FMSE: Lecture 1

The Specification Language Z:
Introduction

Goals of Lecture 1

At the end of this lecture you should be able to:

- write down schemas for simple specification problems
- use quantors, sets and functions in schema invariants
- understand the role of types in Z

The Z Notation

A formal method for software specification and design.

Ingredients:

- set theory and predicate calculus
- typing
- a schema notation for system states and operations

An Example

Specify a library system where registered readers can borrow books from the collection.

- no more than *maxloan* books can be borrowed per reader
- operations: issueing and returning books, registering books and readers, queries (will be covered by the next lectures)

The Library System

$$maxloan : \mathbb{N}$$

$$maxloan > 0$$

$$[BOOK, READER]$$

Library

$$collection : \mathbb{P}\, BOOK$$
$$readers : \mathbb{P}\, READER$$
$$issued : BOOK \pfun READER$$

$$\operatorname{dom} issued \subseteq collection$$
$$\operatorname{ran} issued \subseteq readers$$
$$\forall\, r : \operatorname{ran} issued \bullet \#\{b : BOOK \mid issued(b) = r\} \leq maxloan$$

Axiomatic Descriptions

Z allows for the definition of properties of *global constants* by way of *axiomatic descriptions*.

In our example, we wish to have a maximum to the number of books a reader can borrow. We do not want to specify a value for this maximum, only that it should be bigger than zero.

$$\begin{array}{|l} maxloan : \mathbb{N} \\ \hline maxloan > 0 \end{array}$$

An axiomatic description consists of two parts:

- the *declaration*: *maxloan* is a nonnegative integer
- the *predicate*: *maxloan* must be bigger than 0.

Basic Types

Declaration of the basic types, in our example:

$$[BOOK, READER]$$

This postulates two basic types *BOOK* and *READER* without any properties or structure.

Abstract, so representational issues are not addressed.

Do not try to create e.g. character strings or cartesian products of e.g. names and id's for *BOOK* or *READER* !

specification $\neq$ programming

State Schemas

- A schema has a *name*, here: *Library*
- above the line: a *declaration* of *state variables* (typed!)
  - the values of these variables consitute the state of the system
  - these values can be initialised and changed by operations
- below the line: *invariants*
  - must be true before and after all operations
  - together characterize the admitted states

Sets

Examples of sets in Z:

- $\mathbb{N}$, the set of *natural* numbers: $0, 1, 2, 3, \ldots$ (predefined)
- $\mathbb{N}_1$, the set of *strictly positive integers*: $1, 2, 3, \ldots$ (predefined)
- $\mathbb{Z}$, the set of *integers*: $\ldots, -2, -1, 0, 1, 2, \ldots$ (predefined)
- $\{1, 2, 3, 4, 5, 6\}$ or 1..6 (example of two equivalent set definitions)
- many other constructions (see book).

Types

- Sets in Z are *typed*; elements of the same set must have the same *type*.

  For example, the set $\{2, 4, red, yellow, 6\}$ is NOT well-typed.

- Types enforce structure and discipline, and make it easier to detect errors in a specification.

- Typechecking can be done automatically (see e.g. the tool Z/Eves).

Defining Types

- there is one *predefined* type: $\mathbb{Z}$
- using *free type* definitions, for example,

  $COLOR ::= red \mid green \mid blue \mid yellow \mid cyan \mid white \mid black$
- using *basic type* definitions, for example,

  $$[NAME]$$
- using the *power set operator*: $\mathbb{P}\,\mathbb{Z}$, $\mathbb{P}\,COLOR$, $\mathbb{P}\,NAME$
- using the *Cartesian product operator*: $\mathbb{Z} \times \mathbb{Z}$, $NAME \times COLOR$, etc.

Declarations

- simple declarations of the form *variable* : *set*, for example:

$$i : \mathbb{Z};\ d_1, d_2 : 1..6;\ signal : \{red, yellow, green\}$$

(What are the *types* of i, d_1, d_2 and $signal$?)

- *constrained* declarations, for example:

$$d_1, d_2 : 1..6$$
$$\overline{}$$
$$d_1 + d_2 = 7$$

$$signal : COLOR$$
$$\overline{}$$
$$signal \in \{red, yellow, green\}$$

Pairs & binary relations

A *binary relation* is a set of *pairs*. Example:

$$PHONE == 0 \mathrel{..} 9999$$

$$phone : NAME \leftrightarrow PHONE \qquad [or : \mathbb{P}(NAME \times PHONE)]$$

$$\begin{aligned}
& phone = \{\ldots \\
& (aki, 4117), \\
& (philip, 4107), \\
& (doug, 4107), \\
& (doug, 4136), \\
& (philip, 0113), \\
& (frank, 0110), \\
& (frank, 6190), \\
& \ldots\}
\end{aligned}$$

Domain & Range

For a (binary) relation $R : NAME \leftrightarrow PHONE$ we define

- the *domain* of R, $\operatorname{dom} R = \{x : A \mid \exists\, y : B \bullet (x, y) \in R\}$, i.e. the set of all first elements of pairs in R.
- the *range* of R, $\operatorname{ran} R = \{y : B \mid \exists\, x : A \bullet (x, y) \in R\}$, i.e. the set of all second elements of pairs in R.

Example:

$$\operatorname{dom} phone = \{\ldots, aki, philip, doug, frank, \ldots\}$$
$$\operatorname{ran} phone = \{\ldots, 4117, 4107, 4136, 0113, 0110, 6190, \ldots\}$$

Functions

A *function* $f : A \leftrightarrow B$ is a relation such that each element of $\operatorname{dom} f$ is linked to precisely one element $f(x)$ of $\operatorname{ran} f$, or more formally

$$\forall\, x : \operatorname{dom} f \bullet \#\{y : B \mid (x, y) \in f\} = 1$$

- instead of $f(x)$ we also write $f\ x$ (*function application*)
- $A \rightarrowtail\!\!\!\!\!\!+\;\, B$ denotes the set of all (partial) functions in $A \leftrightarrow B$, enabling declarations of the form $f : A \rightarrowtail\!\!\!\!\!\!+\;\, B$.
  We call such a function *partial* because it does not need to be defined for all $x \in A$.

Total Functions & Injections

- a function $f : A \pfun B$ is a *total function* if $f(x)$ is defined for every element of its source set A, i.e. if $\operatorname{dom} f = A$.

  We write $A \longrightarrow B$ for the set of total functions in $A \pfun B$.

- a function $f : A \pfun B$ is *injective* or *one-to-one* if different elements of $\operatorname{dom} f$ are mapped to different elements or $\operatorname{ran} f$, i.e.

  $$\forall x_1, x_2 : \operatorname{dom} f \bullet x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$$

  We write $A \pinj B$ for the set of injective functions in $A \pfun B$

Logical Connectives

Z has the following standard logical operators:

- *negation*: $\neg\, p$ — (*not p*)
- *conjuction*: $p \wedge q$ — (*p and q*)
- *disjuction*: $p \vee q$ — (*p or q*)
- *implication*: $p \Rightarrow q$ — (*p implies q* or *if p then q*)
- *equivalence*: $p \Leftrightarrow q$ — (*p if and only if q*)

Quantifiers

Quantification introduces *local* variables into predicates:

- *universal quantification*:

  $\forall\, declaration \bullet predicate$ (*for all ... it holds that ...*)

  Example:

  $$\begin{array}{|l} divides : \mathbb{Z} \leftrightarrow \mathbb{Z} \\ \hline \forall\, d, n : \mathbb{Z} \bullet d\ \underline{divides}\ n \Leftrightarrow n \mathsf{\ mod\ } d = 0 \end{array}$$

- *existential quantification*:

  $\exists\, declaration \bullet predicate$ (*there exist ... such that ...*)

  Example: $\exists\, i : ns \bullet i \leq nmax$

Z and Boolean Types

Z does *not* have a built-in Boolean type !!!

So we do NOT write something like:

$$odd : \mathbb{Z} \longrightarrow BOOLEAN$$

$$\forall n : \mathbb{Z} \bullet odd(n) \Leftrightarrow \exists m : \mathbb{Z} \bullet n = 2 * m + 1$$

But we write:

$$odd\_ : \mathbb{P}\,\mathbb{Z}$$

$$\forall n : \mathbb{Z} \bullet$$
$$odd(n) \Leftrightarrow \exists m : \mathbb{Z} \bullet n = 2 * m + 1$$

Set Comprehensions

Sets can be defined using the *set comprehension* format

$$\{declaration \mid predicate\}$$

We can define, for example:

- the set of non-zero numbers:

  $NONZERO == \{i : \mathbb{Z} \mid i \neq 0\}$

- the point on a line with slope m and intercept b

  $line == \{x, y : \mathbb{Z} \mid y = m * x + b\}$

  The elements of this set are the *characteristic tuples* of *line* and have the form (x, y) with $y = m * x + b$.

An Alternative

A set-valued function for *issued*:

Library

$\ldots$

$issued : READER \pfun \mathbb{P}\, BOOK$

$\mathrm{dom}\, issued \subseteq readers$

$\mathrm{ran}\, issued \subseteq \mathbb{P}\, collection$

$\forall\, r : \mathrm{dom}\, issued \bullet \#issued(r) \leq maxloan$

$\forall\, r, r' : \mathrm{dom}\, issued \bullet r \neq r' \Rightarrow issued(r) \cap issued(r') = \varnothing$

Another Alternative

instead of *collection*, record what is on the shelve:

Library

$on\_shelve : \mathbb{P}\, BOOK$

$readers : \mathbb{P}\, READER$

$issued : BOOK \nrightarrow READER$

$\mathrm{dom}\, issued \cap on\_shelve = \varnothing$

$\mathrm{ran}\, issued \subseteq readers$

$\forall\, r : \mathrm{ran}\, issued \bullet \#\{b : BOOK \mid issued(b) = r\} \leq maxloan$

Yet Another Alternative

Record both the collection and what is on the shelve:

Library

$collection : \mathbb{P}\, BOOK$

$on\_shelve : \mathbb{P}\, BOOK$

$readers : \mathbb{P}\, READER$

$issued : BOOK \pfun READER$

$\mathrm{dom}\, issued \cup on\_shelve = collection$

$\mathrm{dom}\, issued \cap on\_shelve = \varnothing$

$\mathrm{ran}\, issued \subseteq readers$

$\forall\, r : \mathrm{ran}\, issued \bullet \#\{b : BOOK \mid issued(b) = r\} \leq maxloan$

Note that there is now redundancy (which can be convenient)

Multiple Copies Of Books

$[TITLE, COPY, READER]$

$$title : COPY \longrightarrow TITLE$$

Library

$collection : \mathbb{P}\, COPY$

$readers : \mathbb{P}\, READER$

$issued : COPY \nrightarrow READER$

$\operatorname{dom} issued \subseteq collection$

$\operatorname{ran} issued \subseteq readers$

$\forall\, r : \operatorname{ran} issued \bullet \#\{b : COPY \mid issued(b) = r\} \leq maxloan$

A Variant

We use *collection* for recording the relation between titles and copies:

$$[TITLE, COPY, READER]$$

Library

$$collection : COPY \leftrightarrow TITLE$$
$$readers : \mathbb{P}\, READER$$
$$issued : COPY \leftrightarrow READER$$

$$\mathrm{dom}\, issued \subseteq \mathrm{dom}\, collection$$
$$\mathrm{ran}\, issued \subseteq readers$$
$$\forall\, r : \mathrm{ran}\, issued \bullet \#\{b : COPY \mid issued(b) = r\} \leq maxloan$$

Conclusions

- There are in general many alternative solutions to a specification problem.
- Which solution to choose is dependent on personal style and preference.
- Which solution is chosen will affect how easy it is to specify certain operations (see next lecture).
- Making a formal specification helps to think about a problem!