups is obvious for the Mizar checker.

```
for G being AbGroup, H being Subgroup of G holds H is add-associative;
```

Of course a subgroup of an Abelian group is again Abelian. The corresponding statement, however, is not automatically accepted by the Mizar checker. The reason is obvious: According to our definition of subgroups the type of a subgroup of G is `Group`, even if G is of type `Abelian Group`. Therefore the Mizar checker cannot infer, that in this case the attribute `Abelian` is also fulfilled. In the following conditional registration we restrict the type of the parameter G to `Abelian Group` and show that then each object of type `Subgroup of G` fulfills the attribute `Abelian`. As usual this registration also enriches the Mizar type checker, so that the type for subgroups of Abelian groups G is changed into `Abelian Subgroup of G`.

```
registration
let G be Abelian Group;
cluster -> Abelian Subgroup of G;
end;
```

As a consequence the fact that subgroups of Abelian subgroups are Abelian is automatically accepted. Even more, because the type `Field` is a subtype of `AbGroup` — note that the defining cluster for `Field` in section 3.1 includes the attribute `Abelian` — this is also true for this type; and in fact for every other algebraic type widening to the type `AbGroup` such as for example `Ring`. Hence all the following statements are now obvious for the Mizar checker.

```
for G being AbGroup, H being Subgroup of G holds H is Abelian;

for F being Field, H being Subgroup of F holds H is Abelian;

for R being Ring, H being Subgroup of R holds H is Abelian;
```

We close this section with a "type version" of a famous result by J. Wedderburn. Skew fields or division rings [8] differ from fields only in that their multiplication is not required to be commutative. Finite skew fields however — as J. Wedderburn showed — are always commutative. This statement can be formulated as a conditional registration as follows.

```
registration
cluster finite -> commutative Skew-Field;
end;
```

Again this fact could just as well be stated as a theorem. But note that the registration in fact changes the type `finite Skew-Field` to `Field`, because now all attributes of the `Field`'s definition are fulfilled. Hence, due to the last registration in section 3.2, polynomial rings over finite skew fields are Euclidean. As a consequence the type `DegreeFunction of Polynom-Ring(1,F)` for finite skew fields `F` is automatically available, which would not be true if Wedderburn's theorem would have been stated as an "ordinary" Mizar theorem.

4 Conclusions

As we have seen Mizar attributes have developed from a pure linguistical phrase into a powerful technique for defining and manipulating types. Though basically all the mathematical definitions and statements in section 3 can be expressed without using attributes and registrations, there are good reasons not to do so, notably in mathematical repositories.

Firstly, we consider the use of attributes in mathematical repositories user-friendly. Attributes allow to enrich algebraic structures with additional properties without changing the original definition. In this way attributes also support the reuse of already proven theorems: Refining a type by adding additional attributes does not exclude the use of theorems formulated for the original type. In addition new type relations can be established. Secondly, typing with attributes shortens the number of theorems to be stored in a mathematical repository. Note that the example theorem from section 3.1 need to be stored only once for the type `Group`, although we presented versions for four different algebraic structures; a number that could be easily enlarged. Thirdly, attributes allow to handle algebraic structures in a mechanized reasoning system similarly to the way mathematicians do. Mathematicians change their view on an algebraic structure if this seems convenient. For example group theorems are applied to fields without thinking about the type coercion that actually takes place. As we have seen Mizar attributes not only enable such a dealing with algebraic structures but in addition allow for automating even ambitious theorems in a mathematical repository.

We believe that especially the linguistical motivation for introducing attributes to the Mizar language permitted in later years the development of a typing technique that is extremely well-suited for the development of mathematical repositories. So, we wait and hope for further "linguistical inspiration" from Mizar. In this spirit

Andrzeju, wszystkiego najlepszego!

References

1. G. Bancerek, *On the Structure of Mizar Types*; Electronic Notes in Theoretical Computer Science, vol. 85(7), Elsevier, 2003.
2. N.G. de Bruijn, *The Mathematical Vernacular, a language for mathematics with typed sets,*; in P. Dybjer et al. (eds.), Proc. of the Workshop on Programming Languages, Marstrand, Sweden, 1987.
3. J.H. Davenport, *MKM from book to computer: a case study*; in: A. Asperti, B. Buchberger, and J. Davenport (eds.), Proc. of MKM 2003, Lecture Notes in Computer Science 2594, pp. 17–29, 2003.
4. E. Kusak, W. Leończuk and M. Muzalewski, *Abelian Groups, Fields and Vector Spaces*; Formalized Mathematics, vol. 1(2), pp. 335-342, 1990.
5. F. Kamareddine and R. Nederpelt, *A refinement of de Bruijn's formal language of mathematics*; Journal of Logic, Language and Information, vol. 13(3), pp. 287–340, 2004.
6. The Mizar Homepage, `www.mizar.org`.
7. R. Matuszewski and P. Rudnicki, *Mizar: The First 30 Years*; Mechanized Mathematics and Its Applications, vol. 4(1), pp. 3–24, 2005. `http://mizar.org/people/romat/MatRud2005.pdf`
8. M. Muzalewski, *Construction of Rings and Left-, Right-, and Bi-Modules over a Ring*; Formalized Mathematics, vol. 2(1), pp. 3-11, 1991.
9. P. Rudnicki and A. Trybulec, *Mathematical Knowledge Management in Mizar*; in: B. Buchberger, O. Caprotti (eds.), Proc. of MKM 2001, Linz, Austria, 2001.
10. P. Rudnicki and A. Trybulec, *Multivariate Polynomials with Arbitrary Numbers of Variables*; Formalized Mathematics, vol. 9(1), pp. 95-110, 2001.
11. P. Rudnicki and A. Trybulec, *On the integrity of a repository of formalized mathematics*; in: A. Asperti, B. Buchberger, and J. Davenport (eds.), Proc. of MKM 2003, Lecture Notes in Computer Science 2594, pp. 162–174, 2003.

12. P. Rudnicki, A. Trybulec and C. Schwarzweller, *Commutative Algebra in the Mizar System*; Journal of Symbolic Computation, vol. 32(1/2), pp. 143-169, 2001.
13. C. Schwarzweller, *The Ring of Integers, Euclidean Rings and Modulo Integers*; Formalized Mathematics, vol. 8(1), pp. 29-34, 1999.
14. A. Trybulec, *Informationslogische Sprache Mizar*; Dokumentation-Information, Heft 33, Ilmenau, 1977.
15. W.A. Trybulec, *Vectors in Real Linear Space*; Formalized Mathematics, vol. 1(2), pp. 291-296, 1990.
16. F. Wiedijk, *Writing a Mizar Article in Nine Easy Steps*; available from `http://www.mizar.org/project/bibliography.html`.