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	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

Overview of ZMathLang

Using the methodology of MathLang for mathematics (section ??), I have created and implemented a step by step way of translating Z specifications into theorem provers with additional checks for correctness along the way. This translation consists of one large framework (executed by a user interface) with many smaller tools to assist the translation. Not only is the translation useful for a novice to translate a formal specification into a theorem prover but it also creates other diagrams and graphs to help with the analysis of a formal system specification.

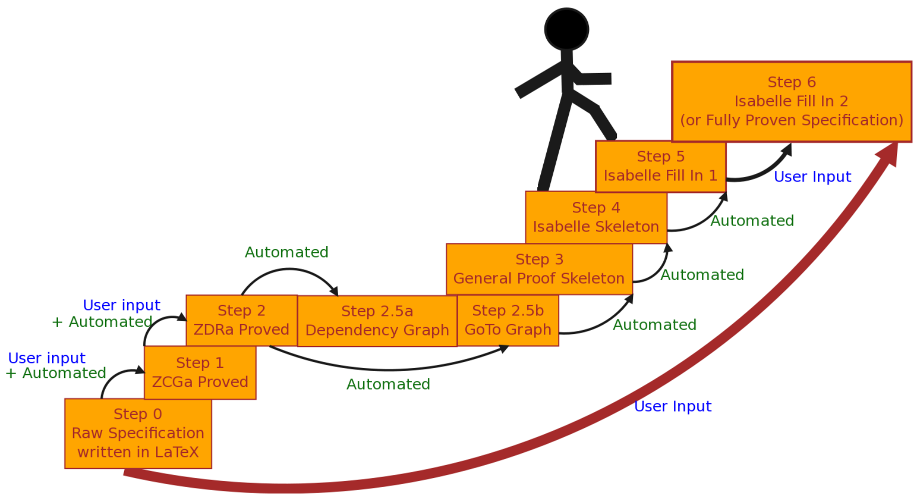


Figure 1.1: The steps required to obtain a full proof from a raw specification.

The framework is targeted at beginners in theorem proving. The users should have some idea of formal specifications but no or little knowledge of the targetted theorem prover. Figure 1.1 shows the outline of the framework. The higher the user goes up the steps the more rigorous the checks for correctness. Step 1 and step 2 are interchangable and can be done in any order. However they both must be completed before moving up to step 3. Step 6 is the highest level of rigour and checks for full correctness in a theorem prover. For this thesis I have chose to translate Z specifications into Isabelle, however this framework is an outline for any formal specification into any theorem prover which could done in the future.

The user doesn't need to go all the way to the top to check for correctness, one advantage of breaking up the translation is that the user gets some level of rigour and can be satisfied with some level of correctness along the way. However the main advantage of breaking up the translation is that the level of expertise needed to check for the correctness of a system specification can be done by someone who has little or no expertise in checking for correctness by a theorem prover.This tool could also aid user in learning theorem proving as it translates their specification and thus they have examples of the syntx used in their theorem prover for their specification. The small black arrows represent the amount of expertise needed for each step. The last step the arrow is slightly thicker as some theorem prover knowledge is needed. However these arrows are still small in comparison to the red thick arrow which represents the translation in one big step.

The framework breaks the translation into 6 steps most of which are partially or fully automated. These are:

- Step 0: Raw LaTeX Z Specification. Start
- Step 1: Check for Core Grammatical correctness (ZCGa). User Input + Automated
- Step 2: Check for Document Rhetorical correctness (ZDRa). User Input + Automated
- Step 3: Generate a General Proof Skeleton (GPSa). Automated
- Step 4: Generate an Isabelle Skeleton. Automated
- Step 5: Fill in the Isabelle Skeleton. Automated

- Step 6: Prove existing lemmas and add more safety properties if needed. User Input

1.1 How far does the automation go?

Figure 1.2 shows a diagram showing how far one can automate a specification on either side. MathLang framework for Z specifications (ZMathLang) is a toolset which assists the user translating and proving a specification (going from left to right). There are also other automating tools within Isabelle which also assist the user with proving specification (going from right to left) in the diagram.

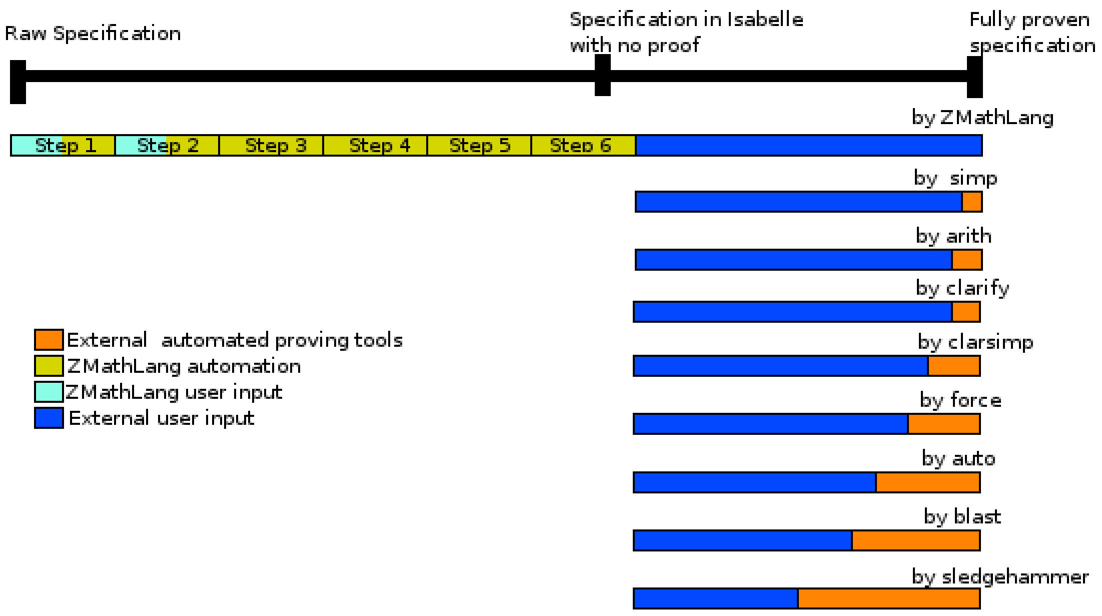


Figure 1.2: How far can one automate a specification proof.

In figure 1.2 we show how far the user can get with automation and how much work is still needed to get the full proof. ZMathLang requires user input for the first two steps (Z Core Grammatical aspect (ZCGa) and Z Document Rhetorical aspect (ZDRa)) however the rest is automated.

The black line shows the path going from a raw specification to a fully proven specification with a milestone in the middle, which signifies when a specification has been translated into Isabelle syntax but yet has no properties or proof. ZMathLang takes the user a little past this milestone as the toolset also generates properties to check the specification for consistency (see chapter **??** section **??**). These properties are added to the specification during step 3 and continued throughout the translation. It is important to note that the ZMathLang toolkit adds these properties to the translation but does not prove them. That is why the rest of the ZMathLang path may require external user input (dark blue) to complete the path. However, the ZMathLang toolkit does assist the user in the translation past the halfway milestone on the diagram.

We have created the ZMathLang toolkit which assist the user from the specification to full proof however there is also ongoing research on proving properties from the theorem prover end. Figure 1.2 shows the amount of proving techniques each automation holds. We have highlighted that ZMathLang only gets the user so far in their proof however they are free to use external automated theorem provers in completing their specification proof.

Even external automated theorem provers have their limitations. For example, the user can use the Isabelle tool '*sledgehammer*' assists the user automatically solving some proofs, but not all. The sledgehammer documentation advises to call '*auto*' or '*safe*' followed by '*simp_all*' before invoking sledgehammer. Depending on the complexity of your proof, this sometimes may prove the users properties, other times it may not and the user will still need to invoke sledgehammer to assist in reaching their goal. Sledgehammer itself is a tool that applies Satisfiability Modulo Theories (SMT) solvers on the current goal e.g. Vampire[4], SPASS [3] and E [5]. We use sledghammer as a collective, to describe all the SMT solvers it covers [1].

Other automated methods include:

- **simp:** simplifies the current goal using term rewriting.
- **arith:** automatically solves linear arithmetic problems.
- **clarify:** like auto but less aggressive.
- **clarsimp:** a combination of clarify and simp.
- **force:** like auto but only applied to the first goal.

- **auto:** applies automated tools to look for a solution.
- **blast:** a powerful first-order prover. [2]

All these automated tools get increasingly further by automating proofs, e.g *clarsimp* covers more proving techniques then *simp* and *blast* covers more proving techniques than *auto* etc. With these tools, one can prove certain properties about their theorem. However, there still doesn't exist an automated proving tool which covers **all** proving techniques. Therefore some user input will be required for more complex proofs.

1.2 Overview of ZMathLang step by step

This section gives an overview of each indvidual step in ZMathLang.

1.2.1 Step 0- The raw LaTeX file

The first step requires the user to write or have a formal specification they wish to check for correctness. This specification can be fully written in Z or partially written in Z (thus a specification written in english on the way to becoming formalised in Z). The specification should be written in LaTeX format and can be a mix of natural language and Z. An example of a specification written in the Z notation can be seen in figure 1.3.

$[NAME, DATE]$

$$\begin{array}{|l}
\hline
\textit{BirthdayBook} \\
known : \mathbb{P}\, NAME \\
birthday : NAME \rightarrowtail DATE \\
\hline
known = \operatorname{dom} birthday \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{InitBirthdayBook} \\
BirthdayBook' \\
\hline
known' = \{\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{AddBirthday} \\
\Delta BirthdayBook \\
name? : NAME \\
date? : DATE \\
\end{array}$$

Figure 1.3: Example of a partial Z specification.

1.2.2 Step 1- The Core Grammatical aspect for Z

The next step in figure 1.1 shows the specification should be ZCGa proved. Although this step is interchangable with step 2 (ZDRa) it is shown as step 2 on the diagram for convinience. In this step the user annoates their document which they have obtained in step 0 with 7 categories and then checks these for correctness. Figure 1.1 show this step is achieved by user input and automation. The user input of this step is the annotations and the automation is the ZCGa checker. This automatically produces a document labeled with the various categories in difference colours and can help identify grammar types to other members interested in the specification. A ZCGa annotated specification is shown in figure 1.4. The ZCGa is further explained in chapter **??**.

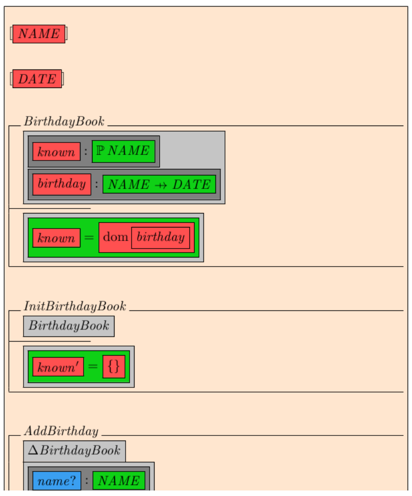


Figure 1.4: Example of a ZCGa annotated specification.

1.2.3 Step 2- The document Rhetorical aspect for Z

The ZDRa (chapter **??**) step shown as step 2 in figure 1.1 comes before or after the ZCGa step. Similarly to the ZCGa step the user annotates their document from step 0 or step 1 with ZDRa instances and relationships. This chunks parts of the specification and allows the user to describe the relationship between these chunks of specification. The annotation is the user input part of this step and the automation is the ZDRa checker which checks if there are any loops in the reasoning and give warnings if the specification still needs to be totalised. Once the user has annotated this document and compiled it the outputing result shows the specification divided into chunks and arrows showing the relations between the chunks. An example of a Z specification annotated in ZDRa is shown in figure 1.5.

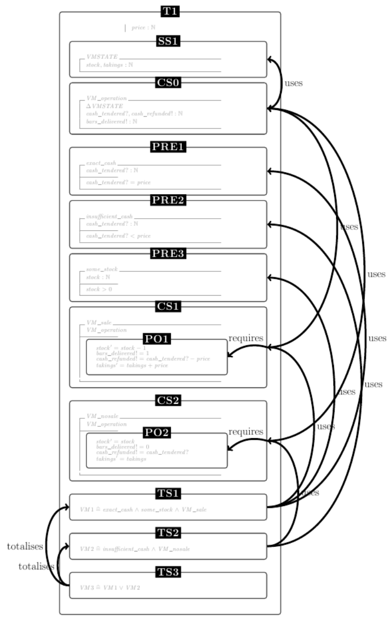


Figure 1.5: Example of a ZDRa annotated specification.

The ZDRa automatically produces a dependency and a goto graph (section **??**), these a shown as 2.5a and 2.5b respectively in figure 1.1. The loops in reasoning are checked in both the dependecy graph and goto graph. An example of a goto graph is shown in figure 1.6.

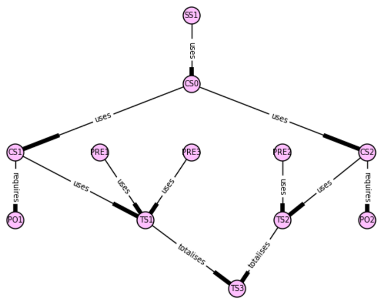
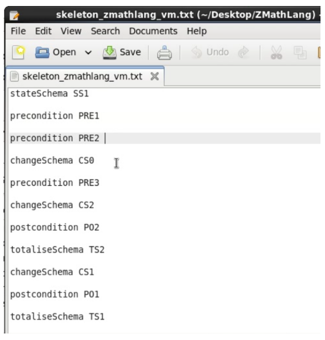


Figure 1.6: Example of an automatically generated goto graph.

1.2.4 Step 3- The General Proof skeleton

The following step is an automatically generated General Proof Skeleton aspect (Gpsa). This document is automated using the goto graph which is generated from the ZDRa annotated LaTeX specification. It uses the goto graph to describe in which logical order to input the specification into any theorem prover. At this stage it also adds simple proof obligations to check for the consitancy of the specification i.e. the specification is not conflictive each part. An example of a general proof skeleton is shown in figure 1.7. The Gpsa is further described in section ??.

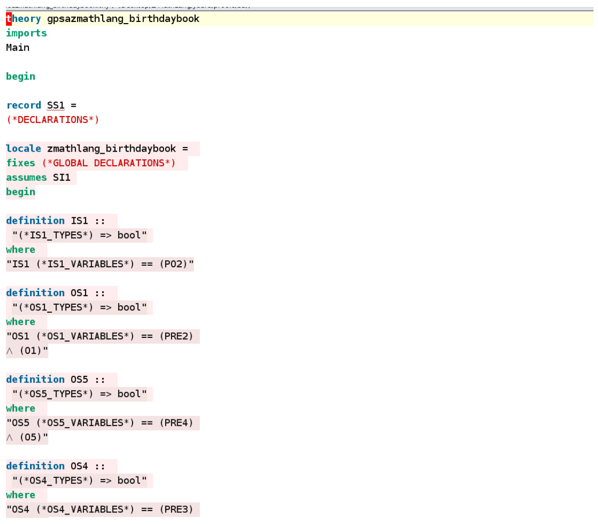

skeleton_zmathlang_vm.txt (~/Desktop/ZMathLang)
File Edit View Search Documents Help
Open
Save
Undo
skeleton_zmathlang_vm.txt

```
stateSchema SS1

precondition PRE1

precondition PRE2

changeSchema CS0

precondition PRE3

changeSchema CS2

postcondition PO2

totaliseSchema TS2

changeSchema CS1

postcondition PO1

totaliseSchema TS1
```

Figure 1.7: Example of a general proof skeleton.

1.2.5 Step 4- The Z specification written as an Isabelle Skeleton

Using the Gpsa in step 3, the instances are then translated into an Isabelle skeleton in step 4. That is the instances of the specification are translated into Isabelle syntax using definitions, lemma's, theorys etc to produce a .thy file. This step is fully automated and thus a user with no Isabelle experience can still get to this stage. An example of a Z specification skeleton written in Isabelle is shown in figure 1.8. Details of how this translation is conducted is described in section ?? of this thesis.

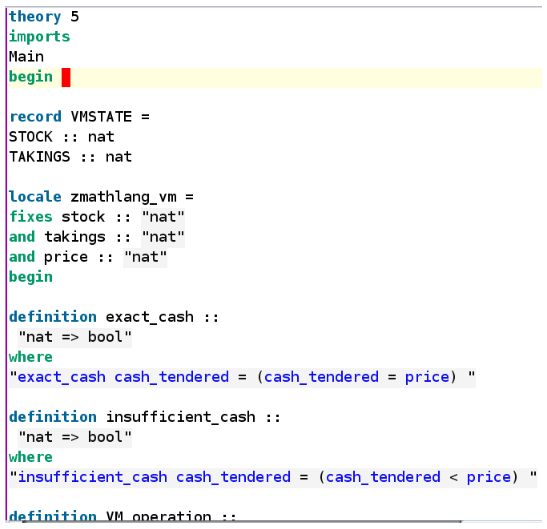


```
theory gpsazmathlang_birthdaybook
imports
Main

begin

record SS1 =
(*DECLARATIONS*)

locale zmathlang_birthdaybook =
fixes (*GLOBAL DECLARATIONS*)
assumes SI1
begin

definition IS1 ::
 "(*IS1_TYPES*) => bool"
where
"IS1 (*IS1_VARIABLES*) == (PO2)"

definition OS1 ::
 "(*OS1_TYPES*) => bool"
where
"OS1 (*OS1_VARIABLES*) == (PRE2)
∧ (O1)"

definition OS5 ::
 "(*OS5_TYPES*) => bool"
where
"OS5 (*OS5_VARIABLES*) == (PRE4)
∧ (O5)"

definition OS4 ::
 "(*OS4_TYPES*) => bool"
where
"OS4 (*OS4_VARIABLES*) == (PRE3)
```

Figure 1.8: Example of an Isabelle skeleton.

1.2.6 Step 5- The Z specification written as in Isabelle Syntax

Step 5 is also automated, using the ZCGa annotated document produced in step 1 and the Isabelle skeleton produced in step 4. This part of the framework fills in the details from the specification using all the declarations, expressions, definition etc in Isabelle syntax. Since the translation can also be done on semi-formal specifications and incomplete formal specification there may be some information missing in the ZCGa such as an expression or a definition. Note the lemmas from the proof obligations created in step 3 will also be filled in, however the actual proofs for these will not and they will be followed by the command '`sorry`' to artificially complete proofs. An example of a filled in isabelle skeleton is shown in figure 1.9.



```
theory 5
imports
Main
begin

record VMSTATE =
STOCK :: nat
TAKINGS :: nat

locale zmathlang_vm =
fixes stock :: "nat"
and takings :: "nat"
and price :: "nat"
begin

definition exact_cash ::
 "nat => bool"
where
"exact_cash cash_tendered = (cash_tendered = price) "

definition insufficient_cash ::
 "nat => bool"
where
"insufficient_cash cash_tendered = (cash_tendered < price) "

definition VM_operation ::
```

Figure 1.9: Example of an Isabelle skeleton automatically filled in.

In this case the Isabelle skeleton will not change. Further information on the translation is described in section **??** of this thesis.

1.2.7 Step 6- A fully proven Z specification

The final step in the ZMathLang framework and the top of the stairs from figure 1.1 is to fill in the Isabelle file from step 5. This final step is represented by a slightly thicker arrow in figure 1.1 compared with the others as the user may need to have some little theorem prover knowledge to prove properties about the specification. Also if there is some missing information such as missing expressions and definitions the user must fill these out as well in order to have a fully proven specification. However this may be slightly easier then writing the specification from scratch in Isabelle as the user would allready have examples of other instances in their Isabelle syntax form. More details on this last step is described in section **??** of this thesis.

1.3 Procedures and products within ZMathLang

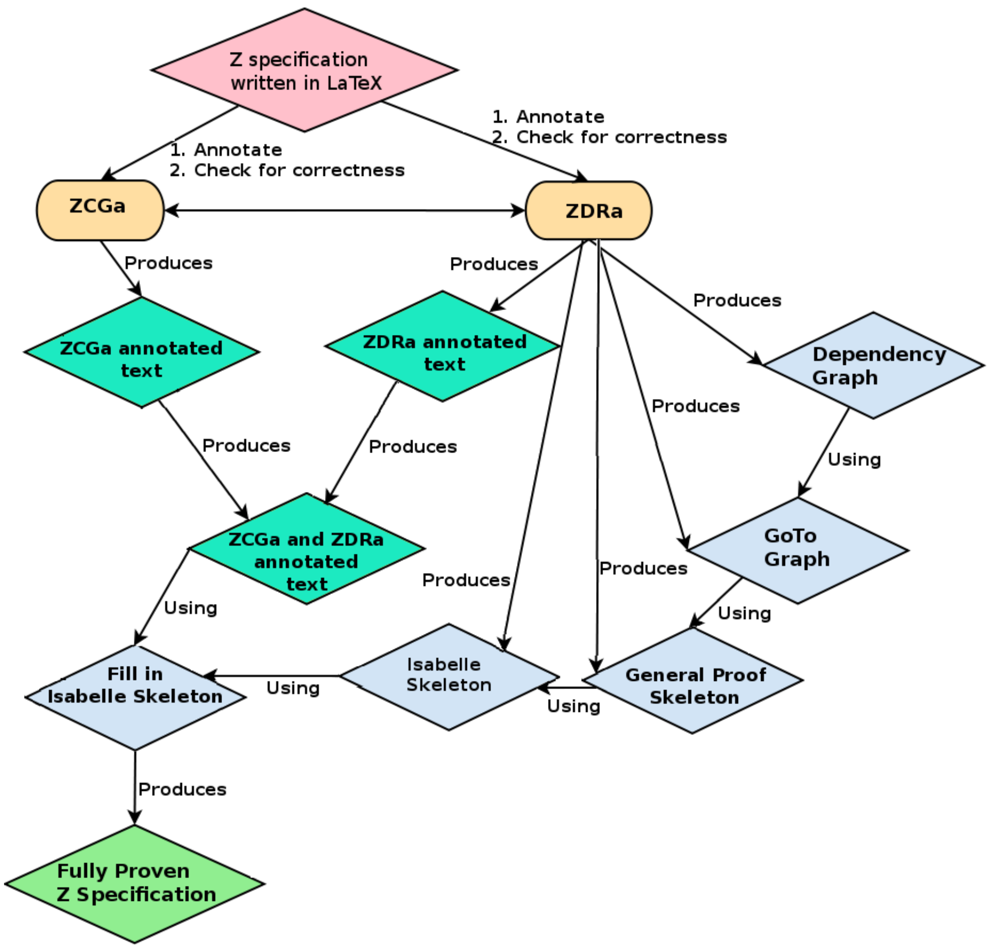

Z specification written in LaTeX
1. Annotate 2. Check for correctness
1. Annotate 2. Check for correctness
ZCGa
ZDRa
Produces
ZCGa annotated text
ZDRa annotated text
Dependency Graph
Using
GoTo Graph
ZCGa and ZDRa annotated text
Using
Fill in Isabelle Skeleton
Isabelle Skeleton
General Proof Skeleton
Fully Proven Z Specification

Figure 1.10: Flow chart of ZMathLang.

Figure 1.10 shows a flow chart describing the documents produced from using the framework and which parts are fully automated, partially automated and user input. Products which are created by full automation are diamonds in blue. Diamonds in green are produced by user input and products shown in aqua diamonds are partial automated.

The pink diamond is the starting point for all users. The orange ovals describe procedures of the ZCGa and ZDRa. The ZCGa procedure requires user input and automation and produces a 'ZCGa annotated text'. The ZDRa procedure requires user input to annotated and the check is automated. Both the ZCGa and ZDRa procedures done together produce a 'ZCGa and ZDRa annotated text'. After completing the ZDRa procedure a 'dependency graph' is automatically generated, which can then in turn generate a 'GoTo graph' which in turn can create a general proof skeleton. From the 'general proof skeleton' we can then create an 'Isabelle skeleton' which can be filled in using information from the 'ZCGa and ZDRa annotated text'. Using the 'Filled in Isabelle skeleton' the user needs to fill in the missing information to obtain a 'fully proven Z specification'.

1.4 The ZMathLang LaTeX Package

The ZMathLang LaTeX package (shown in appendix **??**) was implemented to allow the user to annotate their Z specification document in ZCGa and ZDRa annotations. Coloured boxes will then appear around the grammatical categories when the new ZCGa annotated document is compiled with `pdflatex`. Instances and labelled arrows showing the relations are also displayed when annotated with ZDRa and compiled with `pdflatex`.

1.4.1 Overview

The ZMathLang style file invokes the following packages:

- tcolorbox - Used to draw colours around individual grammatical categories with a black outline for the ZCGa.
- tikz - Used to identify the instances as nodes so the arrows can join from one nodes to another.
- varwidth - Used to chunk each instance as a single entity.
- zed - Used to draw Z specification schemas, freetypes, axiomatic definitions in the zed environment.
- xcolor - Used to define specific colours and gives a wider range of colours compared to the standard.

After invoking the packages we define the colours which are used in the outputting pdf result. We use the same colours as the original MathLang framework for mathematics (MathLang) framework for the grammatical categories which are the same (sets, terms, expressions, declarations, context and definitions) and choose a different colour for the weak type 'specification' as this hasn't been used in the original MathLang framework.

```
\definecolor{term}{HTML}{3A9FF1}
```

Figure 1.11: Part of the syntax to define the colours for ZCGa in the ZMathLang LaTeX file.

The command `\definecolor{*NameOfZCGaType*}{HTML}{*ColourInHtml*}` is used to define a colour for each grammatical category (shown in figure 1.11). Where `*NameOfZCGaType*` is the name of the category i.e. definition, term, set etc and `*ColourInHtml*` is the HTML number for the colour. For example the colour for term in the original ZMathLang is *lightblue* which in HTML format is *3A9FF1*. Therefore we define the colour for *term* as *3A9FF1*.

1.4.2 LaTeX commands to identify ZDRa Instances

The ZDRa section of the LaTeX file provides three new commands: `\draschema`, `\draline` and `\dratheory`. The `\dratheory` annotation is for the entire specification which contins all the intances and relations. The `\draschema` command is to annotate the instances which are entire zed schemas, this command should go before any `\begin{schema}` or `\begin{zed}` command. The `\draline` annotation is to annotate any instance that is a line of text which contains plain text or ZCGa annotated text. But does not include any ZDRa annotated text. For example in figure 1.12 the `\draline{PRE1}` annotation is embedded in the `\draline{CS1}{` which will not compile. Therefore the correct way this schema is labelled is shown in figure 1.13 where the `\draline{PRE1}` annotation is embedded in the `\draschema{CS1}` annotation.

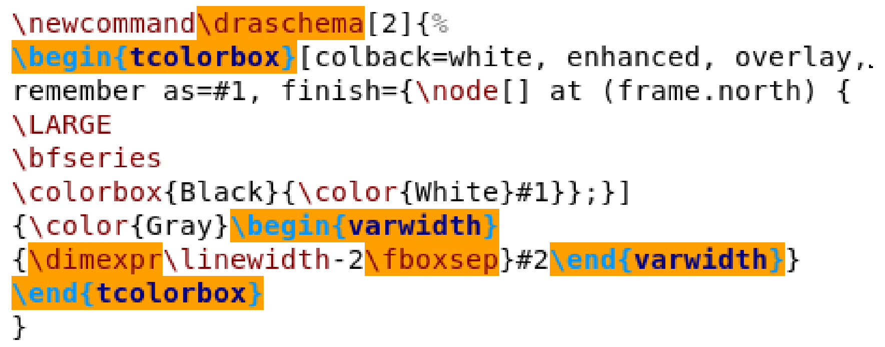
```
\draline{CS1}{

\begin{schema}{B}

\Delta A

\where

\draline{PRE1}{a<b}

\end{schema}

}
```

Figure 1.12: Incorrect annotating of ZDRa.

```
\draschema{CS1}{

\begin{schema}{B}

\Delta A

\where

\draline{PRE1}{a<b}

\end{schema}

}
```

Figure 1.13: Correct annotating of ZDRa.

It is important to note this embedding order as by annotating a chunk of specification using the ZDRa annotation `\draline` it keeps the inside of the part inside as maths mode. Since the annotation `\draschema` is outside the zed commands (eg `\begin{schema}`) then the zed commands make everything inside the instance into math mode.



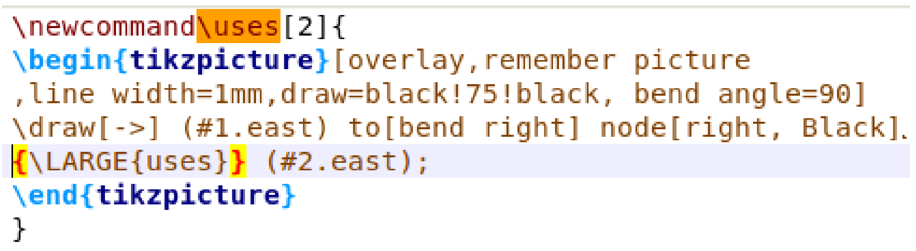
```
\newcommand\draschema[2]{%
\begin{tcolorbox}[colback=white, enhanced, overlay,.
remember as=#1, finish={\node[] at (frame.north) {
\LARGE
\bfseries
\colorbox{Black}{\color{White}#1}};}]
{\color{Gray}\begin{varwidth}
{\dimexpr\linewidth-2\fboxsep}#2\end{varwidth}}
\end{tcolorbox}
}
```

Figure 1.14: The syntax to define a ZDRa schema instance in the ZMathLang LaTeX file.

The new command we are defining for `\draschema` is shown in figure 1.14. The commands for defining `\dratheory` and `\draline` are similar as the `draschema` definition. The command takes two arguments, the first argument will be the name of the instance (e.g SS1, IS4, CS2 etc) and the second argument is the instance itself. Any text within the instance will then become grey so it looks faded as we are only interested in the instance itself and not the context at this point. The background of the box is white with a black outline. We then use the first argument to name the instance and it becomes a node. The name of the instance is also printed in black over the instance itself.

1.4.3 LaTeX commands to identify ZDRa Relations

There are 5 new commands to define the relations for the ZDRa, these are *initialOf*, *uses*, *totalises*, *requires* and *requires*. Information on these relations are described in chapter **??**, however this section of the thesis describes the LaTeX commands implemented so that they can be used to annotated a specification in ZDRa.



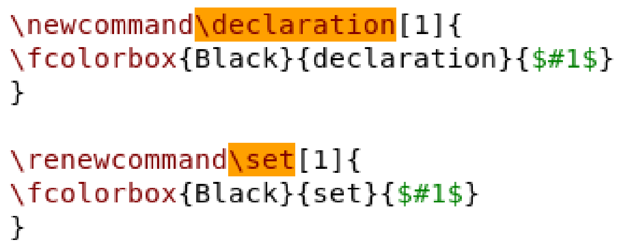
```
\newcommand\uses[2]{
\begin{tikzpicture}[overlay,remember picture
,line width=1mm,draw=black!75!black, bend angle=90]
\draw[->] (#1.east) to[bend right] node[right, Black]
{\LARGE{uses}} (#2.east);
\end{tikzpicture}
}
```

Figure 1.15: The syntax to define a ZDRa schema relation in the ZMathLang LaTeX file.

Figure 1.15 shows how the command `uses` has been implemented. The command takes 2 arguments (which are 2 instances which have been previously annotated) and draws an arrow going from the first instance to the second one. The arrow bend angle is at 90, the arrow width is at 1mm and the arrow goes from the east part of the first instance to the east part of the second instance. The word **uses** is written next to the arrow. All the other relation commands are written in a similar way however the direction of the arrows differ and some arrows bend to the left whilst others bend to the right. The bending of the arrows has been implemented at random so that the compiled document has arrows showing on both sides of the theory and are not overlapping too much.

1.4.4 LaTeX commands to identify ZCGa grammatical types

The ZCGa part of the LaTeX file package uses the colours previously defined in the style file. To define each of the grammatical types we use the `fcolorbox` command. This creates a black outline and a coloured background for each of the grammatical categories.



```
\newcommand\declaration[1]{
\fcolorbox{Black}{declaration}{$#1$}
}

\renewcommand\set[1]{
\fcolorbox{Black}{set}{$#1$}
}
```

Figure 1.16: The syntax to define a ZCGa grammatical categories.

Figure 1.16 shows the commands to define the coloured boxes for *declaration* and *set*. As set is already defined in the mathematical LaTeX library, we redefine the command. The command takes one argument (the text the user which to annotate), changes it to mathmode and draws the box around it. All the grammatical categories are defined in the same way, each with their own background colour. The only exception is the grammatical category of *specification* as this command does not convert the specification into mathmode as it is already in mathmode.

1.5 Conclusion

In total there are 6 steps in order to translate a Z specification into the theorem prover Isabelle. Each of these steps assist the user in understand the specification more, and some steps even produce documents, graphs and charts in order to analyse the specification. These products also allow others in the development team to understand the system better such as clients, stakeholders, developers etc. The majority of the steps are fully automated whilst some a little user input. The next chapter begins to describe step 1 (ZCGa) in more detail.

Bibliography

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

[2] 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`.

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

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

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