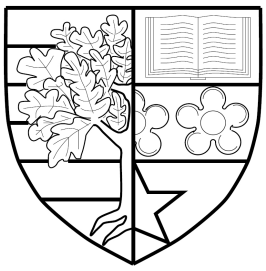
FROM FORMAL SPECIFICATION TO FULL PROOF: A STEPWISE METHOD

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

Write an abstract

Acknowledgements

Contents

List of Tables

List of Figures

Todo list

Acronyms

CGa Core Grammatical aspect.

DRa Document Rhetorical aspect.

TSa Text and Symbol aspect.

Appendix A

Specifications in ZMathLang

A.1 Vending Machine

A.1.1 Raw Latex

```
\documentclass{article}
\usepackage{zed}
\begin{document}
\begin{zed}
price:\nat
\end{zed}

\begin{schema}{VMSTATE}
stock, takings: \nat
\end{schema}

\begin{schema}{VM\_operation}
\Delta VMSTATE \\
cash\_tendered?, cash\_refunded!: \nat \\
bars\_delivered! : \nat
\end{schema}

\begin{schema}{exact\_cash}
cash\_tendered?: \nat
\where
cash\_tendered? = price
\end{schema}

\begin{schema}{insufficient\_cash}
cash\_tendered? : \nat
\where
cash\_tendered? < price
\end{schema}

\begin{schema}{some\_stock}
stock: \nat
\where
stock > 0
\end{schema}

\begin{schema}{VM\_sale}
VM\_operation
\where
stock' = stock -1 \\
bars\_delivered! = 1 \\
cash\_refunded! = cash\_tendered? - price \\
takings' = takings + price
\end{schema}
```

```
\begin{schema}{VM\_nosale}
VM\_operation
\where
stock' = stock \\
bars\_delivered! = 0 \\
cash\_refunded! = cash\_tendered?\\
takings' = takings
\end{schema}

\begin{zed}
VM1 \defs exact\_cash \land some\_stock \land VM\_sale
\end{zed}

\begin{zed}
VM2 \defs insufficient\_cash \land VM\_nosale
\end{zed}

\begin{zed}
VM3 \defs VM1 \lor VM2
\end{zed}
\end{document}
```

A.1.2 Raw Latex output

$$
\begin{array}{|l}
\hline
\textit{VMSTATE} \\
\hline
stock, takings : \mathbb{N} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{VM\_operation} \\
\hline
\Delta VMSTATE \\
cash\_tendered?, cash\_refunded! : \mathbb{N} \\
bars\_delivered! : \mathbb{N} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{exact\_cash} \\
\hline
cash\_tendered? : \mathbb{N} \\
\hline
cash\_tendered? = price \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{insufficient\_cash} \\
\hline
cash\_tendered? : \mathbb{N} \\
\hline
cash\_tendered? < price \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{some\_stock} \\
\hline
stock : \mathbb{N} \\
\hline
stock > 0 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{VM\_sale} \\
\hline
VM\_operation \\
\hline
stock' = stock - 1 \\
bars\_delivered! = 1 \\
cash\_refunded! = cash\_tendered? - price \\
takings' = takings + price \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{VM\_nosale} \\
\hline
VM\_operation \\
\hline
stock' = stock \\
bars\_delivered! = 0 \\
cash\_refunded! = cash\_tendered? \\
takings' = takings \\
\hline
\end{array}
$$

$$VM1 \mathrel{\widehat=} exact\_cash \land some\_stock \land VM\_sale$$

$$VM2 \mathrel{\widehat=} insufficient\_cash \land VM\_nosale$$

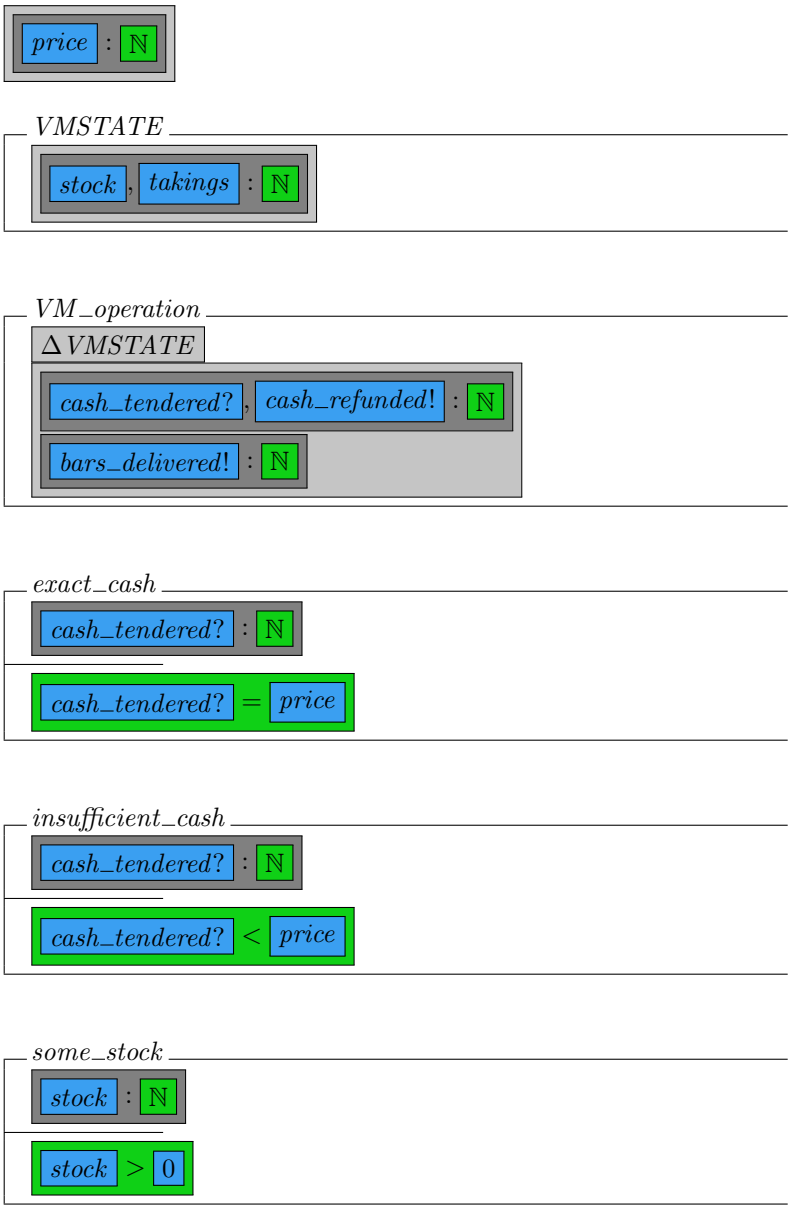
$$VM3 \mathrel{\widehat=} VM1 \lor VM2$$

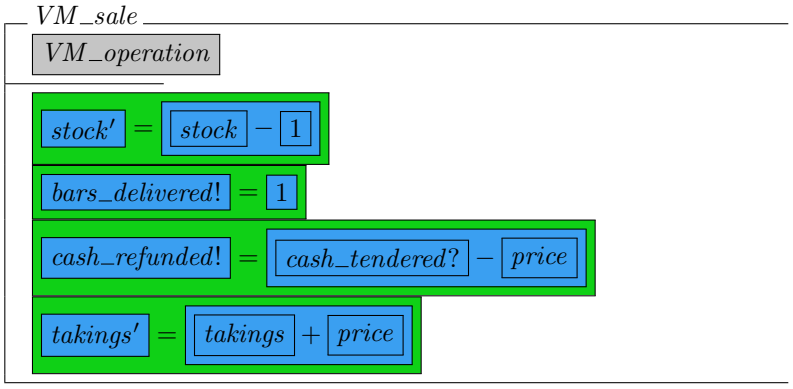
A.1.3 ZCGa Annotated Latex Code

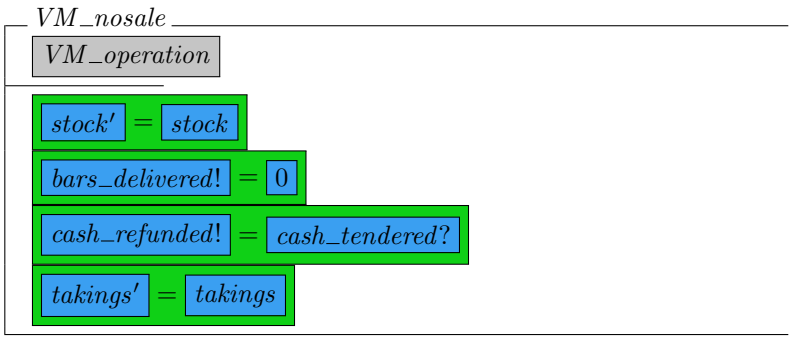
```
\documentclass{article}
\usepackage{zmathlang}
\begin{document}

\begin{zed}
\text{\declaration{\term{price}: \expression{\nat}}}
\end{zed}

\begin{schema}{VMSTATE}
\text{\declaration{\term{stock}, \term{takings}: \expression{\nat}}}
\end{schema}

\begin{schema}{VM\_operation}
\text{\Delta VMSTATE} \\
\text{\declaration{\term{cash\_tendered?},
\term{cash\_refunded!}: \expression{\nat}} \\
\declaration{\term{bars\_delivered!}: \expression{\nat}}}
\end{schema}

\begin{schema}{exact\_cash}
\declaration{\term{cash\_tendered?}: \expression{\nat}}
\where
\expression{\term{cash\_tendered?} = \term{price}}
\end{schema}

\begin{schema}{insufficient\_cash}
\declaration{\term{cash\_tendered?}: \expression{\nat}}
\where
\expression{\term{cash\_tendered?} < \term{price}}
\end{schema}

\begin{schema}{some\_stock}
\declaration{\term{stock}: \expression{\nat}}
\where
\expression{\term{stock} > \term{0}}
\end{schema}

\begin{schema}{VM\_sale}
\text{VM\_operation}
\where
\expression{\term{stock'} = \term{\term{stock} - \term{1}}} \\
\expression{\term{bars\_delivered!} = \term{1}} \\
\expression{\term{cash\_refunded!} =
\term{\term{cash\_tendered?} - \term{price}}} \\
\expression{\term{takings'} = \term{\term{takings} + \term{price}}}
\end{schema}
```

```
\begin{schema}{VM\_nosale}
\text{VM\_operation}
\where
\expression{\term{stock'} = \term{stock}} \\
\expression{\term{bars\_delivered!} = \term{0}} \\
\expression{\term{cash\_refunded!} = \term{cash\_tendered?}}\\
\expression{\term{takings'} = \term{takings}}
\end{schema}

\begin{zed}
VM1 \defs \text{\expression{\text{exact\_cash} \land
\text{some\_stock} \land \text{VM\_sale}}}
\end{zed}

\begin{zed}
VM2 \defs \text{\expression{\text{insufficient\_cash} \land
\text{VM\_nosale}}}
\end{zed}

\begin{zed}
VM3 \defs \text{\expression{\text{VM1} \lor \text{VM2}}}
\end{zed}

\end{document}
```

A.1.4 ZCGa output







$VM1 \mathrel{\widehat=} exact\_cash \land some\_stock \land VM\_sale$

$VM2 \mathrel{\widehat=} insufficient\_cash \land VM\_nosale$

$VM3 \mathrel{\widehat=} VM1 \lor VM2$

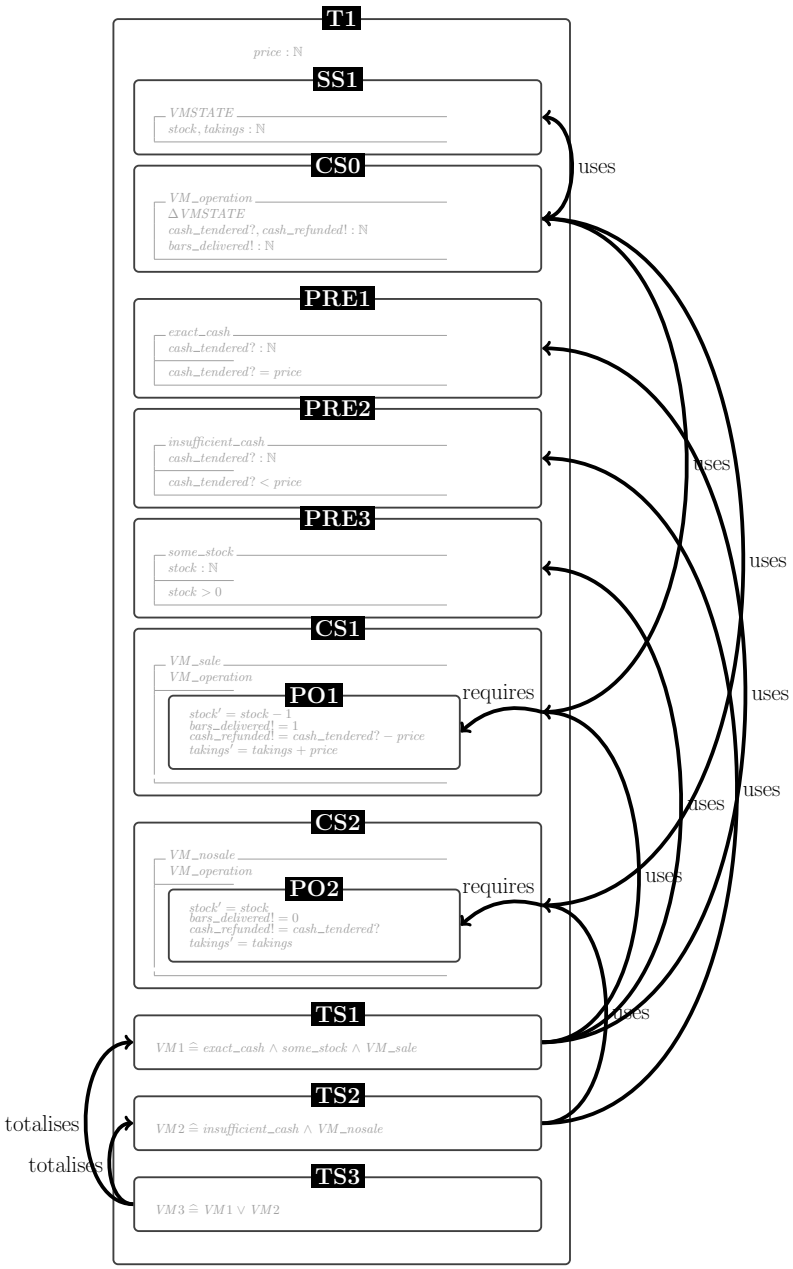
A.1.5 ZDRa Annotated Latex Code

```
\documentclass{article}
\usepackage{zmathlang}
\begin{document}

\dratheory{T1}{0.5}{

\begin{zed}
price:\nat
\end{zed}

\draschema{SS1}{
\begin{schema}{VMSTATE}
stock, takings: \nat
\end{schema}}

\draschema{CS0}{
\begin{schema}{VM\_operation}
\Delta VMSTATE \\
cash\_tendered?, cash\_refunded!: \nat \\
bars\_delivered! : \nat
\end{schema}}

\uses{CS0}{SS1}

\draschema{PRE1}{
\begin{schema}{exact\_cash}
cash\_tendered?: \nat
\where
cash\_tendered? = price
\end{schema}}

\draschema{PRE2}{
\begin{schema}{insufficient\_cash}
cash\_tendered? : \nat
\where
cash\_tendered? < price
\end{schema}}

\draschema{PRE3}{
\begin{schema}{some\_stock}
stock: \nat
\where
stock > 0
\end{schema}}
```

```
\draschema{CS1}{
\begin{schema}{VM\_sale}
VM\_operation
\where
\draline{PO1}{stock' = stock -1 \\
bars\_delivered! = 1 \\
cash\_refunded! = cash\_tendered? - price \\
takings' = takings + price}
\end{schema}}

\uses{CS1}{CS0}
\requires{CS1}{PO1}

\draschema{CS2}{
\begin{schema}{VM\_nosale}
VM\_operation
\where
\draline{PO2}{stock' = stock \\
bars\_delivered! = 0 \\
cash\_refunded! = cash\_tendered?\\
takings' = takings}
\end{schema}}

\uses{CS2}{CS0}
\requires{CS2}{PO2}

\draschema{TS1}{
\begin{zed}
VM1 \defs exact\_cash \land some\_stock \land VM\_sale
\end{zed}}

\uses{TS1}{PRE1}
\uses{TS1}{PRE3}
\uses{TS1}{CS1}

\draschema{TS2}{
\begin{zed}
VM2 \defs insufficient\_cash \land VM\_nosale
\end{zed}}

\uses{TS2}{PRE2}
\uses{TS2}{CS2}
```

```
\draschema{TS3}{
\begin{zed}
VM3 \defs VM1 \lor VM2
\end{zed}}

\totalises{TS3}{TS1}
\totalises{TS3}{TS2}
}
\end{document}
```

A.1.6 ZDRa Output

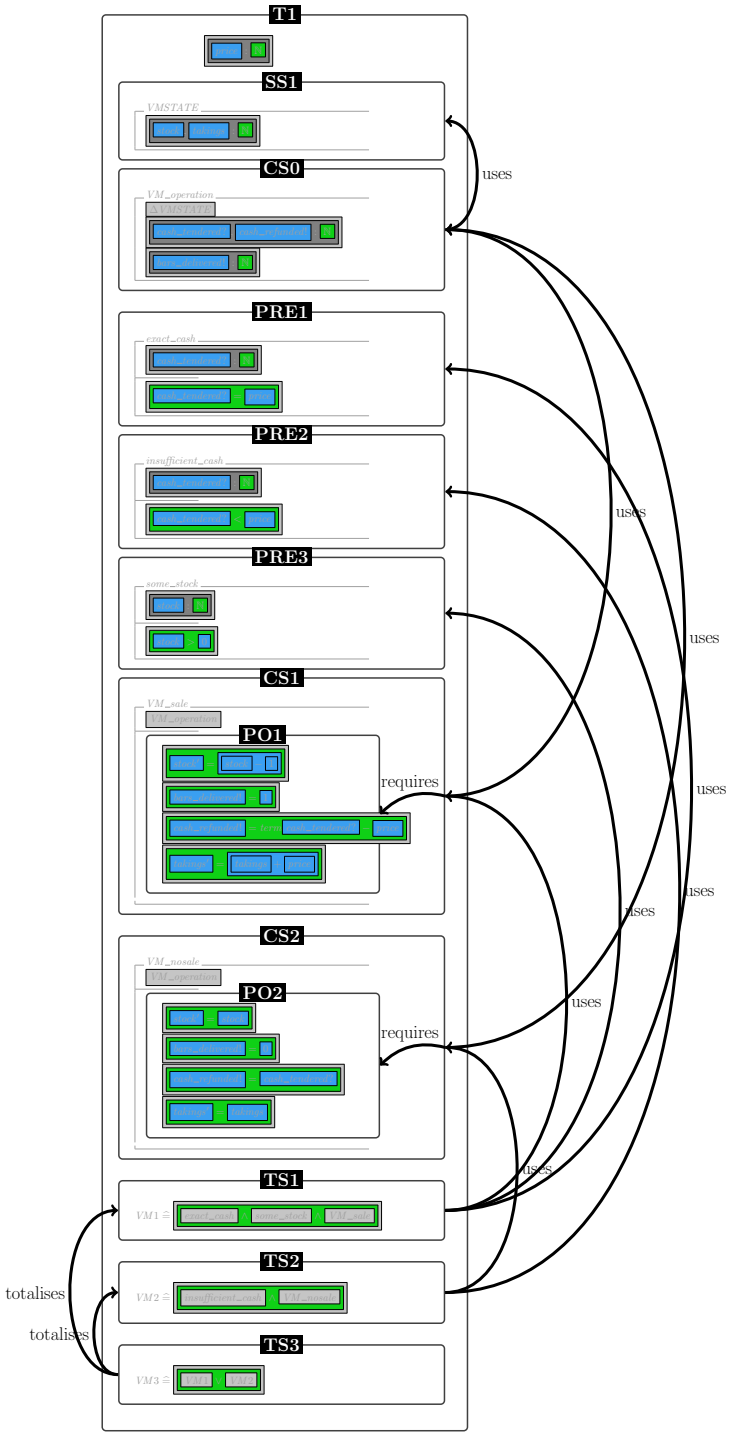


A.1.7 ZCGa and ZDRa Annotated Latex Code

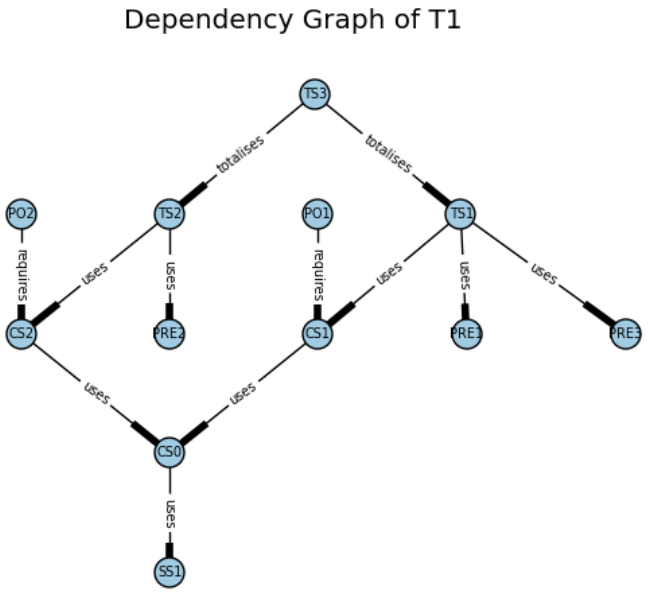
```
\documentclass{article}
\usepackage{zmathlang}
\begin{document}

\dratheory{T1}{0.5}{
\begin{zed}
\text{\declaration{\term{price}:\expression{\nat}}}
\end{zed}

\draschema{SS1}{
\begin{schema}{VMSTATE}
\text{\declaration{\term{stock}, \term{takings}: \expression{\nat}}}
\end{schema}}

\draschema{CS0}{
\begin{schema}{VM\_operation}
\text{\Delta VMSTATE} \\
\text{\declaration{\term{cash\_tendered?}, \term{cash\_refunded!}: \expression{\nat}}} \\
\text{\declaration{\term{bars\_delivered!}: \expression{\nat}}}
\end{schema}}

\uses{CS0}{SS1}

\draschema{PRE1}{
\begin{schema}{exact\_cash}
\text{\declaration{\term{cash\_tendered?}: \expression{\nat}}}
\where
\text{\expression{\term{cash\_tendered?} = \term{price}}}
\end{schema}}

\draschema{PRE2}{
\begin{schema}{insufficient\_cash}
\text{\declaration{\term{cash\_tendered?}: \expression{\nat}}}
\where
\text{\expression{\term{cash\_tendered?} < \term{price}}}
\end{schema}}

\draschema{PRE3}{
\begin{schema}{some\_stock}
\text{\declaration{\term{stock}: \expression{\nat}}}
\where
\text{\expression{\term{stock} > \term{0}}}
\end{schema}}
```

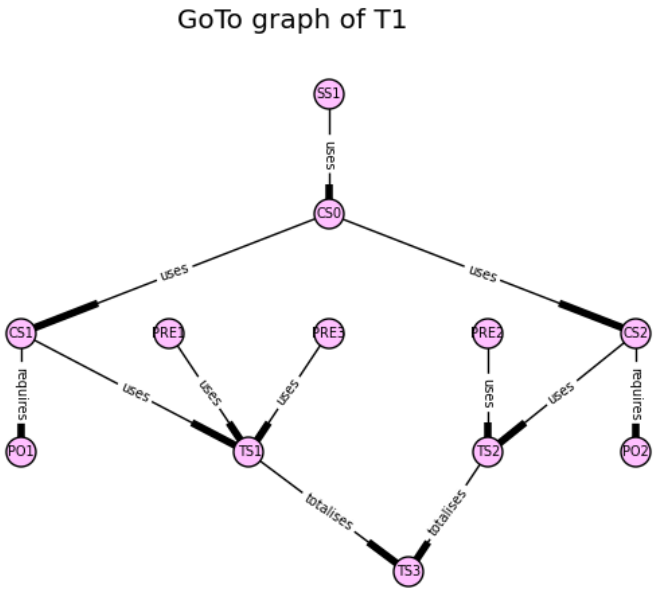
```
\draschema{CS1}{
\begin{schema}{VM\_sale}
\text{VM\_operation}
\where
\draline{PO1}{
\text{\expression{\term{stock'} = \term{\term{stock} - \term{1}}}} \\
\text{\expression{\term{bars\_delivered!} = \term{1}}} \\
\text{\expression{\term{cash\_refunded!} = term{\term{cash\_tendered?} - \term{price}}}} \\
\text{\expression{\term{takings'} = \term{\term{takings} + \term{price}}}}}
\end{schema}}

\uses{CS1}{CS0}
\requires{CS1}{PO1}

\draschema{CS2}{
\begin{schema}{VM\_nosale}
\text{VM\_operation}
\where
\draline{PO2}{\text{\expression{\term{stock'} = \term{stock}}} \\
\text{\expression{\term{bars\_delivered!} = \term{0}}} \\
\text{\expression{\term{cash\_refunded!} = \term{cash\_tendered?}}}\\
\text{\expression{\term{takings'} = \term{takings}}}}
\end{schema}}

\uses{CS2}{CS0}
\requires{CS2}{PO2}

\draschema{TS1}{
\begin{zed}
VM1 \defs \text{\expression{\text{exact\_cash} \land
\text{some\_stock} \land \text{VM\_sale}}}
\end{zed}}

\uses{TS1}{PRE1}
\uses{TS1}{PRE3}
\uses{TS1}{CS1}

\draschema{TS2}{
\begin{zed}
VM2 \defs \text{\expression{\text{insufficient\_cash}
\land\text{VM\_nosale}}}
\end{zed}}

\uses{TS2}{PRE2}
\uses{TS2}{CS2}

\draschema{TS3}{
\begin{zed}
VM3 \defs \text{\expression{\text{VM1} \lor \text{VM2}}}
\end{zed}}

\totalises{TS3}{TS1}
\totalises{TS3}{TS2}
}
\end{document}
```

A.1.8 ZCGa and ZDRa Output



A.1.9 Dependency and Goto Graphs





A.1.10 General Proof Skeleton

```
stateSchema SS1
```

```
precondition PRE1

precondition PRE2

changeSchema CS0

precondition PRE3

changeSchema CS2

postcondition PO2

totaliseSchema TS2

changeSchema CS1

postcondition PO1

totaliseSchema TS1

totaliseSchema TS3
```

A.1.11 Isabelle Proof Skeleton

```
theory isaSkeleton_vendingmachine
imports
Main
begin

record SS1 =
(*DECLARATIONS*)

locale zmathlang_vm =
fixes (*GLOBAL DECLARATIONS*)
begin

definition PRE1 ::
 "(*PRE1_TYPES*) => bool"
where
"PRE1 (*PRE1_VARIABLES*) == (*PRECONDITION*) "

definition PRE2 ::
 "(*PRE2_TYPES*) => bool"
where
"PRE2 (*PRE2_VARIABLES*) == (*PRECONDITION*) "

definition CS0 ::
"(*CS0_TYPES*) => bool"
where
"CS0 (*CS0_VARIABLES*) == True"

definition PRE3 ::
 "(*PRE3_TYPES*) => bool"
where
"PRE3 (*PRE3_VARIABLES*) == (*PRECONDITION*) "
```

```
definition CS0 ::
"(*CS0_TYPES*) => bool"
where
"CS0 (*CS0_VARIABLES*) == True"

definition PRE3 ::
 "(*PRE3_TYPES*) => bool"
where
"PRE3 (*PRE3_VARIABLES*) == (*PRECONDITION*) "

definition CS2 ::
 "(*CS2_TYPES*) => bool"
where
"CS2 (*CS2_VARIABLES*) == (PO2)"

lemma TS2:
"(*TS2_EXPRESSION*)"
sorry

definition CS1 ::
 "(*CS1_TYPES*) => bool"
where
"CS1 (*CS1_VARIABLES*) == (PO1)"

lemma TS1:
"(*TS1_EXPRESSION*)"
sorry

lemma TS3:
"(*TS3_EXPRESSION*)"
sorry

end
end
```

A.1.12 Isabelle Filled In

```
theory 5
imports
Main

begin

record VMSTATE =
STOCK :: nat
TAKINGS :: nat

locale zmathlang_vm =
fixes price :: nat
begin

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

definition exact_cash ::
 "nat  ⇒ bool"
where
"exact_cash cash_tendered  ==
 cash_tendered = price"

 definition VM_operation ::
"VMSTATE => VMSTATE => nat => nat => nat => bool"
where
" VM_operation vmstate vmstate' cash_tendered
cash_refunded bars_delivered == True"

definition some_stock ::
 "nat => bool"
where
" some_stock stock ==  stock > 0  "
```

```
definition VM_nosale ::
 "nat => nat => nat => nat => nat => nat => nat => bool"
where
" VM_nosale stock takings stock' takings'
cash_tendered cash_refunded bars_delivered == ((stock' = stock)
∧ (bars_delivered = 0)
∧ (cash_refunded = cash_tendered)
∧ (takings' = takings))"

definition VM2 ::
 "nat  => nat => nat => nat => nat => nat => nat => bool"
where
" VM2  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered ==
(insufficient_cash  cash_tendered)
 ∧ (VM_nosale stock takings stock' takings' cash_tendered
 cash_refunded bars_delivered)"

definition VM_sale :: " nat => nat => nat ⇒nat => nat =>
nat => nat => bool"
where
" VM_sale  stock takings stock' takings' cash_tendered
cash_refunded bars_delivered ==
(stock' = stock - 1)
∧ (bars_delivered = 1)
∧ (cash_refunded = cash_tendered - price)
∧ (takings' = takings + price) "
```

```
definition VM_sale :: " nat => nat => nat ⇒nat => nat =>
nat => nat => bool"
where
" VM_sale  stock takings stock' takings' cash_tendered
cash_refunded bars_delivered ==
(stock' = stock - 1)
∧ (bars_delivered = 1)
∧ (cash_refunded = cash_tendered - price)
∧ (takings' = takings + price) "

definition VM1 ::
 "nat ⇒ nat  => nat => nat => nat => nat  => nat => bool"
where
" VM1  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered ==
(exact_cash cash_tendered )
∧ (some_stock stock)
∧ (VM_sale  stock takings stock' takings'
 cash_tendered cash_refunded bars_delivered)"

definition VM3 ::
 "nat  => nat => nat => nat => nat => nat => nat => bool"
where
" VM3  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered =
((VM1  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered)
 ∨ (VM2  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered)) "

 end
 end
```

A.1.13 Full Proof in Isabelle

```
theory 6
imports
Main

begin

record VMSTATE =
STOCK :: nat
TAKINGS :: nat

locale zmathlang_vm =
fixes price :: nat
begin

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

definition exact_cash ::
 "nat  ⇒ bool"
where
"exact_cash cash_tendered  ==
 cash_tendered = price"

 definition VM_operation ::
"VMSTATE => VMSTATE => nat => nat => nat => bool"
where
" VM_operation vmstate vmstate' cash_tendered
cash_refunded bars_delivered == True"

definition some_stock ::
 "nat => bool"
where
" some_stock stock ==  stock > 0  "
```

```
definition VM_nosale ::
 "nat => nat => nat => nat => nat => nat => nat => bool"
where
" VM_nosale stock takings stock' takings'
cash_tendered cash_refunded bars_delivered == ((stock' = stock)
∧ (bars_delivered = 0)
∧ (cash_refunded = cash_tendered)
∧ (takings' = takings))"

definition VM2 ::
 "nat  => nat => nat => nat => nat => nat => nat => bool"
where
" VM2  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered ==
(insufficient_cash  cash_tendered)
 ∧ (VM_nosale stock takings stock' takings' cash_tendered
 cash_refunded bars_delivered)"

definition VM_sale :: " nat => nat => nat ⇒nat => nat =>
nat => nat => bool"
where
" VM_sale  stock takings stock' takings' cash_tendered
cash_refunded bars_delivered ==
(stock' = stock - 1)
∧ (bars_delivered = 1)
∧ (cash_refunded = cash_tendered - price)
∧ (takings' = takings + price) "
```

```
definition VM1 ::
 "nat ⇒ nat  => nat => nat => nat => nat  => nat => bool"
where
" VM1  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered ==
(exact_cash cash_tendered )
∧ (some_stock stock)
∧ (VM_sale  stock takings stock' takings'
 cash_tendered cash_refunded bars_delivered)"

definition VM3 ::
 "nat  => nat => nat => nat => nat => nat => nat => bool"
where
" VM3  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered =
((VM1  cash_tendered stock takings stock' takings'
cash_refunded bars_delivered)
 ∨ (VM2  cash_tendered stock takings stock' takings'
 cash_refunded bars_delivered)) "

lemma pre_VM1:
"(∃ stock' takings' cash_refunded bars_delivered.
 VM1 cash_tendered stock takings stock' takings'
 cash_refunded bars_delivered)
 ⟷ (0 < stock) ∧
 (cash_tendered = price) ∧ (0 ≤ takings)"
 apply (unfold VM1_def exact_cash_def
 some_stock_def VM_sale_def)
 apply auto
 done
```

```
  lemma pre_VM2:
"(∃ stock' takings' cash_refunded bars_delivered.
 VM2 cash_tendered stock takings stock' takings'
 cash_refunded bars_delivered)
 ⟷ (cash_tendered < price) ∧
(cash_tendered ≥ 0) ∧ (stock ≥ 0) ∧ (takings ≥ 0)"
apply (unfold VM2_def insufficient_cash_def VM_nosale_def )
apply auto
done

  lemma pre_VM3:
"(∃ stock' takings' cash_refunded bars_delivered.
 VM3 cash_tendered stock takings stock'
 takings' cash_refunded bars_delivered)
 ⟷ (0 < stock ∧
 cash_tendered = price ∧ 0 ≤ takings) ∨ (cash_tendered < price)
∧ (0 ≤ cash_tendered)
∧ (0 ≤ stock)
∧ (0 ≤ takings)"
apply (unfold VM3_def VM2_def VM1_def
some_stock_def exact_cash_def VM_sale_def
  VM_nosale_def insufficient_cash_def)
apply auto
done

lemma cash_lemma: "¬ (insufficient_cash
cash_tendered ∧ exact_cash cash_tendered)"
apply (unfold insufficient_cash_def exact_cash_def)
apply auto
done
```

```
lemma VM3_refines_VM1:
"(∃ stock' takings' cash_refunded bars_delivered.
((VM1 cash_tendered stock takings stock' takings' cash_refunded
bars_delivered)
⟶
(VM3 cash_tendered stock takings stock' takings' cash_refunded
bars_delivered))
∧
(((VM1 cash_tendered stock takings stock' takings' cash_refunded
bars_delivered)
∧
(VM3 cash_tendered stock takings stock' takings' cash_refunded
bars_delivered))
⟶
(VM1 cash_tendered stock takings stock' takings' cash_refunded
bars_delivered)))"
apply (unfold VM3_def VM1_def VM_sale_def
 exact_cash_def some_stock_def)
apply auto
done

lemma VM3_ok:
"(∃ stock' takings cash_refunded bars_delivered.
(VM3 cash_tendered stock takings stock' takings' cash_refunded
bars_delivered)
⟶
((takings' - takings) ≥ price * (stock - stock' )))"
apply (unfold VM3_def VM1_def VM2_def exact_cash_def some_stock_def
  VM_sale_def VM_nosale_def insufficient_cash_def)
apply auto
done

end
end
```

A.2 BirthdayBook

A.2.1 Raw Latex

```
\documentclass{article}
\usepackage{zmathlang}

\begin{document}

\begin{zed}
   [NAME, ~ DATE]
\end{zed}

\begin{schema}{BirthdayBook}
known: \power NAME \\
birthday: NAME \pfun DATE
\where
known=\dom birthday
\end{schema}

\begin{schema}{InitBirthdayBook}
BirthdayBook~'
\where
known' = \{ \}
\end{schema}

\begin{schema}{AddBirthday}
\Delta BirthdayBook \\
name?: NAME \\
date?: DATE
\where
name? \notin known\\
birthday' = birthday \cup \{name? \mapsto date?\}
\end{schema}

\begin{schema}{FindBirthday}
\Xi BirthdayBook \\
name?: NAME \\
date!: DATE
\where
name? \in known\\
date! = birthday(name?)
\end{schema}

\begin{zed}
REPORT ::= ok | already\_known | not\_known
\end{zed}

\begin{schema}{Success}
result!: REPORT
\where
result! = ok
\end{schema}

\begin{schema}{AlreadyKnown}
\Xi BirthdayBook \\
name?: NAME \\
result!: REPORT
\where
name? \in known \\
result! = already\_known
\end{schema}

\begin{schema}{NotKnown}
\Xi BirthdayBook \\
name?: NAME \\
result!: REPORT
\where
name? \notin known \\
result! = not\_known
\end{schema}

\begin{zed}
RAddBirthday ==\\ (AddBirthday \land Success)\\
  \lor AlreadyKnown \\
RFindBirthday ==\\ (FindBirthday \land Success)
 \lor NotKnown \\
\end{zed}

\end{document}
```

A.2.2 Raw Latex ouput

$[NAME,\ DATE]$

BirthdayBook

$$known : \mathbb{P}\ NAME$$
$$birthday : NAME \pfun DATE$$
$$known = \mathrm{dom}\ birthday$$

InitBirthdayBook

$$BirthdayBook'$$
$$known' = \{\}$$

AddBirthday

$$\Delta BirthdayBook$$
$$name? : NAME$$
$$date? : DATE$$
$$name? \notin known$$
$$birthday' = birthday \cup \{name? \mapsto date?\}$$

FindBirthday

$$\Xi BirthdayBook$$
$$name? : NAME$$
$$date! : DATE$$
$$name? \in known$$
$$date! = birthday(name?)$$

$REPORT ::= ok \mid already\_known \mid not\_known$

Success

$$result! : REPORT$$
$$result! = ok$$

AlreadyKnown

$$\Xi BirthdayBook$$
$$name? : NAME$$
$$result! : REPORT$$
$$name? \in known$$
$$result! = already\_known$$

NotKnown

$\Xi BirthdayBook$
$name? : NAME$
$result! : REPORT$

$name? \notin known$
$result! = not\_known$

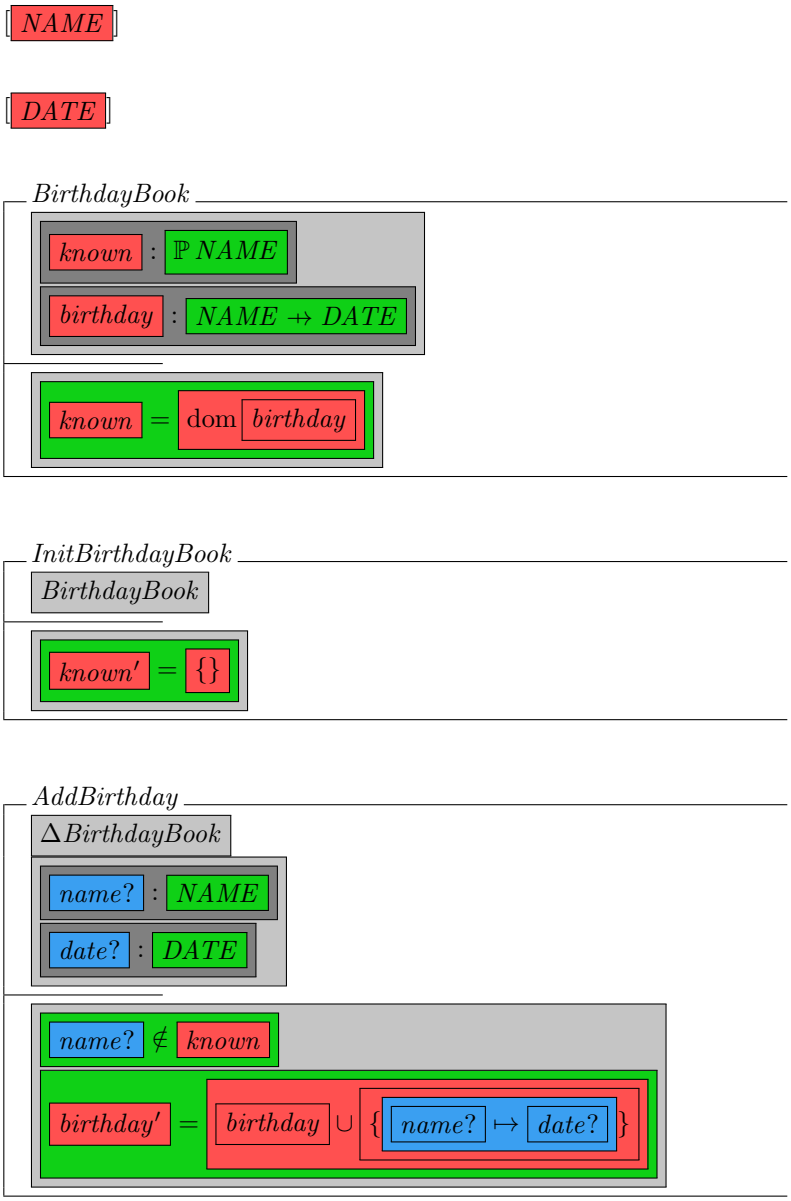
$RAddBirthday ==$
$(AddBirthday \land Success)$
$\lor AlreadyKnown$
$RFindBirthday ==$
$(FindBirthday \land Success) \lor NotKnown$

A.2.3 ZCGa Annotated Latex Code

```
\documentclass{article}
\usepackage{zmathlang}

\begin{document}

\begin{zed}
[\set{NAME}]
\end{zed}

\begin{zed}
[\set{DATE}]
\end{zed}

\begin{schema}{BirthdayBook}
\text{\declaration{\set{known}: \expression{\power NAME}} \\
\declaration{\set{birthday}: \expression{NAME \pfun DATE}}}
\where
\text{\expression{\set{known}=\set{\dom \set{birthday}}}}
\end{schema}

\begin{schema}{InitBirthdayBook}
\text{BirthdayBook}
\where
\text{\expression{\set{known'} = \set{\{ \}}}}
\end{schema}

\begin{schema}{AddBirthday}
\text{\Delta BirthdayBook} \\
\text{\declaration{\term{name?}: \expression{NAME}} \\
\declaration{\term{date?}: \expression{DATE}}}
\where
\text{\expression{\term{name?} \notin \set{known}}\\
\expression{\set{birthday'} = \set{\set{birthday}
\cup \set{\{\term{\term{name?} \mapsto \term{date?}}\}}}}}
\end{schema}

\begin{schema}{FindBirthday}
\text{\Xi BirthdayBook} \\
\text{\declaration{\term{name?}: \expression{NAME}} \\
\declaration{\term{date!}: \expression{DATE}} }
\where
\text{\expression{\term{name?} \in \set{known}}\\
\expression{\term{date!} = \term{\set{birthday}(\term{name?})}}}
\end{schema}

\begin{zed}
\set{REPORT} ::= \term{ok} | \term{already\_known}
| \term{not\_known}
\end{zed}

\begin{schema}{Success}
\text{\declaration{\term{result!}: \expression{REPORT}}}
\where
\text{\expression{\term{result!} = \term{ok}}}
\end{schema}
```

```
\begin{schema}{AlreadyKnown}
\text{\Xi BirthdayBook} \\
\text{\declaration{\term{name?}: \expression{NAME}} \\
\declaration{\term{result!}: \expression{REPORT}}}
\where
\text{\expression{\term{name?} \in \set{known}} \\
\expression{\term{result!} = \term{already\_known}}}
\end{schema}

\begin{schema}{NotKnown}
\text{\Xi BirthdayBook} \\
\text{\declaration{\term{name?}: \expression{NAME}} \\
\declaration{\term{result!}: \expression{REPORT}}}
\where
\text{\expression{\term{name?} \notin \set{known}} \\
\expression{\term{result!} = \term{not\_known}}}
\end{schema}

\begin{zed}
RAddBirthday ==\\
\text{\expression{(\text{AddBirthday} \land
\text{Success})\\  \lor \text{AlreadyKnown}}} \\
RFindBirthday ==\\
\text{\expression{(\text{FindBirthday}
\land \text{Success}) \lor \text{NotKnown}}} \\
\end{zed}

\end{document}
```

A.2.4 ZCGa output



FindBirthday

$\Xi BirthdayBook$

$name? : NAME$

$date! : DATE$

$name? \in known$

$date! = birthday(name?)$

$REPORT ::= ok \mid already\_known \mid not\_known$

Success

$result! : REPORT$

$result! = ok$

AlreadyKnown

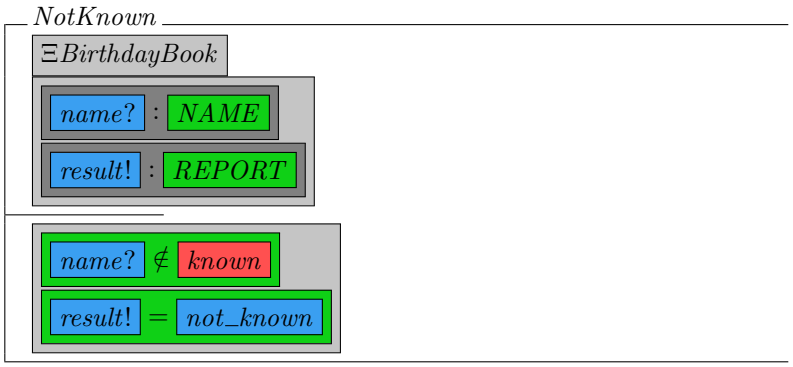
$\Xi BirthdayBook$

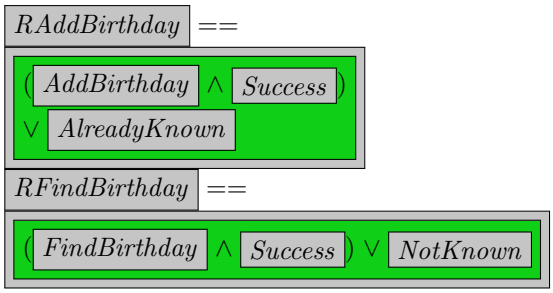
$name? : NAME$

$result! : REPORT$

$name? \in known$

$result! = already\_known$


NotKnown
ΞBirthdayBook
name? : NAME
result! : REPORT
name? ∉ known
result! = not_known

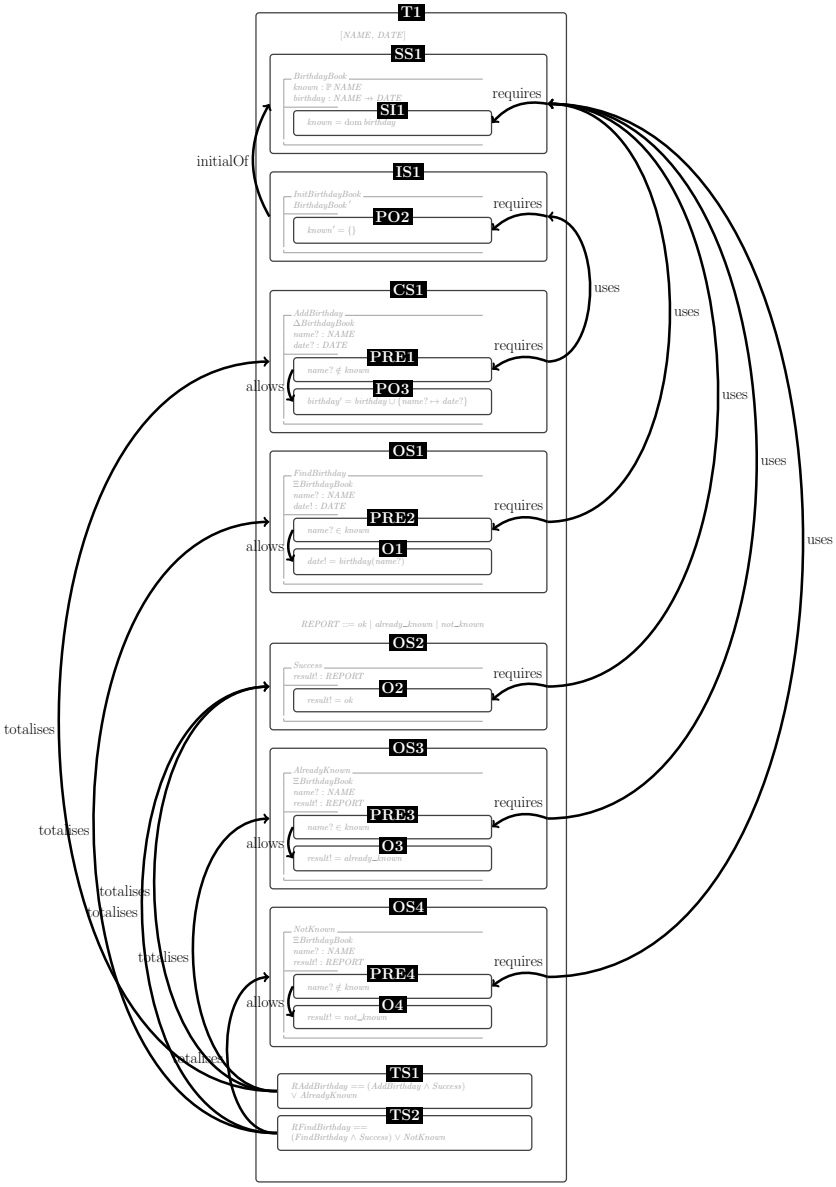

RAddBirthday ==
(AddBirthday ∧ Success)
∨ AlreadyKnown
RFindBirthday ==
(FindBirthday ∧ Success) ∨ NotKnown

A.2.5 ZDRa Annotated Latex Code

```
\documentclass{article}
\usepackage{zmathlang}

\begin{document}

\dratheory{T1}{0.34}{

\begin{zed}
[NAME,  DATE]
\end{zed}

\draschema{SS1}{
\begin{schema}{BirthdayBook}
known: \power NAME \\
birthday: NAME \pfun DATE
\where
\draline{SI1}{known=\dom birthday}
\end{schema}}

\requires{SS1}{SI1}

\draschema{IS1}{
\begin{schema}{InitBirthdayBook}
BirthdayBook~'
\where
\draline{PO2}{known' = \{ \}}
\end{schema}}

\requires{IS1}{PO2}
\initialof{IS1}{SS1}

\draschema{CS1}{
\begin{schema}{AddBirthday}
\Delta BirthdayBook \\
name?: NAME \\
date?: DATE
\where
\draline{PRE1}{name? \notin known}\\
\draline{PO3}{birthday' = birthday \cup
\{name? \mapsto date?\}}
\end{schema}}

\uses{CS1}{IS1}
\requires{CS1}{PRE1}
\allows{PRE1}{PO3}
```

```
\draschema{OS1}{
\begin{schema}{FindBirthday}
\Xi BirthdayBook \\
name?: NAME \\
date!: DATE
\where
\draline{PRE2}{name? \in known}\\
\draline{O1}{date! = birthday(name?)}
\end{schema}}

\allows{PRE2}{O1}
\uses{OS1}{SS1}
\requires{OS1}{PRE2}

\begin{zed}
REPORT ::= ok | already\_known | not\_known
\end{zed}

\draschema{OS2}{
\begin{schema}{Success}
result!: REPORT
\where
\draline{O2}{result! = ok}
\end{schema}}

\requires{OS2}{O2}
\uses{OS2}{SS1}

\draschema{OS3}{
\begin{schema}{AlreadyKnown}
\Xi BirthdayBook \\
name?: NAME \\
result!: REPORT
\where
\draline{PRE3}{name? \in known} \\
\draline{O3}{result! = already\_known}
\end{schema}}

\requires{OS3}{PRE3}
\allows{PRE3}{O3}
\uses{OS3}{SS1}

\draschema{OS4}{
\begin{schema}{NotKnown}
\Xi BirthdayBook \\
name?: NAME \\
result!: REPORT
\where
\draline{PRE4}{name? \notin known} \\
\draline{O4}{result! = not\_known}
\end{schema}}

\requires{OS4}{PRE4}
\allows{PRE4}{O4}
\uses{OS4}{SS1}

\begin{zed}
\draline{TS1}{RAddBirthday == (AddBirthday \land Success)
  \lor AlreadyKnown} \\
   \draline{TS2}{RFindBirthday ==\\ (FindBirthday \land Success)
\lor NotKnown} \\
\end{zed}

\totalises{TS1}{CS1}
\totalises{TS1}{OS2}
\totalises{TS1}{OS3}
\totalises{TS2}{OS1}
\totalises{TS2}{OS2}
\totalises{TS2}{OS4}
}

\end{document}
```

A.2.6 ZDRa Output

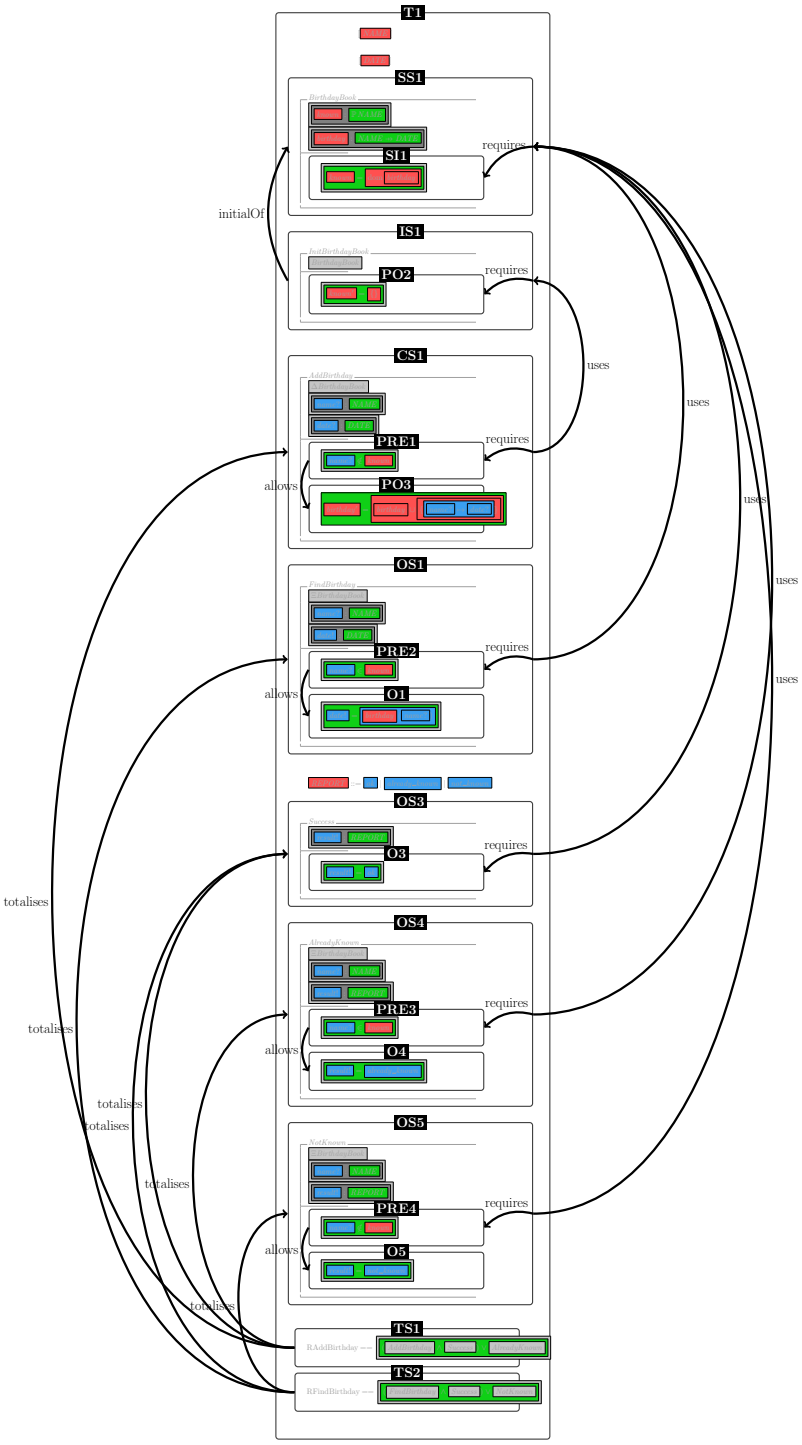


A.2.7 ZCGa and ZDRa Annotated Latex Code

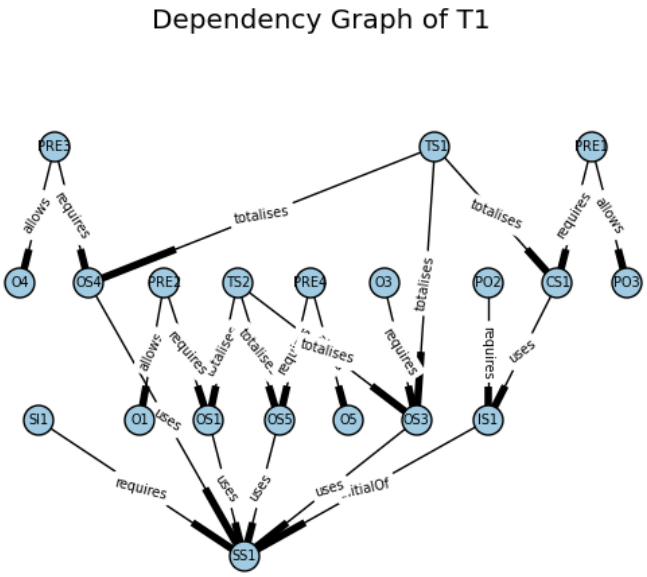
```
\documentclass{article}
\usepackage{zmathlang}

\begin{document}

\dratheory{T1}{0.3}{

\begin{zed}
[\set{NAME}]
\end{zed}

\begin{zed}
[\set{DATE}]
\end{zed}

\draschema{SS1}{
\begin{schema}{BirthdayBook}
\text{\declaration{\set{known}:\expression{\power NAME}}} \\
\text{\declaration{\set{birthday}:\expression{NAME \pfun DATE}}}
\where
\draline{SI1}{\text{\expression{\set{known}=\set{\dom \set{birthday}}}}}
\end{schema}}

\requires{SS1}{SI1}

\draschema{IS1}{
\begin{schema}{InitBirthdayBook}
\text{BirthdayBook}
\where
\draline{PO2}{\text{\expression{\set{known'} = \set{\{ \}}}}}
\end{schema}}

\requires{IS1}{PO2}

\initialof{IS1}{SS1}

\draschema{CS1}{
\begin{schema}{AddBirthday}
\text{\Delta BirthdayBook} \\
\text{\declaration{\term{name?}: \expression{NAME}}} \\
\text{\declaration{\term{date?}: \expression{DATE}}}
\where
\draline{PRE1}{\text{\expression{\term{name?} \notin \set{known}}}}\\
\draline{PO3}{\text{\expression{\set{birthday'} = \set{\set{birthday}
\cup \set{\{\term{\term{name?} \mapsto \term{date?}}\}}}}}}
\end{schema}}

\uses{CS1}{IS1}
\requires{CS1}{PRE1}
\allows{PRE1}{PO3}
```

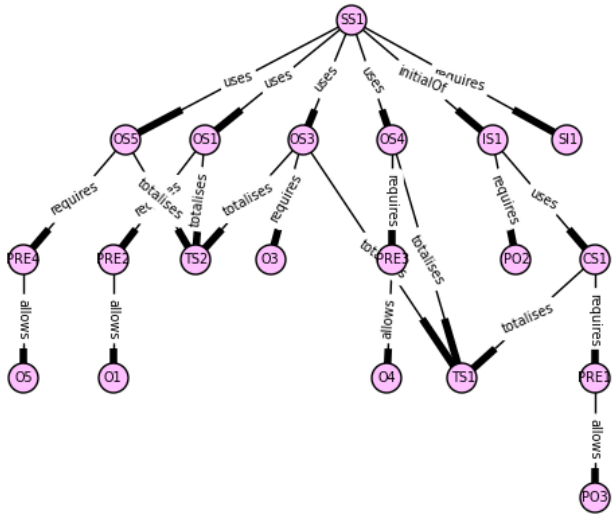
```
\draschema{OS1}{
\begin{schema}{FindBirthday}
\text{\Xi BirthdayBook} \\
\text{\declaration{\term{name?}: \expression{NAME}}} \\
\text{\declaration{\term{date!}: \expression{DATE}}}
\where
\draline{PRE2}{\text{\expression{\term{name?} \in \set{known}}}}\\
\draline{O1}{\text{\expression{\term{date!} =
\term{\set{birthday} (\term{name?})}}}}
\end{schema}}

\allows{PRE2}{O1}
\uses{OS1}{SS1}
\requires{OS1}{PRE2}

\begin{zed}
\set{REPORT} ::= \term{ok} | \term{already\_known} | \term{not\_known}
\end{zed}

\draschema{OS3}{
\begin{schema}{Success}
\text{\declaration{\term{result!}: \expression{REPORT}}}
\where
\draline{O3}{\text{\expression{\term{result!} = \term{ok}}}}
\end{schema}}

\requires{OS3}{O3}
\uses{OS3}{SS1}

\draschema{OS4}{
\begin{schema}{AlreadyKnown}
\text{\Xi BirthdayBook} \\
\text{\declaration{\term{name?}: \expression{NAME}}} \\
\text{\declaration{\term{result!}: \expression{REPORT}}}
\where
\draline{PRE3}{\text{\expression{\term{name?} \in \set{known}}}} \\
\draline{O4}{\text{\expression{\term{result!} = \term{already\_known}}}}
\end{schema}}

\requires{OS4}{PRE3}
\allows{PRE3}{O4}
\uses{OS4}{SS1}

\draschema{OS5}{
\begin{schema}{NotKnown}
\text{\Xi BirthdayBook} \\
\text{\declaration{\term{name?}:\expression{NAME}}} \\
\text{\declaration{\term{result!}:\expression{REPORT}}}
\where
\draline{PRE4}{\text{\expression{\term{name?} \notin \set{known}}}} \\
\draline{O5}{\text{\expression{\term{result!} = \term{not\_known}}}}
\end{schema}}

\requires{OS5}{PRE4}
\allows{PRE4}{O5}
\uses{OS5}{SS1}

\begin{zed}
\draline{TS1}{RAddBirthday == \text{\expression{(\text{AddBirthday}
\land \text{Success})\\  \lor \text{AlreadyKnown}}}} \\
\draline{TS2}{RFindBirthday == \text{\expression{(\text{FindBirthday}
 \land \text{Success}) \lor \text{NotKnown}}}}
\end{zed}

\totalises{TS1}{CS1}
\totalises{TS1}{OS3}
\totalises{TS1}{OS4}
\totalises{TS2}{OS1}
\totalises{TS2}{OS3}
\totalises{TS2}{OS5}

}
\end{document}
```

A.2.8 ZCGa and ZDRa output



A.2.9 Dependency and Goto Graphs





A.2.10 General Proof Skeleton

stateSchema SS1

stateInvarients SI1

initialSchema IS1

postcondition PO2

outputSchema OS1

precondition PRE2

changeSchema CS1

precondition PRE1

outputSchema OS5

precondition PRE4

outputSchema OS4

precondition PRE3

postcondition PO3

output O5

output O4

outputSchema OS3

output O3

output O1

totaliseSchema TS2

totaliseSchema TS1

lemma L1_(CS1)

A.2.11 Isabelle Proof Skeleton

```
theory birthdaybook
imports
Main

begin
(*DATATYPES*)

record SS1 =
(*DECLARATIONS*)

locale 1n2 =
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 CS1 ::
"(*CS1_TYPES*) => bool"
where
"CS1 (*CS1_VARIABLES*) ==
 (PRE1)
∧ (PO3)"
```

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

definition OS4 ::
 "(*OS4_TYPES*) => bool"
where
"OS4 (*OS4_VARIABLES*) == (PRE3)
∧ (O4)"

definition OS3 ::
 "(*OS3_TYPES*) => bool"
where
"OS3 (*OS3_VARIABLES*) == (O3)"

definition TS2 ::
"(*TS2_TYPES*) => bool"
where
"TS2 (*TS2_VARIABLES*) == (*TS2_EXPRESSION*)"

definition TS1 ::
"(*TS1_TYPES*) => bool"
where
"TS1 (*TS1_VARIABLES*) == (*TS1_EXPRESSION*)"

lemma CS1_L1:
"(∃ (*CS1_VARIABLESANDTYPES*).
(PRE1)
∧ (PO3)
∧ (SI1)
∧ (SI1'))"
sorry

end
end
```

A.2.12 Isabelle Filled In

```
theory new5
imports
Main

begin
typedecl NAME
typedecl DATE
datatype REPORT = ok | already_known | not_known


record BirthdayBook =
BIRTHDAY :: "(NAME ⇀ DATE)"
KNOWN :: "(NAME set)"


locale gpsabirthdaybook =
fixes birthday :: "(NAME ⇀ DATE)"
and known :: "(NAME set)"

assumes
"(known = dom birthday)"
begin

definition InitBirthdayBook ::
 "BirthdayBook ⇒ (NAME set) ⇒ (NAME ⇀ DATE) ⇒ BirthdayBook => bool"
where
"InitBirthdayBook birthdaybook' known' birthday' birthdaybook == (known' = {})"

definition FindBirthday ::
 "(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇀ DATE) => bool"
where
"FindBirthday known' name birthdaybook birthdaybook' date birthday' == (name ∈ known)
∧ (date = the (birthday name))"
```

```
definition AddBirthday ::
"(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇀ DATE) => bool"
where
"AddBirthday known' name birthdaybook birthdaybook' date birthday' ==
 (name ∉ known)
∧ (birthday' = birthday (name ↦ date ))"

definition NotKnown ::
 "(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ (NAME ⇀ DATE) ⇒ REPORT => bool"
where
"NotKnown known' name birthdaybook birthdaybook' birthday' result == (name ∉ known)
∧ (result = not_known)"

definition AlreadyKnown ::
 "(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ (NAME ⇀ DATE) ⇒ REPORT => bool"
where
"AlreadyKnown known' name birthdaybook birthdaybook' birthday' result == (name ∈ known)
∧ (result = already_known)"

definition Success ::
 "REPORT => bool"
where
"Success result == (result = ok)"

definition RFindBirthday ::
"(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇀ DATE) ⇒ REPORT => bool"
where
"RFindBirthday known' name birthdaybook birthdaybook' date birthday' result ==
(((FindBirthday known' name birthdaybook birthdaybook' date birthday') ∧
 (Success result)) ∨
 (NotKnown known' name birthdaybook birthdaybook' birthday' result))"
```

```
definition RAddBirthday ::
"(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇸ DATE) ⇒ REPORT => bool"
where
"RAddBirthday known' name birthdaybook birthdaybook' date birthday' result ==
(((AddBirthday known' name birthdaybook birthdaybook' date birthday') ∧
 (Success result)) ∨
 (AlreadyKnown known' name birthdaybook birthdaybook' birthday' result))"

lemma AddBirthday_L1:
"(∃ known' :: (NAME set).
∃ name :: NAME.
∃ date :: DATE.
∃ birthday' :: (NAME ⇸ DATE).
∃ birthday :: (NAME ⇸ DATE).
∃ known :: (NAME set).
(name ∉ known)
∧ (birthday' = birthday (name ↦ date ))
∧ (known = dom birthday)
∧(known' = dom birthday'))"
sorry

end
end
```

A.2.13 Full Proof in Isabelle

```
theory new6
imports
Main

begin
typedecl NAME
typedecl DATE
datatype REPORT = ok | already_known | not_known

record BirthdayBook =
BIRTHDAY :: "(NAME ⇸ DATE)"
KNOWN :: "(NAME set)"

locale gpsabirthdaybook =
fixes birthday :: "(NAME ⇸ DATE)"
and known :: "(NAME set)"

assumes
"(known = dom birthday)"
begin

definition InitBirthdayBook ::
 "BirthdayBook ⇒ (NAME set) ⇒ (NAME ⇸ DATE) ⇒ BirthdayBook => bool"
where
"InitBirthdayBook birthdaybook' known' birthday' birthdaybook == (known' = {})"

definition FindBirthday ::
 "(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇸ DATE) => bool"
where
"FindBirthday known' name birthdaybook birthdaybook' date birthday' == (name ∈ known)
∧ (date = the (birthday name))"
```

```
definition AddBirthday ::
"(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇸ DATE) => bool"
where
"AddBirthday known' name birthdaybook birthdaybook' date birthday' ==
 (name ∉ known)
∧ (birthday' = birthday (name ↦ date ))"

definition NotKnown ::
 "(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ (NAME ⇸ DATE) ⇒ REPORT => bool"
where
"NotKnown known' name birthdaybook birthdaybook' birthday' result == (name ∉ known)
∧ (result = not_known)"

definition AlreadyKnown ::
 "(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ (NAME ⇸ DATE) ⇒ REPORT => bool"
where
"AlreadyKnown known' name birthdaybook birthdaybook' birthday' result == (name ∈ known)
∧ (result = already_known)"

definition Success ::
 "REPORT => bool"
where
"Success result == (result = ok)"

definition RFindBirthday ::
"(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇸ DATE) ⇒ REPORT => bool"
where
"RFindBirthday known' name birthdaybook birthdaybook' date birthday' result ==
(((FindBirthday known' name birthdaybook birthdaybook' date birthday') ∧
 (Success result)) ∨
 (NotKnown known' name birthdaybook birthdaybook' birthday' result))"
definition RAddBirthday ::
"(NAME set) ⇒ NAME ⇒ BirthdayBook ⇒ BirthdayBook ⇒ DATE ⇒ (NAME ⇸ DATE) ⇒ REPORT => bool"
where
"RAddBirthday known' name birthdaybook birthdaybook' date birthday' result ==
 (((AddBirthday known' name birthdaybook birthdaybook' date birthday') ∧
 (Success result)) ∨
 (AlreadyKnown known' name birthdaybook birthdaybook' birthday' result))"

lemma AddBirthday_L1:
"(∃ known' :: (NAME set).
∃ name :: NAME.
∃ date :: DATE.
∃ birthday' :: (NAME ⇸ DATE).
∃ birthday :: (NAME ⇸ DATE).
∃ known :: (NAME set).
(name ∉ known)
∧ (birthday' = birthday (name ↦ date ))
∧ (known = dom birthday)
∧(known' = dom birthday'))"
by auto

(*Here I add my own properties about the birthdaybook specification *)

definition (in gpsabirthdaybook)
birthdaybookstate :: "BirthdayBook ⇒ bool"
where
"birthdaybookstate birthdaybook == (known = dom birthday)"
```

```
lemma AddBirthdayIsHonest:
"(∃ known' birthday' birthdaybook birthdaybook' date.
AddBirthday known' name birthdaybook birthdaybook' date birthday')
⟷
(name ∉ known)"
apply (unfold AddBirthday_def)
apply auto
done

lemma preAddBirthdayTotal:
" (name ∉ known) ∨ (name ∈ known)"
apply (rule excluded_middle)
done

lemma BirthdayBookPredicate:
"(∃ birthdaybook. birthdaybookstate birthdaybook)
⟶ known = dom birthday"
apply (rule impI)
apply (unfold birthdaybookstate_def)
apply auto
done

lemma InitisOk:
"(∃ birthdaybook. InitBirthdayBook birthdaybook' known' birthday' birthdaybook)
⟷ (known' = {}) "
apply (unfold InitBirthdayBook_def)
apply auto
done

lemma RAddBirthdayIsTotal:
"(∃ known' birthday' birthdaybook
birthdaybook' date.
RAddBirthday known' name birthdaybook birthdaybook' date birthday' result)
⟶
(name ∉ known) ∨ (name ∈ known)"
apply (unfold RAddBirthday_def)
apply (unfold AddBirthday_def AlreadyKnown_def Success_def)
apply auto
done

end
end
```

A.3 An example of a specification which fails ZCGa but passes ZDRa

This section shows an example of a specification which is rhetorically correct and passes the ZDRa check however the grammar of the specification is incorrect and therefore fails the ZCGa check. We input the compiled output for each of the examples. For reference to the code the reader is directed to [12].

A.3.1 Raw Latex output

$[NAME]$

$[SURNAME]$

$[TELEPHONE]$

TelephoneDirectory

$$persons : NAME \rightarrow SURNAME$$
$$phoneNumbers : NAME \nrightarrow TELEPHONE$$

$$\operatorname{dom} phoneNumbers = \operatorname{dom} persons$$

InitTelephoneDirectory

$$TelephoneDirectory'$$

$$phoneNumbers' = \{\} persons' = \{\}$$

AddPerson

$$TheTelephoneDirectory$$
$$name? : NAME$$
$$surname? : SURNAMEphone? : TELEPHONE$$

$$name? \mapsto surname? \notin persons$$
$$phoneNumbers' = phoneNumbers \cup \{name? \mapsto phone?\}$$

AddNumber

$$\Delta TelephoneDirectory$$
$$n? : NAME$$
$$s? : SURNAMEp? : TELEPHONE$$

$$n \mapsto s \in persons$$
$$p? \notin \operatorname{ran} phoneNumbers$$
$$phoneNumbers' = phoneNumbers \cup \{n \mapsto phone?\}$$

$$OUTPUT ::= success \mid alreadyInDirectory \mid notInDirectory \mid numberInUse$$

Success

$$message! : OUTPUT$$

$$message! = success$$

AlreadyInDirectory

$$message! : OUTPUT$$
$$n? : NAME$$
$$s? : SURNAME$$

$$n? \mapsto s \in persons$$
$$message! = alreadyInDirectory$$

NameNotInDirectory

$$message! : OUTPUT$$
$$n? : NAME$$
$$s? : SURNAME$$

$$n? \mapsto s? \notin persons$$
$$message! = NamenotInDirectory$$

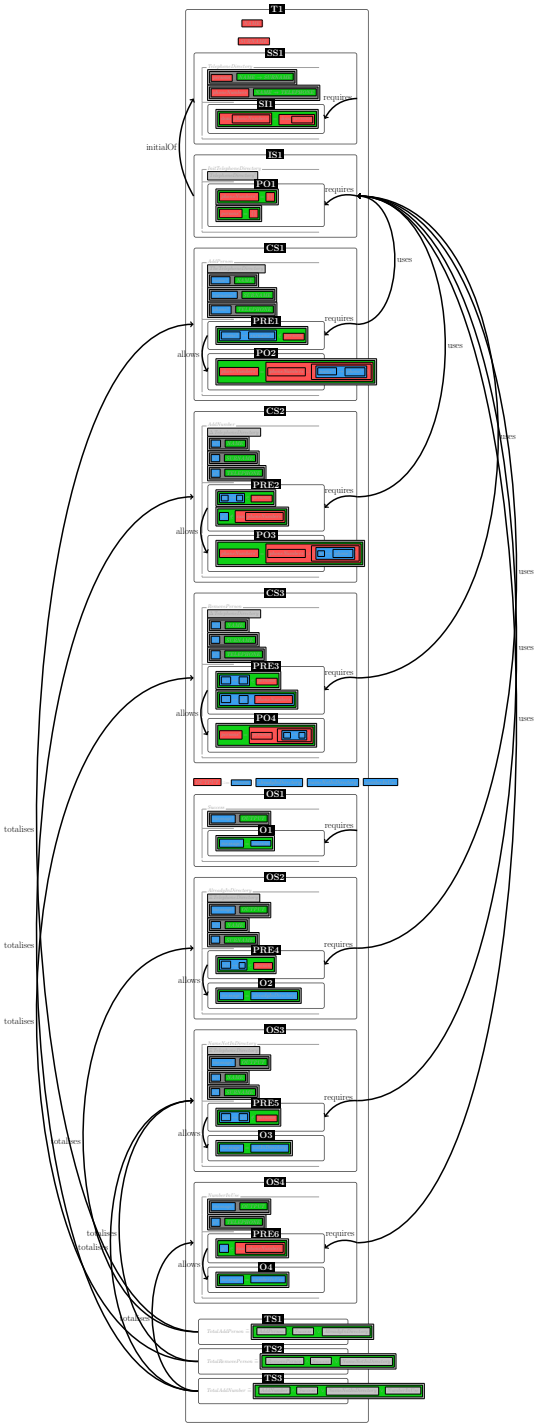
NumberInUse

$$message! : OUTPUT$$
$$p? : TELEPHONE$$

$$p? \in \operatorname{ran} phoneNumbers$$
$$message! = numberInUse$$

$$TotalAddPerson \mathrel{\widehat=} (AddPerson \land Success) \lor AlreadyInDirectory$$
$$TotalRemovePerson \mathrel{\widehat=} (RemovePerson \land Success) \lor NameNotInDirectory$$
$$TotalAddNumber \mathrel{\widehat=} (AddNumber \land Success) \lor NameNotInDirectory \lor Numbe$$

A.3.2 ZCGa and ZDRa output



A.3.3 Messages when running the specification through the ZCGa and ZDRa checks

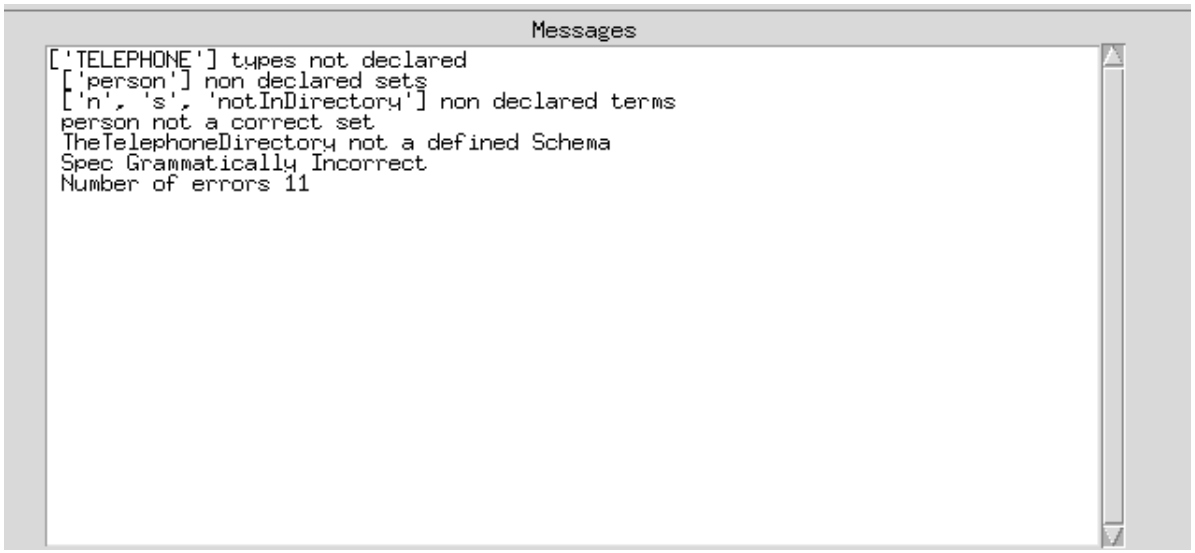


Figure A.1: Message when checking the specification for ZCGa correctness.

Figure A.2: Message when checking the specification for ZDRa correctness.

A.4 An example of a specification which fails ZDRa but passes ZCGa

This section shows an example of a specification which is grammatically correct and passes the ZCGa check however there are loops in it's rhetorical reasoning

and therefore fails the ZDRa check. We input the compiled output for each of the examples. For reference to the code the reader is directed to [12].

A.4.1 Raw Latex output

$[NAME]$

$[SURNAME]$

$[TELEPHONE]$

TelephoneDirectory

$$persons : NAME \rightarrow SURNAME$$
$$phoneNumbers : NAME \nrightarrow TELEPHONE$$

$$\mathrm{dom}\, phoneNumbers = \mathrm{dom}\, persons$$

InitTelephoneDirectory

$$TelephoneDirectory'$$

$$phoneNumbers' = \{\}$$
$$persons' = \{\}$$

AddPerson

$$\Delta TelephoneDirectory$$
$$name? : NAME$$
$$surname? : SURNAME$$
$$phone? : TELEPHONE$$

$$name? \mapsto surname? \notin persons$$
$$persons' = persons \cup \{name? \mapsto surname?\}$$

AddNumber

$$\Delta TelephoneDirectory$$
$$n? : NAME$$
$$s? : SURNAME$$
$$p? : TELEPHONE$$

$$n? \mapsto s? \in persons$$
$$p? \notin \mathrm{ran}\, phoneNumbers$$
$$phoneNumbers' = phoneNumbers \cup \{name? \mapsto phone?\}$$

RemovePerson

$$\Delta TelephoneDirectory$$
$$n? : NAME$$
$$s? : SURNAME$$
$$p? : TELEPHONE$$

$$n? \mapsto s? \in persons$$
$$n? \mapsto p? \notin phoneNumbers$$
$$persons' = persons \setminus \{n? \mapsto s?\}$$

RemoveNumber

$\Delta TelephoneDirectory$
$p? : TELEPHONE$

$p? \in \operatorname{ran} phoneNumbers$
$\exists m : \operatorname{dom} persons \bullet$
$m \mapsto p? \in phoneNumbers \land$
$phoneNumbers' = phoneNumbers \setminus \{m \mapsto p?\}$

$OUTPUT ::= success \mid alreadyInDirectory \mid nameNotInDirectory \mid numberInUse \mid numberDoesntExist$

Success

$message! : OUTPUT$

$message! = success$

AlreadyInDirectory

$\Xi TelephoneDirectory$
$message! : OUTPUT$
$n? : NAME$
$s? : SURNAME$

$n? \mapsto s? \in persons$
$message! = alreadyInDirectory$

NameNotInDirectory

$\Xi TelephoneDirectory$
$message! : OUTPUT$
$n? : NAME$
$s? : SURNAME$

$n? \mapsto s? \notin persons$
$message! = nameNotInDirectory$

NumberInUse

$message! : OUTPUT$
$p? : TELEPHONE$

$p? \in \operatorname{ran} phoneNumbers$
$message! = numberInUse$

NumberDoesntExist

$message! : OUTPUT$
$p? : TELEPHONE$

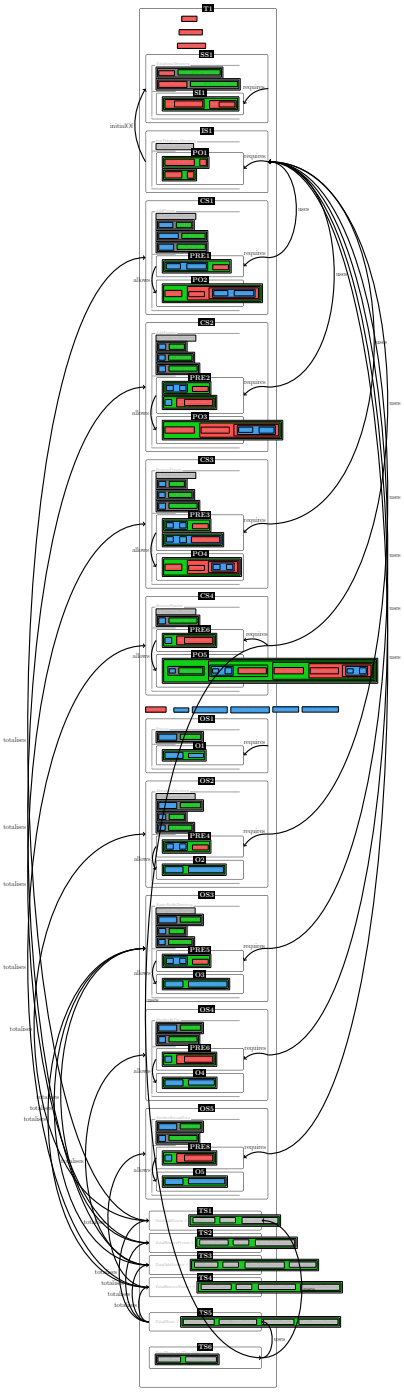
$p? \notin \operatorname{ran} phoneNumbers$
$message! = numberDoesntExist$

$TotalAddPerson \mathrel{\widehat=} (AddPerson \land Success) \lor AlreadyInDirectory$
$TotalRemovePerson \mathrel{\widehat=} (RemovePerson \land Success)$
$\lor NameNotInDirectory$
$TotalAddNumber \mathrel{\widehat=} (AddNumber \land Success)$
$\lor NameNotInDirectory \lor NumberInUse$
$TotalRemoveNumber \mathrel{\widehat=} (RemoveNumber \land Success)$
$\lor NumberDoesntExist \lor NameNotInDirectory$

$TotalPhone \mathrel{\widehat=} TotalAddPerson \lor TotalRemovePerson$
$\lor TotalAddNumber \lor TotalRemoveNumber$

$TotalPhoneAndTotalAddPerson \mathrel{\widehat=} TotalPhone \lor TotalAddPerson$

A.4.2 ZCGa and ZDRa output



A.4.3 Messages when running the specification through the ZCGa and ZDRa checks

Figure A.3: Message when checking the specification for ZCGa correctness.

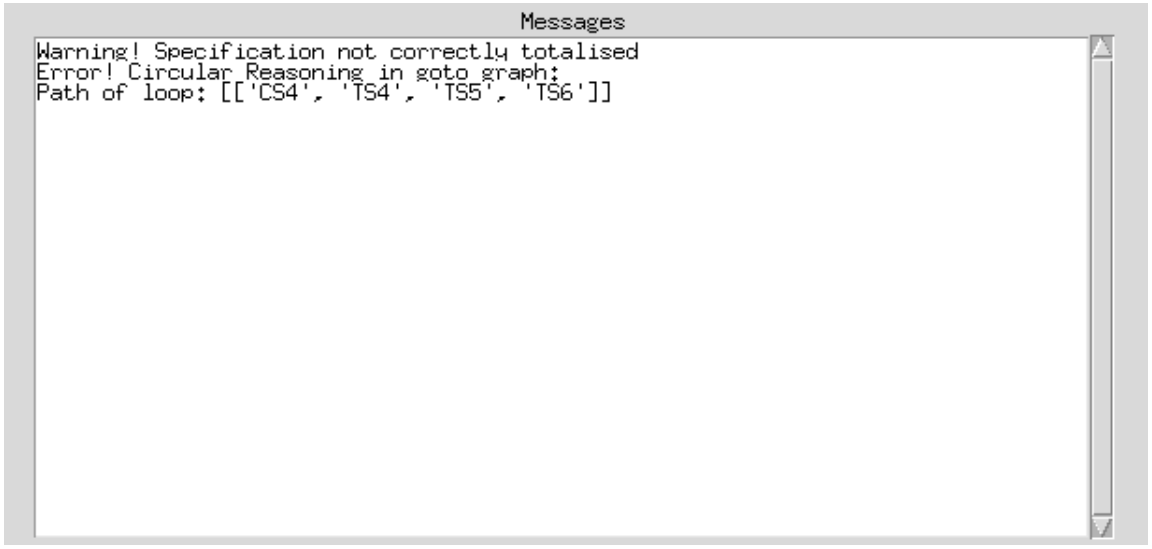


Figure A.4: Message when checking the specification for ZDRa correctness.

A.5 An example of a specification which is semi formal

This section shows an auto pilot specification which is partially written in natural language and partially written formally. Thus it is a natural langauge specification which is on it's way to being formalised.

A.5.1 Raw Latex output

1. The mode-control panel contains four buttons for selecting modes and three displays for dialing in or displaying values. The system supports the following four modes:

   - attitude control wheel steering (att_cws)
   - flight path angle selected (fpa_sel)
   - altitude engage (alt_eng)
   - calibrated air speed (cas_eng)

   $events ::= press\_att\_cws \mid press\_cas\_eng \mid press\_alt\_eng \mid press\_fpa\_sel$

   Only one of the first three modes can be engaged at any time. However, the cas_eng mode can be engaged at the same time as any of the other modes. The pilot engages a mode by pressing the corresponding button on the panel. One of the three modes, att_cws, fpa_sel, or alz_eng, should be engaged at all times. Engaging any of the first three modes will automatically cause the other two to be disengaged since only one of these three modes can be engaged at a time.

   $mode\_status ::= off \mid engaged$

   off_eng

   $mode : mode\_status$

   $mode = off \lor mode = engaged$

   AutoPilot

   $att\_cws : mode\_status$
   $fpa\_sel : mode\_status$
   $alt\_eng : mode\_status$
   $cas\_eng : mode\_status$

   att_cwsDo

   $\Delta AutoPilot$

   $att\_cws = off$
   $att\_cws' = engaged$
   $fpa\_sel' = off$
   $alt\_eng' = off$
   $cas\_eng' = off \lor engaged$

2. There are three displays on the panel: and altitude [ALT], flight path angle [FPA], and calibrated air speed [CAS]. The displays usually show the current values for the altitude, flight path angle, and air speed of the aircraft. However, the pilot can enter a new value into a display by dialing in the value using the knob next to the display. This is the target or "pre-selected" value that the pilot wishes the aircraft to attain. For example, if the pilot wishes to climb to 25,000 feet, he will dial 25,000 into the altitude display window and then press the alz_eng button to engage the altitude mode. Once the target value is achieved or the mode is disengaged, the display reverts to showing the "current" value.

3. If the pilot dials in an altitude that is more than 1,200 feet above the current altitude and then presses the alz_eng button, the altitude mode will not directly engage. Instead, the altitude engage mode will change to "armed" and the flight-path angle select mode is engaged. The pilot must then dial in a flight-path angle for the flight-control system to follow until the aircraft attains the desired altitude. The flight-path angle select mode will remain engaged until the aircraft is within 1,200 feet of the desired altitude, then the altitude engage mode is automatically engaged.

4. The calibrated air speed and the flight-path angle values need not be pre-selected before the corresponding modes are engaged–the current values displayed will be used. The pilot can dial-in a different target value after the mode is engaged. However, the altitude must be pre-selected before the altitude engage button is pressed. Otherwise, the command is ignored.

5. The calibrated air speed and flight-path angle buttons toggle on and off every time they are pressed. For example, if the calibrated air speed button is pressed while the system is already in calibrated air speed mode that mode will be disengaged. However, if the attitude control wheel steering button is pressed while the attitude control wheel steering mode is already engaged, the button is ignored. Likewise, pressing the altitude engage button while the system is already in altitude engage mode has no effect.

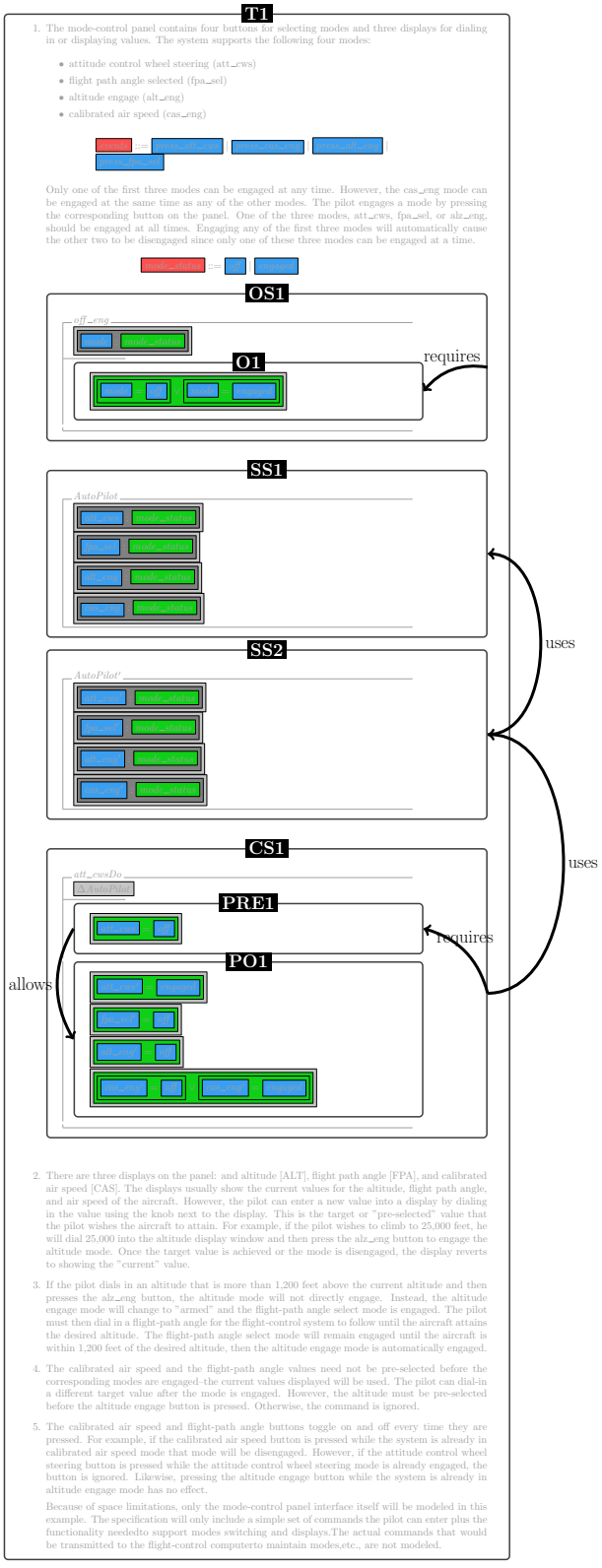
   Because of space limitations, only the mode-control panel interface itself will be modeled in this example. The specification will only include a simple set of commands the pilot can enter plus the functionality neededto support modes switching and displays.The actual commands that would be transmitted to the flight-control computerto maintain modes,etc., are not modeled.

A.5.2 ZCGa and ZDRa Annotated Latex Code

```
\documentclass{article}
\usepackage{zmathlang}

\begin{document}

\dratheory{T1}{0.4}{

\begin{enumerate}
\item The mode-control panel contains four buttons for selecting modes and three displays for
dialling in or displaying values. The system supports the following four modes:

\begin{itemize}
\item attitude control wheel steering (att\_cws)
\item flight path angle selected (fpa\_sel)
\item altitude engage (alt\_eng)
\item calibrated air speed (cas\_eng)
\end{itemize}

\begin{zed}
\set{events} ::= \term{press\_att\_cws} |  \term{press\_cas\_eng} | \term{press\_alt\_eng} | \\
 \term{press\_fpa\_sel}
\end{zed}

Only one of the first three modes can be engaged at any time. However, the cas\_eng mode
can be engaged at the same time as any of the other modes. The pilot engages a mode by pressing
the corresponding button on the panel. One of the three modes, att\_cws, fpa\_sel, or alz\_eng,
should be engaged at all times. Engaging any of the first three modes will automatically cause the
other two to be disengaged since only one of these three modes can be engaged at a time.

\begin{zed}
\set{mode\_status} ::= \term{off} | \term{engaged}
\end{zed}

\draschema{OS1}{
\begin{schema}{off\_eng}
\text{\declaration{\term{mode}: \expression{mode\_status}}}
\where
\draschema{O1}{
\text{\expression{\expression{\term{mode} = \term{off}} \lor \expression{\term{mode} = \term{engaged}}}}}
\end{schema}}

\requires{OS1}{O1}

\draschema{SS1}{
\begin{schema}{AutoPilot}
\text{\declaration{\term{att\_cws}: \expression{mode\_status}}} \\
\text{\declaration{\term{fpa\_sel}: \expression{mode\_status}}} \\
\text{\declaration{\term{alt\_eng}: \expression{mode\_status}}} \\
\text{\declaration{\term{cas\_eng}: \expression{mode\_status}}}
\end{schema}}
```

```
\draschema{SS2}{
\begin{schema}{AutoPilot'}
\text{\declaration{\term{att\_cws'}: \expression{mode\_status}}} \\
\text{\declaration{\term{fpa\_sel'}: \expression{mode\_status}}} \\
\text{\declaration{\term{alt\_eng'}: \expression{mode\_status}}} \\
\text{\declaration{\term{cas\_eng'}: \expression{mode\_status}}}
\end{schema}}

\uses{SS2}{SS1}

\draschema{CS1}{
\begin{schema}{att\_cwsDo}
\text{\Delta AutoPilot }
\where
\draline{PRE1}{
\text{\expression{\term{att\_cws} = \term{off}}}} \\
\draline{PO1}{
\text{\expression{\term{att\_cws'} = \term{engaged}}} \\
\text{\expression{\term{fpa\_sel'} = \term{off}}} \\
\text{\expression{\term{alt\_eng'} = \term{off}}} \\
\text{\expression{\expression{\term{cas\_eng'} = \term{off}} \lor
 \expression{\term{cas\_eng'} = \term{engaged}}}}}
\end{schema}}

\uses{CS1}{SS2}
\allows{PRE1}{PO1}
\requires{CS1}{PRE1}

\item There are three displays on the panel: and altitude [ALT], flight path angle [FPA], and
calibrated air speed [CAS]. The displays usually show the current values for the altitude, flight
path angle, and air speed of the aircraft. However, the pilot can enter a new value into a display by
dialling in the value using the knob next to the display. This is the target or "pre-selected" value that
the pilot wishes the aircraft to attain. For example, if the pilot wishes to climb to 25,000 feet, he
will dial 25,000 into the altitude display window and then press the alz\_eng button to engage the
altitude mode. Once the target value is achieved or the mode is disengaged, the display reverts to
showing the "current" value.

\item If the pilot dials in an altitude that is more than 1,200 feet above the current altitude and
then presses the alz\_eng button, the altitude mode will not directly engage. Instead, the altitude
engage mode will change to "armed" and the flight-path angle select mode is engaged. The pilot
must then dial in a flight-path angle for the flight-control system to follow until the aircraft attains
the desired altitude. The flight-path angle select mode will remain engaged until the aircraft is within
 1,200 feet of the desired altitude, then the altitude engage mode is automatically engaged.

\item The calibrated air speed and the flight-path angle values need not be pre-selected before the
corresponding modes are engaged--the current values displayed will be used. The pilot can dial-in
a different target value after the mode is engaged. However, the altitude must be pre-selected
before the altitude engage button is pressed. Otherwise, the command is ignored.

Because of space limitations, only the mode-control panel interface itself will be modelled in this
example. The specification will only include a simple set of commands the pilot can enter plus the
functionality needed to support modes switching and displays.The actual commands that would
be transmitted to the flight-control computer to maintain modes,etc., are not modelled.

\end{enumerate}
}
\end{document}
```

A.5.3 ZCGa and ZDRa output



A.5.4 Messages when running the specification through the ZCGa and ZDRa checks

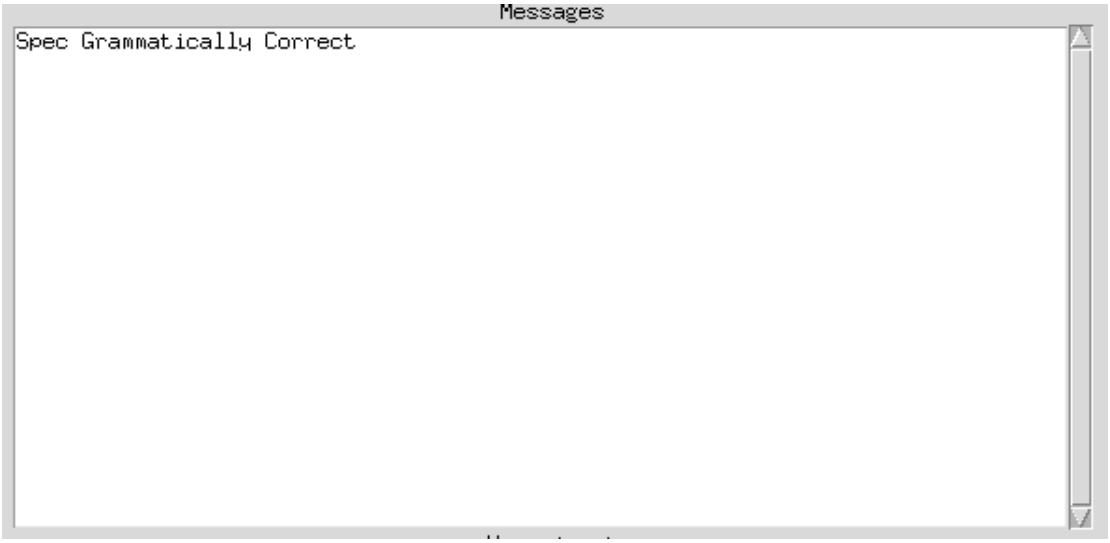


Figure A.5: Message when checking the specification for ZCGa correctness.

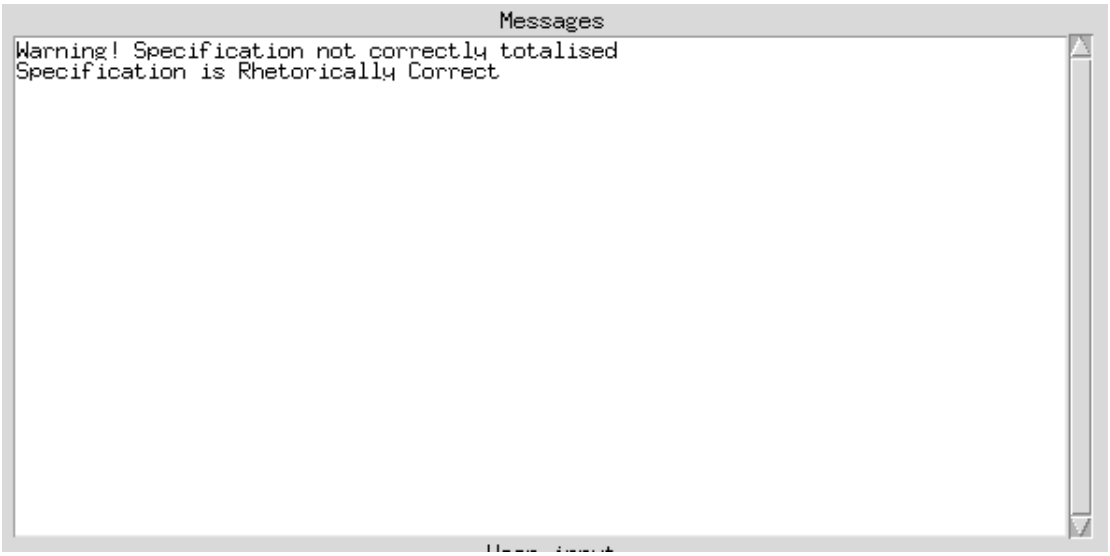


Figure A.6: Message when checking the specification for ZDRa correctness.

A.5.5 General Proof Skeleton

stateSchema SS1

outputSchema OS1

output O1

stateSchema SS2

changeSchema CS1

precondition PRE1

postcondition PO1

A.5.6 Isabelle Proof Skeleton

```
theory gpsa1n2
imports
Main

begin
(*DATATYPES*)

record SS1 =
(*DECLARATIONS*)

locale 1n2 =
fixes (*GLOBAL DECLARATIONS*)
begin

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

definition CS1 ::
"(*CS1_TYPES*) => bool"
where
"CS1 (*CS1_VARIABLES*) ==
 (PRE1)
∧ (PO1)"

end
end
```

A.5.7 Isabelle Filled In

```
theory 5
imports
Main

begin
datatype events = press_att_cws | press_cas_eng | press_alt_eng |
 press_fpa_sel
datatype mode_status = off | engaged


record AutoPilot =
ALT_ENG :: "mode_status"
CAS_ENG :: "mode_status"
ATT_CWS :: "mode_status"
FPA_SEL :: "mode_status"


locale theautopilot =
fixes alt_eng :: "mode_status"
and cas_eng :: "mode_status"
and att_cws :: "mode_status"
and fpa_sel :: "mode_status"

begin

definition off_eng ::
 "mode_status => bool"
where
"off_eng mode == (mode = off ∨ mode = engaged)"

definition att_cwsDo ::
"mode_status ⇒ mode_status ⇒ mode_status ⇒ mode_status => bool"
where
"att_cwsDo fpa_sel' cas_eng' att_cws' alt_eng'  ==
 (att_cws = off)
∧ (att_cws' = engaged)
∧ (fpa_sel' = off)
∧ (alt_eng' = off)
∧ (cas_eng' = off ∨
 cas_eng' = engaged)"

end
end
```

A.6 ModuleReg

A.6.1 ModuleReg Full Proof

This section shows the full proof for the modulereg example from [16]. It includes the filled in Isabelle skeleton. Along with added proofs which have been input by the user.

```
theory new6
imports
Main

begin
typedecl PERSON
typedecl MODULE

record ModuleReg =
STUDENTS :: " PERSON set"
DEGMODULES :: " MODULE set"
TAKING :: "(PERSON * MODULE) set"

locale Themodulereg =
fixes students :: " PERSON set"
and degModules :: " MODULE set"
and taking :: "(PERSON * MODULE) set"
assumes "Domain taking ⊆ students"
 and "Range taking ⊆ degModules"
begin

definition RegForModule ::
"ModuleReg ⇒ ModuleReg ⇒ PERSON ⇒ MODULE => MODULE set ⇒
 PERSON set ⇒ (PERSON * MODULE) set ⇒ bool"
where
"RegForModule modulereg modulereg' p m degModules' students' taking' ==
(p ∈ students)
∧ (m ∈ degModules)
∧ ((p, m) ∉ taking)
∧ (taking' = taking ∪ {(p, m)})
∧ (students' = students)
∧ (degModules' = degModules)"

definition AddStudent ::
"ModuleReg ⇒ ModuleReg => PERSON ⇒  MODULE set ⇒ PERSON set ⇒
(PERSON * MODULE) set ⇒ bool"
where
"AddStudent modulereg modulereg' p degModules' students' taking' ==
 (
(p ∉ students)
∧ (students' = students ∪ {(p)})
∧ (degModules' = degModules)
∧ (taking' = taking))"

(*The proof obligations generated by ZMathLang start here*)
```

```
lemma RegForModule_L1:
"(∃ degModules:: MODULE set.
∃ students :: PERSON set.
∃ taking :: (PERSON * MODULE) set.
∃ p :: PERSON.
∃ degModules':: MODULE set.
∃ students' :: PERSON set.
∃ taking' :: (PERSON * MODULE) set.
∃ m :: MODULE.
(
(p ∈ students)
∧ (m ∈ degModules)
∧ ((p, m) ∉ taking)
∧ (taking' = taking ∪ {(p, m)})
∧ (students' = students)
∧ (degModules' = degModules))
∧ (Domain taking ⊆ students)
∧ (Range taking ⊆ degModules)
∧ (Domain taking' ⊆ students')
∧ (Range taking' ⊆ degModules')
)"
by (smt Domain_empty Domain_insert Range.intros Range_empty
Range_insert Un_empty Un_insert_right empty_iff empty_subsetI
empty_subsetI insert_mono insert_mono singletonI singletonI
singleton_insert_inj_eq' singleton_insert_inj_eq')

lemma AddStudent_L2:
"(∃ degModules:: MODULE set.
∃ students :: PERSON set.
∃ taking :: (PERSON * MODULE) set.
∃ p :: PERSON.
∃ degModules':: MODULE set.
∃ students' :: PERSON set.
∃ taking' :: (PERSON * MODULE) set.
 (
 (students' = students ∪ {(p)})
∧ (degModules' = degModules)
∧ (taking' = taking))
∧ (Domain taking ⊆ students)
∧ (Range taking ⊆ degModules)
∧ (Domain taking' ⊆ students')
∧ (Range taking' ⊆ degModules')
)"
by blast

(*Here I add other safety properties about the ModuleReg specification
which I wish to prove*)

lemma pre_AddStudent:
"(∃ modulereg modulereg' students' degModules' taking'.
AddStudent modulereg modulereg' p degModules' students' taking')
⟷ (p ∉ students)"
apply (unfold AddStudent_def)
apply auto
done
```

```
lemma pre_RegForModule:
"(∃ modulereg modulereg' students' degModules' taking'.
RegForModule modulereg modulereg' p m degModules' students' taking')
 ⟷ (p ∈ students)
 ∧ (m ∈ degModules)
 ∧ ((p,m) ∉ taking)"
 apply (unfold RegForModule_def)
 apply auto
done

definition InitModuleReg::
"ModuleReg ⇒ PERSON set ⇒ MODULE set ⇒ (PERSON * MODULE) set ⇒ bool"
where
"InitModuleReg modulereg' students' degmodules' taking' == ((
(students' = {})
∧ (degmodules' = {})
∧ (taking' = {})))"

lemma InitOK:
"(∃ modulereg'. InitModuleReg modulereg' students' degmodules' taking')
⟶  ((
(students' = {})
∧ (degmodules' = {}))
∧ ((Domain taking' ⊆ students')
∧ (Range taking' ⊆ degModules'))"
by (simp add: InitModuleReg_def)

lemma RegForModuleNotEmpty:
"(∃ modulereg modulereg' students' degModules' taking' p m.
RegForModule modulereg modulereg' students' degModules' taking' p m)
 ⟶ (students ≠ {})
 ∧ (degModules ≠ {})"
by (smt RegForModule_def empty_iff empty_iff)

lemma notEmpty:
"(taking' = taking ∪ {(p,m)}) ⟶ (taking' ≠ {})"
by (smt Un_empty insert_not_empty)

end
end
```

Find year for atelier b proof obligation user manual

Bibliography

[1] HOL-Z 2.0: A Proof Environment for Z-Specifications. *Journal of Universal Computer Science*, 9(2):152–172, Feb. 2003.

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

[3] 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.

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

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

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

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

[8] 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.

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

[10] J. Bowen. Formal Methods Wiki, Z notation, July 2014. Accessed:01/07/2014.

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

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

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

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

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

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

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

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

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

[20] 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.

[21] D. Feller. Using MathLang to check the correctness of specification in Object-Z. 2015.

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

[23] 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.

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

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

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

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

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

[29] 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.

[30] 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.

[31] 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.

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

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

[34] 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.

[35] 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.

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

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

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

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

[40] 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.

[41] 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.

[42] 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.

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

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

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

[46] M. Mihaylova. ZMathLang User Interface Internship Report. 2015.

[47] M. Mihaylova. ZMathLang User Interface User Manual. 2015.

[48] 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.

[49] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. Srivas. PVS: Combining Specification, Proof Checking, and Model Checking. pages 411–414. Springer-Verlag, 1996.

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

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

[52] 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.

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

[54] G. Rossum. Python Reference Manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

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

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

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

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

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

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

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

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

[63] 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.

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

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