Proof in Z

with

ProofPower

Course Objectives

- to describe the basic principles and concepts underlying ProofPower support for Z

- to enable the student to write simple specifications and undertake elementary proofs in Z using ProofPower

- to enable the student to make effective use of the reference documentation for ProofPower-Z

Course Outline

- Introduction to ProofPower-Z
- The Z Predicate Calculus
- Expressions
- Schema Expressions
- Paragraphs and Theories
- The Z ToolKit
- Case Study

Course Prerequisites

We assume a working knowledge of:

- Z as a specification language
- the use of ProofPower with HOL

SML
```
open_theory "z_library";
new_theory "usr023";
new_parent(hd (get_cache_theories()));
set_pc "z_library";
```

Z

$[NAME, DATE]$

Z

$\mathbb{U}[X] \mathrel{\widehat=} X$

Sample Schemas

Z

File

$people : \mathbb{P}\ NAME;$
$age : NAME \pfun DATE$

$dom\ age = people$

Z

File2

$people : \mathbb{P}\ NAME;$
$height : NAME \pfun \mathbb{Z}$

$dom\ height = people$

Z

File3

$people : \mathbb{P}\ NAME$

Z

FileOp

$File; File'; \ i?:\mathbb{N}$

Useful Files

- usr023.dvi - these transparencies for use with previewer.

- usr023_slides.doc - transparencies source file.

- zed_course_work.doc - exercise “work book”.

- zed_course_answers.doc - solutions to exercises.

- sun4example_zed.db - ProofPower database with material loaded in ready to do the exercises.

Reasoning in Z with ProofPower
Facilities 'lifted' from HOL

- Propositional Reasoning
- Predicate Calculus:
  - stripping
  - forward chaining
  - resolution (via $prove\_tac$)
- basic rewriting
- basic integer arithmetic
- arithmetic computations

Reasoning in Z
Areas for Future Enhancement

Function Application

‘set’ inference

Conditional Rewriting

Consistency Proofs

Performance Improvements

Ease of Unfolding Definitions

Methods which contain complexity

Some Z Proofs are Easy with ProofPower

- propositional tautologies

  Propositional reasoning in Z is exactly the same as in HOL, fully automatic and well integrated into the normal proof methods.

- first order predicate calculus

  As in HOL, predicate calculus proofs in Z are either automatic or routine.

- elementary set theory

  A useful class of results from elementary set theory are automatically provable.

- other classes of results

  Whenever a new theory is introduced one or more proof contexts may be developed to solve automatically a range of results in that theory. "Decision procedures" for such classes of results can be made available via "prove_tac".

Simple Predicate Calculus Proofs

- use the subgoaling package
- set the goal

SML

```
open_theory"usr023";
set_pc "z_library";
set_goal([],ⓩ (∀x, y:X• P x ⇒ R y)
              ⇔ (∀v, w:X• ¬ P w ∨ R v)⌝);
```

- initiate proof by contradiction

SML

```
a contr_tac;
```

ProofPower output

```
Tactic produced 2 subgoals:
...
(* 5 *) ⓩ∀ x, y : X • P x ⇒ R y⌝
(* 4 *) ⓩv ∈ X⌝
(* 3 *) ⓩw ∈ X⌝
(* 2 *) ⓩP w⌝
(* 1 *) ⓩ¬ R v⌝

(* ?⊢ *) ⓩfalse⌝
```

- instantiate assumptions as required

SML

```
a (z_spec_asm_tac ⓩ∀ x, y : X • P x ⇒ R y⌝
              ⓩ(x ≙ w, y ≙ v)⌝);
```


ProofPower output

```
Tactic produced 0 subgoals:
(* *** Goal "2" *** *)

(*  5 *)  ⓩ∀ v, w : X • ¬ P w ∨ R v⌝
(*  4 *)  ⓩx ∈ X⌝
(*  3 *)  ⓩy ∈ X⌝
(*  2 *)  ⓩP x⌝
(*  1 *)  ⓩ¬ R y⌝

(* ?⊢ *)  ⓩfalse⌝
```

SML

```
a (z_spec_asm_tac ⓩ∀ v, w : X • ¬ P w ∨ R v⌝
                ⓩ(v ≜ y, w ≜ x)⌝);
```

ProofPower output

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

SML

```
pop_thm();
```

ProofPower output

```
Now 0 goals on the main goal stack
val it = ⊢ (∀ x, y : X • P x ⇒ R y) ⇔
             (∀ v, w : X • ¬ P w ∨ R v) : THM
```


Exercises: 1

Log in;

start Motif Window manager (using `openwin` command);

Select "Z Course" from the Root Menu;

Find Exercises 1 in zed_course_work.doc;

Execute the preliminary commands just before the Exercises;

Work through the the exercises recording your solutions in zed_course_work.doc.

The Z Language in ProofPower

- HOL terms are used to represent Z.

- The "concrete datatype" **Z_TERM** reveals the structure of terms representing values in Z.

- The function:

  SML

  ```
  dest_z_term : TERM -> Z_TERM;
  ```

  may be used to disassemble a TERM which represents Z, and

  SML

  ```
  mk_z_term : Z_TERM -> TERM;
  ```

  may be used to construct a TERM representing a Z construct.

Z Language Quotation

- ## Z Term Quotations

  Predicates, expressions, and schema expressions may be entered in Z using the Z quotation character "$\ulcorner_Z$", e.g.: $\ulcorner_Z\{x{:}\mathbb{Z} \mid x>0 \bullet x*x\}\urcorner$.

- ## Extended Z

  ProofPower accepts an extended Z language for convenience in formal proof, provided that the system control flag $standard\_z\_terms$ is set to $false$.

- ## Standard Z

  Eventually we intend ProofPower to be prepared to check fully against the forthcoming Z standard.

  The norm would then be to check specifications against the standard, but permit the extended language for use in proofs.

Special Extensions

- $\mathbb{U}$

  z

  $\mathbb{U}[X] \mathrel{\widehat{=}} X$

  may be used to avoid explicit typing, or to ensure quantification over entire types rather than sets.

- $\overset{\oplus}{\oplus}$, which type checks like $\in$ (and means the same thing). When used infix $\overset{\oplus}{\oplus}$ and its right hand operand are discarded. It may therefore be used to force the type of an expression without otherwise changing its value.

- Π which take a single operand and creates a context in which a predicate is required. Π is discarded after parsing and type-checking.

The Z Language in ProofPower
declarations

```
datatype     Z_TERM =

  ZLVar     (* local variable ⌜Z x⌝ *)
                     of string      (* variable name *)
                     * TYPE         (* HOL type of variable *)
                     * TERM list              (* generic parameters *)
| ZGVar     (* global variable ⌜Z 𝕌[DATE]⌝ *)
                     of string      (* variable name *)
                     * TYPE         (* HOL type of variable *)
                     * TERM list              (* generic parameters *)

| ZInt      (* positive integer literal ⌜Z 34⌝ *)
                     of string
| ZString   (* string literal ⌜Z "characters"⌝ *)
                     of string

| ZDec                   (* declaration, e.g.
                                    ⌜ML dec_of ⌜Z[x,y:ℤ]⌝⌝ *)
                     of TERM list             (* variables *)
                     * TERM                   (* expression *)
| ZSchemaDec             (* schema reference, e.g.
                                    ⌜ML dec_of ⌜Z[File!]⌝⌝ *)
                     of TERM  (*schema expression*)
                     * string                 (* decoration *)
| ZDecl     (* declaration list, e.g.
                                    ⌜ML decl_of ⌜Z[x,y:ℤ; File!]⌝⌝ *)
                     of TERM list             (* declarations *)
```

Local Variables

Used in variable binding constructs (e.g. quantifiers)

Free variables used in proofs of universal assertions, or in using existential assumptions (by 'skolemisation').

ProofPower allows 'generic' local variables.

Global Variables (i.e. constants)

These are introduced and constrained by various paragraphs.

Subsequent reasoning relies upon utilisation of predicates explicit or implicit in defining paragraph (see later).

Integer Literals

Evaluation of arithmetic expressions involving Integer Literals is built into appropriate proof contexts.

SML

$rewrite\_conv$ [] ⓩ$543*20$⌝;

ProofPower output

$val\ it = \vdash 543 * 20 = 10860 : THM$

String Literals

These are supported by the conversion z_string_conv which converts a string literal into a sequence of HOL character literals:

SML

z_string_conv ⓩ"$string$"⌝;

ProofPower output

$val\ it = \vdash$ "$string$" =

$\langle$⌜‘s‘⌝, ⌜‘t‘⌝, ⌜‘r‘⌝, ⌜‘i‘⌝, ⌜‘n‘⌝, ⌜‘g‘⌝$\rangle : THM$

Declarations

Conversion $z\_dec\_pred\_conv$ converts a declaration into its implicit predicate:

SML

$$val\ pred2 = z\_dec\_pred\_conv$$
$$(dec\_of \text{ⓩ}[x,\ y : \mathbb{Z}]\urcorner);$$

ProofPower output

$$val\ pred2 = \vdash {}_{\mathsf{ML}}\ulcorner dec\_of \text{ⓩ}[x,\ y : \mathbb{Z}]\urcorner\urcorner$$
$$\Leftrightarrow \{x,\ y\} \subseteq \mathbb{Z} : THM$$

Declaration Lists

Conversion $z\_decl\_pred\_conv$ converts a declaration list into its implicit predicate:

SML

$$val\ pred4 = z\_decl\_pred\_conv$$
$$(decl\_of \text{ⓩ}[x,\ y : \mathbb{Z};\ File!]\urcorner);$$

ProofPower output

$$val\ pred4 = \vdash {}_{\mathsf{ML}}\ulcorner decl\_of \text{ⓩ}[x,\ y : \mathbb{Z};\ File!]\urcorner\urcorner$$
$$\Leftrightarrow \{x,\ y\} \subseteq \mathbb{Z} \land (File!) : THM$$

The Z Language in ProofPower
propositional connectives

```
| ZTrue     (* ⌜Z true⌝ *)

| ZFalse    (* ⌜Z false⌝ *)

| Z¬        (* negation, e.g. ⌜Z ¬ p⌝ *)
            of TERM  (* predicate *)

| Z∧        (* conjunction, e.g. ⌜Z p ∧ q⌝ *)
            of TERM * TERM     (* predicates *)

| Z∨        (* disjunction, e.g. ⌜Z p ∨ q⌝ *)
            of TERM * TERM     (* predicates *)

| Z⇒        (* implication, e.g. ⌜Z p ⇒ q⌝ *)
            of TERM * TERM     (* predicates *)

| Z⇔        (* bi−implication, e.g. ⌜Z p ⇔ q⌝ *)
            of TERM * TERM     (* predicates *)
```


Propositional Reasoning

- assume rule:

  SML

  $open\_theory$ "$usr023$";

  $val\ thm1 = asm\_rule\ ⓩ \forall x,\ y{:}\mathbb{N} \bullet\ x{*}y > 0 ⌝$;

  ProofPower Output

  $val\ thm1 = \forall\ x,\ y : \mathbb{N} \bullet x * y > 0$

  $\vdash \forall\ x,\ y : \mathbb{N} \bullet x * y > 0 : THM$

- modus ponens

  SML

  $val\ thm\_a = asm\_rule\ ⓩ\ a : \mathbb{B} ⌝$;

  $val\ thm\_b = asm\_rule\ ⓩ\ a \Rightarrow b ⌝$;

  ProofPower Output

  $val\ thm\_a = a \vdash a : THM$

  $val\ thm\_b = a \Rightarrow b \vdash a \Rightarrow b : THM$

  SML

  $val\ thm\_c = \Rightarrow\_elim\ thm\_b\ thm\_a$;

  ProofPower Output

  $val\ thm\_c = a \Rightarrow b,\ a \vdash b : THM$

The Z Language in ProofPower
quantifiers and relations

```
| ZEq        (* equation, e.g. ⌜Z a = b⌝ *)
                       of TERM * TERM         (* expressions *)

| Z∈         (* membership, e.g. ⌜Z a ∈ b⌝ *)
                       of TERM * TERM         (* expressions *)

| ZSchemaPred          (* schema predicate, e.g.
                              ⌜Z Π (File ')⌝ *)
                       of TERM                (* schema expression *)
                       * string               (* decoration *)

| Z∃         (* existential quantification, ⌜Z ∃ File | p • q⌝ *)
                       of TERM                (* declaration *)
                       * TERM * TERM          (* predicates *)

| Z∃₁        (* unique existential quantification, ⌜Z ∃₁ File | p • q⌝ *)
                       of TERM                (* declaration *)
                       * TERM * TERM          (* predicates *)

| Z∀         (* universal quantification, ⌜Z ∀ File | p • q⌝ *)
                       of TERM                (* declaration *)
                       * TERM * TERM          (* predicates *)
```


Schema Predicates

These are be eliminated in favour of membership statements when rewriting with $z\_library$:

SML

$once\_rewrite\_conv[] ⓩ Π(([x:X])')⌝;$

ProofPower outputval

$val\ it = \vdash (([x : X])') \Leftrightarrow$
$\quad (x \mathrel{\widehat{=}} x') \in [x : X] : THM$

The proof context $z\_library$ which will also eliminate the resulting horizontal schema.

SML

$rewrite\_conv[] ⓩ Π(([x:X])')⌝;$

ProofPower outputval

$val\ it = \vdash (([x : X])') \Leftrightarrow$
$\quad x' \in X : THM$

Reasoning with Quantifiers Specialisation (I)

- most commonly a binding display is used

SML

$z\_\forall\_elim$ ⓩ $(x \widehat{=} 455,\ y \widehat{=} 32)$⌝ $thm1$;

ProofPower Output

$val\ it = \forall\ x,\ y : \mathbb{N} \bullet x * y > 0$
$\vdash \{455,\ 32\} \subseteq \mathbb{N} \wedge true \Rightarrow$
$455 * 32 > 0 : THM$

- any binding expression is acceptable

SML

$z\_\forall\_elim$ ⓩ $exp^{\oplus}_{\oplus}[x,y{:}\mathbb{N}]$⌝ $thm1$;

ProofPower Output

$val\ it = \forall\ x,\ y : \mathbb{N} \bullet x * y > 0$
$\vdash \{exp.x,\ exp.y\} \subseteq \mathbb{N} \wedge true$
$\Rightarrow exp.x * exp.y > 0 : THM$

The signatures of the bindings must match the signature of the declaration exactly.

Reasoning with Quantifiers Specialisation (II)

- where the signature of the declaration contains only a single name an expression which has the same type as that name may be offered:

SML

$z\_\forall\_elim$ ⓩ 45⌝ $z\_\mathbb{N}\_\neg\_plus1\_thm$;

ProofPower Output

$val\ it = \vdash 45 \in \mathbb{N} \land true$
$\Rightarrow \neg\ 45 + 1 = 0 : THM$

Goal Oriented Proof

- Works exactly the same as for HOL.
- Make sure you are in a Z theory.
- Make sure you have a Z proof context.
- Terms should be entered using Z quotes $\ulcorner_Z$ $\urcorner$.

Tactics for Quantifiers

- z_strip_tac:
  - eliminates outer universals in conclusions
  - skolemises existential assumptions
  - pushes in outer negations over universal conclusions
  - pushes in outer negations over existential assumptions
- $z\_spec\_nth\_asm\_tac$:

  specialises universal assumptions
- $z\_\exists\_tac$

  eliminates existential conclusions

Rewriting

Use same facilities as for HOL in appropriate proof contexts.

Most rewrites arising from axiomatic descriptions are effectively conditional, and the conditions must be discharged to achieve the rewrite.

Forward chaining is often an appropriate way to achieve such conditional rewriting.

Chaining

In appropriate proof contexts forward chaining facilities with *all* in name work and stay in Z. Other variants are liable to introduce hol universals.

Rewriting by Chaining - example

z_abs_thm is :

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

Which, because quantified over $\mathbb{N}$, is effectively a *conditional* rewrite.

The proof of:

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

is therefore complicated by the need to establish the necessary conditions for rewriting with z_abs_thm.

First we strip the goal:

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

ProofPower output
```
(*  1 *)  ⓩ0 ≤ a⌝

(* ?⊢ *)  ⓩabs a * abs ~ a = a * a⌝
```

Which places the necessary information in the assumptions.

Rewriting by Chaining - example continued

Then we use forward chaining to establish unconditional equations:

SML

$|\ a\ (all\_fc\_tac\ [z\_abs\_thm]);$

ProofPower output

$|\ (*\ \ 3\ *)\ \ ⌜0 \le a⌝$
$|\ (*\ \ 2\ *)\ \ ⌜abs\ a = a⌝$
$|\ (*\ \ 1\ *)\ \ ⌜abs \sim a = a⌝$
$|$
$|\ (*\ ?\vdash\ *)\ \ ⌜abs\ a * abs \sim a = a * a⌝$

Then rewrite with these equations:

SML

$|\ a\ (asm\_rewrite\_tac[]);$
$|\ pop\_thm();$

(which solves the goal)

Exercises 2: Predicate Calculus

Try Exercises 2 in zed_course_work.doc.

Hints and further exercises may be found in section 7.1 of the Z Tutorial Manual.

The Z Language in ProofPower expressions

```
| ZApp    (* function application ⌜Z f x⌝ *)
                of TERM * TERM        (* expressions *)
| Zλ      (* lambda expression ⌜Z λ x:ℕ | x > 3 • x * x ⌝ *)
                of TERM   (* declaration *)
                * TERM    (* predicate *)
                * TERM    (* expression *)
| Zμ      (* definite description ⌜Z μ x:ℕ | x * x = 4 • x⌝ *)
                of TERM   (* declaration *)
                * TERM    (* predicate *)
                * TERM    (* expression *)
| ZLet    (* let expression ⌜Z let x ≙ 9 • (x, x+x) ⌝ *)
                of (string * TERM) list              (* local definitions *
                * TERM    (* expression *)
| Zℙ      (* power set construction, ⌜Z ℙ ℤ⌝ *)
          of TERM                  (* expression *)
| ZSetd   (* set display, ⌜Z {1,2,3,4} ⌝ *)
          of TYPE                  (* HOL type of elements *)
          * TERM list              (* expressions *)
| ZSeta   (* set abstraction, ⌜Z {x:ℤ | 1≤x≤4 • x*x} ⌝*)
          of TERM   (* declaration *)
          * TERM    (* predicate *)
          * TERM    (* expression *)
```

The Z Language in ProofPower
expressions (continued)

```
| ZTuple   (* tuple displays, ⌜Z (1,2,3,4) ⌝ *)
           of TERM list            (* expressions *)
| ZSel_t   (* tuple element selection, ⌜Z (x,y).2 ⌝ *)
           of TERM  (* expression *)
           * int                   (* element number *)
| Z×       (* cartesian product, ⌜Z (ℤ × ℕ) ⌝ *)
           of TERM list            (* expressions *)
| ZBinding (* binding displays ⌜Z (people ≙ {}, age ≙ {}) ⌝ *)
           of (       string       (* component name *)
                      * TERM       (* component value *)
                      ) list
| Zθ       (* theta term ⌜Z θFile′ ⌝ *)
           of TERM  (* schema expression *)
           * string (* decoration *)
| ZSel_s   (* binding component selection ⌜Z (a ≙ 1, b ≙ "4").b ⌝ *)
           of TERM  (* expression *)
           * string (* component name *)
| Z_s      (* horizontal schema expression
                      ⌜Z [x:ℤ | x>0] ⌝ *)
                      of TERM      (* declaration *)
                      * TERM       (* predicate *)
| Z⟨⟩      (* sequence display ⌜Z ⟨1,2,3⟩ ⌝ *)
           of TYPE  (* type of elements *)
           * TERM list             (* values of elements *)
```


Function Application (I)

Applications of lambda abstractions can be eliminated by (conditional) β-conversion.

SML

z_β_conv ⓩ $(\lambda\ x{:}X \mid P\ x \bullet f\ x)\ a$⌝;

ProofPower outputval

$val\ it = P\ a,\ a \in X \vdash$
$\qquad (\lambda\ x : X \mid P\ x \bullet f\ x)\ a = f\ a : THM$

Other applications may be eliminated in favour of definite descriptions.

SML

z_app_conv ⓩ $f\ a$⌝;

ProofPower output

$val\ it = \vdash f\ a = \mu\ f\_a : \mathbb{U}$
$\qquad \mid (a, f\_a) \in f \bullet f\_a : THM$

More commonly function applications will be eliminated by rewriting with their definitions.

Function Application (II)

For low level reasoning $z\_app\_eq\_tac$ is useful:

SML

$set\_goal([],⌜_Z\ f\ \ a = v⌝);$

$a\ \ z\_app\_eq\_tac;$

ProofPower output

...

$(*\ ?\vdash\ *)\ ⌜_Z(\forall\ f\_a : \mathbb{U} \mid (a,\ f\_a) \in f \bullet f\_a = v)$

$\quad \land\ (a,\ v) \in f⌝$

...

Here the first conjunct expresses the requirement that f is functional at a.

If f is known to be a function this fact may be used more directly with the assistance of the theorem $z\_fun\_app\_clauses$:

$val\ \ z\_fun\_app\_clauses =$

$\quad \vdash \forall\ f : \mathbb{U};\ x : \mathbb{U};\ y : \mathbb{U};\ X : \mathbb{U};\ Y : \mathbb{U}$

$\qquad \bullet (f \in X \nrightarrow Y$

$\qquad\qquad \lor f \in X \rightarrowtail\!\!\!\!\!\!+ Y$

$\qquad\qquad \lor f \in X \twoheadrightarrow\!\!\!\!\!\!+ Y$

$\qquad\qquad \lor f \in X \rightarrow Y$

$\qquad\qquad \lor f \in X \rightarrowtail Y$

$\qquad\qquad \lor f \in X \twoheadrightarrow Y$

$\qquad\qquad \lor f \in X \rightarrowtail\!\!\!\!\!\twoheadrightarrow Y)$

$\qquad\quad \land (x,\ y) \in f$

$\qquad \Rightarrow f\ \ x = y : THM$

Which is most conveniently appplied using forward chaining.

Function Application (III)

SML

```
drop_main_goal();
set_goal([], ⌜f ∈ ℕ ↠ ℤ ⇒
    (4, ~45) ∈ f ⇒ f 4 = ~45⌝);
a (REPEAT z_strip_tac);
```

ProofPower output

```
(* 2 *) ⌜f ∈ ℕ ↠ ℤ⌝
(* 1 *) ⌜(4, ~ 45) ∈ f⌝

(* ?⊢ *) ⌜f 4 = ~ 45⌝
```

SML

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

ProofPower output

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

Often it is necessary to establish that a function application is a member of a set.

The theorem *z_fun_∈_clauses* is of assistance in such cases:

```
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
```


Function Application (IV)

This too is best applied using forward chaining:

SML

```
set_goal([],
 ⓩ[X](∀ b: bag X• count[X] b ∈ X → ℕ)⌝);
a (REPEAT z_strip_tac);
```

ProofPower output

```
(*  1 *)  ⓩb ∈ bag X⌝

(* ?⊢ *)  ⓩcount[X] b ∈ X → ℕ⌝
```

We need the information from the declaration of *count*:

SML

```
a (strip_asm_tac (z_gen_pred_elim
    [ⓩX⌝] (z_get_spec ⓩcount⌝)));
```

ProofPower output

```
(*  3 *)  ⓩb ∈ bag X⌝
(*  2 *)  ⓩcount[X] ∈ bag X ⤖ X → ℕ⌝
...
(* ?⊢ *)  ⓩcount[X] b ∈ X → ℕ⌝
```

Now we can forward chain:

SML

```
a (all_fc_tac [z_fun_∈_clauses]);
val bag_lemma1 = pop_thm ();
```

ProofPower output

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


Lambda Abstraction

For extensional reasoning:

SML

$rewrite\_conv$ [] $ⓩz \in (\lambda\ x{:}X \mid P\ x \bullet f\ x)⌝$;

ProofPower outputval

$val\ it = \vdash z \in \lambda\ x : X \mid P\ x \bullet f\ x \Leftrightarrow$
$z.1 \in X \land P\ z.1 \land f\ z.1 = z.2 : THM$

Lambda abstractions may be transformed into set abstractions.

SML

z_λ_conv $ⓩ \lambda\ x{:}X \mid P\ x \bullet f\ x⌝$;

ProofPower outputval

$val\ it = \vdash \lambda\ x : X \mid P\ x \bullet f\ x =$
$\{x : X \mid P\ x \bullet (x, f\ x)\} : THM$

Definite Description

SML

z_μ_rule ⌜Z $\mu\ x{:}X \mid P \bullet y$ ⌝;

ProofPower output

$val\ it = \vdash \forall\ x' : \mathbb{U}$
$\quad \bullet (\forall\ x : X \mid P \bullet y = x')$
$\quad \wedge (\exists\ x : X \mid P \bullet y = x')$
$\quad\quad \Rightarrow (\mu\ x : X \mid P \bullet y) = x' : THM$

Let Expression

SML

z_let_conv ⌜Z $let\ x \mathrel{\widehat=} 9 \bullet (x,\ x + x)$ ⌝;

ProofPower output

$val\ it = \vdash$
$\qquad (let\ x \mathrel{\widehat=} 9 \bullet (x,\ x + x)) = (9,\ 9 + 9) : THM$

The Power Set Constructor

SML

$z\_\in\_\mathbb{P}\_conv$ ⓩ $z \in \mathbb{P}\ y$⌝;

ProofPower output

$val\ it = \vdash z \in \mathbb{P}\ y \Leftrightarrow$
$(\forall\ x1 : \mathbb{U} \bullet x1 \in z \Rightarrow x1 \in y) : THM$

SML

$rewrite\_conv$[] ⓩ $z \in \mathbb{P}\ y$⌝;

ProofPower output

$val\ it = \vdash z \in \mathbb{P}\ y \Leftrightarrow z \subseteq y : THM$

SML

$rewrite\_conv[z\_\subseteq\_thm]$ ⓩ $z \in \mathbb{P}\ y$⌝;

ProofPower output

$val\ it = \vdash z \in \mathbb{P}\ y$
$\Leftrightarrow (\forall\ x : \mathbb{U} \bullet x \in z \Rightarrow x \in y) : THM$

Set Displays

- sets may be entered as terms by enumeration:

SML

$rewrite\_conv$ [] ⓩ $5 \in \{1,2,3,4,5\}$⌝;

ProofPower output

$val\ it = \vdash 5 \in \{1,\ 2,\ 3,\ 4,\ 5\} \Leftrightarrow true : THM$

SML

$rewrite\_conv$ [] ⓩ $x \in \{1,2,3,4,5\}$⌝;

ProofPower output

$val\ it = \vdash x \in \{1,\ 2,\ 3,\ 4,\ 5\} \Leftrightarrow$
$x = 1 \lor x = 2 \lor x = 3 \lor x = 4 \lor x = 5 : THM$

Set Abstractions

- sets may also be entered as set abstractions:

SML

$rewrite\_conv[]\ ⓩ\ 9 \in \{x{:}\mathbb{N} \mid x < 12\}⌝;$

ProofPower output

$val\ it =$
$\vdash 9 \in \{x : \mathbb{N} \mid x < 12\} \Leftrightarrow 9 \in \mathbb{N} \land 9 < 12 : THM$

SML

$rewrite\_conv[]ⓩ\ z \in \{x, y{:}\mathbb{N} \mid x < y\}⌝;$

ProofPower Output

$val\ it = \vdash z \in \{x, y : \mathbb{N} \mid x < y\}$
$\Leftrightarrow \{z.1, z.2\} \subseteq \mathbb{N} \land z.1 < z.2 : THM$

SML

$rewrite\_conv[]ⓩ\ z \in \{x, y{:}\mathbb{N} \mid x < y \bullet x * y - x\}⌝;$

ProofPower Output

$val\ it = \vdash z \in \{x, y : \mathbb{N} \mid x < y \bullet x * y - x\}$
$\Leftrightarrow (\exists x, y : \mathbb{N} \mid x < y \bullet x * y - x = z) : THM$

Tuples

SML

$rewrite\_conv[]\ ⓩ(x,y) = (a,b)⌝;$

ProofPower output

$val\ it = \vdash (x,\ y) = (a,\ b)$
$\quad\quad \Leftrightarrow x = a \land y = b : THM$

SML

$rewrite\_conv[]\ ⓩ(x,y).1⌝;$

ProofPower output

$val\ it = \vdash (x,\ y).1 = x : THM$

Bindings

SML

$rewrite\_conv[]$
$\quad\quad ⓩ(x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b) = (y \mathrel{\widehat{=}} d,\ x \mathrel{\widehat{=}} c)⌝;$

ProofPower output

$val\ it = \vdash (x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b) = (x \mathrel{\widehat{=}} c,\ y \mathrel{\widehat{=}} d)$
$\quad\quad \Leftrightarrow a = c \land b = d : THM$

SML

$rewrite\_conv[]\ ⓩ(x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b).y⌝;$

ProofPower output

$val\ it = \vdash (x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b).y = b : THM$

Cartesian Products

SML

$rewrite\_conv[]\ ⓩ\ (a,\ b) \in (x \times y)⌝;$

ProofPower output

$val\ it = \vdash (a,\ b) \in x \times y$
$\quad \Leftrightarrow a \in x \land b \in y : THM$

SML

$rewrite\_conv[z\_sets\_ext\_thm]$
$\quad ⓩ\ (x \times y) = (a \times b)⌝;$

ProofPower output

$it = \vdash x \times y = a \times b$
$\quad \Leftrightarrow \quad (\forall z : \mathbb{U} \bullet z.1 \in x \land z.2 \in y$
$\quad\quad \Leftrightarrow z.1 \in a \land z.2 \in b) : THM$

Theta Terms

SML

$z\_\theta\_conv\ ⓩ\ \theta File'⌝;$

ProofPower output

$val\ it = \vdash \theta File' =$
$\quad (age \mathrel{\widehat{=}} age',\ people \mathrel{\widehat{=}} people') : THM$

SML

$rewrite\_conv[z'\theta\_def]\ ⓩ\ \theta File'⌝;$

ProofPower output

$val\ it = \vdash \theta File' =$
$\quad (age \mathrel{\widehat{=}} age',\ people \mathrel{\widehat{=}} people') : THM$

Binding Component Selection

Projection from binding displays is built in to proof context $z\_language$.

SML

$rewrite\_conv[]\ ⓩ\ (x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b).y⌝;$

ProofPower output

$val\ it = \vdash (x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b).y = b : THM$

Projection from theta terms is also built in to proof context $z\_language$.

SML

$rewrite\_conv[]\ ⓩ\ (\theta File').age⌝;$

ProofPower output

$val\ it = \vdash (\theta File').age = age' : THM$

Horizontal Schemas

SML

$rewrite\_conv[]ⓩz \in [x{:}\mathbb{Z};y{:}\mathbb{N}]⌝;$

ProofPower output

$val\ it = \vdash z \in [x : \mathbb{Z};\ y : \mathbb{N}]$
$\quad\Leftrightarrow z.x \in \mathbb{Z} \land z.y \in \mathbb{N} : THM$

SML

$rewrite\_conv[]ⓩ(x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b) \in [x{:}\mathbb{Z};y{:}\mathbb{N}]⌝;$

ProofPower output

$val\ it = \vdash (x \mathrel{\widehat{=}} a,\ y \mathrel{\widehat{=}} b) \in [x : \mathbb{Z};\ y : \mathbb{N}]$
$\quad\Leftrightarrow a \in \mathbb{Z} \land b \in \mathbb{N} : THM$

Sequence Displays

SML

$z\_\langle\rangle\_conv\ ⓩ\langle a,b,c\rangle⌝;$

ProofPower output

$val\ it = \vdash \langle a,\ b,\ c\rangle = \{(1,\ a),\ (2,\ b),\ (3,\ c)\} : THM$

SML

$once\_rewrite\_conv[]ⓩz \in \langle a,b,c\rangle⌝;$

ProofPower output

$val\ it = \vdash z \in \langle a,\ b,\ c\rangle \Leftrightarrow$
$\quad z \in \{(1,\ a),\ (2,\ b),\ (3,\ c)\} : THM$

Exercises 3: Expressions

Try Exercises 3 in zed_course_work.doc.

Hints and further exercises may be found in section 7.2.1 of the Z Tutorial Manual.

The Z Language in ProofPower
schema expressions (I)

```
| Z¬s        (* schema negation ⓩ(¬ File)⊕U⌝ *)
                      of TERM (* schema expression *)
| Z∧s        (* schema conjunction ⓩ(File ∧ File2)⊕U⌝ *)
                      of TERM * TERM (* schema expressions *)
| Z∨s        (* schema disjunction ⓩ(File ∨ File2)⊕U⌝ *)
                      of TERM * TERM (* schema expressions *)
| Z⇒s        (* schema implication ⓩ(File ⇒ File2)⊕U⌝ *)
                      of TERM * TERM (* schema expressions *)
| Z⇔s        (* schema equivalence ⓩ(File ⇔ File2)⊕U⌝ *)
                      of TERM * TERM (* schema expressions *)
| Z∃s        (* schema existential
                      ⓩ(∃ File3 | people = {} • File2)⊕U⌝ *)
                      of TERM   (* declaration *)
                      * TERM    (* predicate *)
                      * TERM    (* schema expression *)
| Z∃1s       (* schema unique existential
                      ⓩ(∃1 File3 | people = {} • File2)⊕U⌝ *)
                      of TERM   (* declaration *)
                      * TERM    (* predicate *)
                      * TERM    (* schema expression *)
| Z∀s        (* schema universal
                      ⓩ(∀ File3 | people = {} • File2)⊕U⌝ *)
                      of TERM   (* declaration *)
                      * TERM    (* predicate *)
                      * TERM    (* schema expression *)
```


The Z Language in ProofPower schema expressions (II)

```
| ZDecor_s (* decoration ⌜Z File ″⌝ *)
                    of TERM   (* schema expression *)
                    * string  (* decoration *)
| ZPre_s    (* pre−condition ⌜Z pre FileOp⌝ *)
                    of TERM   (* schema expression *)
| ZHide_s   (* schema hiding ⌜Z FileOp \_s (age, i?)⌝ *)
                    of TERM   (* schema expression *)
                    * string list (* component names *)
| ZRename_s (* schema renaming
                          ⌜Z File [aged/age, input/i?]⌝ *)
                    of TERM   (* schema expression *)
                    * (string * string) list  (* rename list *)
| Z⌈_s      (* schema projection ⌜Z FileOp ⌈_s File⌝ *)
                    of TERM * TERM    (* schema expressions *)
| Z⨾_s      (* schema composition ⌜Z ΔFile ⨾_s ΔFile⌝ *)
                    of TERM * TERM    (* schema expressions *)
| ZΔ_s      (* delta operation ⌜Z ΔFile⌝ *)
                    of TERM   (* schema expression *)
| ZΞ_s      (* Ξ operation ⌜Z ΞFile⌝ *)
                    of TERM   (* schema expression *)
;
```


Schema Negation

Return to theory where we defined schema $File$:

SML

```
open_theory "usr023";
set_pc "z_language";
```

SML

```
rewrite_conv[]⌜z ∈ (¬ File)⌝;
```

ProofPower output

```
val it = ⊢ z ∈ (¬ File) ⇔ ¬ z ∈ File : THM
```

Schema Conjunction

SML

```
rewrite_conv[]⌜z ∈ (File ∧ File2)⌝;
```

ProofPower output

```
val it = ⊢ z ∈ (File ∧ File2) ⇔
 (age ≙ z.age, people ≙ z.people) ∈ File ∧
 (height ≙ z.height, people ≙ z.people) ∈ File2 : THM
```


Schema Disjunction

SML

$rewrite\_conv[]ⓩz \in (File \lor File2)⌝;$

ProofPower output

$val\ it = \vdash z \in (File \lor File2) \Leftrightarrow$
$(age \mathrel{\widehat{=}} z.age, people \mathrel{\widehat{=}} z.people) \in File \lor$
$(height \mathrel{\widehat{=}} z.height, people \mathrel{\widehat{=}} z.people) \in File2 : THM$

Schema Implication

SML

$rewrite\_conv[]ⓩz \in (File \Rightarrow File2)⌝;$

ProofPower output

$val\ it = \vdash z \in (File \Rightarrow File2) \Leftrightarrow$
$(age \mathrel{\widehat{=}} z.age, people \mathrel{\widehat{=}} z.people) \in File \Rightarrow$
$(height \mathrel{\widehat{=}} z.height, people \mathrel{\widehat{=}} z.people) \in File2 : THM$

Schema Equivalence

SML

$$rewrite\_conv[]ⓩz \in (File \Leftrightarrow File2)⌝;$$

ProofPower output

$$val\ it = \vdash z \in (File \Leftrightarrow File2) \Leftrightarrow$$
$$(age \mathrel{\widehat{=}} z.age,\ people \mathrel{\widehat{=}} z.people) \in File \Leftrightarrow$$
$$(height \mathrel{\widehat{=}} z.height,\ people \mathrel{\widehat{=}} z.people) \in File2 : THM$$

Schema Existential

SML

$$rewrite\_conv[]ⓩz \in (\exists\ File3 \mid people = \{\} \bullet File2)⌝;$$

ProofPower output

$$val\ it = \vdash z \in (\exists\ File3 \mid people = \{\} \bullet File2) \Leftrightarrow$$
$$(\exists\ x1 : \mathbb{U} \bullet$$
$$((people \mathrel{\widehat{=}} x1.people) \in File3$$
$$\land x1.people = \{\})$$
$$\land (height \mathrel{\widehat{=}} z.height,\ people \mathrel{\widehat{=}} x1.people) \in File2)$$
$$: THM$$

Schema Unique Existence

SML

$rewrite\_conv[]ⓩz \in (\exists_1\ File3 \mid people = \{\} \bullet File2)⌝;$

ProofPower output

$val\ it = \vdash z \in (\exists_1\ File3 \mid people = \{\} \bullet File2) \Leftrightarrow$
$(\exists_1\ x1 : \mathbb{U} \bullet$
$((people \mathrel{\widehat{=}} x1.people) \in File3$
$\land x1.people = \{\})$
$\land (height \mathrel{\widehat{=}} z.height, people \mathrel{\widehat{=}} x1.people) \in File2)$
$: THM$

Schema Universal

SML

$rewrite\_conv[]ⓩz \in (\forall\ File3 \mid people = \{\} \bullet File2)⌝;$

ProofPower output

$val\ it = \vdash z \in (\forall\ File3 \mid people = \{\} \bullet File2) \Leftrightarrow$
$(\forall\ x1 : \mathbb{U}$
$\bullet (people \mathrel{\widehat{=}} x1.people) \in File3 \land x1.people = \{\}$
$\Rightarrow (height \mathrel{\widehat{=}} z.height, people \mathrel{\widehat{=}} x1.people) \in File2)$
$: THM$

Decoration

SML

$rewrite\_conv[]⌜_Z z \in File\ ''⌝;$

ProofPower output

$val\ it = \vdash z \in (File'') \quad \Leftrightarrow$
$\ (age \mathrel{\widehat{=}} z.age'', people \mathrel{\widehat{=}} z.people'') \in File : THM$

Pre-Condition

SML

$once\_rewrite\_conv[]⌜_Z z \in (pre\ FileOp)⌝;$

ProofPower output

$val\ it = \vdash z \in (pre\ FileOp) \Leftrightarrow$
$\ z \in \quad [age : \mathbb{U}; i? : \mathbb{U}; people : \mathbb{U}$
$\quad | \exists\ age' : \mathbb{U}; people' : \mathbb{U} \bullet FileOp] : THM$

Schema Hiding

SML

$once\_rewrite\_conv[] ⌜_Z z \in (File \setminus_s (age)) ⌝;$

ProofPower output

$val\ it = \vdash z \in (File \setminus_s (age)) \Leftrightarrow$
$\quad z \in [people : \mathbb{U} \mid \exists\ age : \mathbb{U} \bullet File] : THM$

SML

$rewrite\_conv[] ⌜_Z z \in (File \setminus_s (age)) ⌝;$

ProofPower output

$val\ it = \vdash z \in (File \setminus_s (age))$
$\quad \Leftrightarrow (\exists\ age : \mathbb{U} \bullet$
$\quad\quad (age \mathrel{\widehat{=}} age,\ people \mathrel{\widehat{=}} z.people) \in File) : THM$

Schema Renaming

SML

$once\_rewrite\_conv[]⌜_Z z \in File[aged/age]⌝;$

ProofPower output

$val\ it = \vdash z \in (File\ [aged/age]) \Leftrightarrow$
$\quad (age \mathrel{\widehat{=}} z.aged,\ people \mathrel{\widehat{=}} z.people) \in File : THM$

Schema Projection

SML

$once\_rewrite\_conv[]⌜_Z z \in (FileOp \upharpoonright_s File)⌝;$ (∗ ∗)

ProofPower output

$val\ it = \vdash z \in (FileOp \upharpoonright_s File)$
$\quad \Leftrightarrow z \in ((FileOp \land File) \setminus_s (age', i?, people')) : THM$

Schema Composition

SML

```
once_rewrite_conv[]⌜z ∈ (FileOp ⨾s FileOp)⌝; (* *)
```

ProofPower output

```
val it = ⊢ z ∈ (FileOp ⨾s FileOp)
    ⇔ z
      ∈ [age : 𝕌; i? : 𝕌; people : 𝕌; age′ : 𝕌; people′ : 𝕌
        | ∃ x1 : 𝕌; x2 : 𝕌
          • (age ≙ age, age′ ≙ x1, i? ≙ i?,
                    people ≙ people, people′ ≙ x2)
              ∈ FileOp
            ∧ (age ≙ x1, age′ ≙ age′, i? ≙ i?, people ≙ x2,
                people′ ≙ people′)
              ∈ FileOp] : THM
```


Delta

SML

$once\_rewrite\_conv[]⌜_Z z \in (\Delta File)⌝; (* \ \ *)$

ProofPower output

$val\ it = \vdash z \in (\Delta\ File) \Leftrightarrow$
$\quad z \in [File; File'] : THM$

Xi

SML

$once\_rewrite\_conv[]⌜_Z z \in (\Xi File)⌝; (* \ \ *)$

ProofPower output

$val\ it = \vdash z \in (\Xi\ File) \Leftrightarrow$
$\quad z \in [File; File' \mid \theta File = \theta File'] : THM$

Exercises 4: Schema Expressions

Try Exercises 4 in zed_course_work.doc.

Hints and further exercises may be found in sections 7.2.2 and 7.2.3 of the Z Tutorial Manual.

The exercises show that these operators behave in similiar ways to the predicate calculus versions, and that reasoning is largely automatic.

Entering the goals is tricky because the parser prefers the predicate calculus interpretation of the connectives.

Z Paragraphs

- Fixity declarations
- Given sets
- Abbreviation definitions
- Schema boxes
- Axiomatic descriptions
- Generics
- Free types
- Constraints

Z Paragraphs
Paragraph Processing Modes and Flags

There are several different modes of processing Z paragraphs which are controlled by flags.

- **Type-checking Mode**

  If the flag *z_type_check_only* is set to *true* then only type checking of Z paragrpahs is performed.

  This makes the response faster, and permits greater flexibility in amending paragraphs. This mode is suitable for use while developing specifications prior to undertaking any proof work.

- **Axiomatic Mode**

  If the flag *z_use_axioms* is set to true (and *z_type_check_only* is set to *false*) then axiomatic descriptions and free-type descriptions are introduced using axioms.

- **Conservative Mode**

  If both the above flags is set *false* then all Z axiomatic descriptions are introduced using the ProofPower *new_specification* facility, i.e. by conservative extension.

  Consistency proof obligations, unless discharged automatically, will have to be discharged by the user.

  In a future release it is hoped that free-types will also be supported by conservative extension.

Fixity Declarations

Fixity declarations may be provided for:

- functions

Z

| *fun 10 twice _*

Z

| *fun select ... from _*

- generics

Z

| *gen _ swap _*

- relations

Z

| *rel _ is_even*

The optional numeric value is a priority.

'_' is a space for a parameter
'...' is a space for a sequence of parameters (with sequence brackets elided)

Fixity clauses can only be deleted by deleting the theory they are contained in.

Given Sets

Z

$[G1,\ G2]$

SML

$val\ G1\_def = z\_get\_spec\ ⓩG1⌝;$

ProofPower output

$val\ G1\_def = \vdash\ G1 = \mathbb{U} : THM$

SML

$rewrite\_conv\ [G1\_def]\ ⓩx \in G1⌝;$

ProofPower output

$val\ it = \vdash\ x \in G1 \Leftrightarrow true : THM$

Abbreviation Definitions

SML

```
val _ = set_flag("z_type_check_only", false);
```

Z

$X\ swap\ Y \mathrel{\widehat=} Y \times X$

SML

```
val swap_def = z_get_spec ⓩ(_ swap _)⌝;
```

ProofPower Output

```
val swap_def =
⊢ [X, Y](X swap Y = Y × X) : THM
```

SML

```
rewrite_conv [swap_def] ⓩℤ swap ℕ⌝;
```

ProofPower Output

```
val it = ⊢ ℤ swap ℕ = ℕ × ℤ : THM
```


Schema Boxes

Z

```
┌─ Sch ──────────────────────────────
│        x, y : ℤ;
│        z : ℕ
├────────────────
│        x = y ∨ y = z
└────────────────────────────────────
```

SML

```
val sch_def = z_get_spec ⓩSch⌝;
```

ProofPower Output

```
val sch_def = ⊢ Sch =
  [x, y : ℤ; z : ℕ | x = y ∨ y = z] : THM
```

SML

```
rewrite_conv [sch_def]
 ⓩ∀ x,y:ℤ; z:ℕ • Sch ∨ disjoint ⟨{x},{y},{z}⟩⌝;
```

ProofPower Output

```
val it = ⊢ (∀ x, y : ℤ; z : ℕ • Sch
                    ∨ disjoint ⟨{x}, {y}, {z}⟩)
    ⇔ (∀ x, y : ℤ; z : ℕ
      • [x, y : ℤ; z : ℕ | x = y ∨ y = z]
           ∨ disjoint ⟨{x}, {y}, {z}⟩) : THM
```


Generic Schema Boxes

Z

$DSUBS[X]$

$set1, set2 : \mathbb{P}\ X$

$set1 \cap set2 = \{\}$

SML

$val\ dsubs\_def = z\_get\_spec\ ⓩDSUBS⌝;$

ProofPower Output

$val\ dsubs\_def = \vdash [X](DSUBS[X] =$
$[set1, set2 : \mathbb{P}\ X \mid set1 \cap set2 = \{\}]) : THM$

SML

$rewrite\_conv\ [dsubs\_def]$
$ⓩ\forall\ DSUBS[\mathbb{N}] \bullet set1 \subseteq \mathbb{N} \land set2 \subseteq \mathbb{N}⌝;$

ProofPower Output

$val\ it = \vdash (\forall\ (DSUBS[\mathbb{N}]) \bullet set1 \subseteq \mathbb{N} \land set2 \subseteq \mathbb{N})$
$\Leftrightarrow (\forall\ [set1, set2 : \mathbb{P}\ \mathbb{N} \mid set1 \cap set2 = \{\}] \bullet$
$set1 \subseteq \mathbb{N} \land set2 \subseteq \mathbb{N}) : THM$

Axiomatic Descriptions

Z

$$twice\ \_ : \mathbb{Z} \rightarrow \mathbb{Z}$$

$$\forall i : \mathbb{Z} \bullet twice\ i = 2*i$$

SML

```
val twice_def = z_get_spec ⓩ(twice _)⌝;
```

ProofPower Output

```
val twice_def = ⊢ (twice _) ∈ ℤ → ℤ
        ∧ (∀ i : ℤ • twice i = 2 ∗ i) : THM
```

SML

```
rewrite_conv[twice_def] ⓩtwice 4⌝;
```

ProofPower Output

```
Exception− Fail ∗ no rewriting occurred
```

SML

```
set_goal([],ⓩ∀ n:ℤ • twice n = 2∗n⌝);
a (REPEAT z_strip_tac);
```

ProofPower Output

```
(∗ ∗∗∗ Goal "" ∗∗∗ ∗)
(∗  1 ∗)  ⓩn ∈ ℤ⌝
(∗ ?⊢ ∗)  ⓩtwice n = 2 ∗ n⌝
```

SML

```
a (fc_tac [twice_def]);
```

ProofPower Output

```
Current and main goal achieved
```

Generic Axiomatics

Z

$[X, Y, Z]$

$$select\ ...\ from\ \_ : (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \rightarrow (Y \leftrightarrow Z)$$

$$\forall\ indexed\_set:(X \leftrightarrow Y);\ relation:(Y \leftrightarrow Z) \bullet$$

$$(select\ ...\ from\ \_)\ (indexed\_set,\ relation)$$
$$= (ran\ indexed\_set) \lhd relation$$

ProofPower output

$$val\ select\_from\_def = \vdash [X,\ Y,\ Z]($$
$$(select\ ...\ from\ \_)[X,\ Y,\ Z]$$
$$\in (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \rightarrow Y \leftrightarrow Z$$
$$\land$$
$$(\forall\ indexed\_set : X \leftrightarrow Y;\ relation : Y \leftrightarrow Z \bullet$$
$$(select\ ...\ from\ \_)[X,\ Y,\ Z]\ (indexed\_set,\ relation)$$
$$= ran\ indexed\_set \lhd relation))\ :\ THM$$

Free Types

Z

$TREE ::= tip \mid fork \; (\mathbb{N} \times TREE \times TREE)$

SML

```
val tree_def = z_get_spec ⓩTREE⌝;
```

ProofPower Output

$val\ tree\_def = \vdash TREE = \mathbb{U} : THM$

SML

```
val tip_def = z_get_spec ⓩtip⌝;
```

ProofPower Output

$$
\begin{array}{l}
val\ tip\_def = \vdash (\\
\quad tip \in TREE\\
\ \wedge fork \in \mathbb{N} \times TREE \times TREE \rightarrowtail TREE)\\
\\
\ \wedge disjoint \langle\{tip\}, ran\ fork\rangle\\
\\
\ \wedge (\forall\ W : \mathbb{P}\ TREE \mid\\
\quad \{tip\} \cup fork \ (\!|\ \mathbb{N} \times W \times W \ |\!) \subseteq W \bullet\\
\qquad\qquad TREE \subseteq W) : THM
\end{array}
$$

Mutually Recursive Free Types

Z

$$TYPE ::= Tvar\ G1 \mid Tcon\ (G1 \times seq\ TERM)$$
$$\&$$
$$TERM ::= Con\ (G1 \times TYPE) \mid App\ (TERM \times TERM)$$

SML

$$val\ tvar\_def = z\_get\_spec\ ⓩTvar⌝;$$

ProofPower Output

$$\begin{array}{l}
val\ tvar_def = \vdash (\\
\quad\quad Tvar \in G1 \rightarrowtail TYPE\\
\quad \wedge Tcon \in G1 \times (seq\ TERM) \rightarrowtail TYPE\\
\quad \wedge Con \in G1 \times TYPE \rightarrowtail TERM\\
\quad \wedge App \in TERM \times TERM \rightarrowtail TERM)\\
\\
\quad \wedge (disjoint\ \langle ran\ Tvar, ran\ Tcon\rangle\\
\\
\quad \wedge (\forall\ W : \mathbb{P}\ TYPE\\
\quad\ \ \mid Tvar\ (\!|\ G1\ |\!) \cup Tcon\ (\!|\ G1 \times (seq\ TERM)\ |\!) \subseteq W\\
\quad\ \ \bullet TYPE \subseteq W))\\
\\
\quad \wedge disjoint\ \langle ran\ Con, ran\ App\rangle\\
\\
\quad \wedge (\forall\ W : \mathbb{P}\ TERM\\
\quad\ \ \mid Con\ (\!|\ G1 \times TYPE\ |\!) \cup App\ (\!|\ W \times W\ |\!) \subseteq W\\
\quad\ \ \bullet TERM \subseteq W) : THM
\end{array}$$

Constraints

Z

$$[X]\ ((\exists f : X \rightarrowtail G1 \bullet true) \Leftrightarrow (\exists f : X \rightarrowtail G2 \bullet true))$$

SML

$val\ c1 = get\_axiom$ "−" "$Constraint\ 1$";

ProofPower output

$$val\ c1 = \vdash [X]((\exists\ f : X \rightarrowtail G1 \bullet true) \Leftrightarrow (\exists\ f : X \rightarrowtail G2 \bullet true)) : THM$$

Z

$$\{1\}\ swap\ \{\langle 1 \rangle\} = \{\langle 1 \rangle\} \times \{1\} \land Sch \neq [x, y, z : \mathbb{Z}]$$

Z

$$tip \neq fork(2, tip, tip) \land tip \in TREE$$

Theories

Z Theories contain the following information:

- The theory name and the names of the theories parents and children.

- The names of types (given sets) declared in the theory.

- The names and types of 'global variables' declared in the theory.

- Fixity information.

- Axioms or specifications corresponding to the paragraphs of the Z specification introduced in this theory.

- A collection of saved theorems.

Access to Z Theories

- To use a theory it must be "in context", this can be achieved be opening the theory or one of its descendents:

SML

```
open_theory : string -> unit;
```

- To display the contents of a theory:

SML

```
z_print_theory : string -> unit;
```

- To get things from the theory:

SML

```
get_aliases; get_ancestors; get_axiom; get_axioms;
get_children; get_consts; get_defn;
get_defns; get_descendants; get_parents; get_thm;
get_thms; z_get_spec;
```

- To save things in the theory use Z paragraphs.

Exercises 5: Paragraphs and Theories

Try Exercises 5 in zed_course_work.doc.

Hints and further exercises may be found in section 7.3 of the Z Tutorial Manual.

The Z ToolKit

Available as a set of six theories.

To get the Z ToolKit in context make $z\_library$ a parent.

The theories are:

- z_sets
- $z\_relations$
- $z\_functions$
- $z\_numbers$
- $z\_sequences$
- z_bags

These definitions have been entered in axiomatic mode.

Sets and Relations

- Recommended proof context: z_rel_ext.

- High rate of automatic proof of lemmas in these theories.

- Automatic proof fails if actual generic parameters are supplied.

A Sample Proof About Sets (I)

SML

```
set_pc "z_library_ext";
set_goal([],⌜Z a ∩ (b \ c) = (a ∩ b) \ c⌝);
a z_strip_tac;
```

ProofPower output

```
(∗ ?⊢ ∗)  ⌜Z ∀ x1 : 𝕌 • x1 ∈ a ∩ (b \ c) ⇔ x1 ∈ a ∩ b \ c⌝
```

SML

```
a z_strip_tac;
```

ProofPower output

```
(∗ ?⊢ ∗)  ⌜Z x1 ∈ 𝕌 ∧ true ⇒ (x1 ∈ a ∩ (b \ c) ⇔ x1 ∈ a ∩ b \ c)⌝
```

continuing only using *z_strip_tac* as follows:

ProofPower output

```
(∗ ?⊢ ∗)  ⌜Z x1 ∈ a ∩ (b \ c) ⇔ x1 ∈ a ∩ b \ c⌝
```

ProofPower output

```
(∗ ?⊢ ∗)  ⌜Z (x1 ∈ a ∩ (b \ c) ⇒ x1 ∈ a ∩ b \ c)
              ∧ (x1 ∈ a ∩ b \ c ⇒ x1 ∈ a ∩ (b \ c))⌝
```

ProofPower output

```
(∗ ∗∗∗ Goal "2" ∗∗∗ ∗)
(∗ ?⊢ ∗)  ⌜Z x1 ∈ a ∩ b \ c ⇒ x1 ∈ a ∩ (b \ c)⌝

(∗ ∗∗∗ Goal "1" ∗∗∗ ∗)
(∗ ?⊢ ∗)  ⌜Z x1 ∈ a ∩ (b \ c) ⇒ x1 ∈ a ∩ b \ c⌝
```


A Sample Proof About Sets (II)

ProofPower output

```
(*  3 *)  ⌜Z x1 ∈ a⌝
(*  2 *)  ⌜Z x1 ∈ b⌝
(*  1 *)  ⌜Z ¬ x1 ∈ c⌝

(* ?⊢ *)  ⌜Z x1 ∈ a ∩ b \ c⌝
```

ProofPower output

```
...
(* ?⊢ *)  ⌜Z x1 ∈ a ∩ b ∧ x1 ∉ c⌝
```

ProofPower output

```
(* *** Goal "1.2" *** *)
...
(* ?⊢ *)  ⌜Z x1 ∉ c⌝

(* *** Goal "1.1" *** *)
...
(* ?⊢ *)  ⌜Z x1 ∈ a ∩ b⌝
```

ProofPower output

```
(* *** Goal "1.1" *** *)
...
(* ?⊢ *)  ⌜Z x1 ∈ a ∧ x1 ∈ b⌝
```

ProofPower output

```
(* *** Goal "1.1.2" *** *)
(* ?⊢ *)  ⌜Z x1 ∈ b⌝

(* *** Goal "1.1.1" *** *)
(* ?⊢ *)  ⌜Z x1 ∈ a⌝
```

A Sample Proof About Sets (III)

ProofPower output

```
Tactic produced 0 subgoals:
Current goal achieved, next goal is:
(* *** Goal "1.1.2" *** *)
...
```

ProofPower output

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

(* *** Goal "1.2" *** *)
...
```

ProofPower output

```
...
(* ?⊢ *)  ⌜Z¬ x1 ∈ c⌝
```

ProofPower output

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

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

(* ?⊢ *)  ⌜Z x1 ∈ a ∩ b \ c ⇒ x1 ∈ a ∩ (b \ c)⌝
...
```

Goal 2 being similar to goal 1 we complete its proof in one step:

SML

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

ProofPower output

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


A Sample Proof About Relations (I)

SML

```
set_goal([], ⓩP ⨾ (Q ⨾ R) = (P ⨾ Q) ⨾ R⌝);
a contr_tac;
```

ProofPower output

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

(*  4 *)  ⓩ(x1, y') ∈ P⌝
(*  3 *)  ⓩ(y', y) ∈ Q⌝
(*  2 *)  ⓩ(y, x2) ∈ RSchedule
(*  1 *)  ⓩ∀ y : 𝕌 • ¬ ((x1, y) ∈ P ∧ (y, x2) ∈ Q ⨾ R)⌝

(* ?⊢ *)  ⓩfalse⌝


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

(*  4 *)  ⓩ(x1, y) ∈ P⌝
(*  3 *)  ⓩ(y, y') ∈ Q⌝
(*  2 *)  ⓩ(y', x2) ∈ R⌝
(*  1 *)  ⓩ∀ y : 𝕌 • ¬ ((x1, y) ∈ P ⨾ Q ∧ (y, x2) ∈ R)⌝

(* ?⊢ *)  ⓩfalse⌝
```


A Sample Proof About Relations (II)

SML

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

ProofPower output

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

SML

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

ProofPower output

```
Tactic produced 0 subgoals:
Current and main goal achieved
...
val it = ⊢ P ⨾ Q ⨾ R = (P ⨾ Q) ⨾ R : THM
...
```


Functions

- Automatic proof not very effective.

- Recommended proof contexts: ?

  - Use z_fun_ext for extensional low level reasoning (expands out function arrows).

  - Use $z\_library$ for non-extensional reasoning.

  - For extensional reasoning avoiding expansion of function arrows merge $'z\_fun\_alg$ into z_rel_ext.

  - Use $z\_fun\_$??$\_clauses$ where appropriate instead of expanding function arrows.

- Much further development expected.

Numbers

- Use proof context $z\_library$.
- Theory well populated with results.
- Induction tactics available.
- Linear arithmetic not yet available.
- Theory of # not yet developed.

Induction

Induction principles for Z can be expressed as mixed language theorems in Higher Order Logic and Z e.g.:

- $z\_\mathbb{N}\_induction\_thm$

$$
\begin{array}{|l}
\vdash \forall\ p \\
\quad \bullet\ p\ ⓩ\ 0 ⌝ \\
\quad \land\ (\forall\ i \bullet\ i \in ⓩ\ \mathbb{N} ⌝ \land\ p\ i \Rightarrow p\ ⓩ\ i + 1 ⌝) \\
\quad \Rightarrow (\forall\ m \bullet\ m \in ⓩ\ \mathbb{N} ⌝ \Rightarrow p\ m)\ :\ THM
\end{array}
$$

- $z\_\mathbb{Z}\_induction\_thm$

$$
\begin{array}{|l}
\vdash \forall\ p \\
\quad \bullet\ p\ ⓩ\ 1 ⌝ \\
\quad \land\ (\forall\ i \bullet\ p\ i \Rightarrow p\ ⓩ\ \sim\ i ⌝) \\
\quad \land\ (\forall\ i\ j \bullet\ p\ i \land p\ j \Rightarrow p\ ⓩ\ i + j ⌝) \\
\quad \Rightarrow (\forall\ m \bullet\ p\ m)\ :\ THM
\end{array}
$$

$\forall\_elim$ and all_β_rule may be used to specialise these for use in forward proofs.

Induction Tactics

Special tactics are available to facilitate the use of induction principles:

- induction over natural numbers using $z\_\mathbb{N}\_induction\_tac$

$$\frac{\{\ \Gamma\ \}\ x \in \mathbb{N} \Rightarrow t}{\{\ \Gamma\ \}\ t[0/x]\ ;\quad strip\{t,\ \Gamma\}\ t[x+1/x]} \qquad z\_\mathbb{N}\_induction\_tac\ ⓩ\ x⌝$$

- induction over integers using $z\_\mathbb{Z}\_induction\_tac$

$$\frac{\{\ \Gamma\ \}\ t}{\{\ \Gamma\ \}\ t[1/x];\quad strip\{t[i/x],\ \Gamma\}\ t[\sim i/x];\quad strip\{t[i/x] \wedge t[j/x],\ \Gamma\}\ t[i+j/x]} \qquad z\_\mathbb{Z}\_induction\_tac\ ⓩ\ x⌝$$

Induction - Example (I)

SML

$set\_goal\ ([], ⓩx \in \mathbb{N} \Rightarrow x + y \geq y⌝);$
$a\ (z\_\mathbb{N}\_induction\_tac);$

ProofPower output

$(*\ ***\ Goal\ "2"\ ***\ *)$

$(*\ \ 1\ *)\ \ ⓩ0 \leq i⌝$

$(*\ ?\vdash\ *)\ \ ⓩ(i + 1) + y \geq y⌝$

$(*\ ***\ Goal\ "1"\ ***\ *)$

$(*\ ?\vdash\ *)\ \ ⓩ0 + y \geq y⌝$

Induction - Example (II)

SML

```
set_goal ([], ⓩx * x ≥ 0⌝);
a (z_ℤ_induction_tac ⓩx⌝);
```

ProofPower output

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

(* ?⊢ *)  ⓩ(i + j) * (i + j) ≥ 0⌝

(* *** Goal "2" *** *)
(*  1 *)  ⓩ0 ≤ i * i⌝

(* ?⊢ *)  ⓩ~ i * ~ i ≥ 0⌝

(* *** Goal "1" *** *)
(* ?⊢ *)  ⓩ1 * 1 ≥ 0⌝
```


SML

```
set_goal([],⌜∀ x, y : ℤ •
        x ≤ y ⇒ 0 .. x ⊆ 0 .. y⌝);
a(rewrite_tac[z_get_spec ⌜(_ .. _)⌝]
 THEN REPEAT strip_tac);
```

ProofPower output

```
...
(∗  3 ∗)  ⌜x ≤ y⌝
(∗  2 ∗)  ⌜0 ≤ x1⌝
(∗  1 ∗)  ⌜x1 ≤ x⌝

(∗ ?⊢ ∗)  ⌜x1 ≤ y⌝
```

SML

```
z_≤_trans_thm;
```

ProofPower output

```
val it = ⊢ ∀ i, j, k : 𝕌
        | i ≤ j ∧ j ≤ k • i ≤ k : THM
```

SML

```
a(all_fc_tac[z_≤_trans_thm]);
```

ProofPower output

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

SML

```
set_pc "z_library_ext";
set_goal([],⌜∀ x, y : ℤ • ¬ x ≤ y ⇒
                0 .. y ⊆ 0 .. x − 1⌝);
a(rewrite_tac[z_get_spec ⌜(_ .. _)⌝]
 THEN REPEAT strip_tac);
```

ProofPower output

```
...
(∗  3 ∗)  ⌜y < x⌝
(∗  2 ∗)  ⌜0 ≤ x1⌝
(∗  1 ∗)  ⌜x1 ≤ y⌝

(∗ ?⊢ ∗)  ⌜x1 ≤ x + ∼ 1⌝
```

SML

```
a(all_fc_tac[z_≤_less_trans_thm]);
```

ProofPower output

```
(∗  4 ∗)  ⌜y < x⌝
(∗  3 ∗)  ⌜0 ≤ x1⌝
(∗  2 ∗)  ⌜x1 ≤ y⌝
(∗  1 ∗)  ⌜x1 < x⌝

(∗ ?⊢ ∗)  ⌜x1 ≤ x + ∼ 1⌝
```

SML

```
a(POP_ASM_T (ante_tac o
pure_once_rewrite_rule[z_get_spec⌜(_ < _)⌝]));
```


ProofPower output

```
(*  3 *)  ⓩy < x⌝
(*  2 *)  ⓩ0 ≤ x1⌝
(*  1 *)  ⓩx1 ≤ y⌝

(* ?⊢ *)  ⓩx1 + 1 ≤ x ⇒ x1 ≤ x + ~ 1⌝
```

SML

```
a(once_rewrite_tac[z_≤_≤_0_thm]);
```

ProofPower output

```
(*  3 *)  ⓩy < x⌝
(*  2 *)  ⓩ0 ≤ x1⌝
(*  1 *)  ⓩx1 ≤ y⌝

(* ?⊢ *)  ⓩ(x1 + 1) + ~ x ≤ 0 ⇒
                    x1 + ~ (x + ~ 1) ≤ 0⌝
```

SML

```
a(rewrite_tac[z_∀_elim ⓩ~ x⌝ z_plus_order_thm,
                    z_minus_thm]);
```

ProofPower output

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


Sequences and Bags

- All definitions present.
- Theories otherwise undeveloped.
- Theory of # required before development of this theory.
- If necessary, pro-tem, we recommend adding induction principles and other required results for reasoning in this theory as “constraints”.

Exercises 6: Z ToolKit

Try Exercises 6 in zed_course_work.doc.

Hints and further exercises may be found in section 7.4 of the Z Tutorial Manual.

- 7.4.1 Sets

  (easy)

- 7.4.2 Relations

  (start easy and get harder
  solutions to last set incomplete)

- 7.4.3 Functions

  (no so easy, some solutions missing)

- 7.4.4 Numbers and Finiteness

  (middling to hard)

CASE STUDY - Confidentiality

SML

```
open_theory"usr023";
new_theory"usr023C";
set_pc "z_library";
set_flag("z_type_check_only", false);
set_flag("z_use_axioms", true);
```

Z

[DATA]

Z

STATE

$classified\_data : \mathbb{N} \pfun DATA$

Z

OPERATION

$\Delta STATE;$
clear$? : \mathbb{N}$

Z

READ

$$
\begin{array}{l}
\mathit{OPERATION};\\
\mathbf{class?} : \mathbb{N};\\
\mathbf{data!} : \mathit{DATA}\\
\hline
\mathit{class}? \in \mathrm{dom}\ \mathit{classified\_data};\\
\mathit{class}? \leq \mathit{clear}?;\\
\mathit{data}! = \mathit{classified\_data}\ \mathit{class}?;\\
\mathit{classified\_data}' = \mathit{classified\_data}
\end{array}
$$

Z

COMPUTE

$$
\begin{array}{l}
\mathit{OPERATION};\\
\mathbf{class?} : \mathbb{N};\\
\mathbf{computation?} : (\mathbb{N} \rightarrowtail \mathit{DATA}) \rightarrow \mathit{DATA}\\
\hline
\mathit{class}? \in \mathrm{dom}\ \mathit{classified\_data};\\
\mathit{class}? \geq \mathit{clear}?;\\
\mathit{classified\_data}'\\
= \mathit{classified\_data} \oplus \{\mathit{class}? \mapsto\\
\quad \mathit{computation}?\ ((0\ ..\ \mathit{clear}?) \lhd \mathit{classified\_data})\}
\end{array}
$$

Pre-Condition Proofs

SML

```
set_goal ([], ⓩpre OPERATION ⇔
  classified_data ∈ ℕ ⇸ DATA ∧ 0 ≤ clear?⌝);
```

SML

```
a (rewrite_tac (map z_get_spec
            [ⓩOPERATION⌝, ⓩSTATE⌝]));
```

ProofPower output

```
...
(* ?⊢ *)  ⓩ(∃ classified_data' : 𝕌
    • (classified_data ∈ ℕ ⇸ DATA
      ∧ classified_data' ∈ ℕ ⇸ DATA)
    ∧ 0 ≤ clear?)
 ⇔ classified_data ∈ ℕ ⇸ DATA ∧ 0 ≤ clear?⌝
...
```


SML

```
a (REPEAT z_strip_tac
   THEN_TRY asm_rewrite_tac[]);
```

ProofPower output

```
...
(*  2 *)  ⓩclassified_data ∈ ℕ ⇸ DATA⌝
(*  1 *)  ⓩ0 ≤ clear?⌝

(* ?⊢ *)  ⓩ∃ classified_data′ : 𝕌 •
          classified_data′ ∈ ℕ ⇸ DATA⌝
...
```

SML

```
a (z_∃_tac ⓩ{}⌝ THEN
  PC_T1 "z_library_ext" rewrite_tac[]);
```


An Algorithmic Refinement

Z

BADREAD

$OPERATION$;
class? : $\mathbb{N}$;
data! : $DATA$

$READ \lor$
$(class? > clear?;$
$data! = classified\_data\ class?;$
$classified\_data' = classified\_data)$

SML

```
set_goal([],
 ⌜(pre READ ⇒ pre BADREAD)
 ∧ (pre READ ∧ BADREAD ⇒ READ)⌝);

a (rewrite_tac (map z_get_spec
 [⌜BADREAD⌝, ⌜READ⌝, ⌜OPERATION⌝, ⌜STATE⌝]));

a (REPEAT z_strip_tac THEN rename_tac[]
 THEN asm_rewrite_tac[]);
```


ProofPower output

```
...
(*  9 *)  ⌜classified_data ∈ ℕ ⇸ DATA⌝
(*  8 *)  ⌜classified_data' ∈ ℕ ⇸ DATA⌝
(*  7 *)  ⌜0 ≤ clear?⌝
(*  6 *)  ⌜0 ≤ class?⌝
(*  5 *)  ⌜data! ∈ DATA⌝
(*  4 *)  ⌜class? ∈ dom classified_data⌝
(*  3 *)  ⌜class? ≤ clear?⌝
(*  2 *)  ⌜data! = classified_data class?⌝
(*  1 *)  ⌜classified_data' = classified_data⌝

(* ?⊢ *)  ⌜∃ classified_data'' : 𝕌; data!' : 𝕌
 • (classified_data'' ∈ ℕ ⇸ DATA
   ∧ data!' ∈ DATA)
 ∧ ((classified_data'' ∈ ℕ ⇸ DATA
       ∧ data!' ∈ DATA)
     ∧ data!' = classified_data class?
     ∧ classified_data'' = classified_data
   ∨ clear? < class?
     ∧ data!' = classified_data class?
     ∧ classified_data'' = classified_data)⌝
...
```

SML

```
a (z_∃_tac ⓩ
   (classified_data″ ≙ classified_data,
   data!′ ≙ classified_data class?)⌝
 THEN asm_rewrite_tac[]);
```

ProofPower output

```
...
(*  9 *)  ⓩclassified_data ∈ ℕ ⇸ DATA⌝
...
(*  4 *)  ⓩclass? ∈ dom classified_data⌝
...
(* ?⊢ *)  ⓩclassified_data class? ∈ DATA
              ∧ (classified_data class? ∈ DATA ∨ clear? < class?)⌝
...
```

```
⊢ ∀ 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
```

SML

```
a (all_fc_tac [z_fun_∈_clauses]
            THEN REPEAT strip_tac);
```


Base Types

SML

$open\_theory$ "$usr023C$";

Z

$[\mathbf{IN}, \mathbf{OUT}]$

Z

$\mathbf{STATE2} \mathrel{\widehat{=}} \mathbb{N} \pfun DATA$

Z

$\mathbf{SYSTEM} \mathrel{\widehat{=}} (\mathbb{N} \times IN \times STATE2)$
$\quad \rightarrow (STATE2 \times OUT)$

Critical Property

Z

$$\mathbf{out\_secure} : \mathbb{P}\ SYSTEM$$

$$\forall sys{:}SYSTEM \bullet\ sys \in out\_secure \Leftrightarrow$$

$$(\forall\ clear{:}\mathbb{N};\ inp{:}IN;\ s{,}s'{:}STATE2$$
$$\mid (0\ ..\ \ clear) \lhd s = (0\ ..\ \ clear) \lhd s'$$
$$\bullet\ second\ (sys\ (clear,\ inp,\ s))$$
$$=\ second\ (sys\ (clear,\ inp,\ s')))$$

Z

$$\mathbf{state\_secure} : \mathbb{P}\ SYSTEM$$

$$\forall sys{:}SYSTEM \bullet\ sys \in state\_secure \Leftrightarrow$$

$$(\forall class,\ clear{:}\mathbb{N};\ inp{:}IN;\ s{,}s'{:}STATE2$$
$$\mid ((0\ ..\ \ class) \lhd s) = ((0\ ..\ \ class) \lhd s')$$
$$\bullet\ (0\ ..\ \ class) \lhd (first\ (sys\ (clear,\ inp,\ s)))$$
$$=\ (0\ ..\ \ class) \lhd (first\ (sys\ (clear,\ inp,\ s'))))$$

Z

$$\mathbf{secure} : \mathbb{P}\ SYSTEM$$

$$\forall sys{:}SYSTEM \bullet\ sys \in secure \Leftrightarrow$$
$$sys \in state\_secure \land sys \in out\_secure$$

Secure Architecture

Z

$$\textbf{APPLICATION} \mathrel{\widehat=} (IN \times STATE2) \\ \quad \to (STATE2 \times OUT)$$

Z

$$\textbf{KERNEL} \mathrel{\widehat=} APPLICATION \to SYSTEM$$

Z

$$\textbf{construction} : APPLICATION \times KERNEL \to SYSTEM$$

$$\forall appl{:}APPLICATION;\ kernel{:}KERNEL \bullet \\ \quad construction\ (appl,\ kernel) = kernel\ appl$$

Z

$$\textbf{secure\_kernel} : \mathbb{P}\ KERNEL$$

$$\forall\ kernel{:}KERNEL \bullet\ kernel \in secure\_kernel \Leftrightarrow \\ \quad (\forall appl{:}APPLICATION \bullet \\ \qquad (construction\ (appl,\ kernel)) \in secure)$$

Architectural Correctness

SML

```
set_goal([],⌜∀kernel:KERNEL;appl:APPLICATION•
  kernel ∈ secure_kernel ⇒
  (construction (appl,kernel)) ∈ secure⌝);
```

SML

```
val secure_kernel_sim = z_defn_simp_rule
            (z_get_spec⌜secure_kernel⌝);
```

ProofPower output

```
val secure_kernel_sim = ⊢ ∀ kernel : 𝕌 •
 kernel ∈ secure_kernel ⇔
 kernel ∈ KERNEL
 ∧ (∀ appl : APPLICATION •
     construction (appl, kernel) ∈ secure) : THM
```

SML

```
a (rewrite_tac[secure_kernel_sim]);
```

ProofPower output

```
(∗ ?⊢ ∗)  ⌜∀ kernel : KERNEL; appl : APPLICATION
 • kernel ∈ KERNEL
 ∧ (∀ appl : APPLICATION
     • construction (appl, kernel) ∈ secure)
       ⇒ construction (appl, kernel) ∈ secure⌝
```


SML

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

ProofPower output

```
(∗  3 ∗)  ⌜kernel ∈ KERNEL⌝
(∗  2 ∗)  ⌜appl ∈ APPLICATION⌝
(∗  1 ∗)  ⌜∀ appl : APPLICATION •
              construction (appl, kernel) ∈ secure⌝

(∗ ?⊢ ∗)  ⌜construction (appl, kernel) ∈ secure⌝
```

SML

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

ProofPower output

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


A Secure Kernel

Z

kernel_implementation : $KERNEL$

$\forall\ clear:\mathbb{N};\ inp:IN;$
$\quad state:STATE2;\ appl:APPLICATION\ \bullet$

$kernel\_implementation\ appl\ (clear,\ inp,\ state) =$

$(\quad state \oplus ((0\ ..\ (clear-1)) \lhd$
$\qquad (first\ (appl\ (inp,\ (0\ ..\ clear) \lhd state)))),$
$\qquad\quad second\ (appl\ (inp,\ (0\ ..\ clear) \lhd state))\ )$

Arithmetic Lemmas

SML

```
set_pc "z_library_ext";
```

SML

```
set_goal ([], ⌜∀ x, y : ℤ • x ≤ y ⇒ (0 .. x) ⊆ (0 .. y)⌝);
a (rewrite_tac[z_get_spec ⌜(_ .. _)⌝]
  THEN REPEAT strip_tac);
a (all_fc_tac[z_≤_trans_thm]);
val le_dots_lemma1 = pop_thm ();
```

SML

```
set_goal ([], ⌜∀ x, y : ℤ • ¬ x ≤ y ⇒ (0 .. y) ⊆ (0 .. (x − 1))⌝);
a (rewrite_tac[z_get_spec ⌜(_ .. _)⌝]
  THEN REPEAT strip_tac);
a (all_fc_tac[z_≤_less_trans_thm]);
a (POP_ASM_T (ante_tac o pure_once_rewrite_rule
                        [z_get_spec⌜(_ < _)⌝]));
a (once_rewrite_tac[z_≤_≤_0_thm]);
a (rewrite_tac[z_∀_elim ⌜~ x⌝ z_plus_order_thm, z_minus_thm]);
val le_dots_lemma2 = pop_thm();
```

SML

```
val ×_fc_thm = prove_rule []
  ⌜(∀ v:𝕌; w:𝕌; V:𝕌; W:𝕌 •
  v ∈ V ∧ w ∈ W ⇒ (v,w) ∈ (V × W))⌝;
```


Kernel Security Proof

SML

```
set_pc "z_sets_alg";
set_goal([],⌜kernel_implementation ∈ secure_kernel⌝);
```

ProofPower output

```
(∗ ?⊢ ∗)  ⌜kernel_implementation ∈ secure_kernel⌝
```

SML

```
val specs = map (z_defn_simp_rule o z_get_spec)
  [⌜secure_kernel⌝, ⌜secure⌝, ⌜state_secure⌝, ⌜out_secure⌝];
a (           rewrite_tac specs
              THEN REPEAT strip_tac);
```

ProofPower output

```
Tactic produced 6 subgoals:

(∗ *** Goal "6" *** ∗)
(∗  6 ∗)  ⌜appl ∈ APPLICATION⌝
(∗  5 ∗)  ⌜clear ∈ ℕ⌝
(∗  4 ∗)  ⌜inp ∈ IN⌝
(∗  3 ∗)  ⌜s ∈ STATE2⌝
(∗  2 ∗)  ⌜s′ ∈ STATE2⌝
(∗  1 ∗)  ⌜(0 .. clear) ◁ s = (0 .. clear) ◁ s′⌝

(∗ ?⊢ ∗)
  ⌜(construction (appl, kernel_implementation) (clear, inp, s)).2
 = (construction (appl, kernel_implementation) (clear, inp, s′)).2⌝
```

A Secure Kernel

```
...
(* *** Goal "4" *** *)
(*  7 *)  ⓩappl ∈ APPLICATION⌝
(*  6 *)  ⓩclass ∈ ℕ⌝
(*  5 *)  ⓩclear ∈ ℕ⌝
(*  4 *)  ⓩinp ∈ IN⌝
(*  3 *)  ⓩs ∈ STATE2⌝
(*  2 *)  ⓩs' ∈ STATE2⌝
(*  1 *)  ⓩ(0 .. class) ◁ s = (0 .. class) ◁ s'⌝

(* ?⊢ *)  ⓩ(0 .. class)
                ◁ (construction (appl, kernel_implementation)
                      (clear, inp, s)).1
            = (0 .. class)
                ◁ (construction (appl, kernel_implementation)
                      (clear, inp, s')).1⌝
```

ProofPower output

```
...
(* *** Goal "2" *** *)
(*  1 *)  ⓩappl ∈ APPLICATION⌝

(* ?⊢ *)  ⓩconstruction (appl, kernel_implementation) ∈ SYSTEM⌝
```


ProofPower output

```
...
(* *** Goal "1" *** *)
(* ?⊢ *)  ⌜kernel_implementation ∈ KERNEL⌝

The subgoal 2 duplicates goals labelled 3, 5
The subgoal 3 duplicates goals labelled 2, 5
```

SML

```
val [condec, conpred] = strip_∧_rule (z_get_spec ⌜construction⌝);
val [kidec, kipred] =
        strip_∧_rule (z_get_spec ⌜kernel_implementation⌝);
```

ProofPower output

```
val condec = ⊢ construction ∈
      APPLICATION × KERNEL → SYSTEM : THM
val conpred =
   ⊢ ∀ appl : APPLICATION; kernel : KERNEL
        • construction (appl, kernel) = kernel appl : THM

val kidec = ⊢ kernel_implementation ∈ KERNEL : THM
val kipred =
...
```

SML

```
a (strip_asm_tac kidec);
```

ProofPower output

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


ProofPower output

```
(*  1 *)  ⌜appl ∈ APPLICATION⌝

(* ?⊢ *)  ⌜construction (appl, kernel_implementation) ∈ SYSTEM⌝
...
```

SML

```
a (asm_tac kidec THEN asm_tac condec);
a (LEMMA_T
 ⌜(appl, kernel_implementation) ∈ (APPLICATION × KERNEL)⌝
 asm_tac
 THEN1 contr_tac);
```

ProofPower output

```
...
(* 2 *) ⌜construction ∈ APPLICATION × KERNEL → SYSTEM⌝
(* 1 *) ⌜(appl, kernel_implementation) ∈ APPLICATION × KERNEL⌝

(* ?⊢ *) ⌜construction (appl, kernel_implementation) ∈ SYSTEM⌝
...
```

SML

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

This discharges the current subgoal.

ProofPower output

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


ProofPower output

```
(∗  7 ∗)  ⌜appl ∈ APPLICATION⌝
...
(∗  4 ∗)  ⌜inp ∈ IN⌝
...
(∗  1 ∗)  ⌜(0 .. class ◁ s) = (0 .. class ◁ s′)⌝

(∗ ?⊢ ∗)
  ⌜(0 .. class ◁ (construction
                    (appl, kernel_implementation)
                      (clear, inp, s)).1)
  = (0 .. class ◁ (construction
                    (appl, kernel_implementation)
                      (clear, inp, s′)).1)⌝
```

SML

```
a (strip_asm_tac kidec);
a (ALL_FC_T asm_rewrite_tac [kipred, conpred]);
```

ProofPower output

```
...
(∗  8 ∗)  ⌜appl ∈ APPLICATION⌝
...
(∗  2 ∗)  ⌜(0 .. class) ◁ s = (0 .. class) ◁ s′⌝
(∗  1 ∗)  ⌜kernel_implementation ∈ KERNEL⌝

(∗ ?⊢ ∗)
  ⌜(0 .. class) ◁ (s ⊕ (0 .. clear − 1) ⩤
                  (appl (inp, (0 .. clear) ◁ s)).1)
  = (0 .. class) ◁ (s′ ⊕ (0 .. clear − 1) ⩤
                  (appl (inp, (0 .. clear) ◁ s′)).1)⌝
...
```


If $\ulcorner_{Z} clear \leq class \urcorner$ then:

$$\ulcorner_{Z} (0..clear) \subseteq (0..class) \urcorner$$

and, given:

$$\ulcorner_{Z} (0..class) \lhd s = (0..class) \lhd s' \urcorner$$

we can conclude that:

$$\ulcorner_{Z} (0..clear) \lhd s = (0..clear) \lhd s' \urcorner$$

This fact may be used to rewrite the goal, changing the second occurence of s to s'. The resulting goal will be provable using:

$$\ulcorner_{Z} (0..class) \lhd s = (0..class) \lhd s' \urcorner$$

once more, with the theorem:

$$\ulcorner_{Z} x \lhd z = x \lhd z' \Rightarrow x \lhd (z \oplus y) = x \lhd (z' \oplus y) \urcorner$$

If $\ulcorner_{Z} \neg clear \leq class \urcorner$ then:

$$\ulcorner_{Z} 0..class \subseteq 0..(clear - 1) \urcorner$$

and the theorem:

$$\ulcorner_{Z} (A \subseteq B) \Rightarrow (A \lhd z) = (A \lhd z') \\ \Rightarrow (A \lhd (z \oplus (B \ntriangleleft s))) = (A \lhd (z' \oplus (B \ntriangleleft s'))) \urcorner$$

suffices to prove the subgoal.

SML

```
a (cases_tac ⌜clear ≤ class⌝);
```

ProofPower output

```
Tactic produced 2 subgoals:

(* *** Goal "4.1" *** *)
(*  9 *)  ⌜appl ∈ APPLICATION⌝
...
(*  3 *)  ⌜(0 .. class) ◁ s = (0 .. class) ◁ s'⌝
(*  2 *)  ⌜kernel_implementation ∈ KERNEL⌝
(*  1 *)  ⌜ clear ≤ class⌝

(* ?⊢ *)
  ⌜(0 .. class) ◁ (s ⊕ (0 .. clear − 1) ⩤
                 (appl (inp, (0 .. clear) ◁ s)).1)
 = (0 .. class) ◁ (s' ⊕ (0 .. clear − 1) ⩤
                 (appl (inp, (0 .. clear) ◁ s')).1)⌝
```

SML

```
a (fc_tac [rewrite_rule[z_get_spec⌜ℤ⌝]
  le_dots_lemma1]);
```


ProofPower output

```
...
(* 10 *)  ⌜appl ∈ APPLICATION⌝
...
(*  4 *)  ⌜(0 .. class) ◁ s = (0 .. class) ◁ s'⌝
(*  3 *)  ⌜kernel_implementation ∈ KERNEL⌝
(*  2 *)  ⌜clear ≤ class⌝
(*  1 *)  ⌜0 .. clear ⊆ 0 .. class⌝

(* ?⊢ *)  ⌜(0 .. class) ◁ (s ⊕ (0 .. clear − 1)
              ⩤ (appl (inp, (0 .. clear) ◁ s)).1)
          = (0 .. class) ◁ (s' ⊕ (0 .. clear − 1)
              ⩤ (appl (inp, (0 .. clear) ◁ s')).1)⌝
```

SML

```
val set_lemma_1 = pc_rule1 "z_rel_ext" prove_rule []
  ⌜∀ A, B : 𝕌; x, x' : 𝕌 •
  A ⊆ B ⇒ (B ◁ x) = (B ◁ x') ⇒ (A ◁ x) = (A ◁ x')⌝;
a (ALL_FC_T asm_rewrite_tac[set_lemma_1]);
```


ProofPower output

```
...
(* 10 *)  ⌜appl ∈ APPLICATION⌝
...
(*  4 *)  ⌜(0 .. class ◁ s) = (0 .. class ◁ s′)⌝
...
(*  3 *)  ⌜kernel_implementation ∈ KERNEL⌝
(*  2 *)  ⌜clear ≤ class⌝
(*  1 *)  ⌜0 .. clear ⊆ 0 .. class⌝

(* ?⊢ *)
 ⌜(0 .. class) ◁ (s ⊕ (0 .. clear − 1) ⩤
                    (appl (inp, (0 .. clear) ◁ s′)).1)
 = (0 .. class) ◁ (s′ ⊕ (0 .. clear − 1) ⩤
                     (appl (inp, (0 .. clear) ◁ s′)).1)⌝
```

SML

```
val set_lemma_2 = pc_rule1 "z_rel_ext" prove_rule[]
 ⌜∀ A : 𝕌; x, x′, y : 𝕌 •
   A ◁ x = A ◁ x′ ⇒ A ◁ (x ⊕ y) = A ◁ (x′ ⊕ y)⌝;
a(ALL_FC_T asm_rewrite_tac[set_lemma_2]);
```


state secure - second case

ProofPower output

```
(* *** Goal "4.2" *** *)
(*  9 *)  ⓩappl ∈ APPLICATION⌝
...
(*  3 *)  ⓩ(0 .. class ◁ s) = (0 .. class ◁ s′)⌝
(*  2 *)  ⓩkernel_implementation ∈ KERNEL⌝
(*  1 *)  ⓩ¬ (clear ≤ class)⌝

(* ?⊢ *)
 ⓩ (0 .. class) ◁ (s ⊕ (0 .. clear − 1) ⩤
                   (appl (inp, (0 .. clear) ◁ s)).1)
 = (0 .. class) ◁ (s′ ⊕ (0 .. clear − 1) ⩤
                   (appl (inp, (0 .. clear) ◁ s′)).1)⌝
```

SML

```
(* *** Goal "4.2" *** *)
val set_lemma_3 = pc_rule1 "z_rel_ext" prove_rule[]
  ⓩ∀ A,B:𝕌; x,x′:𝕌; st,st′:𝕌 •
  A ◁ x = A ◁ x′ ⇒ (A ⊆ B)
  ⇒ A ◁ (x ⊕ (B ⩤ st)) = A ◁ (x′ ⊕ (B ⩤ st′))⌝;
a (FC_T (MAP_EVERY ante_tac)
  [rewrite_rule[z_get_specⓩℤ⌝]le_dots_lemma2]
  THEN asm_ante_tac ⓩ(0 .. class) ◁ s = (0 .. class) ◁ s′⌝
  THEN rewrite_tac [set_lemma_3]);
```


The out_secure Subgoal

ProofPower output

```
(∗ ∗∗∗ Goal "6" ∗∗∗ ∗)
(∗  6 ∗)  ⌜appl ∈ APPLICATION⌝
(∗  5 ∗)  ⌜clear ∈ ℕ⌝
(∗  4 ∗)  ⌜inp ∈ IN⌝
(∗  3 ∗)  ⌜s ∈ STATE⌝
(∗  2 ∗)  ⌜s′ ∈ STATE⌝
(∗  1 ∗)  ⌜(0 .. clear) ◁ s = (0 .. clear) ◁ s′⌝


(∗ ?⊢ ∗)
  ⌜(construction (appl, kernel_implementation) (clear, inp, s)).2
 = (construction (appl, kernel_implementation) (clear, inp, s′)).2⌝
```

SML

```
a (MAP_EVERY asm_tac [condec, kidec] THEN
  ALL_FC_T asm_rewrite_tac [conpred, kipred]);
```

SML

```
val kernel_secure_thm = pop_thm();
```

ProofPower output

```
val kernel_secure_thm =
    ⊢ kernel_implementation ∈ secure_kernel : THM
```

A Vending Machine

SML

```
repeat drop_main_goal;
open_theory "usr023";
new_theory "usr023V";
set_flags [
          ("z_type_check_only", false),
          ("z_use_axioms", true)
            ];
```

Z

$price : \mathbb{N}$

Z

VMSTATE

$stock, takings : \mathbb{N}$

Z

VM_operation

$\Delta VMSTATE;$

$cash\_tendered?, cash\_refunded! : \mathbb{N};$

$bars\_delivered! : \mathbb{N}$

Vending Machine Operation Pre-conditions

Z

exact_cash

$cash\_tendered? : \mathbb{N}$

$cash\_tendered? = price$

Z

insufficient_cash

$cash\_tendered? : \mathbb{N}$

$cash\_tendered? < price$

Z

some_stock

$stock : \mathbb{N}$

$stock > 0$

Vending Machine Operations

Z

VM_sale

$$VM\_operation$$

$$stock' = stock - 1;$$
$$bars\_delivered! = 1;$$
$$cash\_refunded! = cash\_tendered? - price;$$
$$takings' = takings + price$$

Z

VM_nosale

$$VM\_operation$$

$$stock' = stock;$$
$$bars\_delivered! = 0;$$
$$cash\_refunded! = cash\_tendered?;$$
$$takings' = takings$$

Z

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

Z

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

Z

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

Exercises 7 : Vending Machine

Turn to Exercises 7 in zed_course_work.doc

1. Prove that the schema VM3 is non-empty. i.e., prove:

$\exists\ VM3 \bullet true$

Hints:

(a) Set the proof context to work with set extensionality by using:

set_pc "$z\_library\_ext$";

(b) Prove this by contradiction using $contr\_tac$.

(c) Try specialising VM3 with a suitable witness.

(d) Does your witness provide values for $cash\_tendered?$, $stock$, $stock'$, $takings$, $takings'$, $cash\_refunded!$ and $bars\_delivered!$?

(e) Is the conclusion $false$? If so, try using $swap\_asm\_concl\_tac$ to help simplify the goal.

(f) Try rewriting with the all the definitions.

(g) Does your goal contain a disjunct, $0 \leq price$? If so try $strip\_asm\_tac\ price$ and rewriting with the assumptions.

Exercises 7 (cont.) : VM Refinement Proof

This exercise is concerned with proving that VM3 is a refinement of VM1. This is a two stage proof.

2. It is useful to prove a lemma that stating that the preconditions $insufficient\_cash$ and $exact\_cash$ are disjoint. i.e., prove:

$$\neg\ (insufficient\_cash \land exact\_cash)$$

Hints:

(a) Set the proof context to work with set extensionality by using:

```
set_pc "z_library_ext";
```

(b) Try rewriting with all the definitions.

(c) If the goal contains inequalities, try rewriting with the specification of $<$. e.g., use z_get_spec. (Avoid looping by using $pure\_rewrite\_tac$.)

(d) z_minus_thm and $plus\_assoc\_thm$ may be useful to normalize any arithmetic expressions.

(e) Repeatedly stripping the goal might be too aggressive; try stripping it in steps, looking for likely opportunities for rewriting with the assumptions.

3. Show that $VM3$ is a refinement of $VM1$. i.e., prove

$$(pre\ VM1 \Rightarrow pre\ VM3) \land (pre\ VM1 \land VM3 \Rightarrow VM1)$$

Hints:

(a) Try rewriting with some of the top-level definitions; the goal can be proved without rewriting will all the definitions!

(b) The lemma proved in part 1 of this exercise will be useful.

(c) If you're stuck, try stripping the goal and seeing what you get.

Exercises 7 : Solutions

For convenience we bind the various specifications to ML variables:

SML

```
val [       price, VMSTATE, VM_operation,
            exact_cash, insufficient_cash, some_stock,
            VM_sale, VM_nosale, VM1, VM2, VM3 ]
   = map z_get_spec [ ⓩprice⌝, ⓩVMSTATE⌝, ⓩVM_operation⌝,
            ⓩexact_cash⌝, ⓩinsufficient_cash⌝, ⓩsome_stock⌝,
            ⓩVM_sale⌝, ⓩVM_nosale⌝, ⓩVM1⌝, ⓩVM2⌝, ⓩVM3⌝ ];
```

SML

```
set_pc "z_library_ext";
set_goal([], ⓩ∃ VM3 • true⌝);
a(contr_tac);
a(z_spec_asm_tac ⓩ∀ VM3 • false⌝
            ⓩ(        cash_tendered? ≙ price,
                      stock ≙ 1, stock' ≙ 0,
                      takings ≙ 0, takings' ≙ price,
                      cash_refunded! ≙ 0,
                      bars_delivered! ≙ 1)⌝);
a(swap_asm_concl_tac
 ⓩ¬ (bars_delivered! ≙ 1, cash_refunded! ≙ 0, cash_tendered? ≙ price,
       stock ≙ 1, stock' ≙ 0, takings ≙ 0, takings' ≙ price)
       ∈ VM3⌝);
```


Exercises 7 : Solutions (cont.)

Proofpower output

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

(*  1 *)  ⌜∀ VM3 • false⌝

(* ?⊢ *)  ⌜(bars_delivered! ≙ 1, cash_refunded! ≙ 0,
                          cash_tendered? ≙ price, stock ≙ 1, stock' ≙ 0,
                          takings ≙ 0, takings' ≙ price)
             ∈ VM3⌝
```

SML

```
a(rewrite_tac[VM1, VM3,
        exact_cash,
        some_stock, VM_sale, VM_operation, VMSTATE]);
a(strip_asm_tac price);
a(asm_rewrite_tac[]);
val VM3_non_empty = pop_thm ();
```

Exercises 7 : Solutions (cont.)

SML

```
set_goal([], ⌜¬ (insufficient_cash ∧ exact_cash)⌝);
a (rewrite_tac [insufficient_cash, exact_cash]);
```

ProofPower output

```
(* ?⊢ *)  ⌜¬
           ((0 ≤ cash_tendered?
              ∧ cash_tendered? < price)
              ∧ 0 ≤ cash_tendered?
              ∧ cash_tendered? = price)⌝
```

SML

```
a (pure_rewrite_tac [z_get_spec ⌜(_ < _)⌝]);
a (rewrite_tac [z_plus_assoc_thm1]);
a (rewrite_tac [z_minus_thm, z_plus_assoc_thm1]);
a (REPEAT_N 3 z_strip_tac);
a (asm_rewrite_tac[]);
val cash_lemma = pop_thm ();
```


Exercises 7 : Solutions (cont.)

To prove the refinement, the previous lemma is useful.

SML

```
set_goal([], ⌜Z (pre VM1 ⇒ pre VM3) ∧ (pre VM1 ∧ VM3 ⇒ VM1)⌝);
a (rewrite_tac [VM1, VM2, VM3]);
```

ProofPower output

```
(* ?⊢ *)  ⌜Z(
 (∃ bars_delivered! : 𝕌;
     cash_refunded! : 𝕌;
     stock′ : 𝕌;
     takings′ : 𝕌
   • exact_cash ∧ some_stock ∧ VM_sale)
   ⇒ (∃ bars_delivered! : 𝕌;
         cash_refunded! : 𝕌;
         stock′ : 𝕌;
         takings′ : 𝕌
       • exact_cash ∧ some_stock ∧ VM_sale
           ∨ insufficient_cash ∧ VM_nosale))
 ∧ ((∃ bars_delivered! : 𝕌;
         cash_refunded! : 𝕌;
         stock′ : 𝕌;
         takings′ : 𝕌
       • exact_cash ∧ some_stock ∧ VM_sale)
     ∧ (exact_cash ∧ some_stock ∧ VM_sale
           ∨ insufficient_cash ∧ VM_nosale)
     ⇒ exact_cash ∧ some_stock ∧ VM_sale)⌝
```


Exercises 7 : Solutions (cont.)

SML

```
a (strip_asm_tac cash_lemma
    THEN asm_rewrite_tac[]);
```

ProofPower output

```
(* 1 *) ⌜¬ insufficient_cash⌝

(* ?⊢ *) ⌜(∃ bars_delivered! : 𝕌;
              cash_refunded! : 𝕌;
              stock' : 𝕌;
              takings' : 𝕌
            • exact_cash ∧ some_stock ∧ VM_sale)
          ∧ exact_cash
          ∧ some_stock
          ∧ VM_sale
        ⇒ exact_cash ∧ some_stock ∧ VM_sale⌝
```

SML

```
a (REPEAT z_strip_tac);
val VM3_refines_VM1 = pop_thm ();
```


Vending Machine Correctness Property

Next we express the requirement that a vending machine does not undercharge:

Z

$$VM\_ok : \mathbb{P}\ \mathbb{P}\ VM\_operation$$

$$\forall\ vm : \mathbb{P}\ VM\_operation \bullet$$
$$vm \in VM\_ok \Leftrightarrow$$
$$(\forall\ VM\_operation \bullet vm \Rightarrow$$
$$takings' - takings \geq price * (stock - stock'))$$

Exercises 8 : Correctness Proof

1. Prove that the Vending Machine VM3 does not undercharge. i.e., prove:

$VM3 \in VM\_ok$

   Hints:

   (a) Set the proof context to work with set extensionality by using:

```
set_pc "z_library_ext";
```

   (b) You will probably need to rewrite the goal with all the definitions.

   (c) Try stripping the goal.

   (d) Do you think that the conclusion is true by dint of arithmetic reasoning?

   If so, you might want to try rewriting with theorems such as z_minus_thm and/or $z\_plus\_assoc1\_thm$.

   (e) $z\_plus\_order\_thm$ may also be useful. You will need to specialise this to some appropriate values if you are going to rewrite with it.

Exercises 8 : Solutions

Before using the definition of VM_ok we convert it into an unconditional rewrite.

SML

```
val VM_ok = z_defn_simp_rule (z_get_spec ⓩVM_ok⌝);
```

ProofPower output

```
val VM_ok = ⊢ ∀ vm : 𝕌
    • vm ∈ VM_ok
      ⇔ vm ∈ ℙ VM_operation
        ∧ (∀ VM_operation
          • vm ⇒ takings′ − takings ≥ price ∗ (stock − stock′)) : THM
```

We now prove that VM3 is a VM_ok.

SML

```
set_pc"z_library_ext";

set_goal([], ⓩVM3 ∈ VM_ok⌝);
a (rewrite_tac [VM1,VM2,VM3,VM_ok,VM_sale,VM_nosale,
                              VM_operation,VMSTATE]);
```


Exercises 8 : Solutions (cont.)

SML

```
a(REPEAT z_strip_tac THEN asm_rewrite_tac[]);
```

Which considerably simplified the problem:

ProofPower output

```
...
(* 2 *)  ⌜cash_refunded! = cash_tendered? + ~ price⌝
(* 1 *)  ⌜takings' = takings + price⌝

(* ?⊢ *)  ⌜price * (stock + ~ (stock + ~ 1)) ≤
                    (takings + price) + ~ takings⌝
...
```

To solve this arithmetic problem, we simplify the lhs of the inequality by
1. pushing in the minus sign

and

2. associating the additions to the left

Exercises 8 : Solutions (cont.)

SML

```
a (rewrite_tac [z_minus_thm, z_plus_assoc_thm1]);
```

which gives the conclusion:

Proofpower output

```
(* ?⊢ *)  ⌜Z price ≤ (takings + price) + ~ takings⌝
```

To solve this problem we move ⌜Z ~ takings⌝ left to place it next to *takings*.

For this we specialise *z_plus_order_thm*:

SML

```
z_plus_order_thm;
```

ProofPower output

```
val it = ⊢ ∀ i : 𝕌
    • ∀ j, k : 𝕌
      • j + i = i + j
      ∧ (i + j) + k = i + j + k
      ∧ j + i + k = i + j + k : THM
```

Exercises 8 : Solutions (cont.)

SML

```
z_∀_elim ⌜Z~ takings⌝ z_plus_order_thm;
```

ProofPower output

```
val it = ⊢ ~ takings ∈ 𝕌 ∧ true
    ⇒ (∀ j, k : 𝕌
      • j + ~ takings = ~ takings + j
        ∧ (~ takings + j) + k = ~ takings + j + k
        ∧ j + ~ takings + k = ~ takings + j + k) : THM
```

SML

```
a (rewrite_tac [z_∀_elim ⌜Z~ takings⌝ z_plus_order_thm]);
a (rewrite_tac [z_plus_assoc_thm1]);
```

SML

```
val VM3_ok_thm = pop_thm();
```

