Department of Computer Science, CUI Lahore Campus

Formal Methods

By

Farooq Ahmad

Schemas and Software Specifications

Schemas, state Schemas, operation schemas

Topics covered

- Schemas
- State schemas
- Operation schemas
- Case study: counter

Z as modeling language

- Z is a model-based notation and use state machine model.
- **In Z you usually model a system by representing its state** which is a collection of state variables and their values.
- And we model some operations that can change its state.

Z for object-oriented programming

- Z is also a natural fit to object-oriented programming.
- Z state variables are like instance variables, and the operations are like methods; Z even provides a kind of inheritance.
- However, Z is not limited just to Abstract Data Types (ADTs) and object- oriented style; you can also use Z in a functional style, among others.

Schema in Z

- The most visible kind of Z paragraph is a naming construct called the **schema**.
- A schema is a piece of mathematical text that specifies some aspect of the software system.
- Schema provides structure to the mathematical text: It provides several structuring constructs called paragraphs.
- Z defines a schema calculus you can use to build big schemas from small ones.
- The schema is the feature that most distinguishes Z from other formal notations.

Use of schemas in Z specification language

- question: how can one structure and combine system descriptions in a specification?
- approach: schemas are used for this purpose in Z
- Now we see how to combine declarations and predicates into structures called schemas.
- Schemas help us model computing systems because they can represent state — memory or storage — and changes of state.
- A schema can be used to describe the abstract state of an abstract data type when the state machine model is used.
- A schema represents a system's state.
- A collection of schemas models the behavior of a computer system.

common distinction, Use case

- state schemas: define variables of a system and their relations
- operation schemas: define the functions that change the state of a system
- Use case: use case represents a set of actions performed by a system for a specific goal.
- Operation schemas, therefore, is a formal representation of use cases of a system

schema

general syntax:

SchemaName

declaration (of variables)

predicates

remarks:

- variables can be listed or separated by semicolons
- predicates in different lines are implicitly conjoined

The system: An example of a counter

- The system we shall look at is a **simple counter**. A counter would be used, for example, to count the number of vehicles passing a census point, the number of people entering a stadium or the number of fleas in a bird's nest.
- When the counter is clicked, its value is increased by one. When the reset button is pressed, the counter's value becomes zero.

System state schema

- Variables are derived from what an object has. A counter has just one variable that we shall name *count*. *count can equal, but never exceed, maximum*.
- A *schema is a set of variables together with a set of predicates constraining those variables.* A schema is drawn as an open box - see below.

Counter

$count : \mathbb{N}$

$count \leq maximum$

Axiomatic definition

The maximum value that count can reach is set or fixed. We express this in an axiomatic definition or description.

$$\begin{array}{|l} maximum : \mathbb{N} \\ \hline maximum = 9999 \end{array}$$

The predicate is optional. We could have written

$$\begin{array}{|l} maximum : \mathbb{N} \end{array}$$

and left its actual value unsaid.

Variables defined in axiomatic definitions are **global**: They can be used anywhere in a Z text after their definition.
This is called declaration before use.

Initial state

We describe the state the system is in when it is first started. The *InitCounter* schema defined below describes the initial state of *Counter*.

```
┌─ InitCounter ──────────────────
│ Counter
├──────────
│ count = 0
└────────────────────────────────
```

The line *Counter* says include all the variables defined in the *Counter* schema (*count* in this instance) and all the predicates ($count \leqslant maximum$ in this instance). To illustrate the point we write out *InitCounter* in full below.

```
┌─ InitCounter ──────────────────
│ count : ℕ
├──────────
│ count ≤ maximum
│ count = 0
└────────────────────────────────
```

Change system state

The value of *count* does not remain the same forever. From time to time the counter will be clicked and the value stored in *count* will be moved on. The *Click* schema shown below updates the *count* system state variable.

Click

$\Delta Counter$

$count < maximum$

$count' = count + 1$

The Δ naming convention is just an abbreviation for the schema that includes both the unprimed "before" state and the primed "after" state. The Δ is not an operator;

Query system state

The schema *QueryCount* shown below outputs the current value of *count*.

QueryCount

$\Xi Counter$

$count! : \mathbb{N}$

$count! = count$

$\Xi Counter$ (say Xi Counter) says include all the variables and predicates defined in the *Counter* schema; the values stored in these variables will not change.

The declaration $count! : \mathbb{N}$ says count output is drawn from the set $\mathbb{N}$. The ! mark stands for output.

The predicate $count! = count$ says count output is the same as the (system variable) *count*.

Cancel, undoing the click operation

Cancel undoes a *Click* operation. We require that *count* is more than zero.

Cancel

$\Delta Counter$

$count > 0$

$count' = count - 1$

Summary of the lecture: Conclusion

- Concept of schema
- System state schema
- Operation schema
- Case study: Counter

Reference and reading material

- Chapter 5: Section 5.1.1 to 5.1.4 of the book "Software Development with Z"