ProofPower

Compliance Tool — Proving VCs

PPTex-2.9.1w2.rda.110727

Copyright © : Lemma 1 Ltd. 2006

Information on the current status of ProofPower is available on the World-Wide Web, at URL:

`http://www.lemma-one.demon.co.uk/ProofPower/index.html`

This document is published by:

Lemma 1 Ltd.
2nd Floor
31A Chain Street
Reading
Berkshire
UK
RG1 2HX
e-mail: `pp@lemma-one.com`

PPTex-2.9.1w2.rda.110727 13:54:16 27/07/2011

CONTENTS

ABOUT THIS PUBLICATION

0.1 Purpose

This document gives guidance on the use of the Compliance Tool proof facilities supplied with ProofPower.

0.2 Readership

This document is intended to be read by users of the Compliance Tool who wish to produce machine-checked proofs of some or all of the VCs generated by the tool. These users are expected to have experience of using ProofPower for proofs in Z; in particular they should be familiar with the material in the ProofPower *Z Tutorial* [1].

0.3 Related Publications

A bibliography is given towards the end of this document.

- How to use the Compliance Tool is described in :

  Compliance Tool — User Guide [4].

- The syntax and semantics of the Compliance Notation as supported by the Compliance Tool is described in:

  Compliance Notation — Language Description [6]

- A description of ProofPower may be found in:

  ProofPower *Software and Services* [3],

  which also contains a full list of other ProofPower documentation.

- How to use ProofPower for formal reasoning about Z specifications may be found in:

  ProofPower *Z Tutorial* [1].

0.4 Area Covered

The user might typically have already undertaken the following tasks:

1. Installed the Compliance Tool on his workstation, by following the procedure described in the *Compliance Tool — Installation and Operation* [5]

2. Loaded a sequence of Compliance Notation scripts into the tool and generated the Z documents from the scripts, by following the procedure described in the *Compliance Tool — User Guide* [4]

This tutorial should then assist the user in working with the VCs, i.e. attempting to prove them, using ProofPower and Compliance Tool facilities. A Compliance Notation example concerning the computational aspects of a simple calculator is included in this tutorial, see chapter 4. Proofs are provided for some of the VCs generated by this example to illustrate the techniques advocated in chapter 3.

0.5 Prerequisites

This Tutorial is designed to assist a Compliance Tool user in the production of machine-checked proofs of the VCs generated by the tool. It is *not* intended to be an introduction to the Z language, or to the Compliance Notation, or indeed to the Compliance Tool itself.

Familiarity with the Compliance Notation is very desirable, although not essential since VC proofs may be conducted independently without prior knowledge of the specification from which the VCs have been generated. Familiarity with the Compliance Tool and with the use of ProofPower for proofs in Z is essential. It is assumed that a user intent on using the Compliance Tool for proving VCs will be familiar with the material in both the ProofPower *Z Tutorial* [1] and the *Compliance Tool — User Guide* [4].

The *Compliance Notation — Language Description* [6] describes the syntax and semantics of the Compliance Notation. The *Compliance Tool — User Guide* [4] gives an introduction to the use of the Compliance Tool for loading a Compliance Notation Script and generating the Z document which contains the VCs. The ProofPower *Z Tutorial* [1] gives an introduction to the use of ProofPower for specification and proof in Z.

The ProofPower user documentation is supplied as part of the ProofPower release included with the Compliance Tool and is available for on-line reference.

0.6 Acknowledgements

Sun Microsystems is a registered trademark of Sun Microsystems Inc. Sun-3, OpenWindows, Sun-4, SPARCstation, SunOS and Solaris are trademarks of Sun Microsystems Inc.

Motif is a registered trademark of the Open Software Foundation, Inc.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

Poly/ML is an implementation of Standard ML with a few non-standard extensions. Poly/ML, and its documentation, is copyright Abstract Hardware Limited.

TEX is copyright the American Mathematical Society and by Donald E. Knuth. The LATEX2ε distribution tape is copyright the LATEX 3 project and its individual authors.

The X Windows System is a trademark of the Massachusetts Institute of Technology.

INTRODUCTION

This tutorial has been designed to assist in the proof of VCs generated by the Compliance Tool. It is divided into three main areas:

- Chapter 2 provides a review of how to do proof in Z. Familiarity with the ProofPower *Z Tutorial* [1] is considered to be essential before embarking on VC proofs, see section 0.5, but the ProofPower *Z Tutorial* [1] covers a lot more than proof in Z. The material presented in chapter 2 concentrates on the proof aspects of ProofPower-Z . There are also some explicit examples in Z of material covered fully in the ProofPower *HOL Tutorial Notes* [2] but only briefly mentioned in the ProofPower *Z Tutorial* [1]. For example, linear arithmetic proofs in ProofPower-Z are very similar to those in ProofPower-HOL and as such are not covered in detail in the ProofPower *Z Tutorial* [1].
- Chapter 3 overviews the extra proof support available in the Compliance Tool and describes how to use these facilities to tackle VC proofs.
- Chapter 4 provides an example sequence of literate scripts concerning the computational aspects of a simple calculator. Selected VCs generated from these scripts are then proven, illustrating the use of techniques described in section 3.

For reference purposes, a complete listing of the calculator example theories is in appendix A. The SPARK program generated by the calculator example literate scripts is in appendix B.

As with many mathematical activities, proving VCs is best learnt by doing rather than reading. The recommended way of using this document is to work through the examples in an interactive session with the Compliance Tool. The source of this document is provided as part of the Compliance Tool release (as `$PPINSTALLDIR/docs/usr503.doc`) to help you do this.

REVIEW OF PROOF IN Z

This chapter provides an overview of the material in the ProofPower *Z Tutorial* [1] which describes how to use ProofPower-Z for doing proofs. It is assumed, in particular, that you have some familiarity with:

- how to do backwards proofs, using *set_goal* and work with the theorems you have proved using *pop_thm* and *save_pop_thm*

- proof contexts. The following have been used in the examples:

  set_pc, *push_pc*, *pop_pc*
  z_library1, *z_library1_ext*
  z_lin_arith
  z_predicates

  The example proofs in this chapter are all conducted in the proof context *z_library1*.

- tactics and tacticals, etc. The following have been used in the examples:

  tactics:

  strip_tac, *z_∀_tac*
  asm_rewrite_tac
  lemma_tac, *cases_tac*
  z_spec_asm_tac
  ante_tac, *discard_tac*
  fc_tac, *all_fc_tac*, *asm_fc_tac*, *all_asm_fc_tac*
  eq_sym_asm_tac, *eq_sym_nth_asm_tac*
  var_elim_asm_tac, *var_elim_nth_asm_tac*
  all_var_elim_asm_tac, *all_var_elim_asm_tac1*
  z_app_eq_tac
  z_≤_induction_tac

  tacticals:

  REPEAT
  ALL_FC_T, *ALL_FC_T1*
  ⇒_T
  THEN, *THEN1*
  PC_T1
  LEMMA_T
  DROP_NTH_ASM_T, *LIST_DROP_NTH_ASM_T*

  canonicalisation functions:

  fc_⇔_canon

- about forward inference: *rewrite_rule* and *∧_right_elim* have been used in the examples

- how to access the specification with *z_get_spec*

You will find detailed information on all the above in the ProofPower *Reference Manual* [8].

A full account of using ProofPower-Z for proofs may be found in the ProofPower *Z Tutorial* [1], which, in turn, refers to the ProofPower *HOL Tutorial Notes* [2]. There are some topics, for example linear arithmetic, which are fully covered in the ProofPower *HOL Tutorial Notes* [2] but to which no special treatment coverage is given in the ProofPower *Z Tutorial* [1]. This is because there is nothing extra to add about working in Z as opposed to HOL.

While it is assumed that you are familiar with the material presented in these tutorials, for convenience, a summary of the main methods advocated for dealing with Z proofs is given here. The recommended way of revising the material in this chapter is to read and try out the examples as you go along. You may also find it instructive to attempt the Z exercises from chapter 7 of the ProofPower *Z Tutorial* [1], the solutions to which are in chapter 8 of that tutorial.

The revision material is divided into the following sections:

2.1 Proof by the "two tactic" method.

2.2 Stripping.

2.3 Automatic Proof.

2.4 Forward Chaining.

2.5 Predicate Calculus with Equality.

2.6 Rewriting.

2.7 Function Application.

2.8 Proving Lemmas "on the fly".

2.9 Case Analysis.

2.10 Induction.

2.1 The Two Tactic Method

This method is given a high priority in the ProofPower *Z Tutorial* [1], although more for pedagogical reasons than because it is a particularly natural way to tackle a proof. Proof by stripping, see section 2.2, is effective in discharging a goal only where the reasoning is mainly propositional. Where the proof will depend either on appropriate specialisation of universally quantified assumptions, or on the choice of a suitable witness for proving an existential conclusion, stripping will not suffice.

The two tactic method injects into the proof process based on stripping, user directed specialisation of universal assumptions. In the context of a proof by contradiction (in which existential conclusions will not arise) this is sufficient to discharge any goals which are reduced to reasoning in the first order predicate calculus. The method is sometimes unnatural because it destroys much of the logical structure of the original goal. Schematically the method is:

SML

```
set_goal([],conjecture);
a contr_tac;                                   (* once suffices *)
a (z_spec_asm_tac ⓩ assumption ⌝ ⓩ value⌝);      (* as many times as necessary *)
```

The choice of universal assumptions and of the values to specialise them to depends on the user identifying one or more specialisations which will result in the derivation of a contradiction from the assumptions. For example, this method transforms a goal with an existentially quantified conclusion into one with a universally quantified assumption:

SML

```
set_goal([],ⓩ[X](∀x,y:X •(∃x:X • x = y))⌝);
a contr_tac;
a (z_spec_asm_tac ⓩ ∀ x : X • ¬ x = y ⌝ ⓩy⌝);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

2.2 Stripping

This method is complete for propositional logic. The tactic *strip_tac* performs a variety of simplifications, and is often usefully applied at the outset of embarking on a proof. The simplifications achieved by *strip_tac* include the following:

- moving the antecedent of an implication from the conclusion to the assumptions of the goal
- proving tautologies
- removing leading universal quantifiers
- using, where possible, relevant assumptions in the assumption-list.

This is often a more natural way to start a proof because it retains some of the structure of the original goal. For example, the proof above in section 2.1 could have been achieved by stripping then applying *z_∃_tac* with a suitable witness then stripping the trivial result:

SML

```
set_goal([],ⓩ[X](∀x,y:X •(∃x:X • x = y))⌝);
a(REPEAT strip_tac);
a(z_∃_tacⓩy⌝);
a(REPEAT strip_tac);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

2.3 Automatic Proof

An automatic proof procedure, in the form of *prove_tac*, is provided for each proof context. In most proof contexts this is capable of solving results which are reducible to simple theorems of the predicate calculus. For example, the proof context *z_library1_ext* reduces the subset relation to a universally quantified membership statement:

SML

```
set_goal([],ⓩA ⊆ B ∧ B ⊆ C ⇒ A ⊆ C⌝);
a(PC_T1"z_library1_ext"prove_tac[]);
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

SML

```
set_goal([],ⓩ{x,y:ℤ | x<y} ⊆ {x,y:ℤ | x<y ∨ x>y+99}⌝);
a(PC_T1"z_library1_ext"prove_tac[]);
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

Even when the application of *prove_tac* fails to prove a goal, it may have resulted in more simplification than would be obtained by other methods. It is also conceivable that if *prove_tac* fails to achieve a proof it may unnecessarily split the goal into subgoals. In this case it would probably be better to undo the application of *prove_tac* and try another approach to solving the goal.

2.3.1 Linear Arithmetic Proof Context

The proof context *z_lin_arith* contains an automatic proof procedure for linear arithmetic. This means terms built up from:

- "Atoms" (numeric literals, variables of type ℤ, etc.)
- Multiplication by numeric literals
- Addition
- $=, \leq, \geq, <, >$
- Logical operators

So, for example, the following are all proved by an application of *prove_tac* in the proof context *z_lin_arith*:

SML

```
set_goal([],ⓩ(∀x,y,z:ℤ | x ≤ y ∧ x + y < z + x • x < z)⌝);
a(PC_T1"z_lin_arith"prove_tac[]);
pop_thm();
```

SML

```
set_goal([],ⓩ(∀x,z:ℤ | (∃ y:Z • x ≥ y ∧ ¬ y < z )• x ≥ z)⌝);
a(PC_T1"z_lin_arith"prove_tac[]);
pop_thm();
```

SML

```
set_goal([],ⓩ(∀x,y:ℤ | x + 2*y < 2*x • y + y < x)⌝);
a(PC_T1"z_lin_arith"prove_tac[]);
pop_thm();
```

SML

```
set_goal([],ⓩ(∀x,y:ℤ • ¬ (2*x + y = 4 ∧ 4*x + 2*y = 7))⌝);
a(PC_T1"z_lin_arith"prove_tac[]);
pop_thm();
```

2.4 Forward Chaining

Forward chaining facilities often provide an easier way of achieving proofs requiring instantiation of universal assumptions.

all_asm_fc_tac will attempt to instantiate universally quantified assumptions which are effectively implications to values which will enable forward inference to take place. This is achieved by matching the antecedent of the implication against other assumptions.

Consider the example in section 2.1. After the application of *contr_tac*, you can derive the required contradiction using *all_asm_fc_tac* with less effort than having to specialise the universally quantified assumption:

SML

```
set_goal([],ⓩ[X](∀x,y:X •(∃x:X • x = y))⌝);
a contr_tac;
a (all_asm_fc_tac[]);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

If forward chaining fails to solve a goal, it may generate irrelevant new assumptions, and so it should be used judiciously.

A related tactic suitable for use with Z is *all_fc_tac*, which chains forward using implications derived from a list of theorems supplied as an argument, matching these against the assumptions, using the assumptions to match the antecedents of the implications.

fc_tac and *asm_fc_tac* are also useful (see ProofPower *Reference Manual* [8]), but these are liable to introduce HOL universal quantifiers, leaving a mixed language subgoal.

2.5 Predicate Calculus with Equality

A variety of additional proof facilities are available to make use of equations.

1. *asm_rewrite_tac*

   may be used to cause equations in the assumptions to rewrite the conclusion of a subgoal. This may sometimes prove sufficient to complete a proof.

2. *eq_sym_asm_tac* or *eq_sym_nth_asm_tac*

   may be used to turn round an equation in an assumption which is the wrong way round to achieve the required rewrite.

3. *var_elim_asm_tac* or *var_elim_nth_asm_tac*

   may be used to completely eliminate from the subgoal occurrences of a variable which appears on one side of an equation in the specified assumption. This causes the conclusion and all the other assumptions to be rewritten with the equation, eliminating occurrences of it. The assumption will then be discarded. These tactics will work whichever way round the equation appears in the assumption.

4. *all_var_elim_asm_tac*, *all_var_elim_asm_tac1*

   automatically eliminate from the assumptions all equations of a sufficiently simple kind, by rewriting every term in the subgoal with them and then discarding the equations. They avoid eliminating equations where this might cause a looping rewrite. The first variant only eliminates equations where both sides are either variables or constants, the second variant will eliminate any equation of which one side is a variable which does not appear on the other side.

2.6 Rewriting

Rewriting using any collection of theorems from which equations are derivable is supported by the standard HOL rewriting facilities (*rewrite_tac* etc.), see the ProofPower *Reference Manual* [8]for details, using Z specific preprocessing of the rewrite theorems (supplied in the Z proof contexts).

Many Z paragraphs give rise to predicates which can be used without further preparation by these standard rewriting facilities. This applies to given sets, abbreviation definitions and schema definitions.

Axiomatic descriptions, and generic axiomatic descriptions will result in equations which are likely to be effectively conditional. In such cases it is necessary to establish the applicability of the rewrite before it can be undertaken.

One way of achieving this is by forward chaining using the conditional equation after establishing the relevant condition. The relevant conditions are usually the membership assertions corresponding to the declaration part of the outer universal quantifier on the theorem to be used for rewriting.

For example, to prove the goal:

SML
```
set_goal([], ⓩ ∀ i:ℕ • abs i = abs ~i⌝);
```

using theorem *z_abs_thm* (which is : $\vdash \forall\ i : \mathbb{N} \bullet abs\ i = i \land abs \sim i = i$). First strip the goal:

SML

```
a (REPEAT z_strip_tac);
```

ProofPower output

```
...
(*  1 *)  ⓩ0 ≤ i⌝

(* ?⊢ *)  ⓩabs i = abs ~ i⌝
...
```

Then forward chain using the theorem and rewrite with the results:

SML

```
a (ALL_FC_T rewrite_tac [z_abs_thm]);
save_pop_thm "abs_eq_abs_minus_thm";
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

In more complicated cases the proof of the required conditions may be non-trivial, often because reasoning about membership of expressions formed with function application is involved. The next section describes the proof support available for use in such cases.

2.7 Function Application

Reasoning at a low level, *z_app_eq_tac* may be used to reduce an equation involving an application to sufficient conditions for its truth, in terms of the membership of the function, e.g.:

SML

```
set_goal([],ⓩ f a = v⌝);
a z_app_eq_tac;
```

ProofPower output

```
...
(* ?⊢ *)  ⓩ(∀ f_a : 𝕌 | (a, f_a) ∈ f • f_a = v) ∧ (a, v) ∈ f⌝
...
```

The first conjunct of this result is needed to ensure that f is functional at a (i.e. maps a to only one value). In the case that f is known to be a function, the theorem *z_fun_app_clauses* may be used with forward chaining, avoiding the need to prove that f is functional at a.

```
val z_fun_app_clauses =
  ⊢ ∀ f : 𝕌; x : 𝕌; y : 𝕌; X : 𝕌; Y : 𝕌
    • (f ∈ X ⇸ Y
          ∨ f ∈ X ⤔ Y
          ∨ f ∈ X ⤀ Y
          ∨ f ∈ X → Y
          ∨ f ∈ X ↣ Y
          ∨ f ∈ X ↠ Y
          ∨ f ∈ X ⤖ Y)
        ∧ (x, y) ∈ f
      ⇒ f x = y : THM
```

In this case the result $(a, v) \in f$ would have to be proven and added to the assumptions before undertaking the forward chaining, e.g.:

SML

```
drop_main_goal();
set_goal([], ⓩ[X,Y](∀ f : X → Y; x:X; y:Y  • (x, y) ∈ f ⇒ f x = y)⌝);
a (REPEAT z_strip_tac);
```

ProofPower output

```
...
(∗  4 ∗)  ⓩf ∈ X → Y⌝
(∗  3 ∗)  ⓩx ∈ X⌝
(∗  2 ∗)  ⓩy ∈ Y⌝
(∗  1 ∗)  ⓩ(x, y) ∈ f⌝

(∗ ?⊢ ∗)  ⓩf x = y⌝
```

SML

```
a (all_fc_tac [z_fun_app_clauses]);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

A common problem is to have to establish that the value of some expression formed by application falls within some particular set. This is often needed to establish the conditions necessary for use of a rewriting equation on the expression.

In these circumstances the theorem *z_fun_∈_clauses* may be used:

```
val z_fun_∈_clauses = ⊢ ∀ f : 𝕌; x : 𝕌; X : 𝕌; Y : 𝕌
 • ((f ∈ X → Y ∨ f ∈ X ↣ Y ∨ f ∈ X ↠ Y ∨ f ∈ X ⤖ Y) ∧ x ∈ X
        ⇒ f x ∈ Y)
 ∧ ((f ∈ X ⇸ Y ∨ f ∈ X ⤔ Y ∨ f ∈ X ⤀ Y) ∧ x ∈ dom f
        ⇒ f x ∈ Y) : THM
```

The claim that a global variable is a member of a function space will often be obtained from the specification of the constant (as part of the predicate implicit in the declaration part of the specification). Where the function is an expression the result is likely to have been established by forward inference using similar methods.

For example, consider the following specifications:

Z
```
[T]
```

Z
```
CONSTSPEC : ℙ1 T ↠ ℕ
─────────────────────────
true
```

where we are required to demonstrate that *CONSTSPEC* applied to something in T is in the set ℕ. From *z_fun_∈_clauses* we have that given an $f \in X \twoheadrightarrow Y$ and an $x \in X$, then we can conclude that $f\ x \in Y$. The current proof context, *z_library1*, will be too agressive for our purposes because it will expand the definition of P_1, the X in this case. So, we will use the proof context *z_predicates* for this example:

SML
```
push_pc"z_predicates";
set_goal([], ⓩ(∀ x:ℙ1 T• CONSTSPEC x ∈ ℕ)⌝);
a(REPEAT strip_tac);
```

ProofPower output
```
...
(∗  1 ∗)  ⓩx ∈ ℙ1 T⌝

(∗ ?⊢ ∗)  ⓩCONSTSPEC x ∈ ℕ⌝
...
```

We need the fact about *CONSTSPEC* which is found in its defining declaration to make the required inference.

This is added to the assumptions as follows:

SML
```
a (strip_asm_tac (z_get_spec ⓩCONSTSPEC⌝));
```

ProofPower output
```
...
(∗  2 ∗)  ⓩx ∈ ℙ1 T⌝
(∗  1 ∗)  ⓩCONSTSPEC ∈ ℙ1 T ↠ ℕ⌝

(∗ ?⊢ ∗)  ⓩCONSTSPEC x ∈ ℕ⌝
...
```

Next we forward chain using the theorem *z_fun_∈_clauses*, which suffices to discharge the goal.

SML

```
a (all_fc_tac[z_fun_∈_clauses]);
pop_thm();
pop_pc();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
...
```

2.8 Using Lemmas

Use of the tactic *lemma_tac* may give a more natural feel to a proof. It allows you to state and prove a lemma “on the fly”. This will generate at least two subgoals - one the statement of the lemma and the rest the result of stripping the lemma into the assumptions. For example:

SML

```
set_goal([],ⓩ(∀x,y :ℤ | x  ≤ y • P (x,y)) ∧ x = y ⇒ P (x,y)⌝);
a(REPEAT strip_tac);
```

ProofPower output

```
...
(*  2 *)  ⓩ∀ x, y : ℤ | x ≤ y • P (x, y)⌝
(*  1 *)  ⓩx = y⌝

(* ?⊢ *)  ⓩP (x, y)⌝
...
```

SML

```
a(lemma_tacⓩx ≤ y⌝);
```

ProofPower output

```
Tactic produced 2 subgoals:

(* *** Goal "2" *** *)

(*  3 *)  ⓩ∀ x, y : ℤ | x ≤ y • P (x, y)⌝
(*  2 *)  ⓩx = y⌝
(*  1 *)  ⓩx ≤ y⌝

(* ?⊢ *)  ⓩP (x, y)⌝


(* *** Goal "1" *** *)

(*  2 *)  ⓩ∀ x, y : ℤ | x ≤ y • P (x, y)⌝
(*  1 *)  ⓩx = y⌝

(* ?⊢ *)  ⓩx ≤ y⌝
```

Rewriting with the assumptions will solve the first goal:

SML

```
a(asm_rewrite_tac[]);
```

ProofPower output

```
Tactic produced 0 subgoals:
Current goal achieved, next goal is:

(* *** Goal "2" *** *)
...
```

and forward chaining will solve the second:

SML

```
a(all_asm_fc_tac[]);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
...
```

In this example, one tactic solved the lemma. If you apply this with *THEN1*, you can avoid the subgoal split:

SML

```
set_goal([],ⓩ(∀x,y :ℤ |x  ≤ y • P (x,y)) ∧ x = y ⇒ P (x,y)⌝);
a(REPEAT strip_tac);
a(lemma_tacⓩx ≤ y⌝ THEN1 asm_rewrite_tac[]);
```

ProofPower output

```
Tactic produced 1 subgoal:

(* *** Goal "" *** *)

(*  3 *)  ⓩ∀ x, y : ℤ | x ≤ y • P (x, y)⌝
(*  2 *)  ⓩx = y⌝
(*  1 *)  ⓩx ≤ y⌝

(* ?⊢ *)  ⓩP (x, y)⌝
...
```

SML

```
a(all_asm_fc_tac[]);
pop_thm();
```

Applying *lemma_tac* is equivalent to applying *LEMMA_T strip_asm_tac*. *LEMMA_T* has an argument of type *THM − > TACTIC* telling you what to do with the new assumption. Sometimes, you really want to put the lemma into the assumptions in exactly the way you have formulated it: *LEMMA_T asm_tac* will achieve this. Another common thing you might want to do is to rewrite your goal with the new assumption: *LEMMA_T rewrite_thm_tac* will achieve this.

2.9 Case Analysis

The tactic *cases_tac condition* lets you reason by cases according as a chosen *condition* is true or false. Consider the example from section 2.8 that was progressed with *lemma_tac*. This time, we will use *cases_tac* which will generate a contradiction in the case where the condition is false:

SML

```
set_goal([],⌜Z(∀x,y :ℤ |x ≤ y • P (x,y)) ∧ x = y ⇒ P (x,y)⌝);
a(REPEAT strip_tac);
```

ProofPower output

```
...
(∗  2 ∗)  ⌜Z∀ x, y : ℤ | x ≤ y • P (x, y)⌝
(∗  1 ∗)  ⌜Zx = y⌝

(∗ ?⊢ ∗)  ⌜ZP (x, y)⌝
...
```

SML

```
a(cases_tac⌜Zx ≤ y⌝);
```

ProofPower output

```
Tactic produced 2 subgoals:
(∗ ∗∗∗ Goal "2" ∗∗∗ ∗)

(∗  3 ∗)  ⌜Z∀ x, y : ℤ | x ≤ y • P (x, y)⌝
(∗  2 ∗)  ⌜Zx = y⌝
(∗  1 ∗)  ⌜Z¬ x ≤ y⌝

(∗ ?⊢ ∗)  ⌜ZP (x, y)⌝


(∗ ∗∗∗ Goal "1" ∗∗∗ ∗)

(∗  3 ∗)  ⌜Z∀ x, y : ℤ | x ≤ y • P (x, y)⌝
(∗  2 ∗)  ⌜Zx = y⌝
(∗  1 ∗)  ⓩx ≤ y⌝

(∗ ?⊢ ∗)  ⓩP (x, y)⌝
```

This time, forward chaining proves the first subgoal:

SML

```
a(all_asm_fc_tac[]);
```

ProofPower output

```
Tactic produced 0 subgoals:
Current goal achieved, next goal is:

(∗ ∗∗∗ Goal "2" ∗∗∗ ∗)
...
```

Eliminating x should derive the expected contradiction to solve subgoal 2:

SML

```
a(all_var_elim_asm_tac1);
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

2.10 Induction

Induction tactics for integers are available. The easiest one to use is *z_≤_induction_tac*. Consider the following goal, for example:

SML

```
set_goal([],ⓩ∀ i,j:ℤ | 0 ≤  i ∧ 0 ≤ j • 0 ≤ i∗j⌝);
a(REPEAT strip_tac);
```

ProofPower putput

```
...
(∗  2 ∗)  ⓩ0 ≤ i⌝
(∗  1 ∗)  ⓩ0 ≤ j⌝

(∗ ?⊢ ∗)  ⓩ0 ≤ i ∗ j⌝
```

Now apply the induction tactic:

SML

```
a(z_≤_induction_tacⓩi⌝);
```

ProofPower output

```
Tactic produced 2 subgoals:

(* *** Goal "2" *** *)

(*  3 *)  ⓩ0 ≤ j⌝
(*  2 *)  ⓩ0 ≤ i⌝
(*  1 *)  ⓩ0 ≤ i * j⌝

(* ?⊢ *)  ⓩ0 ≤ (i + 1) * j⌝


(* *** Goal "1" *** *)

(*  1 *)  ⓩ0 ≤ j⌝

(* ?⊢ *)  ⓩ0 ≤ 0 * j⌝
```

Subgoal 1 is proved by rewriting with the assumptions.

SML

```
a(asm_rewrite_tac[]);
```

Although the original goal was not a linear arithmetic result, induction has reduced the problem to one of linear arithmetic. Subgoal 2 can be proved in the proof context *z_lin_arith*:

SML

```
a(PC_T1"z_lin_arith"asm_prove_tac[]);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

PROVING VCS

As ordinary Z goals, VCs may be proved using all of the normal facilities provided by ProofPower for proof in Z. Chapter 2 provides an overview, based on the ProofPower *Z Tutorial* [1], of the recommended techniques for using ProofPower-Z for doing proofs. In addition, many VCs use Z toolkit extensions which are contained in the theory *cn*. Some custom support is provided in the Compliance Tool to assist with reasoning in this theory, most notably the proof contexts *cn1*, *cn_ext* and *cn*. There are also some purpose built tactics available, e.g. *cn_vc_simp_tac* and *cn_∈_type_tac* which have been designed for use with VC proofs. The main objective of this chapter is to show you how to use these tactics with facilities available in the proof context *cn1* to strip away the "Compliance Tool specific" bits of a VC proof to transform it into an "ordinary" Z proof. From there, hopefully the material in section 2 will guide you in progressing your proof.

3.1 Getting Started

It is assumed that you are using the Compliance Tool and are in a position to access the VCs, i.e. you are either in the same theory as your specification, or in a theory which has your specification as a parent. (*Compliance Tool — User Guide* [4] gives a comprehensive account of the Compliance Notation functions, in particular how to access the VCs generated by the tool from a literate script).

Processing a Compliance Notation script gives rise to definitions and axioms that are not necessarily in a form that makes for easy reasoning. Before you tackle the VC proofs, there is some Compliance Tool support available that generates some theorems for you that should facilitate rewriting when proving the VCs. First call the function *all_cn_make_script_support* with your choice of name as string argument. For example:

```
val my_thms = all_cn_make_script_support "mycn";
```

generates all these supporting theorems, binds them (in a list) to the ML variable *my_thms*, and generates the supporting proof context *mycn*. After a few seconds, you will see a raft of theorems scroll by in the journal window. Each of the definitions and axioms generated by processing the Compliance Notation give rise to one or two supporting theorems. There is always one rewriting theorem, and if this does not contain all the type information implicit in the definition, there will be another signature theorem. These theorems are at least as good as what you get with *z_get_spec*, and in many cases are in a much better form for rewriting.

All supporting theorems are stored in the current theory, and are prefixed by *cn_*. The rewriting theorems finish with *_thm*, and the signature theorems finish with *_sig_thm*. The theorems are also bound to ML variables of the same name. Such theorems will have been generated from all the definitions in your theory. For example, the type *OPERATION* in the package *TC* in the calculator example, see section 4, gives rise to the definition *TCoOPERATION*, for which one supporting theorem has been generated:

```
cn_TCoOPERATION_thm
        ⊢ TCoOPERATION = TCoPLUS .. TCoEQUALS
```

and an attribute definition *TCoOPERATIONvPOS*, for which two supporting theorems have been generated:

$$\begin{array}{l}
cn\_TCoOPERATIONvPOS\_thm \\
\qquad \vdash \forall\ i : TCoOPERATION \bullet TCoOPERATIONvPOS\ i = i \\
cn\_TCoOPERATIONvPOS\_sig\_thm \\
\qquad \vdash TCoOPERATIONvPOS \in TCoOPERATION \rightarrow TCoOPERATION
\end{array}$$

The supporting proof context *mycn* created for you is the proof context *cn1* extended by these supporting theorems. It is not the intention that you should normally set *mycn* as a proof context, because it would typically be too aggressive. The effect of rewriting in proof context *mycn* would be to unwind everything in a goal with its basic definition. However, there are some cases when this is exactly what you want to do, for example, when numerical values in a specification are significant. In this case, you would probably have set the proof context to *cn1* at the start of the proof, and then at some appropriate stage applied the following:

```
a(PC_T1 "mycn" rewrite_tac[]);
```

The theory *cn* contains the Compliance Tool Z toolkit extensions. The recommended user interface to the conversions etc. described there is via the proof context *cn1*. In this proof context the SPARK boolean and relational operators are converted fairly directly into Z. Additional theorems in the theory *cn* are also available for reasoning about the numeric operators *intdiv*, *intmod* and *rem*. The more aggressive proof contexts, *cn* and *cn_ext*, are also available. These, and all the other custom proof tools are described in section 6.1 of the *Compliance Tool — User Guide* [4]. For references purposes, a full listing of the theory *cn* is also provided in that user guide.

3.2 Compliance Tool Proofs

Application of the tactic *cn_vc_simp_tac*, see section 6.1 of *Compliance Tool — User Guide* [4], is the favoured way of beginning a VC proof. In all but the most obscure cases this should simplify the goal, and may even be sufficient to achieve a proof. *cn_vc_simp_tac* first rewrites the conclusion of the goal with the rewriting rules of the current proof context, some associativity theorems, and any theorems of your choice. Outer universal quantifiers are stripped away and any resulting redundancy in the goal removed. For example, using the proof context *cn1*, *cn_vc_simp_tac*, will transform the goal:

$$\begin{array}{lll}
?\vdash & \forall & x : INTEGER;\ y : INTEGER;\ z : INTEGER \\
 & | & (x + y) + 1\ eq\ z = TRUE \land (x \geq 0 \land y \geq 0) \land x \geq 0 \\
 & \bullet & x \geq 0 \land z\ greater\_eq\ 0 = TRUE
\end{array}$$

into:

$$\begin{array}{lll}
?\vdash & & x \in INTEGER \land y \in INTEGER \land z \in INTEGER \\
 & \land & x + y + 1 = z \land 0 \leq x \land 0 \leq y \\
 & \Rightarrow & 0 \leq z
\end{array}$$

You should now be in a position to prove the VCs. The names of the VCs can be obtained from the current theory using *get_conjectures* ” – ”.

First, set the proof context, typically *cn1*, and set the VC, called, say, *vcn_n*, as a ProofPower-Z goal.

SML

```
set_pc"cn1";
set_goal([], get_conjecture"-""vcn_n");
```

Now apply the simplification tactic *cn_vc_simp_tac*. This may achieve the proof; if not and you believe that the goal is simply a predicate calculus result, perhaps with a bit of linear arithmetic, then you may care to attack it with *prove_tac* (in the linear arithmetic proof context if applicable). In general, though, this type of automatic proof procedure would not be appropriate. Next, you might try the following (until the goal is achieved):

- Apply *REPEAT strip_tac*. If this results in the generation of additional subgoals, you may want to “undo 1” and then perform more controlled stripping. For example

  SML

  ```
  set_pc"cn1";
  set_goal([],ⓩ(∀ x : X; y : X
              | P(x) ∧ Q(x)  ∧ x eq y = TRUE
              • P(y) ∧ Q(y))⌝);
  a(cn_vc_simp_tac[]);
  ```

  ProofPower output

  ```

  (* *** Goal "" *** *)

  (* ?⊢ *)  ⓩx ∈ X ∧ y ∈ X ∧ P x ∧ Q x ∧ x = y ⇒ P y ∧ Q y⌝

  ```

  REPEAT strip_tac will generate two subgoals from the conjunction in the right hand side of the implication obtained by *cn_vc_simp_tac*:

  SML

  ```
  a(REPEAT strip_tac);
  ```

  ProofPower output

  ```

  Tactic produced 2 subgoals:

  (* *** Goal "2" *** *)

  (*  5 *)  ⓩx ∈ X⌝
  (*  4 *)  ⓩy ∈ X⌝
  (*  3 *)  ⓩP x⌝
  (*  2 *)  ⓩQ x⌝
  (*  1 *)  ⓩx = y⌝

  (* ?⊢ *)  ⓩQ y⌝

  (* *** Goal "1" *** *)
  ```

```

(* 5 *) ⓩx ∈ X⌝
(* 4 *) ⓩy ∈ X⌝
(* 3 *) ⓩP x⌝
(* 2 *) ⓩQ x⌝
(* 1 *) ⓩx = y⌝

(* ?⊢ *) ⓩP y⌝

```

Whereas *⇒_tac* will strip the left hand side of the implication into the assumptions without affecting the right hand side:

SML

```
undo 1;
a ⇒_tac;
```

ProofPower output

```

Tactic produced 1 subgoal:

(* *** Goal "" *** *)

(* 5 *) ⓩx ∈ X⌝
(* 4 *) ⓩy ∈ X⌝
(* 3 *) ⓩP x⌝
(* 2 *) ⓩQ x⌝
(* 1 *) ⓩx = y⌝

(* ?⊢ *) ⓩP y ∧ Q y⌝
```

- Stripping may not suceed if it generates an assumption which is an equation with a variable on one side of the equals. Eliminating that variable throughout, with a variation on the theme of *var_elim_asm_tac*, before using *asm_rewrite_tac* or *asm_fc_tac* may solve the goal. The example above illustrates this. Eliminating *x* from assumption *1* then rewriting with the assumptions should complete proof:

  SML

  ```
  a(all_var_elim_asm_tac1);
  a(asm_rewrite_tac[]);
  pop_thm();
  ```

  ProofPower output

  ```
  Tactic produced 0 subgoals:
  Current and main goal achieved
  ```

- A For Loop Statement may give rise to a VC with an assumption which is quantified over a range *i* .. *i* − *1*. Such an assumption is false, and the application of *asm_prove_tac* in the linear arithmetic proof context will suffice to prove the VC:

  PC_T1"*z_lin_arith*"*asm_prove_tac*[]

- Rewriting is usually the next thing to try. There are two "levels" of definitions that can be unwound:

  - the SPARK definitions in the specification, e.g. the attributes *TCoDIGITvFIRST* and *TCoDIGITvLAST* in the calculator example, section 4
  - the Z in the specification, e.g. the Z schema *DO_DIGIT* in the calculator example

  Perhaps you could progress your proof by rewriting explicitly with the supporting theorems *cn_TCoDIGITvFIRST_thm*, or *cn_DO_DIGIT_thm* that were generated for you as described in section 3.1.

  It is worth remembering that forward chaining is often required in order to obtain usable rewrites from Z definitions. For example, the theorem available from the definition of *fact* in the calculator example, section 4, is:

  $$\begin{array}{|l} \vdash \mathit{fact} \in \mathbb{N} \rightarrow \mathbb{N} \\ \quad \land\ \mathit{fact}\ \mathit{0} = \mathit{1} \\ \quad \land\ (\forall\ m : \mathbb{N} \bullet \mathit{fact}\ (m + \mathit{1}) = (m + \mathit{1}) * \mathit{fact}\ m) \end{array}$$

  To use the third conjunct of this theorem, it is necessary to have an assumption of the form $m \in \mathbb{N}$ so that you can forward chain, with, say, *fc_tac*, to obtain the result that $\mathit{fact}\ (m + \mathit{1}) = (m + \mathit{1}) * \mathit{fact}\ m$. It is also not uncommon to have the following pattern in a Z axiomatic definition:

  $$\begin{array}{|l} \mathit{SOMEPROPERTY} : X \rightarrow \mathbb{P}\ Y \\ \hline \forall\ x : X;\ y : Y\ \bullet y \in \mathit{SOMEPROPERTY}\ x \Leftrightarrow \mathit{SOMEPREDICATE} \end{array}$$

  Suppose you are required to prove the goal:

  $$\begin{array}{|l} a \in \mathit{SOMEPROPERTY}\ b \end{array}$$

  Then, useful rewriting may be achieved by forward chaining using

  $$\begin{array}{|l} \mathit{ALL\_FC\_T1}\ \mathit{fc\_\Leftrightarrow\_canon}\ \mathit{rewrite\_tac}[\mathit{z\_get\_spec}\ulcorner_{Z}\mathit{SOMEPROPERTY}\urcorner] \end{array}$$

- Some insight as to "what to do next" may be gleaned from chapter 2 . The degree of difficulty experienced in proving VCs rather depends on the refinement steps which generated the VCs in the first place. A complex sequence of intermediate refinements gives rise to complex VCs, whereas refinements which rely heavily on the underlying Z may require you to unwind a lot of definitions.

  It is worth remembering at this time that there is always the possibility that the VC you are trying to prove may not be true, or more likely, may not be provable. You may save yourself pain if, before you embark on the proof using the tool, you spend a few minutes at the outset inspecting the VC to convince yourself that you have a fair chance of achieving a proof. Problems will occur if the pre- condition of a refinement statement is too weak, or the post-condition too strong, to prove the VC.

Typically you may find it advantageous to prove subsidiary lemmas before embarking upon a VC proof. These will be either general purpose lemmas that are not available in ProofPower-Z but can easily be proven with the proof support available, or they will be application specific. Such lemmas are proven separately either because they are results that are found to be needed repeatedly during the process of proving the VCs, or because proving them in isolation is simpler than proving them in the context of a VC proof. Actually, in practice, it is common to experience a sense of *deja vu* when in the middle of a proof. You find that you need a result that you have already proved in an earlier VC proof, and it is at that stage that you decide to backtrack and extract the bits from the previous proof as a separate lemma. Section 3.3 provides examples of the sort of subsidiary lemmas that might typically be useful.

3.3 Additional Techniques

The *Compliance Notation — Language Description* [6] describes how particular SPARK constructs are translated into Z. Some of these constructs give rise to idioms requiring special styles of proof.

3.3.1 Array Component Assignments

Functional overrides occur in the VCs from SPARK array assignments. These assignments can be to an arbitrary level of nesting. Suppose in your VC proofs, the following lemma would be useful:

$$\begin{array}{|l}
\oplus\_lemma \\
[X,\ Y,\ Z](\forall f : X \rightarrow Y \rightarrow Z;\ x{:}X; y2{:}Y; y1{:}\mathbb{U}; z{:}\mathbb{U} \mid \neg y2{=}y1 \bullet \\
\qquad (f \oplus \{x \mapsto f\ x \oplus \{y1 \mapsto z\}\})x\ y2{=}\ f\ x\ y2)
\end{array}$$

Although this does not exist as a theorem in ProofPower-Z, it is straightforward for you to prove as a separate lemma using *z_fun_∈_clauses*, as described in section 2.7, together with the theorems *z_⊕_↦_app_thm* and *z_⊕_↦_app_thm1*:

$$\begin{array}{|l}
z\_\oplus\_\mapsto\_app\_thm \\
\vdash \forall\ f : \mathbb{U};\ x : \mathbb{U};\ y : \mathbb{U} \bullet (f \oplus \{x \mapsto y\})\ x = y
\end{array}$$

$$\begin{array}{|l}
z\_\oplus\_\mapsto\_app\_thm1 \\
\vdash\ [X,Y](\forall\ f : X \rightarrow Y;\ x2 : X;\ x1 : \mathbb{U};\ y : \mathbb{U} \mid \neg\ x2 = x1 \bullet \\
\qquad (f \oplus \{x1 \mapsto y\})\ x2 = f\ x2)
\end{array}$$

SML

```
set_goal([],ⓩ[X, Y, Z](∀f : X → Y → Z; x:X;y2:Y;y1:𝕌;z:𝕌| ¬y2=y1•
        (f ⊕ {x ↦ f x ⊕ {y1 ↦ z}})x y2= f x y2)⌝);
a(REPEAT strip_tac);
a(rewrite_tac[z_⊕_↦_app_thm]);
a(all_asm_fc_tac[z_fun_∈_clauses]);
a(ALL_ASM_FC_T rewrite_tac[z_⊕_↦_app_thm1]);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

3.3.2 Set Membership of Record Components

The form of this class of lemma is $X.field \in set$, which arises from indexing into an array of records in SPARK. The tactic *cn_∈_type_tac* is available to simplify the proof of this type of lemma. A typical example of the set in question is *BOOLEAN*. For example, consider the following specification snippet from a Compliance Notation script:

```
subtype SWITCHTYPE is INTEGER range 1 .. 3;
 type SWITCHDATA is
    record
       STATE : BOOLEAN;  −− on or off
       NEXTSWITCH : SWITCHTYPE;
    end record;

 type SWITCHBOARD is array (SWITCHTYPE) of SWITCHDATA;
```

Suppose the VCs have been generated and the supporting proof context made, as described in section 3.1:

SML

```
val switch_thms = all_cn_make_script_support "switch_cn";
```

Now, suppose we find that for the VC proofs we need to recast something of the form

```
not (sb s).STATE = Boolean false
```

more naturally as

```
(sb s).STATE = Boolean true
```

To do this, we first need to prove that *(sbs).STATE* is a member of *BOOLEAN*:

SML

```
set_goal([],⌜Z∀ sb:SWITCHBOARD; s: SWITCHTYPE•(sb s).STATE ∈ BOOLEAN⌝);
a(REPEAT strip_tac);
```

ProofPower output

```
(∗  2 ∗)  ⌜Zsb ∈ SWITCHBOARD⌝
(∗  1 ∗)  ⌜Zs ∈ SWITCHTYPE⌝

(∗ ?⊢ ∗)  ⌜Z(sb s).STATE ∈ BOOLEAN⌝
```

Now if we apply *cn_∈_type_tac* in the supporting proof context *switch_cn*, the goal will be achieved:

SML

```
a(PC_T1"switch_cn"cn_∈_type_tac[]);
val switch_∈_thm = pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

We can now prove the recasting result using *switch_∈_thm* together with *cn_boolean_clauses1* from the theory *cn*:

```
⊢ (∀ x : BOOLEAN • not x = Boolean (¬ x = Boolean true))
    ∧ (∀ x, y : BOOLEAN
      • x and y = Boolean (x = Boolean true ∧ y = Boolean true))
    ∧ (∀ x, y : BOOLEAN
      • x or y = Boolean (x = Boolean true ∨ y = Boolean true))
    ∧ (∀ x, y : BOOLEAN
      • x xor y = Boolean (¬ x = Boolean true ⇔ y = Boolean true))
```

SML

```
set_goal([],ⓩ∀ sb:SWITCHBOARD; s: SWITCHTYPE •
        not (sb s).STATE = Boolean false ⇔  (sb s).STATE = Boolean true⌝);
a(z_∀_tac THEN REPEAT ⇒_tac);
a(all_asm_fc_tac[switch_∈_thm]);
a(ALL_ASM_FC_T rewrite_tac[cn_boolean_clauses1]);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

3.3.3 Record a Member of Record Set

The form of this class of lemma is $(x \widehat{=} x1, y \widehat{=} y1, ...) \in set\ of\ recs$, which arises from assignment to an array of records in SPARK. Using the switchboard example of the previous section, we could be required to prove the following, say,:

SML

```
set_goal([],ⓩ(STATE ≙ Boolean true, NEXTSWITCH ≙ 2) ∈ SWITCHDATA⌝);
```

Again, applying *cn_∈_type_tac* in the proof context *switch_cn* will prove the goal:

SML

```
a(PC_T1"switch_cn"cn_∈_type_tac[]);
pop_thm();
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
```

As this is a one line proof, you may choose not to prove the result as a separate lemma. It rather depends on how many times the result is needed during the course of the VC proofs, and whether it matters to you that the application of this tactic in the supporting proof context may take some time (depending on the size of your specification).

3.3.4 Real Numbers

Ada fixed point and floating point types (collectively referred to as real types) are represented in Z using the real numbers of pure mathematics as implemented in the ProofPower-Z theory *z_reals*. In addition, the theory *cn* provides some operations on real numbers that are specific to the Compliance Notation (e.g., the operator *_e_* that is used in the translation of real literals).

Several proof contexts are available to assist in working with Ada real types. These are listed in the following table:

Name	Description
'z_reals	This is a component proof context that provides general purpose rules for the theory *z_reals*.
z_ℝ_lin_arith	This is a complete proof context whose main purpose is to provide a decision procedure for the linear fragment of the theory of reals.
'cn_reals	This is a component proof context that provides rules that eliminate the special operators on reals in the theory *cn* in favour of the underlying Z operators. Note that it does not expand the *_e_* operator except in the special case where the exponent is 0. A theorem *cn_e_thm* is provided for use as a rewrite rule to expand other uses of the *_e_* operator.

To see these proof contexts in use, consider the following example of a package specification and body. (For berevity, we have suppressed the *new_script* commmands).

Compliance Notation

```
package real_eg is
  type angle is delta 1.0 / 360.0 range 0.0 .. 1.0 − 1.0 / 360.0;
  procedure interpolate(a, b: in angle; c : out angle);
end real_eg;
```

Compliance Notation

```
package body real_eg is
  procedure interpolate(a, b: in angle; c : out angle)
  Δ C [A +_R REAL_EGoANGLEvDELTA <_R B, A <_R C <_R B]
  is
  begin
     if  a + angle′delta < b
     then c := a + angle′delta;
     end if;
  end interpolate;
end real_eg;
```

This produces the following VCs (in the theory *REAL_EGoINTERPOLATE′proc*):

```
vcREAL_EGoINTERPOLATE_1 ?⊢
   ∀ A, B : REAL_EGoANGLE
      | A +_R REAL_EGoANGLEvDELTA <_R B ∧ A +_R REAL_EGoANGLEvDELTA real_less B = TRUE
      • A <_R A +_R REAL_EGoANGLEvDELTA ∧ A +_R REAL_EGoANGLEvDELTA <_R B
```

```
│ vcREAL_EGoINTERPOLATE_2 ?⊢
│     ∀ A, B, C : REAL_EGoANGLE
│         | A +_R REAL_EGoANGLEvDELTA <_R B ∧ A +_R REAL_EGoANGLEvDELTA real_less B = FALSE
│         • A <_R C ∧ C <_R B
```

To prove them, we work in the proof context obtained by merging *'cn_reals* and *'z_reals* with the standard complete proof context for the *cn* theory, *cn1*:

SML

```
│open_theory "REAL_EGoINTERPOLATE'proc";
│push_merge_pcs["'cn_reals", "'z_reals", "cn1"];
```

We will go through the proof of the first VC. (The second VC is proved immediately with *cn_vc_simp_tac*).

SML

```
│set_goal([], get_conjecture"−""vcREAL_EGoINTERPOLATE_1");
│a(cn_vc_simp_tac[] THEN REPEAT strip_tac);
```

This results in the following:

```
│...
│
│(∗ ?⊢ ∗)  ⌜_Z 0.0 <_R REAL_EGoANGLEvDELTA⌝
```

Here the assumptions are not relevant, we are simply being asked to prove that *REAL_EGoANGLEvDELTA* is positive. We now appeal to the definition of *REAL_EGoANGLEvDELTA*:

SML

```
│a(rewrite_tac[z_get_spec⌜_Z REAL_EGoANGLEvDELTA⌝]);
```

This results in:

```
│...
│
│(∗ ?⊢ ∗)  ⌜_Z real 0 <_R 1.0 /_R 360.0⌝
```

Using the computational rules in the proof context *'z_ℝ_lin_arith* complete the proof:

SML

```
│a(PC_T1 "z_ℝ_lin_arith" prove_tac[]);
```

CALCULATOR EXAMPLE

This Compliance Notation example is concerned with the computational aspects of a simple calculator. Section 4.1 provides the literate scripts for the example and section 4.2 overviews the VCs generated then provides proofs for a subset of them. For reference purposes, a complete set of VC proofs is available in a separate document, [7]. A listing of the theories generated by the calculator example is available in appendix A. In addition, a listing of the generated SPARK program is available in appendix B.

4.1 The Literate Scripts

Though we hide the detail in this document, remember that there will be one literate script per compilation unit (such as a package specification or body).

4.1.1 Basic Definitions

In this section, we define types and constants which will be of use throughout the rest of the scripts.

The SPARK package *TC* below helps record the following facts:

- The calculator deals with signed integers expressed using up to six decimal digits.
- It has a numeric keypad and 6 operation buttons labelled +, −, ×, +/−, √, !, and =.

Compliance Notation

```
package TC is

    BASE : constant INTEGER := 10;
    PRECISION : constant INTEGER := 6;
    MAX_NUMBER : constant INTEGER := BASE ** PRECISION − 1;
    MIN_NUMBER : constant INTEGER := −MAX_NUMBER;

    subtype DIGIT is INTEGER range 0 .. BASE − 1;

    subtype NUMBER is INTEGER range MIN_NUMBER .. MAX_NUMBER;

    type OPERATION is
      (PLUS, MINUS, TIMES, CHANGE_SIGN, SQUARE_ROOT, FACTORIAL, EQUALS);

end TC;
```

4.1.2 The State

In this section, we define a package which contains all the state variables of the calculator.

The package *GV* below defines the global variables we will use to implement the following informal description of part of the calculator's behaviour:

- The calculator has two numeric state variables: the display, which contains the number currently being entered, and the accumulator, which contains the last result calculated.
- The user is considered to be in the process of entering a number whenever a digit button is pressed, and entry of a number is terminated by pressing one of the operation keys.
- When a binary operation key is pressed, the operation is remembered so that the appropriate value can be calculated when the second operand has been entered.

As it is a new package, and thus a new compilation unit, we will require a new script (though here, and elsewhere in this document, we hide the details of this action).

Compliance Notation

```
with TC;
package GV is

    DISPLAY, ACCUMULATOR : TC.NUMBER;

    LAST_OP : TC.OPERATION;

    IN_NUMBER : BOOLEAN;

end GV;
```

4.1.3 The Operations

In this section, we define a package which contains procedures corresponding to pressing the calculator buttons.

4.1.3.1 Package Specification

We now want to introduce a package *OPS* which implements the following informal description of how the calculator responds to button presses:

- The behaviour when a digit button is pressed depends on whether a number is currently being entered into the display. If a number is being entered, then the digit is taken as part of the number. If a number is not being entered (e.g., if an operation button has just been pressed), then the digit is taken as the most significant digit of a new number in the display.
- When a binary operation button is pressed, any outstanding calculation is carried out and the answer (which will be the first operand of the operation) is displayed; the calculator is then ready for the user to enter the other operand of the operation.
- When a unary operation button is pressed, the result of performing that operation to the displayed number is computed and displayed; the accumulator is unchanged, but entry of the displayed number is considered to be complete.
- When the button marked = is pressed, any outstanding calculation is carried out and the answer is displayed.

The package implementing this is defined in section 4.1.3.2 below after we have dealt with some preliminaries.

4.1.3.1.1 Z Preliminaries To abbreviate the description of the package, we do some work in Z first, corresponding to the various sorts of button press.

Note that the use of $\mathbb{Z}$ rather than *TCoNUMBER* reflects the fact that we are ignoring questions of arithmetic overflow here. If we used the Z set which accurately represents the SPARK type, then we would have to add in pre-conditions saying that the operations do not overflow. The following schema defines what happens when a digit button is pressed.

z

DO_DIGIT

$GVoDISPLAY_0,\ GVoDISPLAY : \mathbb{Z};$

$GVoIN\_NUMBER_0,\ GVoIN\_NUMBER : BOOLEAN;$

$D : TCoDIGIT$

$$
\begin{array}{l}
GVoIN\_NUMBER_0 = TRUE \Rightarrow GVoDISPLAY = GVoDISPLAY_0 * TCoBASE + D;\\
GVoIN\_NUMBER_0 = FALSE \Rightarrow \ GVoDISPLAY = D;\\
GVoIN\_NUMBER = TRUE
\end{array}
$$

We now define sets *UNARY* and *BINARY* which partition the two sorts of operation key. Note that = can be considered as a sort of binary operation (which given operands x and y returns x).

z

$$UNARY \mathrel{\widehat{=}} \{TCoCHANGE\_SIGN,\ TCoFACTORIAL,\ TCoSQUARE\_ROOT\}$$

z

$$BINARY \mathrel{\widehat{=}} TCoOPERATION \setminus UNARY$$

We need to define a function for computing factorials in order to define the response to the factorial operation button.

z

$$\begin{array}{|l}
\quad fact : \mathbb{N} \rightarrow \mathbb{N} \\
\hline
\quad fact\ 0 = 1\ ; \\
\quad \forall m{:}\mathbb{N} \bullet\ fact(m{+}1) = (m + 1) * fact\ m
\end{array}$$

Unary operations behave as specified by the following schema. In which we do specify explicitly that the accumulator and last operation values are unchanged for clarity and for simplicity later on (when we group the unary and binary operations together).

z

$$\begin{array}{|l}
\multicolumn{1}{l}{DO\_UNARY\_OPERATION} \\
\hline
GVoACCUMULATOR_0,\ GVoACCUMULATOR : \mathbb{Z}; \\
GVoDISPLAY_0,\ GVoDISPLAY : \mathbb{Z}; \\
GVoLAST\_OP_0,\ GVoLAST\_OP : \mathbb{Z}; \\
GVoIN\_NUMBER : BOOLEAN; \\
O : UNARY \\
\hline
\quad GVoIN\_NUMBER = FALSE; \\
\quad GVoACCUMULATOR = GVoACCUMULATOR_0; \\
\quad GVoLAST\_OP = GVoLAST\_OP_0; \\
\quad O = TCoCHANGE\_SIGN \Rightarrow GVoDISPLAY = {\sim}GVoDISPLAY_0; \\
\quad O = TCoFACTORIAL \land GVoDISPLAY_0 \geq 0 \Rightarrow \\
\qquad\qquad GVoDISPLAY = fact\ GVoDISPLAY_0; \\
\quad O = TCoSQUARE\_ROOT\ \land GVoDISPLAY_0 \geq 0 \Rightarrow \\
\qquad\qquad GVoDISPLAY ** 2 \leq GVoDISPLAY_0 < (GVoDISPLAY + 1) ** 2 \\
\hline
\end{array}$$

The binary operations are specified by the following schema.

z

$DO\_BINARY\_OPERATION$

$$
\begin{array}{l}
GVoACCUMULATOR_0,\ GVoACCUMULATOR : \mathbb{Z};\\
GVoDISPLAY_0,\ GVoDISPLAY : \mathbb{Z};\\
GVoLAST\_OP_0,\ GVoLAST\_OP : \mathbb{Z};\\
GVoIN\_NUMBER : BOOLEAN;\\
O : BINARY
\end{array}
$$

$$
\begin{array}{l}
\quad GVoIN\_NUMBER = FALSE;\\
\quad GVoDISPLAY = GVoACCUMULATOR;\\
\quad GVoLAST\_OP = O;\\
\quad GVoLAST\_OP_0 = TCoEQUALS \Rightarrow\\
\qquad\qquad GVoACCUMULATOR = GVoDISPLAY_0;\\
\quad GVoLAST\_OP_0 = TCoPLUS \Rightarrow\\
\qquad\qquad GVoACCUMULATOR = GVoACCUMULATOR_0 + GVoDISPLAY_0;\\
\quad GVoLAST\_OP_0 = TCoMINUS \Rightarrow\\
\qquad\qquad GVoACCUMULATOR = GVoACCUMULATOR_0 - GVoDISPLAY_0;\\
\quad GVoLAST\_OP_0 = TCoTIMES \Rightarrow\\
\qquad\qquad GVoACCUMULATOR = GVoACCUMULATOR_0 * GVoDISPLAY_0
\end{array}
$$

The disjunction of the schemas for the unary and binary operations is then what is needed to define the response to pressing an arbitrary button press.

z

$$DO\_OPERATION \mathrel{\widehat=} DO\_UNARY\_OPERATION \lor DO\_BINARY\_OPERATION$$

4.1.3.2 The SPARK Package

We now use the schemas of the previous section to define the package *OPS*.

Compliance Notation

```
with TC, GV;
package OPS is
procedure DIGIT_BUTTON (D : in TC.DIGIT)
        Δ GVoDISPLAY, GVoIN_NUMBER [ DO_DIGIT ] ;
procedure OPERATION_BUTTON (O : in TC.OPERATION)
        Δ GVoACCUMULATOR, GVoDISPLAY,
                GVoIN_NUMBER, GVoLAST_OP [ DO_OPERATION ] ;
end OPS;
```

4.1.3.3 Package Implementation

4.1.3.3.1 Package Body The following specification of the package body is derived from the package specification in the obvious way. We leave a k-slot for any extra declarations we may need.

Compliance Notation

```
$references TC, GV;
package body OPS is
procedure DIGIT_BUTTON (D : in TC.DIGIT)
        Δ GVoDISPLAY, GVoIN_NUMBER [ DO_DIGIT ]
    is begin
        Δ GVoDISPLAY, GVoIN_NUMBER [ DO_DIGIT ]                        (3001)
    end DIGIT_BUTTON;
procedure OPERATION_BUTTON (O : in TC.OPERATION)
        Δ GVoACCUMULATOR, GVoDISPLAY,
                GVoIN_NUMBER, GVoLAST_OP [ DO_OPERATION ]
    is
    ⟨ Extra Declarations ⟩                               ( 500 )
    begin
        Δ GVoACCUMULATOR, GVoDISPLAY,
                GVoIN_NUMBER, GVoLAST_OP [ DO_OPERATION ]              (3002)
    end OPERATION_BUTTON;
end OPS;
```

4.1.3.3.2 Supporting Functions We choose to separate out the computation of factorials and square roots into separate functions which replace the k-slot labelled 500. In both cases, we prepare for the necessary algorithms. Our approach for both functions is to introduce and initialise appropriately a variable called *RESULT*, demand that this be set to the desired function return value and return that value.

SML

```
open_scope"OPS.OPERATION_BUTTON";
```

Compliance Notation

```
(500) ≡
  function FACT (M : NATURAL) return NATURAL
      Ξ [ FACT(M) = fact(M) ]
  is
      RESULT : NATURAL;
  begin
      RESULT := 1;
      Δ RESULT [M ≥ 0 ∧ RESULT = 1, RESULT = fact M ]        (1001)
      return RESULT;
  end FACT;

  function SQRT (M : NATURAL) return NATURAL
      Ξ [SQRT(M) ** 2 ≤ M < (SQRT(M) + 1) ** 2]
  is
      RESULT : NATURAL;
     ⟨ other local vars ⟩              (2)
  begin
     RESULT := 0;
     Δ RESULT [RESULT = 0, RESULT ** 2 ≤ M < (RESULT + 1) ** 2](2001)
    return RESULT;
  end SQRT;
```

4.1.3.3.3 Algorithm for Factorial Factorial is implemented by a for-loop with loop-counter *J* and an invariant requiring that as *J* steps from *2* up to *M*, *RESULT* is kept equal to the factorial of *J*:

SML

```
open_scope "OPS.OPERATION_BUTTON.FACT";
```

Compliance Notation

```
 (1001) ⊑
  for J in INTEGER range 2 .. M
  loop
      Δ RESULT [J ≥ 1 ∧ RESULT = fact (J−1), RESULT = fact J] (1002)
  end loop;
```

Now we can complete the implementation of the factorial function by providing the loop body:

Compliance Notation

```
 (1002) ⊑
     RESULT := J * RESULT;
```

4.1.3.3.4 Algorithm for Square Root For square root, we need two extra variables to implement a binary search for the square root.

SML

```
open_scope"OPS.OPERATION_BUTTON.SQRT";
```

Compliance Notation

```
(2) ≡
    MID, HI : INTEGER;
```

The following just says that we propose to achieve the desired effect on *RESULT* using *MID* and *HI* as well.

Compliance Notation

```
(2001) ⊑
        Δ RESULT, MID, HI
            [RESULT = 0, RESULT ** 2 ≤ M < (RESULT + 1) ** 2] (2002)
```

Now we give the initialisation for *HI* and describe the loop which will find the square root:

Compliance Notation

```
(2002) ⊑
        HI := M + 1;
        $till ⟦RESULT ** 2 ≤ M < (RESULT + 1) ** 2⟧
        loop
            Δ RESULT, MID, HI
                [RESULT ** 2 ≤ M < HI ** 2, RESULT ** 2 ≤ M < HI ** 2] (2003)
        end loop;
```

Now we implement the exit for the loop and specify the next step:

Compliance Notation

```
(2003) ⊑
        exit when RESULT + 1 = HI;
        Δ RESULT, MID, HI
            [RESULT ** 2 ≤ M < HI ** 2, RESULT ** 2 ≤ M < HI ** 2] (2004)
```

Now we can fill in the last part of the loop:

Compliance Notation

```
(2004)   ⊑
    MID := (RESULT + HI + 1) / 2;
    if      MID ** 2 > M
    then     HI := MID;
    else    RESULT := MID;
    end if;
```

4.1.3.3.5 Digit Button Algorithm We now continue with the body of the digit button procedure. An if-statement handling the two cases for updating the display, followed by an assignment to the flag should meet the bill here.

SML

```
open_scope"OPS.DIGIT_BUTTON";
```

Compliance Notation

```
(3001) ⊑
   if     GV.IN_NUMBER
   then   GV.DISPLAY := GV.DISPLAY * TC.BASE + D;
   else   GV.DISPLAY := D;
   end if;
   GV.IN_NUMBER := true;
```

4.1.3.3.6 Operation Button Algorithm We now complete the implementation and verification of the package *OPS* by giving the body of the procedure for handling the operation buttons.

SML

```
open_scope "OPS.OPERATION_BUTTON";
```

Compliance Notation

```
(3002) ⊑
   if      O = TC.CHANGE_SIGN
   then    GV.DISPLAY := −GV.DISPLAY;
   elsif   O = TC.FACTORIAL
   then    GV.DISPLAY := FACT(GV.DISPLAY);
   elsif   O = TC.SQUARE_ROOT
   then    GV.DISPLAY := SQRT(GV.DISPLAY);
   else    if      GV.LAST_OP = TC.EQUALS
           then    GV.ACCUMULATOR := GV.DISPLAY;
           elsif   GV.LAST_OP = TC.PLUS
           then    GV.ACCUMULATOR := GV.ACCUMULATOR + GV.DISPLAY;
           elsif   GV.LAST_OP = TC.MINUS
           then    GV.ACCUMULATOR := GV.ACCUMULATOR − GV.DISPLAY;
           elsif   GV.LAST_OP = TC.TIMES
           then    GV.ACCUMULATOR := GV.ACCUMULATOR * GV.DISPLAY;
           end if;
           GV.DISPLAY := GV.ACCUMULATOR;
           GV.LAST_OP := O;
   end if;
   GV.IN_NUMBER := false;
```

4.2 Proving the VCs

The Compliance Tool has generated 37 VCs. The following list outlines where each group of VCs has come from, and proofs of a representative selection from these groups are given below.

8	from the introduction of the package body:	*vcOPS_1*	
		vcOPS_2	
		vcOPS_3	
		vcOPS_4	
		vcOPSoDIGIT_BUTTON_1	
		vcOPSoDIGIT_BUTTON_2	
		vcOPSoOPERATION_BUTTON_1	
		vcOPSoOPERATION_BUTTON_2	
4	from the introduction of supporting functions:	*vcOPSoOPERATION_BUTTONoFACT_1*	
		vcOPSoOPERATION_BUTTONoFACT_2	
		vcOPSoOPERATION_BUTTONoSQRT_1	
		vcOPSoOPERATION_BUTTONoSQRT_2	
5	from the refinement steps in the body of *FACT*:	*vc1001_1*	
		vc1001_2	
		vc1001_3	
		vc1001_4	
		vc1002_1	
10	from the refinement steps in the body of *SQRT*:	*vc2001_1*	
		vc2001_2	
		vc2002_1	
		vc2002_2	
		vc2002_3	
		vc2003_1	
		vc2003_2	
		vc2003_3	
		vc2004_1	
		vc2004_2	
2	from if-statement in the digit button procedure:	*vc3001_1*	
		vc3001_2	
8	from if-statement in the operations button procedure:	*vc3002_1*	1-3
		vc3002_2	
		vc3002_3	
		vc3002_4	4-8
		vc3002_5	
		vc3002_6	
		vc3002_7	
		vc3002_8	

4.2.1 Preliminaries

The first thing to do is to generate the supporting theorems and supporting proof context for the calculator example, as described in section 3.1. We do this in the theory *calc_prelims*.

SML

```
open_theory "calc_prelims";
val calc_thms = all_cn_make_script_support "calc_prelims";
```

We will conduct the VC proofs in the proof context *cn1*; any subsidiary general purpose results that we may need will be proved in the proof context *z_library1*.

4.2.2 Package Body VCs

Eight VCs are produced from the introduction of the package body, see section 4.1.3.3.1. These are trivial because the package body is derived directly from the package specification. For example:

$$\begin{array}{l}\mathit{vcOPS\_1}\\ \qquad \mathit{true} \Rightarrow \mathit{true}\end{array}$$

$$\begin{array}{l}\mathit{vcOPS\_2}\\ \qquad \forall\ \mathit{GVoDISPLAY},\ \mathit{GVoDISPLAY}_0 : \mathit{TCoNUMBER};\\ \qquad\quad \mathit{GVoIN\_NUMBER},\ \mathit{GVoIN\_NUMBER}_0 : \mathit{BOOLEAN};\\ \qquad\quad D : \mathit{TCoDIGIT} \mid \mathit{true} \land \mathit{DO\_DIGIT} \bullet \mathit{DO\_DIGIT}\end{array}$$

cn_vc_simp_tac will solve all of these VCs. For example:

SML

```
open_scope "OPS";
set_pc"cn1";
set_goal([],get_conjecture"-""vcOPS_2");
a(cn_vc_simp_tac[]);
save_pop_thm"vcOPS_2";
```

4.2.3 Function Definition VCs

These four VCs result from the requirement to show that the function bodies of *FACT* and *SQRT* achieve the statement of their specifications, see section 4.1.3.3.2. For example:

$$\begin{array}{l}\mathit{vc500\_1}\\ \forall\ M : \mathit{NATURAL} \bullet M \geq 0 \land 1 = 1\end{array}$$

$$\begin{array}{l}\mathit{vc500\_2}\\ \mathit{FACT} : \mathit{NATURAL} \rightarrow \mathit{NATURAL};\ \mathit{RESULT} : \mathit{NATURAL};\ M : \mathit{NATURAL}\\ \qquad\quad \mid \mathit{true} \land \mathit{RESULT} = \mathit{fact}\ M \land \mathit{FACT}\ M = \mathit{RESULT}\\ \qquad\quad \bullet \mathit{FACT}\ M = \mathit{fact}\ M\end{array}$$

The proof of *vc500_1* requires the following general purpose lemma about SPARK natural numbers:

SML

```
open_theory "calc_prelims";
push_pc"z_library1";
set_goal([], ⌜Z∀m : NATURAL• m ≥ 0⌝);
a(rewrite_tac[z_get_spec⌜ZNATURAL⌝] THEN REPEAT strip_tac);
val natural_thm = save_pop_thm"natural_thm";
```

Now *cn_vc_simp_tac* followed by forward chaining with *natural_thm* will solve *vcOPSoOPERATION_BUTTONo*

SML

```
open_scope "OPS.OPERATION_BUTTON.FACT";
set_goal([],get_conjecture"−""vcOPSoOPERATION_BUTTONoFACT_1");
a(cn_vc_simp_tac[]);
a(REPEAT strip_tac THEN all_fc_tac[natural_thm]);
save_pop_thm"vcOPSoOPERATION_BUTTONoFACT_1";
```

After simplifying and stripping, *vcOPSoOPERATION_BUTTONoFACT_2* is an example of predicate calculus with equality, see section 2.5, and is straightforward to prove using variable elimination, or just rewriting with the assumptions:

SML

```
set_goal([],get_conjecture"−""vcOPSoOPERATION_BUTTONoFACT_2");
a(cn_vc_simp_tac[]);
a(REPEAT strip_tac);
a(all_var_elim_asm_tac1);
save_pop_thm"vcOPSoOPERATION_BUTTONoFACT_2";
```

4.2.4 *FACT* Refinement Steps VCs

The implementation of factorial produces five VCs, see section 4.1.3.3.3. For example:

```
vc1001_1
∀ RESULT : NATURAL; M : NATURAL
  | (M ≥ 0 ∧ RESULT = 1) ∧ 2 ≤ M
  • 2 ≥ 1 ∧ RESULT = fact (2 − 1)
```

```
vc1001_2
∀ RESULT : NATURAL; M : NATURAL
  | (M ≥ 0 ∧ RESULT = 1) ∧ 2 > M
  • RESULT = fact M
```

First, a general purpose lemma about the first two values of factorial (needed because our algorithm avoids the unnecessary pass through the loop with $J = 1$). The theorem *cn_fact_thm* is one of the supporting theorems generated in section 4.2.1.

SML

```
open_theory"calc_prelims";
push_pc"z_library1";
set_goal([], ⓩfact 0 = 1 ∧ fact 1 = 1⌝);
a(rewrite_tac[cn_fact_thm, rewrite_rule[cn_fact_thm]
        ((z_∀_elimⓩ0⌝ o ∧_right_elim) cn_fact_thm)]);
val fact_thm  = save_pop_thm"fact_thm";
pop_pc();
```

To prove *vc1001_1*, simplify the VC, then strip and rewrite with *fact_thm* and the assumptions:

SML

```
open_scope "OPS.OPERATION_BUTTON.FACT";
set_pc"cn1";
set_goal([],get_conjecture"−""vc1001_1");
a(cn_vc_simp_tac[]);
a(REPEAT strip_tac THEN asm_rewrite_tac[fact_thm]);
save_pop_thm"vc1001_1";
```

For *vc1001_2*, first simplify and then strip:

SML

```
set_goal([],get_conjecture"−""vc1001_2");
a(cn_vc_simp_tac[]);
a(REPEAT strip_tac);
```

ProofPower output

```
(∗ ∗∗∗ Goal "" ∗∗∗ ∗)

(∗  5 ∗)  ⓩRESULT ∈ NATURAL⌝
(∗  4 ∗)  ⓩM ∈ NATURAL⌝
(∗  3 ∗)  ⓩ0 ≤ M⌝
(∗  2 ∗)  ⓩRESULT = 1⌝
(∗  1 ∗)  ⓩ¬ 2 ≤ M⌝

(∗ ?⊢ ∗)  ⓩRESULT = fact M⌝

```

Assumption 1 gives you $M = 0$ or $M = 1$, you will need the linear arithmetic proof context for this, then rewriting with *fact_thm* completes the proof:

SML

```
a(lemma_tacⓩM = 0 ∨ M = 1⌝
        THEN1 PC_T1"z_lin_arith"asm_prove_tac[]
        THEN asm_rewrite_tac[fact_thm]);
save_pop_thm"vc1001_2";
```

4.2.5 *SQRT* Refinement Steps VCs

The implementation of square root produces ten VCs, see section 4.1.3.3.4. Eight of them are trivial and are proved by *cn_vc_simp_tac*. For example:

```
vc2002_2
∀ HI : INTEGER; RESULT, RESULT₀ : NATURAL; M : NATURAL
  | RESULT₀ = 0 ∧ RESULT ** 2 ≤ M ∧ M < HI ** 2
  • RESULT ** 2 ≤ M ∧ M < HI ** 2
```

SML

```
open_scope "OPS.OPERATION_BUTTON.SQRT";
set_pc "cn1";
set_goal([],get_conjecture"−""vc2002_2");
a(cn_vc_simp_tac[]);
save_pop_thm"vc2002_2";
```

vc2003_1 also requires stripping and variable elimination to achieve a proof. Extra work is needed in the proof of *vc2002_1*:

```
vc2002_1
∀ RESULT : NATURAL; M : NATURAL
             | RESULT = 0
             • RESULT ** 2 ≤ M ∧ M < (M + 1) ** 2
```

This depends on some facts about the exponentiation operator, which you would probably provide as separate lemmas:

SML

```
push_pc"z_library1";
set_goal([], ⓩ∀x: ℤ•  x ** 1 = x⌝);
a(REPEAT strip_tac);
a(rewrite_tac[rewrite_rule[](
    z_∀_elimⓩ(x ≜ x, y ≜ 0)⌝ (∧_right_elim(z_get_specⓩ(_**_)⌝)))]);
val star_star_1_thm = save_pop_thm"star_star_1_thm";
```

SML

```
set_goal([], ⓩ∀x: ℤ•  x ** 2 = x * x⌝);
a(REPEAT strip_tac);
a(rewrite_tac[star_star_1_thm, rewrite_rule[](
    z_∀_elimⓩ(x ≜ x, y ≜ 1)⌝ (∧_right_elim(z_get_specⓩ(_**_)⌝)))]);
val star_star_2_thm = save_pop_thm"star_star_2_thm";
pop_pc();
```

To prove *vc2002_1*, first simplify the goal, strip the assumptions and eliminate *RESULT*:

SML

```
set_goal([], get_conjecture "−" "vc2002_1");
a(cn_vc_simp_tac[]);
a(REPEAT ⇒_tac THEN all_var_elim_asm_tac1);
```

ProofPower output

```
...
(* 2 *) ⓩM ∈ NATURAL⌝
(* 1 *) ⓩ0 ∈ NATURAL⌝

(* ?⊢ *) ⓩ0 ** 2 ≤ M ∧ ¬ (M + 1) ** 2 ≤ M⌝
```

Throw away the irrelevant assumption then forward chain with *natural_thm*:

SML

```
a(POP_ASM_T discard_tac THEN all_fc_tac[natural_thm]);
```

ProofPower output

```
...
(* 2 *) ⓩM ∈ NATURAL⌝
(* 1 *) ⓩ0 ≤ M⌝

(* ?⊢ *) ⓩ0 ** 2 ≤ M ∧ ¬ (M + 1) ** 2 ≤ M⌝
```

We no longer need assumption 2. Then rewriting with the assumptions and *star_star_2_thm* will deal with the first conjunct in the goal and expand the second:

SML

```
a(DROP_NTH_ASM_T 2 discard_tac);
a(asm_rewrite_tac[star_star_2_thm]);
```

The proof is progressed by induction on M:

SML

```
a(z_≤_induction_tacⓩM⌝);
```

ProofPower output

```
Tactic produced 2 subgoals:

(* *** Goal "2" *** *)

(* 2 *) ⓩ0 ≤ i⌝
(* 1 *) ⓩ¬ (i + 1) * (i + 1) ≤ i⌝

(* ?⊢ *) ⓩ¬ ((i + 1) + 1) * ((i + 1) + 1) ≤ i + 1⌝


(* *** Goal "1" *** *)

(* ?⊢ *) ⓩ¬ (0 + 1) * (0 + 1) ≤ 0⌝
```

The first subgoal is proved by rewriting in the current proof context. The second is proved automatically in the linear arithmetic proof context:

SML

```
(* *** Goal "1" *** *)
a(rewrite_tac[]);
(* *** Goal "2" *** *)
a(PC_T1 "z_lin_arith" asm_prove_tac[]);
save_pop_thm "vc2002_1";
```

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
...
```

4.2.6 Digit Button VCs

Two VCs result from the if_statement in the digit button procedure, see section 4.1.3.3.5. For example:

```
∀ GVoDISPLAY : TCoNUMBER; GVoIN_NUMBER : BOOLEAN; D : TCoDIGIT
            | true ∧ GVoIN_NUMBER = TRUE
            • (D ≙ D, GVoDISPLAY ≙ GVoDISPLAY ∗ TCoBASE + D,
                  GVoDISPLAY₀ ≙ GVoDISPLAY, GVoIN_NUMBER ≙ TRUE,
                  GVoIN_NUMBER₀ ≙ GVoIN_NUMBER)
              ∈ DO_DIGIT
```

Both are proven by simplifying, stripping and then rewriting with the definition of *DO_DIGIT*:

SML

```
open_scope "OPS.DIGIT_BUTTON";
set_pc "cn1";
set_goal([],get_conjecture"−""vc3001_1");
a(cn_vc_simp_tac[]);
a(REPEAT strip_tac THEN asm_rewrite_tac[cn_DO_DIGIT_thm]);
save_pop_thm"vc3001_1";
```

4.2.7 Operations Button VCs

Eight VCs result from the branches in the operations button procedure, 3 unary and 5 binary, see section 4.1.3.3.6. For example, the unary operation produces:

```
vc3002_1
∀ GVoDISPLAY, GVoACCUMULATOR : TCoNUMBER;
    GVoLAST_OP : TCoOPERATION;
    O : TCoOPERATION
  | true ∧ O eq TCoCHANGE_SIGN = TRUE
  • (GVoACCUMULATOR ≙ GVoACCUMULATOR,
         GVoACCUMULATOR0 ≙ GVoACCUMULATOR,
        GVoDISPLAY ≙ ∼ GVoDISPLAY, GVoDISPLAY0 ≙ GVoDISPLAY,
        GVoIN_NUMBER ≙ FALSE, GVoLAST_OP ≙ GVoLAST_OP,
        GVoLAST_OP0 ≙ GVoLAST_OP, O ≙ O)
    ∈ DO_OPERATION
```

To prove this VC, we do need to unwind the basic definitions, so we will use the list of supporting theorems *calc_thms* that we generated before starting out on the proofs, section 4.2.1. (We could equally well have chosen to rewrite in the supporting proof context *calc_cn* with no additional theorems.) This proof gives an example of where *REPEAT strip_tac* would be too brutal because it would generate many subgoals.

SML

```
open_scope "OPS.OPERATION_BUTTON";
set_pc "cn1";
set_goal([], get_conjecture"-""vc3002_1");
a(cn_vc_simp_tac calc_thms);
a ⇒_tac;
a(asm_rewrite_tac[]);
save_pop_thm"vc3002_1";
```

The remaining two unary operations VCs require us to make the (reasonable) assumption that a non-negative number of the precision handled by the calculator will fit in a SPARK *NATURAL*. This amounts to the following axiom:

Z

```
  TCoMAX_NUMBER ≤ INTEGERvLAST
```

SML

```
val number_ax = get_axiom"-""Constraint 1";
```

To prove *vc3002_2*, apply the same tactics as for *vc3002_1*:

SML

```
set_goal([], get_conjecture"-""vc3002_2");
a(cn_vc_simp_tac calc_thms);
a ⇒_tac;
a(asm_rewrite_tac[]);
```

ProofPower output

```
...
(∗ ∗∗∗ Goal "" ∗∗∗ ∗)

(∗ 10 ∗)  ⓩ∼ 999999 ≤ GVoACCUMULATOR⌝
(∗  9 ∗)  ⓩGVoACCUMULATOR ≤ 999999⌝
(∗  8 ∗)  ⓩ∼ 999999 ≤ GVoDISPLAY⌝
(∗  7 ∗)  ⓩGVoDISPLAY ≤ 999999⌝
(∗  6 ∗)  ⓩ0 ≤ GVoLAST_OP⌝
(∗  5 ∗)  ⓩGVoLAST_OP ≤ 6⌝
(∗  4 ∗)  ⓩ0 ≤ O⌝
(∗  3 ∗)  ⓩO ≤ 6⌝
(∗  2 ∗)  ⓩ¬ O = 3⌝
(∗  1 ∗)  ⓩO = 5⌝

(∗ ?⊢ ∗)  ⓩ0 ≤ GVoDISPLAY ⇒ FACT GVoDISPLAY = fact GVoDISPLAY⌝

```

Then remove the redundant assumptions and strip the goal:

SML

```
a(LIST_DROP_NTH_ASM_T[1,2,3,4,5,6,8,9,10] (MAP_EVERY discard_tac));
a strip_tac;
```

ProofPower output

```
...
(∗ ∗∗∗ Goal "" ∗∗∗ ∗)

(∗  2 ∗)  ⓩGVoDISPLAY ≤ 999999⌝
(∗  1 ∗)  ⓩ0 ≤ GVoDISPLAY⌝

(∗ ?⊢ ∗)  ⓩFACT GVoDISPLAY = fact GVoDISPLAY⌝
```

All we need now is to add to the assumptions the fact that *GVoDISPLAY* ∈ *NATURAL*, then forward chain with the definition of *FACT*. Notice that during the proof of the lemma we switch to the proof context *calc_cn* which contains the supporting theorems generated in section 4.2.1.

SML

```
a(lemma_tac ⓩGVoDISPLAY ∈ NATURAL⌝);
(∗ ∗∗∗ Goal "1" ∗∗∗ ∗)
a(ante_tac number_ax);
a(PC_T1"calc_prelims"asm_rewrite_tac[z_get_specⓩNATURAL⌝]);
a(DROP_NTH_ASM_T 2 ante_tac THEN PC_T1 "z_lin_arith" prove_tac[]);
(∗ ∗∗∗ Goal "2" ∗∗∗ ∗)
a(ALL_FC_T rewrite_tac[z_get_specⓩFACT⌝]);
save_pop_thm"vc3002_2";
```

The proof of *vc3002_3* will be almost identical, except that the last step will be to forward chain with the definition of *SQRT*.

Because the binary operations only involve built-in arithmetic operators, they are a little easier to verify than the unary ones. For example the VC:

$$\begin{array}{l}
vc3002\_4\\
\forall\ GVoDISPLAY,\ GVoACCUMULATOR : TCoNUMBER;\\
\quad GVoLAST\_OP : TCoOPERATION;\\
\quad O : TCoOPERATION\\
\ \ \mid true\\
\quad \land\ O\ eq\ TCoCHANGE\_SIGN = FALSE\\
\quad \land\ O\ eq\ TCoFACTORIAL = FALSE\\
\quad \land\ O\ eq\ TCoSQUARE\_ROOT = FALSE\\
\quad \land\ GVoLAST\_OP\ eq\ TCoEQUALS = TRUE\\
\ \ \bullet\ (GVoACCUMULATOR \mathrel{\widehat=} GVoDISPLAY,\ GVoACCUMULATOR_0 \mathrel{\widehat=} GVoACCUMULATOR,\\
\qquad GVoDISPLAY \mathrel{\widehat=} GVoDISPLAY,\ GVoDISPLAY_0 \mathrel{\widehat=} GVoDISPLAY,\\
\qquad GVoIN\_NUMBER \mathrel{\widehat=} FALSE,\ GVoLAST\_OP \mathrel{\widehat=} O,\ GVoLAST\_OP_0 \mathrel{\widehat=} GVoLAST\_OP,\\
\qquad O \mathrel{\widehat=} O)\\
\quad \in DO\_OPERATION
\end{array}$$

will be proved by the following:

SML

```
set_goal([], get_conjecture"−""vc3002_4");
a(PC_T1"calc_prelims"cn_vc_simp_tac[]);
a(⇒_tac THEN asm_rewrite_tac[]);
save_pop_thm"vc3002_4";
```

The same proof will suffice for the remaining four binary operation VCs.

EXAMPLE THEORY LISTINGS

The examples in this document give rise to 13 theories as follows:

A.1 THE Z THEORY EX'proc

A.1.1 Parents

cn

A.1.2 Global Variables

T	$\mathbb{P}\ T$
CONSTSPEC	$\mathbb{P}\ T \leftrightarrow \mathbb{Z}$
SWITCHTYPE	$\mathbb{P}\ \mathbb{Z}$
SWITCHTYPEvFIRST	$\mathbb{Z}$
SWITCHTYPEvLAST	$\mathbb{Z}$
SWITCHTYPEvSUCC	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
SWITCHTYPEvPRED	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
SWITCHTYPEvPOS	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
SWITCHTYPEvVAL	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
SWITCHDATA	$\mathbb{P}\ [NEXTSWITCH, STATE : \mathbb{Z}]$
SWITCHDATAuSTATE[**g1**, **g2**]	$[NEXTSWITCH : g2; STATE : g1] \times g1 \leftrightarrow [NEXTSWITCH : g2; STATE : g1]$
SWITCHDATAuNEXTSWITCH[**g1**, **g2**]	$[NEXTSWITCH : g2; STATE : g1] \times g2 \leftrightarrow [NEXTSWITCH : g2; STATE : g1]$
SWITCHBOARD	$\mathbb{P}\ (\mathbb{Z} \leftrightarrow [NEXTSWITCH, STATE : \mathbb{Z}])$
SWITCHBOARDvFIRST	$\mathbb{Z}$
SWITCHBOARDvLAST	$\mathbb{Z}$
SWITCHBOARDvLENGTH	$\mathbb{Z}$
SWITCHBOARDvRANGE	$\mathbb{P}\ \mathbb{Z}$

SWITCHBOARDvFIRSTv1
$\mathbb{Z}$
SWITCHBOARDvLASTv1
$\mathbb{Z}$
SWITCHBOARDvLENGTHv1
$\mathbb{Z}$
SWITCHBOARDvRANGEv1
$\mathbb{P}\ \mathbb{Z}$

A.1.3 Axioms

CONSTSPEC $\vdash CONSTSPEC \in \mathbb{P}_1\ T \twoheadrightarrow \mathbb{N} \land true$
SWITCHDATAuSTATE
SWITCHDATAuNEXTSWITCH

$$\begin{array}{l}\vdash [g1,\\ \quad g2]((SWITCHDATAuSTATE[g1,\ g2]\\ \qquad \in [STATE : g1;\ NEXTSWITCH : g2] \times g1\\ \qquad\quad \rightarrow [STATE : g1;\ NEXTSWITCH : g2]\\ \quad\ \land\ SWITCHDATAuNEXTSWITCH[g1,\ g2]\\ \qquad \in [STATE : g1;\ NEXTSWITCH : g2] \times g2\\ \qquad\quad \rightarrow [STATE : g1;\ NEXTSWITCH : g2])\\ \quad\ \land\ (\forall\ r : [STATE : g1;\ NEXTSWITCH : g2];\\ \qquad x1 : g1;\\ \qquad x2 : g2\\ \qquad \bullet\ SWITCHDATAuSTATE[g1,\ g2]\ (r,\ x1)\\ \qquad\quad = (NEXTSWITCH \mathrel{\widehat{=}} r.NEXTSWITCH,\ STATE \mathrel{\widehat{=}} x1)\\ \qquad\ \land\ SWITCHDATAuNEXTSWITCH[g1,\ g2]\ (r,\ x2)\\ \qquad\quad = (NEXTSWITCH \mathrel{\widehat{=}} x2,\ STATE \mathrel{\widehat{=}} r.STATE)))\end{array}$$

A.1.4 Definitions

T $\vdash T = \mathbb{U}$
SWITCHTYPE $\vdash SWITCHTYPE = 1\ ..\ 3$
SWITCHTYPEvFIRST
$\vdash SWITCHTYPEvFIRST = 1$
SWITCHTYPEvLAST
$\vdash SWITCHTYPEvLAST = 3$
SWITCHTYPEvSUCC
$\vdash SWITCHTYPEvSUCC = INTEGERvSUCC$
SWITCHTYPEvPRED
$\vdash SWITCHTYPEvPRED = INTEGERvPRED$
SWITCHTYPEvPOS
$\vdash SWITCHTYPEvPOS = INTEGERvPOS$
SWITCHTYPEvVAL
$\vdash SWITCHTYPEvVAL = INTEGERvVAL$
SWITCHDATA $\vdash SWITCHDATA$
$= [STATE : BOOLEAN;\ NEXTSWITCH : SWITCHTYPE]$
SWITCHBOARD $\vdash SWITCHBOARD = SWITCHTYPE \rightarrow SWITCHDATA$
SWITCHBOARDvFIRST
$\vdash SWITCHBOARDvFIRST = SWITCHTYPEvFIRST$
SWITCHBOARDvLAST

$\vdash SWITCHBOARDvLAST = SWITCHTYPEvLAST$

SWITCHBOARDvLENGTH

$\vdash SWITCHBOARDvLENGTH = \# SWITCHTYPE$

SWITCHBOARDvRANGE

$\vdash SWITCHBOARDvRANGE = SWITCHTYPE$

SWITCHBOARDvFIRSTv1

$\vdash SWITCHBOARDvFIRSTv1 = SWITCHTYPEvFIRST$

SWITCHBOARDvLASTv1

$\vdash SWITCHBOARDvLASTv1 = SWITCHTYPEvLAST$

SWITCHBOARDvLENGTHv1

$\vdash SWITCHBOARDvLENGTHv1 = \# SWITCHTYPE$

SWITCHBOARDvRANGEv1

$\vdash SWITCHBOARDvRANGEv1 = SWITCHTYPE$

A.1.5 Theorems

abs_eq_abs_minus_thm

$\vdash \forall i : \mathbb{N} \bullet abs\ i = abs\ {\sim}\ i$

cn_SWITCHBOARDvRANGEv1_thm

$\vdash SWITCHBOARDvRANGEv1 = SWITCHTYPE$

cn_SWITCHBOARDvLENGTHv1_thm

$\vdash SWITCHBOARDvLENGTHv1 = \# SWITCHTYPE$

cn_SWITCHBOARDvLASTv1_thm

$\vdash SWITCHBOARDvLASTv1 = SWITCHTYPEvLAST$

cn_SWITCHBOARDvFIRSTv1_thm

$\vdash SWITCHBOARDvFIRSTv1 = SWITCHTYPEvFIRST$

cn_SWITCHBOARDvRANGE_thm

$\vdash SWITCHBOARDvRANGE = SWITCHTYPE$

cn_SWITCHBOARDvLENGTH_thm

$\vdash SWITCHBOARDvLENGTH = \# SWITCHTYPE$

cn_SWITCHBOARDvLAST_thm

$\vdash SWITCHBOARDvLAST = SWITCHTYPEvLAST$

cn_SWITCHBOARDvFIRST_thm

$\vdash SWITCHBOARDvFIRST = SWITCHTYPEvFIRST$

cn_SWITCHBOARD_thm

$\vdash SWITCHBOARD = SWITCHTYPE \rightarrow SWITCHDATA$

cn_SWITCHDATA_thm

$\vdash SWITCHDATA$
$= [NEXTSWITCH : SWITCHTYPE; STATE : BOOLEAN]$

cn_SWITCHTYPEvVAL_thm

$\vdash SWITCHTYPEvVAL = INTEGERvVAL$

cn_SWITCHTYPEvPOS_thm

$\vdash SWITCHTYPEvPOS = INTEGERvPOS$

cn_SWITCHTYPEvPRED_thm

$\vdash SWITCHTYPEvPRED = INTEGERvPRED$

cn_SWITCHTYPEvSUCC_thm

$\vdash SWITCHTYPEvSUCC = INTEGERvSUCC$

cn_SWITCHTYPEvLAST_thm

$\vdash SWITCHTYPEvLAST = 3$

cn_SWITCHTYPEvFIRST_thm

$\vdash SWITCHTYPEvFIRST = 1$

cn_SWITCHTYPE_thm

$\vdash \mathit{SWITCHTYPE} = \mathit{1} \mathrel{..} \mathit{3}$

cn_T_thm $\vdash T = \mathbb{U}$

cn_SWITCHDATAuSTATE_sig_thm

$$\begin{array}{l}
\vdash [\mathit{g1},\\
\quad \mathit{g2}](\mathit{SWITCHDATAuSTATE}[\mathit{g1}, \mathit{g2}]\\
\qquad \in [\mathit{STATE} : \mathit{g1}; \mathit{NEXTSWITCH} : \mathit{g2}] \times \mathit{g1}\\
\qquad\quad \rightarrow [\mathit{STATE} : \mathit{g1}; \mathit{NEXTSWITCH} : \mathit{g2}])
\end{array}$$

cn_SWITCHDATAuNEXTSWITCH_sig_thm

$$\begin{array}{l}
\vdash [\mathit{g1},\\
\quad \mathit{g2}](\mathit{SWITCHDATAuNEXTSWITCH}[\mathit{g1}, \mathit{g2}]\\
\qquad \in [\mathit{STATE} : \mathit{g1}; \mathit{NEXTSWITCH} : \mathit{g2}] \times \mathit{g2}\\
\qquad\quad \rightarrow [\mathit{STATE} : \mathit{g1}; \mathit{NEXTSWITCH} : \mathit{g2}])
\end{array}$$

cn_SWITCHDATAuNEXTSWITCH_thm

cn_SWITCHDATAuSTATE_thm

$$\begin{array}{l}
\vdash [\mathit{g1},\\
\quad \mathit{g2}](\forall\ r : [\mathit{STATE} : \mathit{g1}; \mathit{NEXTSWITCH} : \mathit{g2}];\\
\qquad\quad \mathit{x1} : \mathit{g1};\\
\qquad\quad \mathit{x2} : \mathit{g2}\\
\qquad \bullet\ \mathit{SWITCHDATAuSTATE}[\mathit{g1}, \mathit{g2}]\ (r, \mathit{x1})\\
\qquad\qquad = (\mathit{NEXTSWITCH} \mathrel{\widehat{=}} r.\mathit{NEXTSWITCH}, \mathit{STATE} \mathrel{\widehat{=}} \mathit{x1})\\
\qquad\quad \wedge\ \mathit{SWITCHDATAuNEXTSWITCH}[\mathit{g1}, \mathit{g2}]\ (r, \mathit{x2})\\
\qquad\qquad = (\mathit{NEXTSWITCH} \mathrel{\widehat{=}} \mathit{x2}, \mathit{STATE} \mathrel{\widehat{=}} r.\mathit{STATE}))
\end{array}$$

cn_CONSTSPEC_sig_thm

$\vdash \mathit{CONSTSPEC} \in \mathbb{P}_{\mathit{1}}\ T \twoheadrightarrow \mathbb{N}$

A.2 THE Z THEORY REAL_EG'spec

A.2.1 Parents

cn

A.2.2 Children

REAL_EGoINTERPOLATE′context REAL_EG′body

A.2.3 Global Variables

REAL_EGoANGLE

$\mathbb{P}\ \mathbb{R}$

REAL_EGoANGLEvDELTA

$\mathbb{R}$

REAL_EGoANGLEvFIRST

$\mathbb{R}$

REAL_EGoANGLEvLAST

$\mathbb{R}$

A.2.4 Definitions

REAL_EGoANGLE

$$\vdash \mathit{REAL\_EGoANGLE} = 0.0 \mathrel{..}_R 1.0 -_R 1.0 /_R 360.0$$

REAL_EGoANGLEvDELTA

$$\vdash \mathit{REAL\_EGoANGLEvDELTA} = 1.0 /_R 360.0$$

REAL_EGoANGLEvFIRST

$$\vdash \mathit{REAL\_EGoANGLEvFIRST} = 0.0$$

REAL_EGoANGLEvLAST

$$\vdash \mathit{REAL\_EGoANGLEvLAST} = 1.0 -_R 1.0 /_R 360.0$$

A.3 THE Z THEORY REAL_EG'body

A.3.1 Parents

$\mathit{REAL\_EG'spec}$ $\qquad$ cn

A.4 THE Z THEORY REAL_EGoINTERPOLATE'proc

A.4.1 Parents

$\mathit{REAL\_EGoINTERPOLATE'context}$

A.4.2 Conjectures

vcREAL_EGoINTERPOLATE_1

$$\begin{array}{l}
\forall A, B : \mathit{REAL\_EGoANGLE} \\
\quad | \; A +_R \mathit{REAL\_EGoANGLEvDELTA} <_R B \\
\qquad \land A +_R \mathit{REAL\_EGoANGLEvDELTA} \mathbin{\mathit{real\_less}} B = \mathit{TRUE} \\
\quad \bullet \; A <_R A +_R \mathit{REAL\_EGoANGLEvDELTA} \\
\qquad \land A +_R \mathit{REAL\_EGoANGLEvDELTA} <_R B
\end{array}$$

vcREAL_EGoINTERPOLATE_2

$$\begin{array}{l}
\forall A, B, C : \mathit{REAL\_EGoANGLE} \\
\quad | \; A +_R \mathit{REAL\_EGoANGLEvDELTA} <_R B \\
\qquad \land A +_R \mathit{REAL\_EGoANGLEvDELTA} \mathbin{\mathit{real\_less}} B = \mathit{FALSE} \\
\quad \bullet \; A <_R C \land C <_R B
\end{array}$$

A.4.3 Theorems

vcREAL_EGoINTERPOLATE_1

$$\begin{array}{l}
\vdash \forall A, B : \mathit{REAL\_EGoANGLE} \\
\qquad | \; A +_R \mathit{REAL\_EGoANGLEvDELTA} <_R B \\
\qquad\quad \land A +_R \mathit{REAL\_EGoANGLEvDELTA} \mathbin{\mathit{real\_less}} B = \mathit{TRUE} \\
\qquad \bullet \; A <_R A +_R \mathit{REAL\_EGoANGLEvDELTA} \\
\qquad\quad \land A +_R \mathit{REAL\_EGoANGLEvDELTA} <_R B
\end{array}$$

vcREAL_EGoINTERPOLATE_2

$$\begin{array}{l}
\vdash \forall A, B, C : \mathit{REAL\_EGoANGLE} \\
\qquad | \; A +_R \mathit{REAL\_EGoANGLEvDELTA} <_R B \\
\qquad\quad \land A +_R \mathit{REAL\_EGoANGLEvDELTA} \mathbin{\mathit{real\_less}} B = \mathit{FALSE} \\
\qquad \bullet \; A <_R C \land C <_R B
\end{array}$$

A.5 THE Z THEORY TC'spec

A.5.1 Parents

cn

A.5.2 Children

OPSoOPERATION_BUTTON′context
OPSoDIGIT_BUTTON′context
OPS′body
OPS′spec
GV′spec

A.5.3 Global Variables

TCoBASE	$\mathbb{Z}$
TCoPRECISION	$\mathbb{Z}$
TCoMAX_NUMBER	$\mathbb{Z}$
TCoMIN_NUMBER	$\mathbb{Z}$
TCoDIGIT	$\mathbb{P}\ \mathbb{Z}$
TCoDIGITvFIRST	$\mathbb{Z}$
TCoDIGITvLAST	$\mathbb{Z}$
TCoDIGITvSUCC	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoDIGITvPRED	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoDIGITvPOS	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoDIGITvVAL	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoNUMBER	$\mathbb{P}\ \mathbb{Z}$
TCoNUMBERvFIRST	$\mathbb{Z}$
TCoNUMBERvLAST	$\mathbb{Z}$
TCoNUMBERvSUCC	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoNUMBERvPRED	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoNUMBERvPOS	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoNUMBERvVAL	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoPLUS	$\mathbb{Z}$
TCoMINUS	$\mathbb{Z}$
TCoTIMES	$\mathbb{Z}$
TCoCHANGE_SIGN	$\mathbb{Z}$
TCoSQUARE_ROOT	$\mathbb{Z}$

TCoFACTORIAL $\mathbb{Z}$
TCoEQUALS $\mathbb{Z}$
TCoOPERATION $\mathbb{P}\ \mathbb{Z}$
TCoOPERATIONvFIRST
$\mathbb{Z}$
TCoOPERATIONvLAST
$\mathbb{Z}$
TCoOPERATIONvSUCC
$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoOPERATIONvPRED
$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoOPERATIONvPOS
$\mathbb{Z} \leftrightarrow \mathbb{Z}$
TCoOPERATIONvVAL
$\mathbb{Z} \leftrightarrow \mathbb{Z}$

A.5.4 Axioms

TCoBASE $\vdash \mathit{TCoBASE} \in \mathit{INTEGER} \land \mathit{TCoBASE} = \mathit{10}$
TCoPRECISION $\vdash \mathit{TCoPRECISION} \in \mathit{INTEGER} \land \mathit{TCoPRECISION} = \mathit{6}$
TCoMAX_NUMBER
$\vdash \mathit{TCoMAX\_NUMBER} \in \mathit{INTEGER}$
$\land\ \mathit{TCoMAX\_NUMBER} = \mathit{TCoBASE} ** \mathit{TCoPRECISION} - \mathit{1}$
TCoMIN_NUMBER
$\vdash \mathit{TCoMIN\_NUMBER} \in \mathit{INTEGER}$
$\land\ \mathit{TCoMIN\_NUMBER} = \sim \mathit{TCoMAX\_NUMBER}$

A.5.5 Definitions

TCoDIGIT $\vdash \mathit{TCoDIGIT} = \mathit{0} \mathinner{..} \mathit{TCoBASE} - \mathit{1}$
TCoDIGITvFIRST
$\vdash \mathit{TCoDIGITvFIRST} = \mathit{0}$
TCoDIGITvLAST
$\vdash \mathit{TCoDIGITvLAST} = \mathit{TCoBASE} - \mathit{1}$
TCoDIGITvSUCC
$\vdash \mathit{TCoDIGITvSUCC} = \mathit{INTEGERvSUCC}$
TCoDIGITvPRED
$\vdash \mathit{TCoDIGITvPRED} = \mathit{INTEGERvPRED}$
TCoDIGITvPOS $\vdash \mathit{TCoDIGITvPOS} = \mathit{INTEGERvPOS}$
TCoDIGITvVAL $\vdash \mathit{TCoDIGITvVAL} = \mathit{INTEGERvVAL}$
TCoNUMBER $\vdash \mathit{TCoNUMBER} = \mathit{TCoMIN\_NUMBER} \mathinner{..} \mathit{TCoMAX\_NUMBER}$
TCoNUMBERvFIRST
$\vdash \mathit{TCoNUMBERvFIRST} = \mathit{TCoMIN\_NUMBER}$
TCoNUMBERvLAST
$\vdash \mathit{TCoNUMBERvLAST} = \mathit{TCoMAX\_NUMBER}$
TCoNUMBERvSUCC
$\vdash \mathit{TCoNUMBERvSUCC} = \mathit{INTEGERvSUCC}$
TCoNUMBERvPRED
$\vdash \mathit{TCoNUMBERvPRED} = \mathit{INTEGERvPRED}$
TCoNUMBERvPOS
$\vdash \mathit{TCoNUMBERvPOS} = \mathit{INTEGERvPOS}$

TCoNUMBERvVAL
$\vdash \mathit{TCoNUMBERvVAL} = \mathit{INTEGERvVAL}$

TCoPLUS $\vdash \mathit{TCoPLUS} = \mathit{0}$

TCoMINUS $\vdash \mathit{TCoMINUS} = \mathit{1}$

TCoTIMES $\vdash \mathit{TCoTIMES} = \mathit{2}$

TCoCHANGE_SIGN
$\vdash \mathit{TCoCHANGE\_SIGN} = \mathit{3}$

TCoSQUARE_ROOT
$\vdash \mathit{TCoSQUARE\_ROOT} = \mathit{4}$

TCoFACTORIAL $\vdash \mathit{TCoFACTORIAL} = \mathit{5}$

TCoEQUALS $\vdash \mathit{TCoEQUALS} = \mathit{6}$

TCoOPERATION $\vdash \mathit{TCoOPERATION} = \mathit{TCoPLUS} \mathrel{..} \mathit{TCoEQUALS}$

TCoOPERATIONvFIRST
$\vdash \mathit{TCoOPERATIONvFIRST} = \mathit{TCoPLUS}$

TCoOPERATIONvLAST
$\vdash \mathit{TCoOPERATIONvLAST} = \mathit{TCoEQUALS}$

TCoOPERATIONvSUCC
$\vdash \mathit{TCoOPERATIONvSUCC}$
$= (\mathit{TCoOPERATION} \setminus \{\mathit{TCoOPERATIONvLAST}\}) \lhd \mathit{succ}$

TCoOPERATIONvPRED
$\vdash \mathit{TCoOPERATIONvPRED} = \mathit{TCoOPERATIONvSUCC}\ ^\sim$

TCoOPERATIONvPOS
$\vdash \mathit{TCoOPERATIONvPOS} = \mathit{id}\ \mathit{TCoOPERATION}$

TCoOPERATIONvVAL
$\vdash \mathit{TCoOPERATIONvVAL} = \mathit{TCoOPERATIONvPOS}\ ^\sim$

A.6 THE Z THEORY GV'spec

A.6.1 Parents

TC′spec *cn*

A.6.2 Children

OPSoOPERATION_BUTTON′context *OPS′spec*
OPSoDIGIT_BUTTON′context *calc_prelims*
OPS′body

A.7 THE Z THEORY calc_prelims

A.7.1 Parents

GV′spec

A.7.2 Children

OPS′spec

A.7.3 Global Variables

DO_DIGIT $\mathbb{P}$
$[D, GVoDISPLAY, GVoDISPLAY_0, GVoIN\_NUMBER,$
$GVoIN\_NUMBER_0 : \mathbb{Z}]$

UNARY $\mathbb{P}\ \mathbb{Z}$

BINARY $\mathbb{P}\ \mathbb{Z}$

fact $\mathbb{Z} \leftrightarrow \mathbb{Z}$

DO_UNARY_OPERATION
$\mathbb{P}$
$[GVoACCUMULATOR, GVoACCUMULATOR_0, GVoDISPLAY,$
$GVoDISPLAY_0, GVoIN\_NUMBER, GVoLAST\_OP,$
$GVoLAST\_OP_0, O : \mathbb{Z}]$

DO_BINARY_OPERATION
$\mathbb{P}$
$[GVoACCUMULATOR, GVoACCUMULATOR_0, GVoDISPLAY,$
$GVoDISPLAY_0, GVoIN\_NUMBER, GVoLAST\_OP,$
$GVoLAST\_OP_0, O : \mathbb{Z}]$

DO_OPERATION $\mathbb{P}$
$[GVoACCUMULATOR, GVoACCUMULATOR_0, GVoDISPLAY,$
$GVoDISPLAY_0, GVoIN\_NUMBER, GVoLAST\_OP,$
$GVoLAST\_OP_0, O : \mathbb{Z}]$

A.7.4 Axioms

fact $\vdash fact \in \mathbb{N} \rightarrow \mathbb{N}$
$\land\ fact\ 0 = 1$
$\land\ (\forall\ m : \mathbb{N} \bullet fact\ (m + 1) = (m + 1) * fact\ m)$

A.7.5 Definitions

DO_DIGIT $\vdash DO\_DIGIT$
$= [GVoDISPLAY_0, GVoDISPLAY : \mathbb{Z};$
$GVoIN\_NUMBER_0, GVoIN\_NUMBER : BOOLEAN;$
$D : TCoDIGIT$
$\mid (GVoIN\_NUMBER_0 = TRUE$
$\Rightarrow GVoDISPLAY = GVoDISPLAY_0 * TCoBASE + D)$
$\land\ (GVoIN\_NUMBER_0 = FALSE \Rightarrow GVoDISPLAY = D)$
$\land\ GVoIN\_NUMBER = TRUE]$

UNARY $\vdash UNARY$
$= \{TCoCHANGE\_SIGN, TCoFACTORIAL, TCoSQUARE\_ROOT\}$

BINARY $\vdash BINARY = TCoOPERATION \setminus UNARY$

DO_UNARY_OPERATION
$\vdash DO\_UNARY\_OPERATION$
$= [GVoACCUMULATOR_0, GVoACCUMULATOR : \mathbb{Z};$
$GVoDISPLAY_0, GVoDISPLAY : \mathbb{Z};$
$GVoLAST\_OP_0, GVoLAST\_OP : \mathbb{Z};$
$GVoIN\_NUMBER : BOOLEAN;$
$O : UNARY$
$\mid GVoIN\_NUMBER = FALSE$
$\land\ GVoACCUMULATOR = GVoACCUMULATOR_0$

$$\begin{array}{l}
\land\ GVoLAST\_OP = GVoLAST\_OP_0 \\
\land\ (O = TCoCHANGE\_SIGN \\
\quad \Rightarrow GVoDISPLAY = \sim GVoDISPLAY_0) \\
\land\ (O = TCoFACTORIAL \land GVoDISPLAY_0 \geq 0 \\
\quad \Rightarrow GVoDISPLAY = \mathit{fact}\ GVoDISPLAY_0) \\
\land\ (O = TCoSQUARE\_ROOT \land GVoDISPLAY_0 \geq 0 \\
\quad \Rightarrow GVoDISPLAY ** 2 \leq GVoDISPLAY_0 \\
\qquad \land GVoDISPLAY_0 < (GVoDISPLAY + 1) ** 2)]
\end{array}$$

DO_BINARY_OPERATION

$$\begin{array}{l}
\vdash DO\_BINARY\_OPERATION \\
\quad = [GVoACCUMULATOR_0,\ GVoACCUMULATOR : \mathbb{Z}; \\
\qquad GVoDISPLAY_0,\ GVoDISPLAY : \mathbb{Z}; \\
\qquad GVoLAST\_OP_0,\ GVoLAST\_OP : \mathbb{Z}; \\
\qquad GVoIN\_NUMBER : BOOLEAN; \\
\qquad O : BINARY \\
\qquad |\ GVoIN\_NUMBER = FALSE \\
\qquad\ \land\ GVoDISPLAY = GVoACCUMULATOR \\
\qquad\ \land\ GVoLAST\_OP = O \\
\qquad\ \land\ (GVoLAST\_OP_0 = TCoEQUALS \\
\qquad\quad \Rightarrow GVoACCUMULATOR = GVoDISPLAY_0) \\
\qquad\ \land\ (GVoLAST\_OP_0 = TCoPLUS \\
\qquad\quad \Rightarrow GVoACCUMULATOR \\
\qquad\qquad = GVoACCUMULATOR_0 + GVoDISPLAY_0) \\
\qquad\ \land\ (GVoLAST\_OP_0 = TCoMINUS \\
\qquad\quad \Rightarrow GVoACCUMULATOR \\
\qquad\qquad = GVoACCUMULATOR_0 - GVoDISPLAY_0) \\
\qquad\ \land\ (GVoLAST\_OP_0 = TCoTIMES \\
\qquad\quad \Rightarrow GVoACCUMULATOR \\
\qquad\qquad = GVoACCUMULATOR_0 * GVoDISPLAY_0)]
\end{array}$$

DO_OPERATION $\vdash DO\_OPERATION$

$$= (DO\_UNARY\_OPERATION \lor DO\_BINARY\_OPERATION)$$

A.7.6 Theorems

cn_TCoOPERATIONvVAL_thm

$$\vdash \forall\ i : TCoOPERATION \bullet TCoOPERATIONvVAL\ i = i$$

cn_TCoOPERATIONvVAL_sig_thm

$$\vdash TCoOPERATIONvVAL \in TCoOPERATION \rightarrow TCoOPERATION$$

cn_TCoOPERATIONvPOS_thm

$$\vdash \forall\ i : TCoOPERATION \bullet TCoOPERATIONvPOS\ i = i$$

cn_TCoOPERATIONvPOS_sig_thm

$$\vdash TCoOPERATIONvPOS \in TCoOPERATION \rightarrow TCoOPERATION$$

cn_TCoOPERATIONvPRED_thm

$$\begin{array}{l}
\vdash \forall\ i : TCoPLUS + 1 \mathrel{..} TCoEQUALS \\
\quad \bullet\ TCoOPERATIONvPRED\ i = i + \sim 1
\end{array}$$

cn_TCoOPERATIONvPRED_sig_thm

$$\begin{array}{l}
\vdash TCoOPERATIONvPRED \\
\quad \in TCoPLUS + 1 \mathrel{..} TCoEQUALS \\
\qquad \rightarrow TCoPLUS \mathrel{..} TCoEQUALS + \sim 1
\end{array}$$

cn_TCoOPERATIONvSUCC_thm

$$\vdash \forall\ i : TCoPLUS \mathrel{..} TCoEQUALS + \sim 1$$

$\bullet\ \mathit{TCoOPERATIONvSUCC}\ i = i + 1$

cn_TCoOPERATIONvSUCC_sig_thm

$$\begin{array}{l} \vdash \mathit{TCoOPERATIONvSUCC} \\ \quad \in \mathit{TCoPLUS} \mathrel{..} \mathit{TCoEQUALS} + \sim 1 \\ \qquad \to \mathit{TCoPLUS} + 1 \mathrel{..} \mathit{TCoEQUALS} \end{array}$$

cn_TCoOPERATIONvLAST_thm

$\vdash \mathit{TCoOPERATIONvLAST} = \mathit{TCoEQUALS}$

cn_TCoOPERATIONvFIRST_thm

$\vdash \mathit{TCoOPERATIONvFIRST} = \mathit{TCoPLUS}$

cn_TCoOPERATION_thm

$\vdash \mathit{TCoOPERATION} = \mathit{TCoPLUS} \mathrel{..} \mathit{TCoEQUALS}$

cn_TCoEQUALS_thm

$\vdash \mathit{TCoEQUALS} = 6$

cn_TCoFACTORIAL_thm

$\vdash \mathit{TCoFACTORIAL} = 5$

cn_TCoSQUARE_ROOT_thm

$\vdash \mathit{TCoSQUARE\_ROOT} = 4$

cn_TCoCHANGE_SIGN_thm

$\vdash \mathit{TCoCHANGE\_SIGN} = 3$

cn_TCoTIMES_thm

$\vdash \mathit{TCoTIMES} = 2$

cn_TCoMINUS_thm

$\vdash \mathit{TCoMINUS} = 1$

cn_TCoPLUS_thm

$\vdash \mathit{TCoPLUS} = 0$

cn_TCoNUMBERvVAL_thm

$\vdash \mathit{TCoNUMBERvVAL} = \mathit{INTEGERvVAL}$

cn_TCoNUMBERvPOS_thm

$\vdash \mathit{TCoNUMBERvPOS} = \mathit{INTEGERvPOS}$

cn_TCoNUMBERvPRED_thm

$\vdash \mathit{TCoNUMBERvPRED} = \mathit{INTEGERvPRED}$

cn_TCoNUMBERvSUCC_thm

$\vdash \mathit{TCoNUMBERvSUCC} = \mathit{INTEGERvSUCC}$

cn_TCoNUMBERvLAST_thm

$\vdash \mathit{TCoNUMBERvLAST} = \mathit{TCoMAX\_NUMBER}$

cn_TCoNUMBERvFIRST_thm

$\vdash \mathit{TCoNUMBERvFIRST} = \mathit{TCoMIN\_NUMBER}$

cn_TCoNUMBER_thm

$\vdash \mathit{TCoNUMBER} = \mathit{TCoMIN\_NUMBER} \mathrel{..} \mathit{TCoMAX\_NUMBER}$

cn_TCoDIGITvVAL_thm

$\vdash \mathit{TCoDIGITvVAL} = \mathit{INTEGERvVAL}$

cn_TCoDIGITvPOS_thm

$\vdash \mathit{TCoDIGITvPOS} = \mathit{INTEGERvPOS}$

cn_TCoDIGITvPRED_thm

$\vdash \mathit{TCoDIGITvPRED} = \mathit{INTEGERvPRED}$

cn_TCoDIGITvSUCC_thm

$\vdash \mathit{TCoDIGITvSUCC} = \mathit{INTEGERvSUCC}$

cn_TCoDIGITvLAST_thm

$\vdash \mathit{TCoDIGITvLAST} = \mathit{TCoBASE} + \sim 1$

cn_TCoDIGITvFIRST_thm

$\vdash \mathit{TCoDIGITvFIRST} = 0$

cn_TCoDIGIT_thm

$$\vdash \mathit{TCoDIGIT} = \mathit{0} \mathrel{..} \mathit{TCoBASE} + \sim \mathit{1}$$

cn_TCoMIN_NUMBER_sig_thm

$$\vdash \mathit{TCoMIN\_NUMBER} \in \mathit{INTEGER}$$

cn_TCoMIN_NUMBER_thm

$$\vdash \mathit{TCoMIN\_NUMBER} = \sim \mathit{TCoMAX\_NUMBER}$$

cn_TCoMAX_NUMBER_sig_thm

$$\vdash \mathit{TCoMAX\_NUMBER} \in \mathit{INTEGER}$$

cn_TCoMAX_NUMBER_thm

$$\vdash \mathit{TCoMAX\_NUMBER} = \mathit{TCoBASE} ** \mathit{TCoPRECISION} + \sim \mathit{1}$$

cn_TCoPRECISION_sig_thm

$$\vdash \mathit{TCoPRECISION} \in \mathit{INTEGER}$$

cn_TCoPRECISION_thm

$$\vdash \mathit{TCoPRECISION} = \mathit{6}$$

cn_TCoBASE_sig_thm

$$\vdash \mathit{TCoBASE} \in \mathit{INTEGER}$$

cn_TCoBASE_thm

$$\vdash \mathit{TCoBASE} = \mathit{10}$$

cn_DO_OPERATION_thm

$$\begin{array}{l}
\vdash \mathit{DO\_OPERATION} \\
\quad = (\mathit{DO\_UNARY\_OPERATION} \lor \mathit{DO\_BINARY\_OPERATION})
\end{array}$$

cn_DO_BINARY_OPERATION_thm

$$\begin{array}{l}
\vdash \mathit{DO\_BINARY\_OPERATION} \\
\quad = [\mathit{GVoACCUMULATOR} : \mathbb{Z}; \\
\quad\quad \mathit{GVoACCUMULATOR}_0 : \mathbb{Z}; \\
\quad\quad \mathit{GVoDISPLAY} : \mathbb{Z}; \\
\quad\quad \mathit{GVoDISPLAY}_0 : \mathbb{Z}; \\
\quad\quad \mathit{GVoIN\_NUMBER} : \mathit{BOOLEAN}; \\
\quad\quad \mathit{GVoLAST\_OP} : \mathbb{Z}; \\
\quad\quad \mathit{GVoLAST\_OP}_0 : \mathbb{Z}; \\
\quad\quad O : \mathit{BINARY} \\
\quad\quad | \ \mathit{GVoIN\_NUMBER} = \mathit{FALSE} \\
\quad\quad\quad \land \mathit{GVoDISPLAY} = \mathit{GVoACCUMULATOR} \\
\quad\quad\quad \land \mathit{GVoLAST\_OP} = O \\
\quad\quad\quad \land (\mathit{GVoLAST\_OP}_0 = \mathit{TCoEQUALS} \\
\quad\quad\quad\quad \Rightarrow \mathit{GVoACCUMULATOR} = \mathit{GVoDISPLAY}_0) \\
\quad\quad\quad \land (\mathit{GVoLAST\_OP}_0 = \mathit{TCoPLUS} \\
\quad\quad\quad\quad \Rightarrow \mathit{GVoACCUMULATOR} \\
\quad\quad\quad\quad\quad = \mathit{GVoACCUMULATOR}_0 + \mathit{GVoDISPLAY}_0) \\
\quad\quad\quad \land (\mathit{GVoLAST\_OP}_0 = \mathit{TCoMINUS} \\
\quad\quad\quad\quad \Rightarrow \mathit{GVoACCUMULATOR} \\
\quad\quad\quad\quad\quad = \mathit{GVoACCUMULATOR}_0 - \mathit{GVoDISPLAY}_0) \\
\quad\quad\quad \land (\mathit{GVoLAST\_OP}_0 = \mathit{TCoTIMES} \\
\quad\quad\quad\quad \Rightarrow \mathit{GVoACCUMULATOR} \\
\quad\quad\quad\quad\quad = \mathit{GVoACCUMULATOR}_0 * \mathit{GVoDISPLAY}_0)]
\end{array}$$

cn_DO_UNARY_OPERATION_thm

$$\begin{array}{l}
\vdash \mathit{DO\_UNARY\_OPERATION} \\
\quad = [\mathit{GVoACCUMULATOR} : \mathbb{Z}; \\
\quad\quad \mathit{GVoACCUMULATOR}_0 : \mathbb{Z}; \\
\quad\quad \mathit{GVoDISPLAY} : \mathbb{Z}; \\
\quad\quad \mathit{GVoDISPLAY}_0 : \mathbb{Z};
\end{array}$$

$GVoIN\_NUMBER : BOOLEAN;$
$GVoLAST\_OP : \mathbb{Z};$
$GVoLAST\_OP_0 : \mathbb{Z};$
$O : UNARY$
$| \; GVoIN\_NUMBER = FALSE$
$\quad \land GVoACCUMULATOR = GVoACCUMULATOR_0$
$\quad \land GVoLAST\_OP = GVoLAST\_OP_0$
$\quad \land (O = TCoCHANGE\_SIGN$
$\qquad \Rightarrow GVoDISPLAY = \sim GVoDISPLAY_0)$
$\quad \land (O = TCoFACTORIAL \land GVoDISPLAY_0 \geq 0$
$\qquad \Rightarrow GVoDISPLAY = fact\ GVoDISPLAY_0)$
$\quad \land (O = TCoSQUARE\_ROOT \land GVoDISPLAY_0 \geq 0$
$\qquad \Rightarrow GVoDISPLAY ** 2 \leq GVoDISPLAY_0$
$\qquad\quad \land GVoDISPLAY_0 < (GVoDISPLAY + 1) ** 2)]$

cn_BINARY_thm
$\vdash BINARY = TCoOPERATION \setminus UNARY$

cn_UNARY_thm $\vdash UNARY$
$= \{TCoCHANGE\_SIGN, TCoFACTORIAL, TCoSQUARE\_ROOT\}$

cn_DO_DIGIT_thm
$\vdash DO\_DIGIT$
$= [D : TCoDIGIT;$
$\quad GVoDISPLAY : \mathbb{Z};$
$\quad GVoDISPLAY_0 : \mathbb{Z};$
$\quad GVoIN\_NUMBER : BOOLEAN;$
$\quad GVoIN\_NUMBER_0 : BOOLEAN$
$\quad | \; (GVoIN\_NUMBER_0 = TRUE$
$\qquad \Rightarrow GVoDISPLAY = GVoDISPLAY_0 * TCoBASE + D)$
$\quad \land (GVoIN\_NUMBER_0 = FALSE \Rightarrow GVoDISPLAY = D)$
$\quad \land GVoIN\_NUMBER = TRUE]$

cn_fact_sig_thm
$\vdash fact \in \mathbb{N} \rightarrow \mathbb{N}$

cn_fact_thm $\vdash fact\ 0 = 1$
$\quad \land (\forall m : \mathbb{N} \bullet fact\ (m + 1) = (m + 1) * fact\ m)$

natural_thm $\vdash \forall m : NATURAL \bullet m \geq 0$

fact_thm $\vdash fact\ 0 = 1 \land fact\ 1 = 1$

A.8 THE Z THEORY OPS'spec

A.8.1 Parents

GV'spec *TC'spec* *calc_prelims* *cn*

A.8.2 Children

OPSoOPERATION_BUTTON'context *OPS'body*
OPSoDIGIT_BUTTON'context

A.9 THE Z THEORY OPS'body

A.9.1 Parents

$OPS'spec \qquad GV'spec \qquad TC'spec \qquad cn$

A.9.2 Conjectures

vcOPS_1 $true \Rightarrow true$

vcOPS_2

$$\begin{array}{l}\forall\ GVoIN\_NUMBER, GVoIN\_NUMBER_0 : BOOLEAN;\\ \quad D : TCoDIGIT;\\ \quad GVoDISPLAY, GVoDISPLAY_0 : TCoNUMBER\\ \ \mid true \land DO\_DIGIT\\ \ \bullet\ DO\_DIGIT\end{array}$$

vcOPS_3 $true \Rightarrow true$

vcOPS_4

$$\begin{array}{l}\forall\ GVoIN\_NUMBER : BOOLEAN;\\ \quad GVoACCUMULATOR, GVoACCUMULATOR_0, GVoDISPLAY,\\ \quad\ \ GVoDISPLAY_0 : TCoNUMBER;\\ \quad GVoLAST\_OP, GVoLAST\_OP_0, O : TCoOPERATION\\ \ \mid true \land DO\_OPERATION\\ \ \bullet\ DO\_OPERATION\end{array}$$

A.9.3 Theorems

vcOPS_2

$$\begin{array}{l}\vdash \forall\ GVoIN\_NUMBER, GVoIN\_NUMBER_0 : BOOLEAN;\\ \quad D : TCoDIGIT;\\ \quad GVoDISPLAY, GVoDISPLAY_0 : TCoNUMBER\\ \ \mid true \land DO\_DIGIT\\ \ \bullet\ DO\_DIGIT\end{array}$$

A.10 THE Z THEORY OPSoDIGIT_BUTTON'proc

A.10.1 Parents

$OPSoDIGIT\_BUTTON'context$

A.10.2 Conjectures

vcOPSoDIGIT_BUTTON_1

$true \Rightarrow true$

vcOPSoDIGIT_BUTTON_2

$$\begin{array}{l}\forall\ GVoIN\_NUMBER, GVoIN\_NUMBER_0 : BOOLEAN;\\ \quad D : TCoDIGIT;\\ \quad GVoDISPLAY, GVoDISPLAY_0 : TCoNUMBER\\ \ \mid true \land DO\_DIGIT\\ \ \bullet\ DO\_DIGIT\end{array}$$

vc3001_1

$$\begin{array}{l}\forall\ GVoIN\_NUMBER : BOOLEAN;\\ \quad D : TCoDIGIT;\\ \quad GVoDISPLAY : TCoNUMBER\end{array}$$

$$\begin{array}{l}
\quad \mid \mathit{true} \land \mathit{GVoIN\_NUMBER} = \mathit{TRUE} \\
\quad \bullet (D \mathrel{\widehat{=}} D,\ \mathit{GVoDISPLAY} \mathrel{\widehat{=}} \mathit{GVoDISPLAY} * \mathit{TCoBASE} + D, \\
\qquad \mathit{GVoDISPLAY}_0 \mathrel{\widehat{=}} \mathit{GVoDISPLAY},\ \mathit{GVoIN\_NUMBER} \mathrel{\widehat{=}} \mathit{TRUE}, \\
\qquad \mathit{GVoIN\_NUMBER}_0 \mathrel{\widehat{=}} \mathit{GVoIN\_NUMBER}) \\
\quad\ \in \mathit{DO\_DIGIT}
\end{array}$$

vc3001_2

$$\begin{array}{l}
\forall\ \mathit{GVoIN\_NUMBER} : \mathit{BOOLEAN}; \\
\quad D : \mathit{TCoDIGIT}; \\
\quad \mathit{GVoDISPLAY} : \mathit{TCoNUMBER} \\
\quad \mid \mathit{true} \land \mathit{GVoIN\_NUMBER} = \mathit{FALSE} \\
\quad \bullet (D \mathrel{\widehat{=}} D,\ \mathit{GVoDISPLAY} \mathrel{\widehat{=}} D,\ \mathit{GVoDISPLAY}_0 \mathrel{\widehat{=}} \mathit{GVoDISPLAY}, \\
\qquad \mathit{GVoIN\_NUMBER} \mathrel{\widehat{=}} \mathit{TRUE}, \\
\qquad \mathit{GVoIN\_NUMBER}_0 \mathrel{\widehat{=}} \mathit{GVoIN\_NUMBER}) \\
\quad\ \in \mathit{DO\_DIGIT}
\end{array}$$

A.10.3 Theorems

vc3001_1

$$\begin{array}{l}
\vdash \forall\ \mathit{GVoIN\_NUMBER} : \mathit{BOOLEAN}; \\
\qquad D : \mathit{TCoDIGIT}; \\
\qquad \mathit{GVoDISPLAY} : \mathit{TCoNUMBER} \\
\quad \mid \mathit{true} \land \mathit{GVoIN\_NUMBER} = \mathit{TRUE} \\
\quad \bullet (D \mathrel{\widehat{=}} D,\ \mathit{GVoDISPLAY} \mathrel{\widehat{=}} \mathit{GVoDISPLAY} * \mathit{TCoBASE} + D, \\
\qquad \mathit{GVoDISPLAY}_0 \mathrel{\widehat{=}} \mathit{GVoDISPLAY}, \\
\qquad \mathit{GVoIN\_NUMBER} \mathrel{\widehat{=}} \mathit{TRUE}, \\
\qquad \mathit{GVoIN\_NUMBER}_0 \mathrel{\widehat{=}} \mathit{GVoIN\_NUMBER}) \\
\quad\ \in \mathit{DO\_DIGIT}
\end{array}$$

A.11 THE Z THEORY OPSoOPERATION_BUTTON'proc

A.11.1 Parents

OPSoOPERATION_BUTTON'context

A.11.2 Global Variables

FACT	$\mathbb{Z} \leftrightarrow \mathbb{Z}$
SQRT	$\mathbb{Z} \leftrightarrow \mathbb{Z}$

A.11.3 Axioms

FACT

$$\begin{array}{l}
\vdash \mathit{FACT} \in \mathit{NATURAL} \rightarrow \mathit{NATURAL} \\
\quad \land (\forall\ M : \mathit{NATURAL} \bullet \mathit{true} \Rightarrow \mathit{FACT}\ M = \mathit{fact}\ M)
\end{array}$$

SQRT

$$\begin{array}{l}
\vdash \mathit{SQRT} \in \mathit{NATURAL} \rightarrow \mathit{NATURAL} \\
\quad \land (\forall\ M : \mathit{NATURAL} \\
\qquad \bullet \mathit{true} \\
\qquad\ \Rightarrow \mathit{SQRT}\ M ** \mathit{2} \leq M \\
\qquad\quad \land M < (\mathit{SQRT}\ M + \mathit{1}) ** \mathit{2})
\end{array}$$

Constraint 1

$$\vdash \mathit{TCoMAX\_NUMBER} \leq \mathit{INTEGERvLAST}$$

A.11.4 Conjectures

vcOPSoOPERATION_BUTTON_1

$$\mathit{true} \Rightarrow \mathit{true}$$

vcOPSoOPERATION_BUTTON_2

$$\begin{array}{l}
\forall\ \mathit{GVoIN\_NUMBER} : \mathit{BOOLEAN};\\
\quad \mathit{GVoACCUMULATOR},\ \mathit{GVoACCUMULATOR}_0,\ \mathit{GVoDISPLAY},\\
\quad\quad \mathit{GVoDISPLAY}_0 : \mathit{TCoNUMBER};\\
\quad \mathit{GVoLAST\_OP},\ \mathit{GVoLAST\_OP}_0,\ O : \mathit{TCoOPERATION}\\
\ \mid \mathit{true} \land \mathit{DO\_OPERATION}\\
\ \bullet\ \mathit{DO\_OPERATION}
\end{array}$$

vc3002_1

$$\begin{array}{l}
\forall\ \mathit{GVoACCUMULATOR},\ \mathit{GVoDISPLAY} : \mathit{TCoNUMBER};\\
\quad \mathit{GVoLAST\_OP},\ O : \mathit{TCoOPERATION}\\
\ \mid \mathit{true} \land O\ \mathit{eq}\ \mathit{TCoCHANGE\_SIGN} = \mathit{TRUE}\\
\ \bullet\ (\mathit{GVoACCUMULATOR} \mathrel{\widehat{=}} \mathit{GVoACCUMULATOR},\\
\quad\quad \mathit{GVoACCUMULATOR}_0 \mathrel{\widehat{=}} \mathit{GVoACCUMULATOR},\\
\quad\quad \mathit{GVoDISPLAY} \mathrel{\widehat{=}} \sim \mathit{GVoDISPLAY},\\
\quad\quad \mathit{GVoDISPLAY}_0 \mathrel{\widehat{=}} \mathit{GVoDISPLAY},\\
\quad\quad \mathit{GVoIN\_NUMBER} \mathrel{\widehat{=}} \mathit{FALSE},\ \mathit{GVoLAST\_OP} \mathrel{\widehat{=}} \mathit{GVoLAST\_OP},\\
\quad\quad \mathit{GVoLAST\_OP}_0 \mathrel{\widehat{=}} \mathit{GVoLAST\_OP},\ O \mathrel{\widehat{=}} O)\\
\quad \in \mathit{DO\_OPERATION}
\end{array}$$

vc3002_2

$$\begin{array}{l}
\forall\ \mathit{GVoACCUMULATOR},\ \mathit{GVoDISPLAY} : \mathit{TCoNUMBER};\\
\quad \mathit{GVoLAST\_OP},\ O : \mathit{TCoOPERATION}\\
\ \mid \mathit{true}\\
\quad \land\ O\ \mathit{eq}\ \mathit{TCoCHANGE\_SIGN} = \mathit{FALSE}\\
\quad \land\ O\ \mathit{eq}\ \mathit{TCoFACTORIAL} = \mathit{TRUE}\\
\ \bullet\ (\mathit{GVoACCUMULATOR} \mathrel{\widehat{=}} \mathit{GVoACCUMULATOR},\\
\quad\quad \mathit{GVoACCUMULATOR}_0 \mathrel{\widehat{=}} \mathit{GVoACCUMULATOR},\\
\quad\quad \mathit{GVoDISPLAY} \mathrel{\widehat{=}} \mathit{FACT}\ \mathit{GVoDISPLAY},\\
\quad\quad \mathit{GVoDISPLAY}_0 \mathrel{\widehat{=}} \mathit{GVoDISPLAY},\\
\quad\quad \mathit{GVoIN\_NUMBER} \mathrel{\widehat{=}} \mathit{FALSE},\ \mathit{GVoLAST\_OP} \mathrel{\widehat{=}} \mathit{GVoLAST\_OP},\\
\quad\quad \mathit{GVoLAST\_OP}_0 \mathrel{\widehat{=}} \mathit{GVoLAST\_OP},\ O \mathrel{\widehat{=}} O)\\
\quad \in \mathit{DO\_OPERATION}
\end{array}$$

vc3002_3

$$\begin{array}{l}
\forall\ \mathit{GVoACCUMULATOR},\ \mathit{GVoDISPLAY} : \mathit{TCoNUMBER};\\
\quad \mathit{GVoLAST\_OP},\ O : \mathit{TCoOPERATION}\\
\ \mid \mathit{true}\\
\quad \land\ O\ \mathit{eq}\ \mathit{TCoCHANGE\_SIGN} = \mathit{FALSE}\\
\quad \land\ O\ \mathit{eq}\ \mathit{TCoFACTORIAL} = \mathit{FALSE}\\
\quad \land\ O\ \mathit{eq}\ \mathit{TCoSQUARE\_ROOT} = \mathit{TRUE}\\
\ \bullet\ (\mathit{GVoACCUMULATOR} \mathrel{\widehat{=}} \mathit{GVoACCUMULATOR},\\
\quad\quad \mathit{GVoACCUMULATOR}_0 \mathrel{\widehat{=}} \mathit{GVoACCUMULATOR},\\
\quad\quad \mathit{GVoDISPLAY} \mathrel{\widehat{=}} \mathit{SQRT}\ \mathit{GVoDISPLAY},\\
\quad\quad \mathit{GVoDISPLAY}_0 \mathrel{\widehat{=}} \mathit{GVoDISPLAY},\\
\quad\quad \mathit{GVoIN\_NUMBER} \mathrel{\widehat{=}} \mathit{FALSE},\ \mathit{GVoLAST\_OP} \mathrel{\widehat{=}} \mathit{GVoLAST\_OP},\\
\quad\quad \mathit{GVoLAST\_OP}_0 \mathrel{\widehat{=}} \mathit{GVoLAST\_OP},\ O \mathrel{\widehat{=}} O)\\
\quad \in \mathit{DO\_OPERATION}
\end{array}$$

vc3002_4

$$\begin{array}{l}
\forall\ \mathit{GVoACCUMULATOR},\ \mathit{GVoDISPLAY} : \mathit{TCoNUMBER};\\
\quad \mathit{GVoLAST\_OP},\ O : \mathit{TCoOPERATION}\\
\ \mid \mathit{true}\\
\quad \land\ O\ \mathit{eq}\ \mathit{TCoCHANGE\_SIGN} = \mathit{FALSE}\\
\quad \land\ O\ \mathit{eq}\ \mathit{TCoFACTORIAL} = \mathit{FALSE}
\end{array}$$

$$\begin{array}{l}
\quad \land\ O\ eq\ TCoSQUARE\_ROOT = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoEQUALS = TRUE \\
\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR, \\
\qquad GVoDISPLAY \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,\ GVoLAST\_OP \mathrel{\widehat{=}} O, \\
\qquad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O) \\
\quad \in DO\_OPERATION
\end{array}$$

vc3002_5

$$\begin{array}{l}
\forall\ GVoACCUMULATOR,\ GVoDISPLAY : TCoNUMBER; \\
\quad GVoLAST\_OP,\ O : TCoOPERATION \\
\mid true \\
\quad \land\ O\ eq\ TCoCHANGE\_SIGN = FALSE \\
\quad \land\ O\ eq\ TCoFACTORIAL = FALSE \\
\quad \land\ O\ eq\ TCoSQUARE\_ROOT = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoEQUALS = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoPLUS = TRUE \\
\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoACCUMULATOR + GVoDISPLAY, \\
\qquad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR, \\
\qquad GVoDISPLAY \mathrel{\widehat{=}} GVoACCUMULATOR + GVoDISPLAY, \\
\qquad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,\ GVoLAST\_OP \mathrel{\widehat{=}} O, \\
\qquad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O) \\
\quad \in DO\_OPERATION
\end{array}$$

vc3002_6

$$\begin{array}{l}
\forall\ GVoACCUMULATOR,\ GVoDISPLAY : TCoNUMBER; \\
\quad GVoLAST\_OP,\ O : TCoOPERATION \\
\mid true \\
\quad \land\ O\ eq\ TCoCHANGE\_SIGN = FALSE \\
\quad \land\ O\ eq\ TCoFACTORIAL = FALSE \\
\quad \land\ O\ eq\ TCoSQUARE\_ROOT = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoEQUALS = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoPLUS = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoMINUS = TRUE \\
\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoACCUMULATOR - GVoDISPLAY, \\
\qquad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR, \\
\qquad GVoDISPLAY \mathrel{\widehat{=}} GVoACCUMULATOR - GVoDISPLAY, \\
\qquad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,\ GVoLAST\_OP \mathrel{\widehat{=}} O, \\
\qquad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O) \\
\quad \in DO\_OPERATION
\end{array}$$

vc3002_7

$$\begin{array}{l}
\forall\ GVoACCUMULATOR,\ GVoDISPLAY : TCoNUMBER; \\
\quad GVoLAST\_OP,\ O : TCoOPERATION \\
\mid true \\
\quad \land\ O\ eq\ TCoCHANGE\_SIGN = FALSE \\
\quad \land\ O\ eq\ TCoFACTORIAL = FALSE \\
\quad \land\ O\ eq\ TCoSQUARE\_ROOT = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoEQUALS = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoPLUS = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoMINUS = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoTIMES = TRUE \\
\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoACCUMULATOR * GVoDISPLAY,
\end{array}$$

$\quad\quad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoDISPLAY \mathrel{\widehat{=}} GVoACCUMULATOR * GVoDISPLAY,$
$\quad\quad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY,$
$\quad\quad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,\ GVoLAST\_OP \mathrel{\widehat{=}} O,$
$\quad\quad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O)$
$\quad \in DO\_OPERATION$

vc3002_8 $\forall\ GVoACCUMULATOR,\ GVoDISPLAY : TCoNUMBER;$
$\quad GVoLAST\_OP,\ O : TCoOPERATION$
$\mid true$
$\quad \land\ O\ eq\ TCoCHANGE\_SIGN = FALSE$
$\quad \land\ O\ eq\ TCoFACTORIAL = FALSE$
$\quad \land\ O\ eq\ TCoSQUARE\_ROOT = FALSE$
$\quad \land\ GVoLAST\_OP\ eq\ TCoEQUALS = FALSE$
$\quad \land\ GVoLAST\_OP\ eq\ TCoPLUS = FALSE$
$\quad \land\ GVoLAST\_OP\ eq\ TCoMINUS = FALSE$
$\quad \land\ GVoLAST\_OP\ eq\ TCoTIMES = FALSE$
$\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoDISPLAY \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY,$
$\quad\quad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,\ GVoLAST\_OP \mathrel{\widehat{=}} O,$
$\quad\quad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O)$
$\quad \in DO\_OPERATION$

A.11.5 Theorems

vc3002_1 $\vdash \forall\ GVoACCUMULATOR,\ GVoDISPLAY : TCoNUMBER;$
$\quad GVoLAST\_OP,\ O : TCoOPERATION$
$\mid true \land O\ eq\ TCoCHANGE\_SIGN = TRUE$
$\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoDISPLAY \mathrel{\widehat{=}} \sim GVoDISPLAY,$
$\quad\quad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY,$
$\quad\quad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,$
$\quad\quad GVoLAST\_OP \mathrel{\widehat{=}} GVoLAST\_OP,$
$\quad\quad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O)$
$\quad \in DO\_OPERATION$

vc3002_2 $\vdash \forall\ GVoACCUMULATOR,\ GVoDISPLAY : TCoNUMBER;$
$\quad GVoLAST\_OP,\ O : TCoOPERATION$
$\mid true$
$\quad \land\ O\ eq\ TCoCHANGE\_SIGN = FALSE$
$\quad \land\ O\ eq\ TCoFACTORIAL = TRUE$
$\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR,$
$\quad\quad GVoDISPLAY \mathrel{\widehat{=}} FACT\ GVoDISPLAY,$
$\quad\quad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY,$
$\quad\quad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,$
$\quad\quad GVoLAST\_OP \mathrel{\widehat{=}} GVoLAST\_OP,$
$\quad\quad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O)$
$\quad \in DO\_OPERATION$

vc3002_4 $\vdash \forall\ GVoACCUMULATOR,\ GVoDISPLAY : TCoNUMBER;$

$$\begin{array}{l}
\quad GVoLAST\_OP,\ O : TCoOPERATION \\
\mid\ true \\
\quad \land\ O\ eq\ TCoCHANGE\_SIGN = FALSE \\
\quad \land\ O\ eq\ TCoFACTORIAL = FALSE \\
\quad \land\ O\ eq\ TCoSQUARE\_ROOT = FALSE \\
\quad \land\ GVoLAST\_OP\ eq\ TCoEQUALS = TRUE \\
\bullet\ (GVoACCUMULATOR \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoACCUMULATOR_0 \mathrel{\widehat{=}} GVoACCUMULATOR, \\
\qquad GVoDISPLAY \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoDISPLAY_0 \mathrel{\widehat{=}} GVoDISPLAY, \\
\qquad GVoIN\_NUMBER \mathrel{\widehat{=}} FALSE,\ GVoLAST\_OP \mathrel{\widehat{=}} O, \\
\qquad GVoLAST\_OP_0 \mathrel{\widehat{=}} GVoLAST\_OP,\ O \mathrel{\widehat{=}} O) \\
\quad \in DO\_OPERATION
\end{array}$$

A.12 THE Z THEORY OPSoOPERATION_BUTTONoFACT'func

A.12.1 Parents

$OPSoOPERATION\_BUTTONoFACT'context$

A.12.2 Conjectures

vcOPSoOPERATION_BUTTONoFACT_1

$$\forall\ M : NATURAL \bullet M \geq 0 \land 1 = 1$$

vcOPSoOPERATION_BUTTONoFACT_2

$$\begin{array}{l}
\forall\ M,\ RESULT : NATURAL;\ FACT : NATURAL \to NATURAL \\
\quad \mid\ true \land RESULT = fact\ M \land FACT\ M = RESULT \\
\quad \bullet\ FACT\ M = fact\ M
\end{array}$$

vc1001_1

$$\begin{array}{l}
\forall\ M,\ RESULT : NATURAL \\
\quad \mid\ (M \geq 0 \land RESULT = 1) \land 2 \leq M \\
\quad \bullet\ 2 \geq 1 \land RESULT = fact\ (2 - 1)
\end{array}$$

vc1001_2

$$\begin{array}{l}
\forall\ M,\ RESULT : NATURAL \\
\quad \mid\ (M \geq 0 \land RESULT = 1) \land 2 > M \\
\quad \bullet\ RESULT = fact\ M
\end{array}$$

vc1001_3

$$\begin{array}{l}
\forall\ J : INTEGER;\ M,\ RESULT,\ RESULT_0 : NATURAL \\
\quad \mid\ (M \geq 0 \\
\qquad \land\ RESULT_0 = 1) \\
\qquad \land\ J \in 2\ ..\ M \\
\qquad \land\ J \neq M \\
\qquad \land\ RESULT = fact\ J \\
\quad \bullet\ J + 1 \geq 1 \land RESULT = fact\ (J + 1 - 1)
\end{array}$$

vc1001_4

$$\begin{array}{l}
\forall\ M,\ RESULT,\ RESULT_0 : NATURAL \\
\quad \mid\ (M \geq 0 \land RESULT_0 = 1) \land RESULT = fact\ M \\
\quad \bullet\ RESULT = fact\ M
\end{array}$$

vc1002_1

$$\begin{array}{l}
\forall\ J : INTEGER;\ RESULT : NATURAL \\
\quad \mid\ J \geq 1 \land RESULT = fact\ (J - 1) \\
\quad \bullet\ J * RESULT = fact\ J
\end{array}$$

A.12.3 Theorems

vcOPSoOPERATION_BUTTONoFACT_1

$\vdash \forall\ M : NATURAL \bullet M \geq 0 \land 1 = 1$

vcOPSoOPERATION_BUTTONoFACT_2

$\vdash \forall\ M, RESULT : NATURAL; FACT : NATURAL \rightarrow NATURAL$
$\mid true \land RESULT = fact\ M \land FACT\ M = RESULT$
$\bullet FACT\ M = fact\ M$

vc1001_1 $\vdash \forall\ M, RESULT : NATURAL$
$\mid (M \geq 0 \land RESULT = 1) \land 2 \leq M$
$\bullet 2 \geq 1 \land RESULT = fact\ (2 - 1)$

vc1001_2 $\vdash \forall\ M, RESULT : NATURAL$
$\mid (M \geq 0 \land RESULT = 1) \land 2 > M$
$\bullet RESULT = fact\ M$

A.13 THE Z THEORY OPSoOPERATION_BUTTONoSQRT'func

A.13.1 Parents

OPSoOPERATION_BUTTONoSQRT′context

A.13.2 Conjectures

vcOPSoOPERATION_BUTTONoSQRT_1

$true \Rightarrow 0 = 0$

vcOPSoOPERATION_BUTTONoSQRT_2

$\forall\ M, RESULT : NATURAL; SQRT : NATURAL \rightarrow NATURAL$
$\mid true$
$\land (RESULT ** 2 \leq M$
$\land M < (RESULT + 1) ** 2)$
$\land SQRT\ M = RESULT$
$\bullet SQRT\ M ** 2 \leq M \land M < (SQRT\ M + 1) ** 2$

vc2001_1 $\forall\ RESULT : NATURAL \mid RESULT = 0 \bullet RESULT = 0$

vc2001_2 $\forall\ M, RESULT, RESULT_0 : NATURAL$
$\mid RESULT_0 = 0$
$\land RESULT ** 2 \leq M$
$\land M < (RESULT + 1) ** 2$
$\bullet RESULT ** 2 \leq M \land M < (RESULT + 1) ** 2$

vc2002_1 $\forall\ M, RESULT : NATURAL$
$\mid RESULT = 0$
$\bullet RESULT ** 2 \leq M \land M < (M + 1) ** 2$

vc2002_2 $\forall\ HI : INTEGER; M, RESULT, RESULT_0 : NATURAL$
$\mid RESULT_0 = 0 \land RESULT ** 2 \leq M \land M < HI ** 2$
$\bullet RESULT ** 2 \leq M \land M < HI ** 2$

vc2002_3 $\forall\ M, RESULT, RESULT_0 : NATURAL$
$\mid RESULT_0 = 0$
$\land RESULT ** 2 \leq M$
$\land M < (RESULT + 1) ** 2$
$\bullet RESULT ** 2 \leq M \land M < (RESULT + 1) ** 2$

vc2003_1 $\forall\ HI : INTEGER; M, RESULT : NATURAL$

$$\begin{array}{l}
\quad \mid (RESULT ** 2 \leq M \\
\qquad \land M < HI ** 2) \\
\qquad \land RESULT + 1\ eq\ HI = TRUE \\
\quad \bullet RESULT ** 2 \leq M \land M < (RESULT + 1) ** 2
\end{array}$$

vc2003_2

$$\begin{array}{l}
\forall HI : INTEGER; M, RESULT : NATURAL \\
\quad \mid (RESULT ** 2 \leq M \\
\qquad \land M < HI ** 2) \\
\qquad \land RESULT + 1\ eq\ HI = FALSE \\
\quad \bullet RESULT ** 2 \leq M \land M < HI ** 2
\end{array}$$

vc2003_3

$$\begin{array}{l}
\forall HI, HI_0 : INTEGER; M, RESULT, RESULT_0 : NATURAL \\
\quad \mid (RESULT_0 ** 2 \leq M \\
\qquad \land M < HI_0 ** 2) \\
\qquad \land RESULT ** 2 \leq M \\
\qquad \land M < HI ** 2 \\
\quad \bullet RESULT ** 2 \leq M \land M < HI ** 2
\end{array}$$

vc2004_1

$$\begin{array}{l}
\forall HI : INTEGER; M, RESULT : NATURAL \\
\quad \mid (RESULT ** 2 \leq M \\
\qquad \land M < HI ** 2) \\
\qquad \land ((RESULT + HI + 1)\ intdiv\ 2) ** 2\ greater\ M \\
\qquad\quad = TRUE \\
\quad \bullet RESULT ** 2 \leq M \\
\qquad \land M < ((RESULT + HI + 1)\ intdiv\ 2) ** 2
\end{array}$$

vc2004_2

$$\begin{array}{l}
\forall HI : INTEGER; M, RESULT : NATURAL \\
\quad \mid (RESULT ** 2 \leq M \\
\qquad \land M < HI ** 2) \\
\qquad \land ((RESULT + HI + 1)\ intdiv\ 2) ** 2\ greater\ M \\
\qquad\quad = FALSE \\
\quad \bullet ((RESULT + HI + 1)\ intdiv\ 2) ** 2 \leq M \land M < HI ** 2
\end{array}$$

A.13.3 Theorems

vc2002_2

$$\begin{array}{l}
\vdash \forall HI : INTEGER; M, RESULT, RESULT_0 : NATURAL \\
\qquad \mid RESULT_0 = 0 \land RESULT ** 2 \leq M \land M < HI ** 2 \\
\qquad \bullet RESULT ** 2 \leq M \land M < HI ** 2
\end{array}$$

star_star_1_thm

$$\vdash \forall x : \mathbb{Z} \bullet x ** 1 = x$$

star_star_2_thm

$$\vdash \forall x : \mathbb{Z} \bullet x ** 2 = x * x$$

vc2002_1

$$\begin{array}{l}
\vdash \forall M, RESULT : NATURAL \\
\qquad \mid RESULT = 0 \\
\qquad \bullet RESULT ** 2 \leq M \land M < (M + 1) ** 2
\end{array}$$

CALCULATOR EXAMPLE ADA PROGRAM

```
package TC
is
  BASE : constant INTEGER := 10;
  PRECISION : constant INTEGER := 6;
  MAX_NUMBER : constant INTEGER := BASE ** PRECISION - 1;
  MIN_NUMBER : constant INTEGER := - MAX_NUMBER;
  subtype DIGIT is INTEGER range 0..BASE - 1;
  subtype NUMBER is INTEGER range MIN_NUMBER..MAX_NUMBER;
  type OPERATION is (PLUS, MINUS, TIMES, CHANGE_SIGN, SQUARE_ROOT, FACTORIAL,
  EQUALS);
end TC;

with TC;
package GV
is
  DISPLAY, ACCUMULATOR : TC.NUMBER;
  LAST_OP : TC.OPERATION;
  IN_NUMBER : BOOLEAN;
end GV;

with TC, GV;
package OPS
is
  procedure DIGIT_BUTTON (D : in TC.DIGIT);
    -- Spec ...
  procedure OPERATION_BUTTON (O : in TC.OPERATION);
    -- Spec ...
end OPS;
```

```
package body OPS
is
  procedure DIGIT_BUTTON (D : in TC.DIGIT)
    -- Spec ...
  is
  begin
    if GV.IN_NUMBER
    then
      GV.DISPLAY := GV.DISPLAY * TC.BASE + D;
    else
      GV.DISPLAY := D;
    end if;
    GV.IN_NUMBER := true;
  end DIGIT_BUTTON;
  procedure OPERATION_BUTTON (O : in TC.OPERATION)
    -- Spec ...
  is
    function FACT (M : NATURAL) return NATURAL
      -- Spec ...
    is
      RESULT : NATURAL;
    begin
      RESULT := 1;
      for J in INTEGER range 2..M
      loop
        RESULT := J * RESULT;
      end loop;
      return RESULT;
    end FACT;
    function SQRT (M : NATURAL) return NATURAL
      -- Spec ...
    is
      RESULT : NATURAL;
      MID, HI : INTEGER;
    begin
      RESULT := 0;
      HI := M + 1;
      -- $TILL ...
      loop
        exit when RESULT + 1 = HI;
        MID := (RESULT + HI + 1) / 2;
        if MID ** 2 > M
        then
          HI := MID;
        else
          RESULT := MID;
        end if;
      end loop;
      return RESULT;
    end SQRT;
  begin
    if O = TC.CHANGE_SIGN
    then
      GV.DISPLAY := - GV.DISPLAY;
    elsif O = TC.FACTORIAL
    then
      GV.DISPLAY := FACT (GV.DISPLAY);
    elsif O = TC.SQUARE_ROOT
    then
      GV.DISPLAY := SQRT (GV.DISPLAY);
    else
      if GV.LAST_OP = TC.EQUALS
      then
        GV.ACCUMULATOR := GV.DISPLAY;
      elsif GV.LAST_OP = TC.PLUS
      then
        GV.ACCUMULATOR := GV.ACCUMULATOR + GV.DISPLAY;
      elsif GV.LAST_OP = TC.MINUS
      then
        GV.ACCUMULATOR := GV.ACCUMULATOR - GV.DISPLAY;
      elsif GV.LAST_OP = TC.TIMES
      then
        GV.ACCUMULATOR := GV.ACCUMULATOR * GV.DISPLAY;
      end if;
```

```
      GV.DISPLAY := GV.ACCUMULATOR;
      GV.LAST_OP := O;
    end if;
    GV.IN_NUMBER := false;
  end OPERATION_BUTTON;
end OPS;
```

REFERENCES

[1] DS/FMU/IED/USR011. *ProofPower Z Tutorial.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[2] DS/FMU/IED/USR013. *ProofPower HOL Tutorial Notes.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[3] DS/FMU/IED/USR014. *ProofPower Software and Services.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[4] ISS/HAT/DAZ/USR501. *Compliance Tool — User Guide.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[5] ISS/HAT/DAZ/USR502. *Compliance Tool — Installation and Operation.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[6] ISS/HAT/DAZ/USR504. *Compliance Notation — Language Description.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[7] ISS/HAT/DAZ/WRK513. *Calculator Example VCs Proof Scripts.* R.D. Arthan and G.M. Prout, Lemma 1 Ltd., `http://www.lemma-one.com`.

[8] LEMMA1/HOL/USR029. *ProofPower HOL Reference Manual.* Lemma 1 Ltd., `rda@lemma-one.com`.

INDEX