LECTURE 11: Z

Software Engineering

Mike Wooldridge

1 Introduction

- In this lecture, we introduce *schemas*, the most distinctive feature of the Z specification language.
- We show how a simple computer system can be specified in Z.

Model-Based Specification

- Z — like VDM, its main rival — is a *model-based specification framework*.
- The idea is to contruct an *abstract model* of the system we desire to build.

  This model is:
  - *high level*;
  - *idealised*;
  - *does not detail with implementation specifics*.
- What does the model consist of?
  - description of system *state space*;
  - description of system *operations*.
- System state-space is the set of all states that the system could be in.
- The state of a system describes the value of each variable (and memory location).

- The most fundamental operation we use is the assignment statement, ':=' ... such statements *change the state of a system.*
- In Z, we represent the state space of a system as a collection of functions, sets, relations, sequences, bags, etc., together with a collection of *invariant properties* on these objects.
- These invariant properties describe regularities between state changes.
- How about operations? What level of abstraction to we deal with them? Lowest level would be assignment statement level.

  We start with more abstract descriptions.
- Operations are usually defined in terms of *pre-* and *post-* conditions.
- Operations define acceptable state transitions.

2 Schemas

- The Z schema is a 2-dimensional graphical notation for describing:
  - state spaces;
  - operations.
- **Definition:** A vertical-form schema is either of the form

$$\begin{array}{|l}
\hline
\textit{SchemaName} \\
\textit{Declarations} \\
\hline
\textit{Predicate}_1; \ \cdots ; \ \textit{Predicate}_n \\
\hline
\end{array}$$

  or of the form

$$\begin{array}{|l}
\hline
\textit{SchemaName} \\
\textit{Declarations} \\
\hline
\end{array}$$

- In the latter case, the predicate part is assumed to be 'true'.

- Once introduced, *SchemaName* will be associated with the schema proper, which is the contents of the box.
- The declarations part of the schema will contain:
  - a list of variable declarations; and
  - references to other schemas (this is called schema inclusion).
- Variable declarations have the usual form:

$$x_1, x_2, \ldots, x_n : T;$$

- The predicate part of a schema contains a list of predicates, separated either by semi-colons or new lines.

2.1 State Space Schemas

- Here is an example state-space schema, representing part of a system that records details about the phone numbers of staff.

  (Assume that *NAME* is a set of names, and *PHONE* is a set of phone numbers.)

$$
\begin{array}{|l}
\hline
\textit{PhoneBook} \\
\hline
\textit{known} : \mathbb{P}\,\textit{NAME} \\
\textit{tel} : \textit{NAME} \pto \textit{PHONE} \\
\hline
\operatorname{dom} \textit{tel} = \textit{known} \\
\hline
\end{array}
$$

- The declarations part of this schema introduces two variables: *known* and *tel*.
- The value of *known* will be a subset of *NAME*, i.e., a set of names.

  This variable will be used to represent all the names that we know about — those that we can give a phone number for.
- The value of *tel* will be a partial function from *NAME* to *PHONE*, i.e., it will associate names with phone numbers.

- The declarations part is separated from the predicate part by the horizontal line.
- The predicate part contains the following invariant:

  The domain of *tel* is always equal to the set *known*.

2.2 Operation Schemas

- In specifying a system operation, we must consider:
  - the objects that are *accessed* by the operation, and of these:
    - the objects that are *known to remain unchanged* by the operation (cf. value parameters);
    - the objects that *may be altered* by the operation (cf. variable parameter);
  - the *pre-conditions* of the operation, i.e., the things that must be true for the operation to succeed;
  - the *post-conditions* — the things that will be true after the operation, if the pre-condition was satisfied before the operation.

- Return to the telephone book example, and consider the 'lookup' operation: we put a name in, and get a phone number out.
  - this operation accesses the *PhoneBook* schema;
  - it does not change it;
  - it takes a single 'input' — a name for which we want to find a phone number;
  - it produces a single output — a phone number.
  - it has the pre-condition that the name is known to the database.
- Here is a Z schema specifying the lookup operation:

Find

$$\Xi PhoneBook$$
$$name? : NAME$$
$$phone! : PHONE$$

$$name? \in known$$
$$phone! = tel(name?)$$

This illustrates the following Z conventions:

- placing the name of the schema in the declarations part 'includes' that schema — it is as if the variables were declared where the name is;
- 'input' variable names are terminated by a question mark;
- ... the only input is $name?$
- 'output' variables are terminated by an exclamation mark;
- ... the only output is $phone!$
- the Ξ (Xi) symbol means that the *PhoneBook* schema is not changed;
- if we have written a Δ (delta) instead of Ξ, it would mean that the *PhoneBook* schema *did* change.
- the pre-condition is that $name?$ is a member of *known*;
- the post-condition is that $phone!$ is set to $tel(name?)$.

- Here is another schema: this one add's a name/phone pair to the phone book.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\mathit{AddName}\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta \mathit{PhoneBook} \\
\mathit{name}? : \mathit{NAME} \\
\mathit{phone}? : \mathit{PHONE} \\
\hline
\mathit{name}? \notin \mathit{known} \\
\mathit{tel}' = \mathit{tel} \cup \{\mathit{name}? \mapsto \mathit{phone}?\} \\
\hline
\end{array}
$$

- This schema accesses *PhoneBook* and *does* change it (hence the use of Δ rather that Ξ.)
- Two inputs: a name ($\mathit{name}?$) and phone number ($\mathit{phone}?$).
- Pre-condition: the name is not already in the database.
- Post-condition: *tel* after the operation is the same as *tel* before the operation with the addition of maplet $\mathit{name}? \mapsto \mathit{phone}?$.
- Appending a $'$ to a variable means 'the variable *after* the operation is performed'.

- EXERCISE. Rewrite this schema to get rid of post-condition, and allow overwriting of existing names.

3 CADIZ

- CADIZ is an automated checker and typesetter for Z specifications.
- It takes as its input a plain ASCII file, prepared using an ordinary text editor. This file contains various instructions describing Z schemas.
- It then performs some checks on this specification, and depending on what command-line options you gave, it will:
  - typeset your spec., producing a binary file with a `.dit` extension, which can be printed off with the `printz` command;
  - allow you to browse through the spec., and get feedback on certain parts of it.