FROM FORMAL SPECIFICATION TO FULL PROOF: A STEPWISE METHOD

by

Lavinia Burski

Submitted for the degree of
Doctor of Philosophy

DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES
HERIOT-WATT UNIVERSITY

March 2016

Acronyms

ASM Abstract state machine.

CGa Core Grammatical aspect.

DRa Document Rhetorical aspect.

GPSa General Proof Skeleton aspect.

Gpsa General Proof Skeleton aspect.

GpsaOL General Proof Skeleton ordered list.

Hol-Z Hol-Z.

IEC International Electrotechnical Commission.

MathLang MathLang framework for mathematics.

PPZed Proof Power Z.

SIL Safety Integrity Levels.

SMT Satisfiability Modulo Theories.

TSa Text and Symbol aspect.

UML Unified Modeling Language.

UTP Unifying theories of programming.

ZCGa Z Core Grammatical aspect.

ZDRa Z Document Rhetorical aspect.

ZMathLang a toolkit for checking various degrees of correctness for Z specifications.

Glossary

computerisation The process of putting a document in a computer format.

formal methods Mathermatically rigorous techniques and tools for the specification, design and verification of software and hardware systems.

formalisation The process of extracting the essence of the knowledge contained in a document and providing it in a complete, correct and unambiguous format.

halfbaked proof The automatically filled in skeleton also known as the Half-Baked Proof.

partial correctness A total correctness specification [P] C [Q] is true if and only if, whenever C is executed in a state satisfying P and if the execution of C terminates, then the state in which Cs execution terminates satisfies Q.

semi-formal specification A specification which is partially formal, meaning it has a mix of natural language and formal parts.

total correctness A total correctness specification [P] C [Q] is true if and only if, whenever C is executed in a state satisfying P, then the execution of C terminates, after C terminates Q holds.

Chapter 1

Background

Formal methods are a specific type of mathematical notation which is based on the techniques of the specification, verification and development of software and hardware systems [5]. Since our thesis presents a toolkit for checking various degrees of correctness for Z specifications (ZMathLang) we go right to the beginning of the framework, to describe how mathematical notation came about. Then we describe the original MathLang framework (the framework which ZMathLang is an adaptation of) and then give the reader an idea of other formal methods and languages. In the next section we wish to describe what is the language of Z and give more details of it's syntax and semantics. We then highlight other proving techniques which have been done for maths, formal methods and Z.

1.1 Mathematical Notations

Computer science (and thus computer systems) have evolved from basic mathematics. We can say that formal specification writers are practicing mathematicians as they write system specifications in a formal manner. Therefore we must start right at the beginning at the foundation of mathematical notation.

1.1.1 Right from the begininning

The relationship between mathematical reasoning and practicing mathematicians started out early on during the ancient Greeks where logic was already being studied.

Reasoning in logic was used for just about anything not just mathematics such as law, medicine and farming. This very early form of mathematics made very famous discoveries such as Aristotles logic [24], Euclid's geometry [8] and Leibniz Calculus [16].

Further on in the 1800's, Frege wrote *Die Grundlagen der Arithmetik* [10] and other works where he noted that mathematics is a branch of logic. In this works, he began building a solid foundation for mathematics. This early foundation along with Cantors set theory [6] was argued to be incosistant and thus Russel found a pardox in this work.

In the late 19th century and beginning of 20th century, Russell & Whitehead [23] started to form a basis for mathematical notation. Their three volume work describes a set of rules from which all mathematical truths could be proven. In these early stages the authors try to dervive all maths from logic. This ambitious project was the first stepping stone in collaborating all mathematics under one notation.

Further to Russell & Whitehead's work, Bourbanki [1] wrote a series of books beginning n the 1935's with the aim of grounding mathematics. Their main works is included in the Elements of Mathematics series [4] which does not need any special of knowledge of mathematics. It describes mathematics from the very beginning and goes through core mathematical concepts such as set theory, algebra, function etc and gives complete proofs for these concepts.

Adding to Russell's work, Zermelo introduced an axiomatisation of set teory which was later extended by Frankel and Skolem to form ZF set theory [22]. This new theory is what we will later see the Z notation is based on and the notation this thesis checks the correctness of.

1.1.2 Computerisation of Maths and Proof Systems

In the 21st century, a great area of research is how use, store and support this mathematical knowledge. Since automation has become more and more used, mathematicians have looked into ways in which they can use computers to reason about

[1] A name given to a collective of mathematicians

and provide services to mathematics. This would include all areas of mathematics, such as logic, mechanics and software specifications. Mathematical knowedge can be represent in lots of different ways including the following:

- One can typeset mathematics into a computer using a system such as LaTeX [15] . These systems can edit and format mathematical knowledge so that it can be stored or printed. These systems provide good visual appearence and thus can be used for storing and archiving ones documents. Even Z specification, have their own package for LaTeX and thus the structure of the specification can be represented both formally and informally. However, it is difficult to represent the logical structure of mathematical formulas and the logical structure of mathematics is embedded in natural language. Therefore, there isn't a lot of support for checking the correctness of general mathematics represented in the system (Z specifications on their own can be checked but this is discussed further in section 1.5.4).

- Systems such as proof assistants (e.g. Isabelle [21], Coq [20] and ProofPower-Z [2]) and automated theorem provers (e.g. Boyer-Moore, Otter) and jointly called proof systems. Proof systems each provide a formal language for writing mathematics. Early work on proof systems was done by De Bruijn when he worked on the AutoMath [17] project. AutoMath (automating mathematics) was the first attempt to digitize and formally prove mathematics which was assited by a computer. AutoMath is described as a language for formalising mathematical texts and for automating the validation of mathematics. The AutoMath project is what brought uniform notations and automated proof together.

  Further to this work, there has been many other proof systems implemented to implement and check mathematics for total correctness. It is possible to access the semantics of mathematics in these systems. However, with these proof systems, a user must choose a specific proof system and one of these have their advatages and disadvantages. Also, each of these proof systems also take quite some time to learn. There is much documentation on some of these systems (e.g. Isabelle) and some are very well supported. But this in turn can be a downfall, as there is so much documentation, it is difficult to know how much one must learn and where to start. The best way of learning one of these system is from someone who already is an expert in the chosen proof system. A lot of the prood systsems use proof tactics to constructs proofs and make them smaller, however to prove certain properties in a proof system, one can use various tactics to get to the same goal and may sometimes be difficult to find which tactic is best to use.

With these disadvantages many academic and industrial mathematicians do not generally use the mathematics written in the language of the proof system and usually are not willing to spend the time to check the correctness of their own work in this system.

- There also exists semantical oriented document representations like OpenMath [1] and OMDoc [13]. These systems are better than the typesetting systems such as LaTeX to produce readable and printable version of the mathematical knowledge written. Some aspects of the semantics of the mathematics can be represented in these type systems. However, when using these systems it is difficult to control the presentation and therefore a typesetting system is more likely to be used. Like the typesetting system, systems like OMDoc also have difficult reading the logical structure of mathematics embedded in the natural language of the text. Systems like OMDoc can associate symbolic formulas with chunks of natural language text, however these chunks can not be seen bu the computer and thus can not be checked if it is correct.

  Another disadvantage is that although there is support for the semantics of the mathematics, these systems can not support the semantics in terms of logical foundations for mathematics (unlike proof systems).

1.1.3 Conclusion

In summary the MathLang framework for mathematics (MathLang) framework has been developed to be used as a bridge between the categories metioned above as a way to represent and automatically check mathematical knowledge. Since the Z notation has stemmed from the origins of mathematics and industry is starting to use formal methods in there system development we have chosen to adapt the MathLang framework to accomadate Z specification and have developed a set of tools to do so.

1.2 MathLang for mathematics

MathLang originally started in 2000. It's original goals was to allow gradual computerisation and formalisation of mathematical texts.

MathLang is not a system for proof verification but a framework to computerise and translate information (such as mathematical text) into a form on which proof checkers can operate.

The MathLang framework provides extra features supporting more rigour to translation of the common mathematical language. One can define further levels of translations into more semantically and logically complete versions. This gradual computerisation method should be more accessible than direct formalisation, because a number of first levels do nor require any particular expertise in formalisation.

So far Mathlang has given alternative and complete paths which transform mathematical texts into new computerised and formalised versions. Dividing the formalisation of mathematical texts into a number of different stages was first proposed by N.G. de Bruijn to relate common mathematical language to his Mathematical Vernacular [7] and his proof checking system Automath.

1.2.1 Overview and Goals

The MathLang Framework instructs the computerisation process to be broken up into a number of levels called **aspects**. Each aspect can be worked out independently, simultaneously or sequentially without prior knowledge of another aspect. The current MathLang Framework contains three well-developed aspects, the Core

Grammatical aspect (CGa), the Text and Symbol aspect (TSa) and the Document Rhetorical aspect (DRa), and has further aspects such as the Formal Proof Sketch.

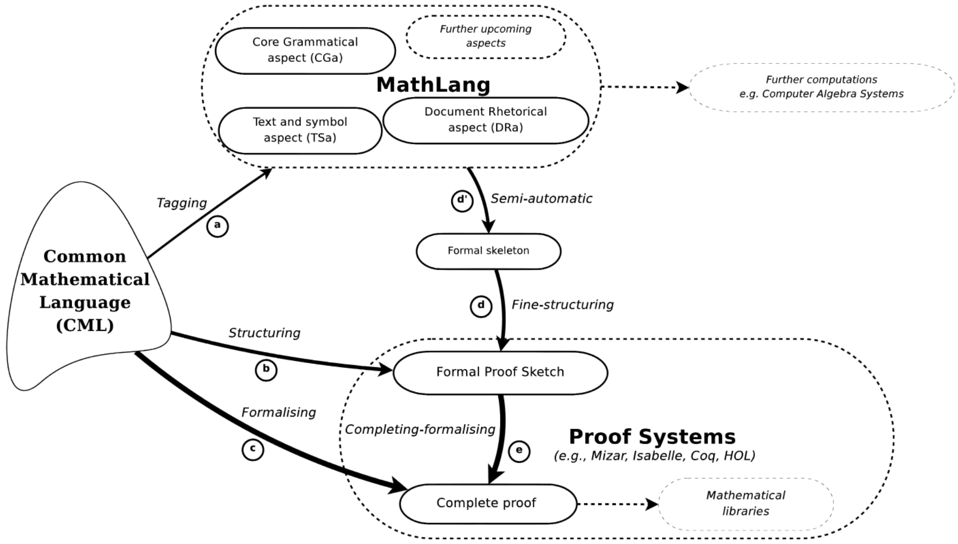

Figure 1.1: The MathLang approach to computerisation/formalisation [12]

Figure 1.1 shows the overall situation of work in the current MathLang Framework. The labelled arrows show the computerisation paths from the common mathematical language to any proof system. The width of the arrow representing each path segment increases according to the expertise required. The level of expertise needed to computerise a CML text straight into a complete proof is very high, however the level of expertise is much smaller by using the MathLang framework to help form a formal skeleton and then into a complete proof. The dashed arrows illustrate further computerisation that one can envision.

1.2.2 Detailed information on CGa

The current CGa in MathLang uses a finite set of grammatical *categories* to identify the structure and common concepts used in mathematical texts. The aims of the CGa is to make explicit the grammatical role played by the elements of mathematical texts and to allow the validation of the grammatical and reasoning structure within the CGa encoding in a mathematical text. The CGa checks for grammatical correctness and finds errors like an identifier being used without and prior introduction or the wrong number of arguments being given to a function [18].

- Reference Zenglars quote
- Weak type theory into CGa

1.2.3 Detailed information on DRa

The Document Rhetorical aspects checks that the correctness of the reasoning in the mathematical document is correct and that there are no loops. The DRa mark-up system is simple and more concentrated on the narrative structure of the mathematical documents whereas other previous systems (such as DocBook [2], Text Encoding Initiative [3], OMDoc [4]) were more concentrated on the subtleties of the documents. It is used to describe and annotate chunks of texts according to their narrative role played within the document [18]. Using the DRa annotation system we can capture the role that a chunk of text imposes on the rest of the document or on another chunk of text. This leads to generating dependency graphs which play an important role on mathematical knowledge representation. With these graphs, the reader can easily find their own way while reading the original text without the need to understand all of its subtleties. Processing DRa annotations can flag problems such as cicular reasoning and poorly-supported theorems. ———————————

———

- relations
- instances
- Dependency and goto graph

1.2.4 Detailed information on skeletons

- General Proof Skeleton
- Half baked proof

[2]http://www.docbook.org
[3]http://www.tei-c.org/index.xml
[4]http://www.omdoc.org

- Filled in skeleton

1.2.5 information on TSa

The TSa builds the bridge between a mathematical text and its grammatical interpretation. The TSa is a way of rewriting parts of the text so they have the same meaning. For example some mathematicians may prefer to write "a=b and b=c and c=d", others may prefer "a=b, b=c, c=d" and some others may prefer "a=b=c=d". As you can see all these methods of writing have the same meaning however some symbols are different. The TSa annotates each expression in the text with a string of words or symbols which aim to act as the mathematical representation of which this expression is. This allows everything in the text to be uniform.

1.2.6 A full worked examples in mathlang

show step by step translation of mathematical text into isabelle from laamars phd thesis.

1.2.7 Conclusion

1.3 Formal Methods and Languages

Formal methods are usually used to assist in formalising in a variety of texts including systems, software and even language itself.

Definition 1.3.1 (Formal Language)**.** *A language designed for use in situations in which natural language is unsuitable, as for example in mathematics, logic, or computer programming.*[5]

Definition 1.3.2 (Formal Specification)**.** *The specification of a program's properties in a language defined by a mathematical logic.*[6]

[5] www.dictionary.com/browse/formal-language
[6] www.wiki.c2.com/?FormalSpecification

Definition 1.3.3 (Formal methods). *Mathematical approaches to software and system development which support the rigorous specification, design and verification of computer systems.*[7]

Formal methods are mathematical approaches to software and system development which support the rigorous specification, design and verification of computer systems [9]. Specifications are a collection of statements describing how a proposed system should act and function. Formal specifications use notations with defined mathematical meanings to describe systems with precision and no ambiguity. The properties of these specifications can then be worked out with more confidence and can be described to the customers and other stakeholders. This can uncover bugs in the stated requirements which may not have found in a natural language specification. With this, a more complete requirements validation can take place earlier in the development life-cycle and thus save costs and time of the overall project. The rigor using formal methods eliminates design errors earlier and results in substantially redecued time [11].

1.3.1 A brief history of formal methods

The first known formal language is thought to be used by Frege in his Begriffsschift (1879), Begriffschift meaning 'concept of writing' described as 'formal language of pure thoguht'. Frege formalised propositional logic as an axiomatic system.

Formal methods then grew in the following:

- 1940's, Alan Turing annotated the properties of program states to simplify the logical analysus of sequential programs
- 1960's Floyd, Hoare and Naur recommended using axiomatic techniques to prove programs meet their specification.
- 1970's Dijkstra used formal caluculus to aid development of non-deterinist programs

[7] www.fmeurope.org/?page_id=2

Formal methods are used today are just as important as when they were used before. Formal methods have a large presence in academia and have also made their way into various industries to prevent design flaws in high integrity systems. Previous desing errors have been found in systems such as the Therac-25 machine, which was used for radiation therapy produced by Atomic Energy of Canada Limited in 1982. It was involved in multiple incidents in which patients were give massive overdoses of radiation [3]. Another major fault which led to disastrous results was NASAs Checkout Launch and Control System (CLCS) cancelled 9/2002 [8].

1.3.2 Types of formal methods

Today there are many different types of formal methods used both in industry and academia. Specification languages are expressive languages with general proof methods, such as VDM, Z, B. Another type of formal method could be program correctness proofs which associates logical inference rules with programming syntax, e.g. Hoare triples and Gries Methodolgy.There may also be model based approaches to formal methods, which are domain specific languages with precise algorithms for correctness proofs, e.g. Temporal logic [?], Fuzzy logic. Formal methods are used to precisley communicate specifications and the function of programs. They are also used to ensure the correctness of systems particularly safety critical systems. Formal methods have been a success in a variety of projects. For example, in the Sholis project [?], using a formal specification was most effective for fault finding, therefore if the specifications are correct, then the program implemented should then in turn contain less errors if it follows the correct specification. King, Hammond, Chapman and Pryor's paper [?] was based on the SHOLIS defence system. It highlighted the importance of having a formal specification on a system to check for errors. It was found that the Z proof was the most cost effective for fault finding. The Z specification found 75% of

[8] www.spaceref.com/news/viewnews.html?id=475

the total faults for the system. Since Z specifications are important for finding faults in SIL4 systems (based on the sholis project), then checking the correctness of the Z specification is itself very important. Note that the specifications found 75% of errors and not 100%. As human error can still occur in formal specifications, using the ZMathLang approach may increase the percentage of errors found.Success of formal methods (B27 Traffic Control System, SHOLIS project, Data Acquisition, Monitoring and Commanding of Space Equipment) Weakness of formal methods (Low-level ontologies, Limited Scope,Cost,Poor tool feedback) What needs to be done to make formal methods industrial strength?

- - Bridge gap between real world and mathematics
  - Mapping from formal specifications to code (preferably automated)
  - Patterns identified
  - Level of abstraction should be supported
  - Tools needed to hide complexity of formalism
  - Provide visualization of specifications
  - Certain activities not yet formulizable methods
  - No one model has been identified which should be used for software)

1.3.3 Conclusion

1.4 Z Syntax and semantics

1.4.1 Why Z would work with MathLang

Bridges formal method and discrete mathematical notation.

1.4.2 Introduction to Z

Z is based on predicate Calculus, Zermelo-Frankel set theory...

Ivented by j-R Abriel, ISO standard.+ Spivey standard

1.4.3 Propositional and predicate logic

1.4.4 Sets and Types

1.4.5 Definitions

- AxDef
- Freetypes
- Schema (declarations and expressions)

1.4.6 A full example in Z

1.4.7 Conclusion

1.5 Proving systems for Z

Intro....

1.5.1 Levels of Rigor

- Level 1 represents the use of mathematical logic to specify the system.
- Level 2 uses pencil-and-paper proofs.
- Level 3 is the most rigorous application of formal methods.

1.5.2 Proving systems for maths

e.g. Mizar, Isabelle, Coq

1.5.3 Proving systems for formal method

e.g. Dafny, ALC2, PVS

1.5.4 Proving Systems specific for Z

e.g. Fuzz, Hol-z ProofPower-z

1.5.5 Other proeprties to prove

1.5.6 Conclusion

1.6 Background Conclusion

1.6.1 MathLang for Z

- [19] states what ro do to make formal methods industrial strength
- [14] stating in future work mathlang should be developed to cope with more mathematics (formal spec is a type of mathematics)
- diagram of math text to theorem prover using mathlang + diagram of specification to theorem prover using mathlang

ZMathLang covers items 1, 3, 5, 6, 7 from section 1.3.

Bibliography

[1] J. Abbott, A. Díaz, and R. S. Sutor. A report on openmath: A protocol for the exchange of mathematical information. *SIGSAM Bull.*, 30(1):21–24, Mar. 1996.

[2] R. Arthan. Proof Power. `http://www.lemma-one.com/ProofPower/index/`, February 2011.

[3] S. Baase. *A Gift of Fire: Social, Legal, and Ethical Issues for Computers and the Internet.* An Alan R. Apt book. Pearson Education, 2003.

[4] N. Bourbaki. *General topology. Chapters 1-4.* Elements of mathematics. Springer-Verlag, Berlin, Heidelberg, Paris, 1989. Trad. de : Topologie gnrale chapitres 1-4.

[5] R. Butler, G. Hagen, J. Maddalon, C. Muñoz, A. Narkawicz, and G. Dowek. How formal methods impels discovery: A short history of an air traffic management project. In C. Muñoz, editor, *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010), NASA/CP-2010-216215*, pages 34–46, Langley Research Center, Hampton VA 23681-2199, USA, April 2010. NASA.

[6] G. Cantor. Ueber eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen. *Journal fr die reine und angewandte Mathematik*, 1874(77):258–262, 1847.

[7] N. de Bruijn. The mathematical vernacular, a language for mathematics with typed set. *In Workshop on Programming Logic*, 1987.

[8] R. Fitzpatrick and J. Heiberg. *Euclid's Elements.* Richard Fitzpatrick, 2007.

[9] Formal Methods Europe, L-H Eriksson. Formal methods europe. http://www.fmeurope.org/?page_id=2, May 2016.

[10] F. Gottlob. Die grundlagen der arithmetik. *Eine logisch mathematische Untersuchung ber den Begriff der Zahl, Bres*, 292, 1884.

[11] J. Groote, A. Osaiweran, and Wesselius2. Benefits of Applying Formal Methods to Industrial Control Software. Technical report, Eindhoven University of Technology, 2011.

[12] F. Kamareddine, M. Maarek, K. Retel, and J. B. Wells. Gradual computerisation/formalisation of mathematical texts into Mizar. In *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, pages 81–95. Springer-Verlag, 2007.

[13] M. Kohlhase. *OMDoc – An Open Markup Format for Mathematical Documents [Version 1.2]: Foreword by Alan Bundy (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[14] R. Lamar. *A Partial Translation Path from MathLang to Isabelle*. PhD thesis, Heriot-Watt University, 2011.

[15] Leslie Lamport et al. The latex project. http://www.latex-project.org/, September 2016.

[16] L. Mastin. 17th century mathematics - Leibniz. http://www.storyofmathematics.com/17th_leibniz.html, 2010.

[17] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath*, volume 133. Elsevier, 1994.

[18] K. Retel. *Gradual Computerisation and Verification of Mathematics: MathLang's Path into Mizar*. PhD thesis, Heriot-Watt University, 2009.

[19] C. V. Stringfellow. Formal methods presentation. September 2016.

[20] Thierry Coquand and Grard Huet. Coq. http://coq.inria.fr/, September 2016.

[21] University of Cambridge and Technische Universitat Munchen. Isabelle. `http://www.isabelle.in.tum.de`, May 2015.

[22] E. W. Weisstein. Zermelo-fraenkel set theory. `http://mathworld.wolfram.com/Zermelo-FraenkelSetTheory.html`.

[23] A. Whitehead and B. Russell. *Principia Mathematica*. Number v. 2 in Principia Mathematica. University Press, 1912.

[24] J. Woods. *Aristotle's Earlier Logic (2nd Edition)*. College Publications, 2014.