CO508: Formal Specification using Z

Rogério de Lemos
(based on the slides by Keith Hanna)

- Introduction to Z specification language
- Z schemas

Last Lecture - Summary

The motivation to formally represent complex/critical software systems:

- Software engineering requires some form of mathematical underpinning for a precise description of software systems:

Recap of set theory:

- Set comprehension

  $odd\_no = \{x : \mathbb{N} \mid x \in even\_no \bullet x+1\}$

- Cartesian product

  $\{1, 2\} \times \{a, b\} = \{ (1,a), (2,a), (1,b), (2,b) \}$

- Powerset

  $\mathbb{P} \{a, b\} = \{ \{ \}, \{a\}, \{b\}, \{a, b\} \}$

Exercises

List the following sets:

- $\{n:\mathbb{N} \mid n^2 < 17\}$;
- $\{n:\mathbb{Z} \mid n^2 < 17\}$;
- $\{n:\mathbb{N} \mid n > 2 \wedge n < 4\}$;
- $\{n:\mathbb{N} \mid n < 4 \wedge n > 4 \bullet n\}$.

Describe the following sets:

- $\{0, 3, 6, 9, 12\}$;
- $\{-2, -1, 0, 1, 2\}$.

Exercises

Given A={a,b,c}, B={b, c, d, e} and C={a, b, c, e, f,}, find:

- $\#A$, $A \cup B$, $A \setminus B$, $A \cap (B \cup C)$, $A \cap (B \cap C)$.

Given X={red, blue} and Y={1,2}, write out each of the following as a list:

- $X \times X$, $X \times Y$, $Y \times X$, $\mathbb{P}X$, $\mathbb{P}(\mathbb{P}X)$, $(\mathbb{P}X) \times (\mathbb{P}Y)$, $\mathbb{P}(X \times Y)$, $(\mathbb{P}X) \setminus \{red\}$, $\{S: \mathbb{P}X \mid \#S=2\}$.

If a set X has n members how many members does $\mathbb{P}X$ have?

If a set X has n members and a set Y has m members, how many members does $X \times Y$ have?

Objectives

Introduction to Z specification language

Z schemas:

- Declarations
- Predicates

State schemas:

- Schema inclusion

Operator schemas:

- Schema disjunction

Exercises

The Z Specification Language

It is a widely-used specification language designed during the 1980's.

- It is based on set theory and predicate logic.
- It is strongly typed.
- It augments set theory with constructs ("schemas") allowing specifications to be expressed in a:
  - high-level of abstraction;
  - modular form.

The Z Specification Language

- It aims to specify the desired behaviour of the system in terms of a "model". The model has:
  - a state space (a set, described in terms of simpler sets),
  - a set of operations
    - relations on the 'before' and 'after' states, and input and output variables.
- It has an extensive toolkit (or library) of predefined set theory/logic operators which allow specifications to be defined concisely.

Example of a Z Specification

Part of a Z specification for the well-known "Tower of Hanoi" problem (from *Z in Practice*):

$$
\begin{array}{l}
\textit{Move1} \\
\hline
\Delta\ \textit{Hanoi1} \\
\textit{from!}, \textit{to!} : \mathbb{N} \\
\hline
\{\textit{from!}, \textit{to!}\}? \subseteq 1\,..\,\textit{numberOfPoles} \\
\textit{from!} \neq \textit{to!} \\
\textit{poleseq}_1\ \ \textit{from!} \neq \langle\,\rangle \\
\textit{poleseq}_1' = \\
\quad \textit{poleseq}_1 \oplus \\
\quad \{\textit{from!} \mapsto \textit{front}\ \ \textit{poleseq}_1\ \ \textit{from!}, \\
\quad\ \textit{to!} \mapsto \textit{poleseq}_1\ \textit{to!} \frown \langle \textit{last}\ \ \textit{poleseq}_1\ \ \textit{from!}\,\rangle\} \\
\hline
\end{array}
$$

Principles of Model-based Specification

A system is characterised by:

- Defining a set that will represent the *abstract state* of the system.
- Specifying the allowable *initial states*.
- Specifying the intended effect of each operation by defining a relation between:
  - the state before the operation
  - the inputs
  - the state after the operation
  - the outputs.

What is Z?

Z is a notation based on set theory.

- It is a *typed* notation (the variables have types and every expression must be well-typed).
- It uses *schemas* to group together related declarations and constraints. This allows large, complex specifications to be constructed in a modular way.
- It includes a "toolkit" of useful definitions (this saves having to "reinvent the wheel" every time one writes a specification).

Z Schema

A schema is a basic unit of formal specification, describe:

- the states and the operations of a system;
- the relationship between the states of the system at different levels of abstraction.

Z Schema

There are two ways of writing a schema definition.

- The in-line form:

  SchemaName $\widehat{=}$ [*variable declarations* | *predicates*]

- The boxed form:

```
┌── SchemaName ──────
│ variable declarations
├─────────────
│ predicates
└────────────────
```

Z Schema

A Z specification contains:

- *State schemas* - describe the variables and the relationship between the variables:
  - *system invariant* is a predicate over the relationship between the variables;
    - these predicates should be true in every state of the system.
- *Operator schemas* - describe the operations of the system;
  - *pre-condition* is a predicate that must be satisfied before an operation is performed;
  - *post-condition* is a predicate that must be satisfied after an operation is performed;

Declarations

The purpose of a declaration is to introduce a new identifier and to define its *type.*

A *declaration* is of the form

name : *expression*

where *expression* is a set-valued expression.

Suppose that PERSON is the set of all people, then these are well-formed declarations:

n : $\mathbb{N}$
secretary : PERSON
evens : $\mathbb{P}\ \mathbb{Z}$
students : $\mathbb{P}$ PERSON

Declarations

Note that:

- $\mathbb{N}$ is the set of natural numbers {0, 1, 2, . . . }
- $\mathbb{Z}$ is the set of integers { . . ., -2, -1, 0, 1 , 2 , . . . }
- In Z, types are synonymous with (maximal) sets. Thus, from the declaration n : $\mathbb{N}$, we can infer the predicate n $\in$ $\mathbb{N}$.
- The type $\mathbb{P}\ \mathbb{Z}$ is the set of all subsets of $\mathbb{Z}$. Thus, an element of this set (such as *evens*) is a subset of $\mathbb{Z}$. Hence we can infer
  - evens $\subseteq$ $\mathbb{Z}$

Declarations

- Likewise, *students* is an element of the set of all subsets of PERSON. Hence we can infer
  - students $\subseteq$ PERSON
- These two declarations are *not* well-formed:
  - m : 3 --- "3" is not a set!
  - s : secretary --- "secretary" is not a set!
- [Tricky point!] Even though *s* is declared by
  - s: students

  it is not, in fact, of type *students*.

  Rather, it is the "maximal" set of which *students* is a subset. Thus, *s* is of type PERSON.

Predicates

A *proposition* is an expression which is either true or false.

- Keanu Reeves played in 'The Matrix'

A *predicate* is an expression containing one or more variables which, depending on the values taken by the variables, is either true or false.

- Keanu Reeves played in *film*
  - Keanu Reeves played in 'Speed' (T)
  - Keanu Reeves played in 'The Phantom Menace' (F)
- $n > 3$
- $s \in$ students
- $\forall m : \mathbb{Z} \bullet (m \in evens) \Rightarrow ((m+1) \notin evens)$

Predicates

Universal quantification ($\forall$ - for all)

- An expression that talks about all of the elements of a set;
  - $\forall$ s : Student $\bullet$ s has a student number
  - $\forall x : \mathbb{N} \bullet x < x+1$

Existential quantification ($\exists$ - there exists)

- At least one element of the set satisfies an expression;
  - $\exists$ s : Student $\bullet$ s does have a library card
  - $\exists x : \mathbb{N} \bullet x < 1$

State Schemas

Here is a schema that introduces a state space that could be used in modelling a timer:

Timer

minute : $\mathbb{N}$
hour : $\mathbb{N}$

minute < 60
hour < 24

This schema:

- introduces two variables (*minute* and *hour*);
- constrains the values they can take.

State Schemas

The two predicates are implicitly joined by an "$\wedge$".

The $\wedge$ connective is explicitly present if the same schema is written in linear form:

- Timer $\widehat{=}$ [minute, hour : $\mathbb{N}$ | (minute < 60) $\wedge$ (hour < 24)]

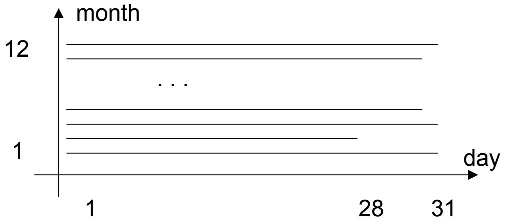
State Schemas

A schema that describes the allowable states of a calendar:

Calendar

day : $\mathbb{N}$
month : $\mathbb{N}$

$0 < month \leqslant 12$
$0 < day \leqslant nDays(month)$



KENT UNIVERSITY OF KENT AT CANTERBURY

Data Type Declaration

Often we need to be able to declare simple data types. Such declarations take the form

$newType ::= ident_1 \mid \ldots \mid ident_n$

For example, to introduce a new type named *RESPONSE* having two values, *ok* and *fail*, we write:

RESPONSE ::= ok | fail

KENT UNIVERSITY OF KENT AT CANTERBURY

Schema Inclusion

One schema can be *included* in another simply by including its name in the *declarations* section of a schema.

For example, the schema defined by

TimeStamp

Timer
Calendar

is an abbreviation for the schema:

KENT UNIVERSITY OF KENT AT CANTERBURY

Schema Inclusion

TimeStamp

minute : $\mathbb{N}$
hour : $\mathbb{N}$
day : $\mathbb{N}$
month : $\mathbb{N}$

$minute < 60$
$hour < 24$
$0 < month \leqslant 12$
$0 < day \leqslant nDays(month)$

Exercises

Write the following expressions in English, and state whether they are true or false:

- $\forall n:\mathbb{N} \bullet \exists m:\mathbb{N} \bullet n+m=2$
- $\exists n:\mathbb{N} \bullet \forall m:\mathbb{N} \bullet n+m=2$
- $\exists n:\mathbb{N} \bullet \exists m:\mathbb{N} \bullet n+m=2$

Exercises

List the following set:

- $\{n:\mathbb{N} \mid n < 7 \land \exists m:\mathbb{N} \bullet 3m=n\}$

Given A={p, q} and B={1, 2, 3}, state whether the following predicates are true:

- $\forall S : \mathbb{P}A \bullet \#S = 2$
- $\forall (x,y) : B \times B \bullet x = y$
- $\exists (x,y) : B \times B \bullet x + y = 6$