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

Abstract

Proving formal specifications in order to find logical errors is often a difficult and labour-intensive task. This thesis introduces a new and stepwise toolkit to assist in the translation of formal specifications into theorem provers, using a number of simple steps based on the MathLang Framework. By following these steps, the translation path between a Z specification and a formal proof in Isabelle could be carried out even by one who is not proficient in theorem proving.

Acknowledgements

I dedicate this thesis to my loving and supportive boyfriend, Jeff.

Contents

List of Tables

List of Figures

Todo list

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 MathLang framework 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

Evaluation and Discussion

In this chapter we go through a few case studies and discuss the difference between the specification translations if any. Table 1.1 shows the specifications we have translated into Isabelle using ZMathLang. We have classified these examples to show the different types of specifications which can be translated using the ZMathLang toolkit. In this chapter we take one example from each class and describe in more detail how the translation was done.

Examples using only terms	**Examples using sets and terms**
Vending Machine	Birthday Book
SteamBoiler	ClubState
Incomplete translations	Clubstate2
Autopilot	GenDB
A specification which fails ZCGa	ModuleReg
A specification which fails ZDRa	ProjectAlloc
	Timetable
	Videoshop
	TelephoneDirectory
	ZCGa

Table 1.1: A table showing the specifications we have translated into Isabelle using ZMathLang

We have categoriesed the specification into three groups; specifications which only use terms, specifications which use both terms and set and specifications which the translation is incomplete for a variety of reasons. All the specifications we have translated are 'state based specifications', which means they operate within a state and to change the state their may become precondition and postconditions within the state. Some specifications are described differently such as functional specifications, however those type of specifications are out of the scope of this thesis.

1.1 Complexity of specifications

This section we analyse the complexity of the specifications we have translate using ZMathLang. First we check the complexity of the raw LaTeX specification file, without any annoatation. Then we discuss the complexity of the Z Core Grammatical aspect (ZCGa) annotated specifications and Z Document Rhetorical aspect (ZDRa) annotated specifications and how this affects the translation into Isabelle.

1.1.1 Raw Latex Count

Table 1 how long each specification is by amount of lines of code and environments uses. We have listed the specifications in decreasing complexity of how many lines of LaTeX the raw specification has.

Specification	**Environment**				**Lines of LaTeX**
	Zed	Schema	Axdef	**Total**	
Steamboiler	10	34	3	47	507
ProjectAlloc	4	17	0	21	213
VideoShop	3	15	0	18	166
TelephoneDirectory	6	11	0	17	133
ClubState	4	11	1	16	129
ZCGa	2	9	0	11	128
GenDB	2	7	0	9	114
Timetable	1	6	1	8	92
BirthdayBook	3	7	0	10	83
AutoPilot	2	3	0	5	83
ClubState2	1	6	1	8	80
Vending Machine	4	7	0	11	68
ModuleReg	1	3	0	4	43

Table 1.2: How many zed, schema and axdef environments and lines of LaTeX code makes up each specification

We list information about how many different envronments and lines of LaTeX make up each specification in table 1.2. The environment numbers count how many different types of environments exist within the specification. That is how many '`\begin{schema}...\end{schema}`' or '`\begin{zed}`... etc. We add up the total amount of environments in the specification. From the table we can see that for most of the specifications the more lines of LaTeX there is then the total amount of environments increse. However, there are three exceptions to this trend. The '*BirthdayBook*' specification, '*ClubState2*' specification and '*Vendine Machine*' specification. Specifications for systems are can always be written in a variety of ways and still have the same meaning. Even formal specifications can be written different ways. For example one may have the following declarations:

$t : \mathbb{N}$

$l : \mathbb{N}$

However, this declaration can also be written as the following:

$t, l :; \mathbb{N}$

Thus removing a line. Formal specifications can also include comments written in natural language which are not part of the formal script. These extra comments about the specification may have also added to the line count in table 1.2.

1.1.2 ZCGa Count

In this section, we evaluate the ZCGa annotations on the specifications. We describe how many of each ZCGa annotations occurs for each specification we have translated.

Specification	**ZCGa WeakTypes**					
	■	■	■	■	■	■
Steamboiler	297	26	282	595	4	0
ProjectAlloc	98	43	113	154	165	0
VideoShop	87	31	75	119	95	0
TelephoneDirectory	78	26	53	72	50	0
ClubState	75	17	51	55	51	0
ZCGa	73	27	67	35	133	0
GenDB	45	24	71	117	121	1
Timetable	35	15	53	48	114	0
BirthdayBook	26	11	24	28	19	0
AutoPilot	16	9	19	31	2	0
ClubState2	34	7	37	22	72	0
Vending Machine	16	7	21	37	0	0
ModuleReg	20	6	18	13	31	0

Table 1.3: How many of each grammatical category exists in each specification.

The amount of times a ZCGa weak type occurs in each specification is shown in table 1.3. We remind the reader the colours corresponding to each grammatical type are: *schematext*, *declaration*, *expression*, *term*, *set* and *definition*. In this instance we don't use specification as we assume each document contains a single specification.

In our sample set we only have one specification (GenDB) with a '`definition`' annotation. This `definition` is locally defined within the specification. The '*Vending Machine*' specification only uses `terms` and therefore there are no ZCGa `term` annotations. However the '*StemBoiler*' specification also only uses term yet there are 4 `set` ZCGa annotations. This is because some of the `terms` used in the specification have to be intrduced by a `set`. For example in the *SteamBoiler* specification we have the following annotation:

```
\begin{zed}
\set{State} ::= \term{init} | \term{norm} |
\term{broken} | \term{stop}
\end{zed}
```

Although the set `State` is annotated as a set, it is not used in any of the schema's in the rest of the specification. It is only defined to present the terms `init`, `norm`, `broken` and `stop` which are used in the specification.

We expect there to be more `schemaText`'s then `declarations` and `expressions` combined as `schemaText` contains all `declarations`, `expressions` and SchemaNames however, from the table we can see that this is not always the case. For example in the *ProjectAlloc* example, there are 98 `schemaText`, 43 `declarations` and 113 `expressions`. The reason for this could be because a single `expression` can in itself contain many `expressions`. For example the following `schemaText` has been taken from the *ProjectAlloc* specification:

```
\text{\expression{\forall
\declaration{\term{lec}: \expression{\dom maxPlaces}}\\
@ \expression{\term{\# (\set{\set{allocation}
\rres \set{\{\term{lec}\}}})} \leq \term{\set{maxPlaces}~\term{lec}}}}}
```

In this example we can see that there contains 1 annotated `schemaText` but 3 `expressions`. Another reason why there may be more `expressions` than `schemaText` is because when annotating a specification with ZCGa, `declarations` also contain `expressions`. If we have the following example, again taken from the ProjectAlloc specification:

```
\text{\declaration{\set{studInterests}, \set{lecInterests}:
\expression{PERSON \pfun\iseq TOPIC}}}
```

The ZCGa text contains 1 annotation of `SchemaText`, 1 annotation of a `declaration`, 2 annotations of `sets` and 1 annotation of an `expression`. Since this is the case we expect to see more expressions than declarations in every specification, which is true according to table 1.3.

1.1.3 ZDRa Count

In this section we analyse the amount of ZDRa instances and relations are labeled for each of the specifications we translated. We give details of the amount of instances in table 1.4 and give details of the amount of relations in each specification in table 1.5.

Specification	ZDRa Instances									
	A	**SS**	**IS**	**CS**	**OS**	**TS**	**PRE**	**PO**	**O**	**SI**
Steamboiler	6	2	2	21	6	6	21	23	12	1
ProjectAlloc	0	1	1	5	11	0	11	6	22	1
VideoShop	0	1	1	3	10	0	13	4	20	1
TelephoneDirectory	0	1	1	4	5	5	8	5	10	1
ClubState	1	1	1	4	6	4	9	6	11	0
ZCGa	0	1	1	6	1	0	6	7	2	1
GenDB	0	1	1	4	2	0	6	5	4	1
Timetable	1	1	1	4	0	0	4	5	0	1
BirthdayBook	0	1	1	1	4	2	4	2	8	1
AutoPilot	0	2	0	1	1	0	1	1	2	0
ClubState2	1	2	1	3	0	0	3	4	0	2
Vending Machine	0	1	0	3	0	3	3	2	0	0
ModuleReg	0	1	0	2	0	0	2	2	0	1

Table 1.4: How many of each ZDRa instances exists in each specification.

From table 1.4 we can see that all specifications have either 1 or 2 statesSchema's. For state base specification it should be the case that then specification has at least 1 state. Most state based specifications have stateInvariants that must be conformed to through all the changes of the specification. However this is not a must and some specification (even from our sample) do not have any stateInvariants.

All precondition must have a corresponding postcondition or output, therefore we can say:

Lemma 1.1.1. *precondition* $\longrightarrow$ *postcondition* $\vee$ *output*

The table supports this informatio as there are more combined postconditions and outputs then there are precondition. However not all postconditions and outputs need to have a precondition, they can be executed without one. Therefore the number of preconditions does not need to equal the total number of postcondition and outputs.

Specification	**ZDRa Relations**				
	initiaOf	**requires**	**allows**	**totalises**	**uses**
Steamboiler	2	28	21	24	92
ProjectAlloc	1	16	11	0	16
VideoShop	0	15	13	0	142
TelephoneDirectory	1	11	8	14	8
ClubState	1	12	9	14	12
ZCGa	1	9	6	0	7
GenDB	1	8	6	0	6
Timetable	1	6	4	0	6
BirthdayBook	1	7	4	6	5
AutoPilot	0	2	1	0	2
ClubState2	1	6	3	0	6
Vending Machine	0	2	0	2	8
ModuleReg	0	3	2	0	2

Table 1.5: How many of each ZDRa relations exists in each specification.

We can cross reference the table showing the amount of instances (table 1.4) with the table showing the relations (table 1.5). For example, the relation *initialOf* can only occur if the specification has an *initialSchema*. Not all specifictions have an *initialSchema* and therefore do not have an *initialOf* relation.

There is also an equal amount of *allows* relations as there is *preconditions*. As was written previously, all preconditions must have a corresponding output or postcondition, therefore the relation '*allows*' links each precondition to its corresponding postcondition or output. However, the vendingMachine specification is an exception to this as the preconditions are written as entire schema's. For example we have the following instance in the vending machine specification:

```
\draschema{PRE3}{
\begin{schema}{some\_stock}
stock: \nat
```

```
\where
stock > 0
\end{schema}}
```

This chunk of specification describes an entire schema as a precondition. The totalising schema then joints the precondition to their corresponding output or postcondition. The specification is written in this way as it is a personal choice of writing the specification formally. All other specifications in our sample set are written in the style where the precondition and corresponding output or postcondition are written inside the same schema environment.

Obviously, the relation '*totalises*' only occurs in specifications where totaliseSchema's are present. Therefore the '*totalise*' relation is not necessary in all specifications.

VideoShop specification is one of the largest specifications (in terms of lines of LaTeX) in our sample set however it has quite a small amount of relations

Complete Evaluation and discussion chapter

1.2 Case Studies

This section describes a few specification case studies in which we have used the ZMathLang tool kit to translate and prove formal specifications into the Isabelle automated theorem prover. The first case study present a formal specification only using terms, the second is a formal specification where both sets and terms are used and therefore the syntax used in Isabelle is more complex. The final case study we present is a partial translation of a specification which is not fully formalised but on it's way to becoming fully formal.

1.2.1 Case Study 1: A specification using only terms.

The following case study is based on the *Steamboiler* specification which has been translated and proved in Isabelle using the ZMathLang framework. This case study only uses variables which are terms.

1.2.2 Case Study 2: A specification using both terms and sets.

This case study based is on the *Project Allocation* specification which uses both terms and sets. The specification has been translated into Isabelle using the ZMathLang framework.

1.2.3 Case Study 3: A semi formal specification.

In this case study we present the *AutoPilot* specification. The specification is a semi formal specification and has been partially translated into Isabelle. The parts which have been translated are written formally and have been annotated accordingly. This gives an example of a specification which is written in natural language and is on it's way to being formalised.

1.3 Analysing examples

1.4 Reflection and Discussion

1.5 Conclusion

Bibliography

[1] J.-R. Abrial. Event Based Sequential Program Development: Application to Constructing a Pointer Program. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 51–74. Springer, 2003.

[2] J.-R. Abrial. Formal methods in industry: achievements, problems, future. *Software Engineering, International Conference on*, 0:761–768, 2006.

[3] M. Adams. Proof auditing formalised mathematics. *Journal of Formalized Reasoning*, 9(1):3–32, 2016.

[4] A. Álvarez. *Automatic Track Gauge Changeover for Trains in Spain.* Vía Libre monographs. Vía Libre, 2010.

[5] A. W. Appel. Foundational Proof-Carrying Code. In *LICS*, pages 247–256, 2001.

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

[7] H. P. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1991. http://citeseer.ist.psu.edu/barendregt92lambda.htmlElectronic Edition.

[8] J. C. Blanchette. *Hammering Away, A user's guide to Sledgehammer for Isabelle/HOL.* Institut fur Informatik, Technische Universitat Munchen, May 2015.

[9] E. Borger and R. F. Stark. *Abstract State Machines: A Method for High-Level System Design and Analysis.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

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

[11] J. Bowen. Formal Methods Wiki, Z notation. `http://formalmethods.wikia.com/wiki/Z_notation`, July 2014.

[12] A. D. Brucker, H. Hiss, and B. Wolff. HOL-Z 2.0: A Proof Environment for Z-Specifications. *Journal of Universal Computer Science*, 9(2):152–172, feb 2003.

[13] L. Burski. Zmathlang. `http://www.macs.hw.ac.uk/~lb89/zmathlang/`, Jan 2016.

[14] L. Burski. ZMathLang Website. `http://www.macs.hw.ac.uk/~lb89/zmathlang/examples`, June 2016.

[15] R. W. Butler. An introduction to requirements capture using PVS: Specification of a simple autopilot. NASA Technical Memorandum 110255, NASA Langley Research Center, Hampton, VA, May 1996.

[16] R. W. Butler. What is Formal Methods. `http://shemesh.larc.nasa.gov/fm/fm-what.html`, March 2001.

[17] W. Chantatub. *The Integration of Software Specification Verification and Testing Techniques with Software Requirements and Design Processes.* PhD thesis, University of Sheffield, 1995.

[18] Clearsy Systems Engineering. B Methode. `http://www.methode-b.com/en/`, 2013.

[19] J. Coleman, C. Jones, I. Oliver, A. Romanovsky, and E. Troubitsyna. RODIN (rigorous open development environment for complex systems). In *EDCC-5, Budapest, Supplementary Volume*, pages 23–26, Apr. 2005.

[20] I. E. Commission. IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems. Technical report, International Electrotechnical Commission, 2010.

[21] E. Currie. *The Essence of Z.* Prentice-Hall Essence of Computing Series. Prentice Hall Europe, 1999.

[22] H. Curry. Functionality in combinatorial logic. In *Proceedings of National Academy of Sciences*, volume 20, pages 584–590, 1934.

[23] C.Weidenbach, D.Dimova, A.Fietzke, R.Kumar, M.Suda, and P. Wischnewski. Isabelle cheat sheet. `http://www.phil.cmu.edu/~avigad/formal/FormalCheatSheet.pdf`.

[24] C.Weidenbach, D.Dimova, A.Fietzke, R.Kumar, M.Suda, and P. Wischnewski. Spass. `http://www.spass-prover.org/publications/spass.pdf`.

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

[26] L. De Moura and N. Bjørner. Satisfiability Modulo Theories: Introduction and Applications. *Commun. ACM*, 54(9):69–77, Sept. 2011.

[27] D. Fellar, F. Kamareddine, and L. Burski. Using MathLang to Check the Correctness of Specifications in Object-Z. In E. Venturino, H. M. Srivastava, M. Resch, V. Gupta, and V. Singh, editors, *In Modern Mathematical Methods and High Performance Computing in Science and Technology*, Ghaziabad, India, 2016. M3HPCST, Springer Proceedings in Mathematics and Statistics.

[28] D. Feller. Using MathLang to check the correctness of specification in Object-Z. Master Thesis Report, 2015.

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

[30] S. Fraser and R. Banach. Configurable Proof Obligations in the Frog Toolkit. In *Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2007), 10-14 September 2007, London, England, UK*, pages 361–370. IEEE Computer Society, 2007.

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

[32] S. L. Hantler and J. C. King. An Introduction to Proving the Correctness of Programs. *ACM Comput. Surv.*, 8(3):331–353, Sept. 1976.

[33] E. C. R. Hehner. Specifications, Programs, and Total Correctness. *Sci. Comput. Program.*, 34(3):191–205, 1999.

[34] A. Ireland. Rigorous Methods for Software Engineering, High Integrity Software Intensive Systems. Heriot Watt Universtiy, MACS, Lecture Slides.

[35] F. Kamareddine and J.B.Wells. A research proposal to UK funding body. Formath, 2000.

[36] F. Kamareddine, R. Lamar, M. Maarek, and J. B. Wells. Restoring Natural Language as a Computerised Mathematics Input Method. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Calculemus/MKM*, volume 4573 of *Lecture Notes in Computer Science*, pages 280–295. Springer, 2007.

[37] 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.

[38] F. Kamareddine, M. Maarek, and J. B. Wells. Toward an Object-Oriented Structure for Mathematical Text. In M. Kohlhase, editor, *MKM*, volume 3863 of *Lecture Notes in Computer Science*, pages 217–233. Springer, 2005.

[39] F. Kamareddine and R. Nederpelt. A refinement of de Bruijn's formal language of mathematics. *Logic, Language and Information*, 13(3):287–340, 2004.

[40] F. Kamareddine, J. B. Wells, and C. Zengler. Computerising mathematical texts in MathLang. Technical report, Heriot-Watt University, 2008.

[41] Khosrow-Pour and Mehdi, editors. *Encyclopedia of Information Science and Technology*. IGI Global,, 2 edition.

[42] S. King, J. Hammond, R. Chapman, and A. Pryor. Is Proof More Cost-Effective Than Testing? *IEEE Trans. Software Eng.*, 26(8):675–686, 2000.

[43] Kolyang, T. Santen, and B. Wolff. *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96 Turku, Finland, August 26–30, 1996 Proceedings*, chapter A structure preserving encoding of Z in isabelle/HOL, pages 283–298. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[44] Kolyang, T. Santen, B. Wolff, R. Chaussee, I. Gmbh, and D.-S. Augustin. Towards a Structure Preserving Encoding of Z in HOL, 1986.

[45] A. Krauss. Defining Recursive Functions in Isabelle/HOL , 2008.

[46] R. Lamar. The MathLang Formalisation Path into Isabelle – A Second-Year report, 2003.

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

[48] R. Lamar, F. Kamareddine, and J. B. Wells. MathLang Translation to Isabelle Syntax. In J. Carette, L. Dixon, C. S. Coen, and S. M. Watt, editors, *Calculemus/MKM*, volume 5625 of *Lecture Notes in Computer Science*, pages 373–388. Springer, 2009.

[49] P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl, and M. Verhoef. The overture initiative integrating tools for vdm. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6, Jan. 2010.

[50] I. Lee, J. Y.-T. Leung, and S. H. Son. *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC, 1 edition, 2007.

[51] K. R. M. Leino. Dafny: An Automatic Program Verifier for Functional Correctness. In E. M. Clarke and A. Voronkov, editors, *LPAR (Dakar)*, Lecture Notes in Computer Science, pages 348–370. Springer, 2010.

[52] M. Lindgren, C. Norstrm, A. Wall, and R. Land. Importance of Software Architecture during Release Planning. In *WICSA*, pages 253–256. IEEE Computer Society, 2008.

[53] I. E. U. Ltd and H. . S. Laboratory. A methodology for the assignment of safety integrity levels (SILs) to safety-related control functions implemented by safety-related electrical, electronic and programmable electronic control systems of machines. Standard, Health and Safety Executive (HSE), Mar. 2004.

[54] M. Maarek. Mathematical documents faithfully computerised:the grammatical and text & symbol aspects of the MathLang framework, First Year Report, 2003.

[55] M. Maarek. *Mathematical documents faithfully computerised: the grammatical and test & symbol aspects of the MathLang Framework*. PhD thesis, Heriot-Watt University, 2007.

[56] M. Mahajan. Proof Carrying Code. *INFOCOMP Journal of Computer Science*, 6(4):01–06, 2007.

[57] M. Mihaylova. ZMathLang User Interface Internship Report. Internship Report, 2015.

[58] M. Mihaylova. ZMathLang User Interface User Manual. Intern User Manual, 2015.

[59] G. C. Necula and P. L. 0001. Safe, Untrusted Agents Using Proof-Carrying Code. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 61–91. Springer, 1998.

[60] I. C. Office. International Electrotechnical Commission. `http://www.iec.ch/`, July 2016.

[61] S. Owre, S. Rajan, J. 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, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[62] R. L. Page. Engineering Software Correctness. *J. Funct. Program.*, 17(6):675–686, 2007.

[63] B. C. Pierce. *Types and Programming Languages*. MIT Press, Cambridge, MA, USA, 2002.

[64] W. R. Plugge and M. N. Perry. American Airlines' "Sabre" Electronic Reservations System. In *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '61 (Western), pages 593–602, New York, NY, USA, 1961. ACM.

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

[66] A. Riazanov and A. Voronkov. The design and implementation of vampire. *Journal of AI Communications*, 15(2/3):91–110, 2002.

[67] G. Rossum. Python Reference Manual. Technical report, Python Software Foundation, Amsterdam, The Netherlands, The Netherlands, 1995.

[68] M. Saaltink and O. Canada. The Z/EVES 2.0 User's Guide, 1999.

[69] S. Schulz. E–a brainiac theorm prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[70] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[71] M. Spivey. Z Reference Card. https://spivey.oriel.ox.ac.uk/mike/fuzz/refcard.pdf. Accessed on November 2014.

[72] M. Spivey. Towards a Formal Semantics for the Z Notation. Technical Report PRG41, OUCL, October 1984.

[73] M. Spivey. The fuzz manual. *Computing Science Consultancy*, 34, 1992.

[74] S. Stepney. A tale of two proofs. In *BCS-FACS third Northern formal methods workshop, Ilkley*, 1998.

[75] I. UK. *Customer Information Control System (CICS) Application Programmer's Reference Manual*. White Plains, New York.

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

[77] Z. Wen, H. Miao, and H. Zeng. Generating Proof Obligation to Verify Object-Z Specification. In *Proceedings of the International Conference on Software Engineering Advances (ICSEA 2006), October 28 - November 2, 2006, Papeete, Tahiti, French Polynesia*, page 38. IEEE Computer Society, 2006.

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

[79] J. Woodcock and A. Cavalcanti. A tutorial introduction to designs in unifying theories of programming. In *Integrated Formal Methods*, pages 40–66. Springer, 2004.

[80] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[81] C. Zengler. MathLang- Towards a Better Usability and Building the Path into Coq, First Year Report. Technical report, Heriot-Watt University, November 2008.