Didier Bert
Jonathan P. Bowen
Martin C. Henson
Ken Robinson (Eds.)

LNCS 2272

ZB 2002: Formal Specification and Development in Z and B

2nd International Conference of B and Z Users
Grenoble, France, January 2002, France
Proceedings

ZB 2002

Springer

Didier Bert   Jonathan P. Bowen
Martin C. Henson   Ken Robinson (Eds.)

ZB 2002: Formal Specification and Development in Z and B

2nd International Conference of B and Z Users
Grenoble, France, January 23-25, 2002
Proceedings

Springer

Volume Editors

Didier Bert
CNRS, Laboratoire LSR, IMAG
681, rue de la Passerelle
38402 Saint Martin d'Hères Cedex, France
E-mail: didier.bert@imag.fr

Jonathan P. Bowen
South Bank University, SCISM, Centre for Applied Fromal Methods
Borough Road, London SE1 0AA, UK
E-mail: jonathan.bowen@sbu.ac.uk

Martin C. Henson
University of Essex, Department of Computer Science
Wivenhoe Park, Colchester CO4 3SQ, UK
E-mail: hensm@essex.ac.uk

Ken Robinson
The University of New South Wales, UNSW
CAESER, The School of Computer Science and Engineering
Sydney NSW 2052, Australia
E-mail: k.robinson@unsw.edu.au

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

ZB 2002: formal specification and development in Z and B : proceedings / 2nd International Conference of B and Z Users, Grenoble, France, January 23 - 25, 2002. Didier Bert ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2272)
ISBN 3-540-43166-7

CR Subject Classification (1998): D.2.1, D.2.2, D.2.4, F.3.1, F.4.2, F.4.3

ISSN 0302-9743
ISBN 3-540-43166-7 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10846165 06/3142 5 4 3 2 1 0

Preface

These proceedings record the papers presented at the second International Conference of B and Z Users (ZB 2002), held on 23–25 January 2002 in the city of Grenoble in the heart of the French Alps. This conference built on the success of the first conference in this series, ZB 2000, held at the University of York in the UK. The location of ZB 2002 in Grenoble reflects the important work in the area of formal methods carried out at the *Laboratoire Logiciels Systèmes Réseaux* within the *Institut d'Informatique et Mathématiques Appliquées de Grenoble* (LSR-IMAG), especially involving the B method.

B and Z are two important formal methods that share a common conceptual origin; each are leading approaches applied in industry and academia for the specification and development (using formal refinement) of computer-based systems. At ZB 2002 the B and Z communities were brought together to hold a second joint conference that simultaneously incorporated the 13th International Z User Meeting and the 4th International Conference on the B method. Although organized logistically as an integral event, editorial control of the joint conference remained vested in two separate but cooperating program committees that respectively determined its B and Z content, but in a coordinated manner.

All the submitted papers in these proceedings were peer reviewed by at least three reviewers drawn from the B or Z committee depending on the subject matter of the paper. Reviewing and initial selection were undertaken electronically. The Z committee met at South Bank University in London on 27th September 2001 to determine the final selection of Z papers. The B committee met on the morning of 28th September 2001 at the Conservatoire National des Arts et Métiers (CNAM) in Paris to select B papers. A joint committee meeting was held at the same location in the afternoon to resolve the final paper selection and to draft a program for the conference. Sergiy Vilkomir of the Centre for Applied Formal Methods (CAFM) at South Bank University aided in the local organization of the Z meeting. Véronique Viguié Donzeau-Gouge helped in the organization of the meetings at CNAM.

The conference featured a range of contributions by distinguished invited speakers drawn from both industry and academia. The invited speakers addressed significant recent industrial applications of formal methods, as well as important academic advances serving to enhance their potency and widen their applicability. Our invited speakers for ZB 2002 were drawn from Finland, France, and Canada. Ralph-Johan Back, Professor of Computer Science at Åbo Akademi University and Director of the Turku Centre for Computer Science (TUCS) has made important contributions in the development of the refinement calculus, influential and relevant to many formal methods, including B and Z. Pierre Chartier of RATP (Régie Autonome des Transports Parisiens), central in rail transport for Paris, is a leading expert in the industrial application of the B method. Eric C.R. Hehner, Professor of Computer Science at the University of Toronto, has always presented his novel ideas for formal methods using an elegant simplicity.

Besides its formal sessions, the conference included tool demonstrations, exhibitions, and tutorials. In particular, a workshop on *Refinement of Critical Systems: Methods, Tools, and Experience* (RCS 2002) was organized on 22 January 2001 with the support of the EU IST-RTD Project *MATISSE: Methodologies and Associated Technologies for Industrial Strength Systems Engineering*, in association with the ZB 2002 meeting. Other conference sessions included a presentation on the status of the international Z Standard, in its final stages of acceptance. In addition, the International B Conference Steering Committee (APCB) and the Z User Group (ZUG) used the conference as a convenient venue for open meetings intended for those interested in the B and Z communities respectively.

The topics of interest to the conference included: Industrial applications and case studies using Z or using B; Integration of model-based specification methods in the software development lifecycle; Derivation of hardware-software architecture from model-based specifications; Expressing and validating requirements through formal models; Theoretical issues in formal development (e.g., issues in refinement, proof process, or proof validation, etc.); Software testing versus proof-oriented development; Tools supporting tools for the Z notation and the B method; Development by composition of specifications; Validation of assembly of COTS by model-based specification methods; Z and B extensions and/or standardization.

The ZB 2002 conference was jointly initiated by the Z User Group (ZUG) and the International B Conference Steering Committee (APCB). LSR-IMAG provided all local organization and financial backing for the conference. Without the great support from many local staff at LSR-IMAG and others in Grenoble, ZB 2002 would not have been possible. In particular, we would like to thank the Local Committee Chair, Marie-Laure Potet. ZB 2002 was supported by CNRS (Centre National de la Recherche Scientifique), INPG (Institut National Polytechnique de Grenoble), Université Joseph Fourier (Grenoble), and IMAG. ClearSy System Engineering, Gemplus, the Institut National de Recherche sur les Transports et leur Securité (INRETS), and RATP provided sponsorship. We are grateful to all those who contributed to the success of the conference.

On-line information concerning the conference is available under the following Uniform Resource Locator (URL):

`http://www-lsr.imag.fr/zb2002/`

This also provides links to further on-line resources concerning the B method and Z notation.

We hope that all participants and other interested readers benefit scientifically from these proceedings and also find them stimulating in the process.

November 2001

Didier Bert
Jonathan Bowen
Martin Henson
Ken Robinson

Program and Organizing Committees

The following people were members of the ZB 2002 Z program committee:

Conference Chair: Jonathan Bowen, South Bank University, London, UK
Program Chair: Martin Henson, University of Essex, UK

Ali Abdallah, South Bank University, London, UK
Rob Arthan, Lemma 1, Reading, UK
Paolo Ciancarini, University of Bologna, Italy
Neville Dean, Anglia Polytechnic University, UK
John Derrick, The University of Kent at Canterbury, UK
Mark d'Inverno, University of Westminster, UK
Wolfgang Grieskamp, Microsoft Research, USA
Henri Habrias, University of Nantes, France
Jonathan Hammond, Praxis Critical Systems, UK
Ian Hayes, University of Queensland, Australia
Jonathan Jacky, University of Washington, USA
Randolph Johnson, National Security Agency, USA
Steve King, University of York, UK
Kevin Lano, Kings College London, UK
Yves Ledru, LSR-IMAG, Grenoble, France
Jean-Francois Monin, France Telecom R&D, France
Fiona Polack, University of York, UK
Norah Power, University of Limerick, Ireland
Steve Reeves, University of Waikato, New Zealand
Mark Saaltink, ORA, Ottawa, Canada
Thomas Santen, Technical University of Berlin, Germany
Graeme Smith, University of Queensland, Australia
Susan Stepney, Logica Cambridge, UK
Sam Valentine, LiveDevices, York, UK
John Wordsworth, The University of Reading, UK

The following served on the ZB 2002 B program committee:

Program Chair: Didier Bert, CNRS, LSR-IMAG, Grenoble, France
Co-chair: Ken Robinson, The University of New South Wales, Australia

Christian Attiogbé, University of Nantes, France
Richard Banach, University of Manchester, UK
Juan Bicarregui, CLRC, Oxfordshire, UK
Pierre Bieber, CERT, Toulouse, France
Egon Börger, University of Pisa, Italy
Michael Butler, University of Southampton, UK
Dominique Cansell, LORIA, University of Metz, France
Pierre Chartier, RATP, Paris, France
Steve Dunne, University of Teesside, UK
Mark Frappier, University of Sherbrooke, Canada
Andy Galloway, University of York, UK
Jacques Julliand, University of Besançon, France
Jean-Louis Lanet, GemPlus Research Lab, France
Brian Matthews, CLRC, Oxfordshire, UK
Luis-Fernando Mejia, Alstom Transport Signalisation, France
Jean-Marc Meynadier, Matra Transport, France
Louis Mussat, DCSSI, France
Marie-Laure Potet, LSR-IMAG, Grenoble, France
Emil Sekerinski, McMaster University, Canada
Bill Stoddart, University of Teesside, UK
Helen Treharne, Royal Holloway, UK
Véronique Viguié Donzeau-Gouge, CNAM, Paris, France
Marina Walden, Åbo Akademi, Finland

The following people helped with the organization of the conference in various capacities:

B submissions:	Ken Robinson, The University of New South Wales Didier Bert, LSR-IMAG, Grenoble
Z submissions:	Martin Henson, University of Essex Sonia Oakden, University of Essex
Invited speakers:	Ken Robinson, The University of New South Wales
Tool demonstrations:	Mark d'Inverno, University of Westminster Yves Ledru, LSR-IMAG, Grenoble
Tutorials:	Henri Habrias, University of Nantes
Proceedings:	Didier Bert, LSR-IMAG, Grenoble
Local committee:	Marie-Laure Potet (chair), LSR-IMAG, Grenoble Pierre Berlioux, Jean-Claude Reynaud

We are especially grateful to the above for their efforts in ensuring the success of the conference.

External Referees

We are grateful to the following people who aided the program committees in the reviewing of papers, providing additional specialist expertise:

Yamine Ait Ameur, ENSAE/Aérospatiale and ONERA-CERT Toulouse, France
Françoise Bellegarde, Université de Franche-Comté, France
Eerke Boiten, The University of Kent at Canterbury, UK
Lilian Burdy, Laboratoire CEDRIC, CNAM, France
Alessandra Cavarra, Oxford University Computing Laboratory, UK
Fabien Combret, GemPlus, France
Axel Dold, University of Ulm, Germany
Benoit Fraikin, University of Sherbrooke, Canada
Lindsay Groves, Victoria University, New Zealand
Paul Howells, University of Westminster, UK
Olga Kouchnarenko, Université de Franche-Comté, France
Leonid Mikhailov, University of Southampton, UK
Pascal Poizat, Université d'Évry, France
Mike Poppleton, Open University, UK
Antoine Requet, GemPlus, France
Hector Ruiz Barradas, Universidad Autónoma Metropolitana de México
Marianne Simonot, Laboratoire CEDRIC, CNAM, France
Carsten Sühl, GMD, Berlin, Germany
Bruno Tatibouet, Université de Franche-Comté, France
Ray Turner, University of Essex, UK
Mark Utting, University of Waikato, New Zealand
Norbert Völker, University of Essex, UK
Jim Woodcock, The University of Kent at Canterbury, UK

Support

ZB 2002 greatly benefited from the support of the following organizations:

CNRS
IMAG
INP Grenoble
Université Joseph Fourier, Grenoble
Ministère français des Affaires Etrangères

and sponsorship from:

ClearSy System Engineering
GemPlus
INRETS
RATP

Table of Contents

Theories, Implementations, and Transformations

Eric Hehner and Ioannis T. Kassios

Department of Computer Science, University of Toronto,
Toronto ON M5S 3G4 Canada
{hehner, ykass}@cs.utoronto.ca

Abstract. The purpose of this paper is to try to put theory presentation and structuring in the simplest possible logical setting in order to improve our understanding of it. We look at how theories can be combined, and compared for strength. We look at theory refinement and implementation, and what constitutes proof of correctness. Our examples come from both the functional style and imperative (state-changing) style of theory. Finally, we explore how one implementation can be transformed to another.

1 Introduction

A classic paper by Burstall and Goguen in 1977 [2] taught us to think about data types used in computer programs as logical theories, presented by axioms, whose properties can be explored by logical deduction. The following year, a paper by Guttag and Horning [4] developed the idea further, showing us the algebraic properties of data types presented as theories. Another important contribution came from Abrial [8] in the design of Z, and more recently B [1]. He brought to theory design all the structuring and scoping that programming languages provide, enabling us to build large theories by composing smaller ones. With the work of the Z and B community, and a change of terminology, theory design became an important part of software development.

The purpose of this paper is to try to put theory presentation and structuring in the simplest possible logical setting in order to improve our understanding of it. It is not the purpose of this paper to provide a notation or language for practical engineering use; for that task the Z and B community are the leaders.

2 Notation

Notation is not the point of this paper; as much as possible, we will use standard, or at least familiar, notations. The two booleans are $\top$ and $\bot$, and the boolean operators are $\neg \wedge \vee = \neq \Rightarrow \Leftarrow$. The same equality $=$ and unequality $\neq$ will be used with any type. we also use a large version $=\ \Rightarrow\ \Leftarrow$ of equality and implication that are identical to the small version except for their precedence; the only purpose is to save

D. Bert et al. (Eds.): ZB 2002, LNCS 2272, pp. 1-21, 2002.

a clutter of parentheses. The empty bunch is *null* . The comma (,) is bunch union, which is commutative, idempotent, and associative. The colon (:) is bunch inclusion. For example,

$$2, 9 : 0, 2, 5, 9$$

is a boolean expression with value T because the left operand of colon is included in the right operand. We use the asymmetric notation $x,..y$ for the bunch of integers from and including x up to but excluding y . The empty list is [*nil*] , and the list [2; 6; 4; 8] contains four items. The notation $[x;..y]$ is used for the list of integers from and including x up to but excluding y . Lists are indexed from 0 . List formation distributes over bunch union, so if *nat* is the natural numbers, then [*nat*] is the list whose one item is the bunch of natural numbers, or equally, the bunch of all lists whose one item is a natural number. A star denotes repetition of an item, so [**nat*] is all lists of natural numbers. We use # for list length. We use a standard lambda notation $\lambda x\colon D\cdot fx$ for functions, and juxtaposition for function application. We use $A \rightarrow B$ for the bunch of all functions with domain at least A and range at most B . Quantifiers $\forall\ \exists$ apply to functions, but for the sake of familiarity they replace the lambda.

Here are all the notations of the paper in a precedence table.

0.	$\top$ $\bot$ ( ) [ ] numbers names	(true, false, precedence, list brackets)
1.	juxtaposition	(function application) right-to-left
2.	# * $\rightarrow$	(list length, item repetition, function space) right-to-left
3.	+ – +	(addition, subtraction, catenation) left-to-right
4.	; ;..	(sequencing of list items) associative
5.	, ,.. |	(bunch union, function selection) associative
6.	$=$ $\neq$ $<$ $>$ $\leq$ $\geq$:	(equality, unequality, order, inclusion) continuing
7.	$\neg$	(negation) right-to-left
8.	$\wedge$	(conjunction) associative
9.	$\vee$	(disjunction) associative
10.	$\Rightarrow$ $\Leftarrow$	(implication) continuing
11.	:=	(assignment)
12.	**if then else**	(if then else)
13.	;	(sequential composition) associative
14.	$\lambda\cdot$ $\forall\cdot$ $\exists\cdot$	(function, quantifiers)
15.	$=$ $\Longrightarrow$ $\Longleftarrow$	(equality, implication) continuing

To say that $=$ is continuing is to say that $a = b = c$ neither associates to the left nor associates to the right, but means $a = b \wedge b = c$. A mixture of continuing operators can be used; for example, $a \leq b < c$ means $a \leq b \wedge b < c$. For further details on notation and basic theories please consult [5] or [6].

3 Theories

Here is a little theory presented in a style similar to [2] and [4].

Theory0:	names:	*chain, start, link, isStart*
	signatures:	*start*: *chain*
		link: $chain \to chain$
		isStart: $chain \to bool$
	axioms:	*isStart start*
		$\forall c$: *chain*· $\neg$ *isStart* (*link c*)

Theory0 introduces four new names into our vocabulary. The signatures section tells us something about the role these names will play in the theory. Then the axioms tell us what can be proven, what are the theorems, in this theory.

The first problem with this presentation of Theory0 is that names cannot be attached to theories. For example, this theory uses the name *bool* , and many others do too, and each of them is telling us something about *bool*. And when we build large theories by composing smaller ones, no particular theory in the composition can claim a name as its own. And it isn't just names that get introduced by theories; symbols like $\le$, or in our example $\forall$ and $\neg$, and even $=$, are used in many theories, and each of them is telling us something more about the use of those symbols. Names and symbols are defined by their use in all theories where they appear; and we can always add more theories to the collection. As part of a library of theories, we need a linked, browsable dictionary of names and symbols, telling us which theories use them. This dictionary should be generated automatically from the library of theories, so that it is always up-to-date. The first change to theory presentation is to remove the list of names.

The next change to theory presentation is to consider a signature to be a kind of boolean expression. One of the uses of Bunch Theory is as a fine-grained type theory. The boolean expression

$$5:\ 0, 3, 5, 8$$

has value T and says, " 5 is included among 0, 3, 5, 8 ". But we can also read it as " 5 has type 0, 3, 5, 8 ". Defining *nat* as the bunch of all natural numbers, the boolean expression 5: *nat* has value T . And so *x*: *nat* can be given as an axiom about *x* . So too *x*, *y*: *nat* can be an axiom, just as 3, 5: 0, 3, 5, 8 has value T . The expression $A \to B$ consists of all functions with domain at least A and range at most B . For example,

$$(\lambda n: nat \cdot n+1) :\ nat \to nat$$

has value T . And so *f*: $nat \to nat$ can be an axiom about *f*. By "currying", $A \to B \to C$ consists of two-variable functions, and so on.

The final change to theory presentation is just to write all the axioms as one big axiom by taking their conjunction. Now a theory consists of one single axiom, so there is now no difference between a theory and an axiom. Theory0 can be written as follows.

$$
\begin{aligned}
\mathit{Theory0} \quad = \quad & \mathit{start}: \mathit{chain} \\
\wedge \quad & \mathit{link}: \mathit{chain} \to \mathit{chain} \\
\wedge \quad & \mathit{isStart}: \mathit{chain} \to \mathit{bool} \\
\wedge \quad & \mathit{isStart}\ \mathit{start} \\
\wedge \quad & \forall c: \mathit{chain} \cdot\ \neg\, \mathit{isStart}\ (\mathit{link}\ c)
\end{aligned}
$$

4 Composition

The original paper by Burstall and Goguen [2] presents four operations on theories: combination, enrichment, induction, and derivation. To illustrate theory combination, here is a second theory.

$$
\begin{aligned}
\mathit{Theory1} \quad = \quad & \mathit{start}: \mathit{chain} \\
\wedge \quad & \mathit{link}: \mathit{chain} \to \mathit{chain} \\
\wedge \quad & \forall c: \mathit{chain} \cdot\ \mathit{start} \neq \mathit{link}\ c \\
\wedge \quad & \forall c, d: \mathit{chain} \cdot\ (c=d) = (\mathit{link}\ c = \mathit{link}\ d)
\end{aligned}
$$

Theory0 and *Theory1* have much in common, but also some differences; there are theorems in each that are not theorems in the other. With our form of theory presentation, we can combine the two theories with ordinary boolean conjunction.

$$\mathit{Theory2} \quad = \quad \mathit{Theory0} \wedge \mathit{Theory1}$$

Burstall and Goguen's next theory operation, enrichment, is also just conjunction, but with further axioms rather than with a named theory. Here is an example.

$$
\begin{aligned}
\mathit{Theory3} \quad = \quad & \mathit{Theory2} \\
\wedge \quad & \forall c: \mathit{chain} \cdot\ \mathit{start} \le c < \mathit{link}\ c
\end{aligned}
$$

The next of Burstall and Goguen's theory operations adds a structural induction scheme over the generators of the new data type. For us, it is again just conjunction of another axiom.

$$
\begin{aligned}
\mathit{Theory4} \quad = \quad & \mathit{Theory3} \\
\wedge \quad & \forall P: (\mathit{chain} \to \mathit{bool}) \cdot \\
& \quad P\ \mathit{start} \wedge (\forall c: \mathit{chain} \cdot\ P\ c \Rightarrow P\ (\mathit{link}\ c)) \\
\Rightarrow \quad & \quad \forall c: \mathit{chain} \cdot\ P\ c
\end{aligned}
$$

That is the familiar form of induction; a neater, equivalent form is as follows.

$$\begin{aligned} Theory4 \quad = \quad & Theory3 \\ \wedge \quad & \forall C \cdot start, link\ C : C \ \Rightarrow \ chain : C \end{aligned}$$

To briefly explain this axiom, most operators and functions distribute over bunch union. For example,

$$(2, 5, 9) + 1 \ = \ (3, 6, 10)$$

So *link C* consists of all the results of applying *link* to things in C. The axiom says that if *start* and all the links of things in C are included in C, then *chain* is included in C. The antecedent can be rewritten as

$$start : C \quad \wedge \quad link : C \rightarrow C$$

and, regarding C as the unknown, *chain* is one solution. The axiom therefore says that *chain* is the smallest solution.

Burstall and Goguen's final operation on theories, derivation, allows part of a theory to be hidden from the theory users. For us, that's existential quantification.

$$Theory5 \quad = \quad \exists start : chain \cdot Theory4$$

Theory5 has all the same theorems as *Theory4* minus those that mention *start*. If we want to keep all the theorems of *Theory4* but rename *start* as *new*, define

$$Theory6 \quad = \quad \exists start : chain \cdot start = new \wedge Theory4$$

We can combine theories with other boolean operators too, such as disjunction and implication. For example,

$$Theory7 \quad = \quad (\forall c : chain \cdot new \leq c) \Rightarrow Theory6$$

This makes *Theory7* such that if we had the axiom $\forall c : chain \cdot new \leq c$ then we would have *Theory6*. In a vague sense, *Theory7* is *Theory6* without $\forall c : chain \cdot new \leq c$. To be precise, if we take *Theory7* and add the axiom $\forall c : chain \cdot new \leq c$, we get back *Theory6*.

$$Theory6 \quad = \quad Theory7 \wedge \forall c : chain \cdot new \leq c$$

New theories are not always built by additions to old theories; sometimes they are built by deletions. One of the problems with object-orientation is that, although subclassing allows us to add attributes, there is no way to delete attributes and make a superclass, nor to make an interclass between two existing classes.

These examples illustrate that our theory presentation is both a simplification and a generalization of the early work. By reducing theories to boolean expressions we understand them in the simplest possible way, and we allow all combinations that make logical sense.

5 Refinement and Implementation

A theory can serve as a specification of a data type, and of computation in general. Specifications can be refined, usually by resolving nondeterminism. Specification A refines specification B if all computer behavior satisfying A also satisfies B . If theories are expressed as single boolean expressions,

theory A refines theory B means $A \Rightarrow B$

theory B is refined by theory A means $B \Leftarrow A$

Refinement is just implication. So far, we have

$$
\begin{aligned}
&\mathit{Theory6} \Rightarrow \mathit{Theory7}\\
&\mathit{Theory4} \Rightarrow \mathit{Theory5}\\
&\mathit{Theory4} \Rightarrow \mathit{Theory3}\\
&\mathit{Theory3} \Rightarrow \mathit{Theory2}\\
&\mathit{Theory2} \Rightarrow \mathit{Theory1}\\
&\mathit{Theory2} \Rightarrow \mathit{Theory0}
\end{aligned}
$$

When we define a theory, and especially when we combine theories, there is always the danger of inconsistency. The only way to prove the consistency of a theory is to implement it. As software engineers, our goal is to design useful theories (they must be consistent to be useful), and to implement them. A theory is said to be implemented when all names and symbols appearing in it have been implemented. A name or symbol is implemented by defining it in terms of other names and symbols that are implemented. Ultimately, the computing machinery provides the ground theory on top of which all other theories are implemented. (To logicians, an implementation is known as a "model", and the ultimate machinery is usually taken to be set theory, although they might claim that the model is the sets themselves and not set theory.)

An implementation can be expressed in exactly the same form as a theory: a boolean expression. Here is an example implementation of *Theory4* , assuming that *nat* is an implemented data type, and that functions are implemented.

$$
\begin{aligned}
\mathit{Imp} \quad = \quad & \mathit{chain} = \mathit{nat}\\
\wedge \quad & \mathit{start} = 0\\
\wedge \quad & \mathit{isStart} = (\lambda c\colon \mathit{nat}\cdot c{=}0)\\
\wedge \quad & \mathit{link} = (\lambda c\colon \mathit{nat}\cdot c{+}1)
\end{aligned}
$$

An implementation is also a theory, but of a particular form. It is a conjunction of equations, and each equation has a left side consisting of one of the names needing an implementation, and a right side employing only names and symbols that are already implemented.

The benefit in expressing an implementation in the same form as a theory is that the proof of correctness of the implementation is now just a boolean implication. We prove that *Imp* correctly implements *Theory4* by proving

$$Imp \Rightarrow Theory4$$

Implementation is just refinement by an implemented theory. By the transitivity of implication we have immediately that *Imp* also implements *Theory5* , *Theory3* , *Theory2* , *Theory1* , and *Theory0* .

6 Functional Stack

From a typical mathematician's viewpoint, a stronger theory is a better theory because it allows us to prove more. But the theory must not be so strong as to be inconsistent, for then we can prove everything trivially. The game is to add axioms, approaching the brink of inconsistency as closely as possible without falling over. For example, here a strong but consistent theory of stacks.

$$
\begin{array}{lll}
Stack0 = \lambda X\cdot & & empty\colon stack \\
& \wedge & push\colon stack \to X \to stack \\
& \wedge & pop\colon stack \to stack \\
& \wedge & top\colon stack \to X \\
& \wedge & (\forall S\cdot\ empty,\ push\ S\ X\colon S \ \ \Rightarrow\ \ stack\colon S) \\
& \wedge & (\forall s\colon stack\cdot\ \forall x\colon X\cdot\ push\ s\ x \neq empty) \\
& \wedge & (\forall s, t\colon stack\cdot\ \forall x, y\colon X\cdot \\
& & push\ s\ x = push\ t\ y \ \ = \ \ s{=}t \wedge x{=}y) \\
& \wedge & (\forall s\colon stack\cdot\ \forall x\colon X\cdot\ pop\ (push\ s\ x) = s) \\
& \wedge & (\forall s\colon stack\cdot\ \forall x\colon X\cdot\ top\ (push\ s\ x) = x)
\end{array}
$$

And here is an implementation, assuming lists, functions, and integers are already implemented.

$$
\begin{array}{lll}
Stack = & & stack = [*int] \\
& \wedge & empty = [nil] \\
& \wedge & push = (\lambda s\colon stack\cdot\ \lambda x\colon int\cdot\ s^{+}[x]) \\
& \wedge & pop = (\lambda s\colon stack\cdot\ \mathbf{if}\ s{=}empty\ \mathbf{then}\ empty\ \mathbf{else}\ s\ [0;..\#s-1]) \\
& \wedge & top = (\lambda s\colon stack\cdot\ \mathbf{if}\ s{=}empty\ \mathbf{then}\ 0\ \mathbf{else}\ s\ (\#s-1))
\end{array}
$$