8 SCHEMA CALCULUS

In Chapter Six we saw how declarations and predicates were combined in structures called schemas. We saw that a schema represents a system's state, that a schema describes the change in a system, and that a collection of schemas models the behaviour of a computer system.

Now we go on to see how larger schemas may be formed by combining smaller ones using conjunction (and) and disjunction (or).

8.1 THE TWO SETS MODEL

The Scenario

First, we set the scene. Students may join the Z class providing the class is not full. Those who successfully complete all the assignments may leave with a certificate.

Given Sets

We are not concerned with details such as students' number, name, date of birth and so on. So we introduce the basic type *STUDENT* as a given set.

[*STUDENT*]

Axiomatic Definitions

There is a limit to the number of students who can join the class. We define *maxClassSize*, the maximum number of students that can be in the class. Here, we have set this size arbitrarily (for no special reason) to 20 even though its actual value is not relevant.

$$\begin{array}{|l}
maxClassSize : \mathbb{Z} \\
\hline
maxClassSize = 20
\end{array}$$

System State Scenario

We model the class system as two sets of students. *enrolled* is the set of students who have joined the class. *passed* is a subset of *enrolled* and represents the set of enrolled students who have passed their assignments. Those who have enrolled but not yet passed is represented by *enrolled* \ *passed*. The size of the class is constrained by *maxClassSize*.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Class}} \\
\hline
\mathit{enrolled} : \mathbb{F}\ \mathit{STUDENT} \\
\mathit{passed} : \mathbb{F}\ \mathit{STUDENT} \\
\hline
\#\mathit{enrolled} \leqslant \mathit{maxClassSize} \\
\mathit{passed} \subseteq \mathit{enrolled} \\
\hline
\end{array}$$

The Initial State

To begin with there are no enrolled students and no student has passed.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{InitClass}} \\
\hline
\mathit{Class} \\
\hline
\mathit{enrolled} = \emptyset \\
\mathit{passed} = \emptyset \\
\hline
\end{array}$$

Enrol

A student may join the class if the class is not already full and if the student has not already enrolled. A new student cannot have passed all their assignments.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Enrol}} \\
\hline
\Delta \mathit{Class} \\
\mathit{student?} : \mathit{STUDENT} \\
\hline
\#\mathit{enrolled} < \mathit{maxClassSize} \\
\mathit{student?} \notin \mathit{enrolled} \\
\mathit{enrolled}' = \mathit{enrolled} \cup \{\ \mathit{student?}\ \} \\
\mathit{passed}' = \mathit{passed} \\
\hline
\end{array}$$

The ? in *student*? means input.

Complete

An existing student is transferred to the *passed* set provided they have passed all their assignments and have not already been transferred. Every student in *passed* must also be in *enrolled*. *enrolled* remains unchanged.

Complete

$\Delta Class$

$student? : STUDENT$

$student? \subseteq enrolled$

$student? \notin passed$

$passed' = passed \cup student?$

$enrolled' = enrolled$

Leave With Certificate

Only those existing students who have passed may leave with a certificate; they are removed from both *passed* and *enrolled*.

LeaveWithCertificate

$\Delta Class$

$student? : STUDENT$

$student? \in passed$

$passed' = passed \setminus \{ student? \}$

$enrolled' = enrolled \setminus \{ student? \}$

EXERCISE 8.1

1 Explain each line of Z introduced so far in Section 8.1 above as if to a beginning Z student.

2 Write and explain the schema *LeaveWithoutCertificate* that specifies the process of a student leaving the class without having passed all their assignments.

3 Write and explain the schema *ReportNumberEnrolled* that outputs the number of students who are currently enrolled.

8.2 FREE TYPE DEFINITIONS

In Section 8.1 above, in which we introduced the Z class system, we were concerned with the simple, straightforward, no problem scenarios. For example, we did not concern ourselves with the possibility that the class is full and so no one can be enrolled on it. We ignored the possibility that a student could be enrolled twice. We ignored the possibility that a student who is not enrolled could be transferred to the passed set. We now address these error scenarios.

First, we draw up a table of pre-conditions, the set of states for which successful outcomes are defined. We add on to that table the conditions for failure.

Schema	**Pre-condition for success**	**Conditions for failure**
Enrol	$\#enrolled < maxClassSize$ $student? \notin enrolled$	class full: $\#enrolled \geqslant maxClassSize$ already enrolled: $student? \in enrolled$
Complete	$student? \in enrolled$ $student? \notin passed$	not enrolled: $student? \notin enrolled$ already passed: $student? \in passed$
LeaveWithCertificate	$student? \in passed$	not passed: $student? \notin passed$

Then, in a *free type definition*, we define *REPORT* to be the set of values that describe either a schema's success or the reasons for its failure.

REPORT ::= *ok* | *classFull* | *alreadyEnrolled* | *notEnrolled* | *alreadyPassed* | *notPassed*

::= stands for free type definition. The | separates one element from the next. A variable of type *REPORT* has a value drawn from the given list. Notice that each element name begins with a lower case letter and contains no spaces.

EXERCISE 8.2

1 For the schema *LeaveWithoutCertificate* written in Exercise 8.1 above define and explain

- **a** the pre-conditions for success
- **b** the conditions for failure

2 Which two values of the *REPORT* type defined in Section 8.2 above include the reasons why the *LeaveWithoutCertificate* process could fail.

8.3 SUCCESS AND ERROR SCHEMAS

The *Success* schema has just one declaration and one predicate. It will be combined with other schemas to indicate their successful outcome.

Success

Success outputs *ok*.

Success

$report! : REPORT$

$report! = ok$

Class Full

We define a schema for each identified error case. We do not expect an error case to update any system variables.

The class is full if the number of enrolled students has reached (or by some mistake has exceeded) the maximum class size.

ClassFull

$\Xi Class$

$report! : REPORT$

$\#enrolled \geqslant maxClassSize$

$report! = classFull$

Already Enrolled

A student cannot be enrolled again if they are already enrolled.

AlreadyEnrolled

$\Xi Class$

$student? : STUDENT$

$report! : REPORT$

$student? \in enrolled$

$report! = alreadyEnrolled$

Not Enrolled

If a student is not enrolled they cannot be passed.

$$\begin{array}{|l}
\textit{NotEnrolled} \\
\hline
\Xi \textit{Class} \\
\textit{student}? : \textit{STUDENT} \\
\textit{report}! : \textit{REPORT} \\
\hline
\textit{student}? \notin \textit{enrolled} \\
\textit{report}! = \textit{notEnrolled} \\
\hline
\end{array}$$

Already Passed

The same student cannot be passed twice.

$$\begin{array}{|l}
\textit{AlreadyPassed} \\
\hline
\Xi \textit{Class} \\
\textit{student}? : \textit{STUDENT} \\
\textit{report}! : \textit{REPORT} \\
\hline
\textit{student}? \in \textit{passed} \\
\textit{report}! = \textit{alreadyPassed} \\
\hline
\end{array}$$

Not Passed

A student who has not passed cannot leave with a certificate.

$$\begin{array}{|l}
\textit{NotPassed} \\
\hline
\Xi \textit{Class} \\
\textit{student}? : \textit{STUDENT} \\
\textit{report}! : \textit{REPORT} \\
\hline
\textit{student}? \notin \textit{passed} \\
\textit{report}! = \textit{notPassed} \\
\hline
\end{array}$$

EXERCISE 8.3

Explain each line of Z introduced in Section 8.3 above, as if to a beginning Access student.

8.4 SCHEMA CALCULUS

We use conjunction (and) to combine two schemas.

$$Enrol \wedge Success$$

Say *enrol and success*.

We use disjunction (or) to represent alternatives.

$$(Enrol \wedge Success) \vee ClassFull \vee AlreadyEnrolled$$

Say *enrol ok and success, or class full, or already enrolled*.

Now we can define the total, complete version of the enrol process.

$$EnrolTot \mathrel{\widehat{=}} (Enrol \wedge Success) \vee ClassFull \vee AlreadyEnrolled$$

$\widehat{=}$ stands for *schema definition*. So, the *EnrolTot* schema is defined to be *Enrol* and *Success*, or *ClassFull*, or *AlreadyEnrolled*.

Similarly,

$$CompleteTot \mathrel{\widehat{=}} (Complete \wedge Success) \vee NotEnrolled \vee AlreadyPassed$$

$$LeaveWithCertificateTot \mathrel{\widehat{=}} (LeaveWithCertificate \wedge Success) \vee NotPassed$$

EXERCISE 8.4

1 Write and explain the total process *LeaveWithoutCertificate*.

2 A hotel management system is used to maintain a record of the current state of the hotel rooms, whether occupied or not. Write and explain a Z specification for the system described below. A full explanation includes:

- example data and sets
- ordinary English prose describing what is specified - as if to a non computing specialist, and
- a technical explanation of the Z you used - as if to a beginning Z student.

Use Case	**Commission**
Goal	To add a new room to the Accommodation System
Pre-condition	The room has not already been added
Initiating Actor	Accommodation Manager

Main Success Scenario

1 Manager inputs new room
2 System confirms new room added
3 Exit success

Exceptions

2a Room already in the system
2a1 Exit failure

Use Case	**Occupy**
Goal	To inform the system that a room is now occupied by a guest
Pre-condition	The room is in the Accommodation System and the room is not occupied
Initiating Actor	Receptionist

Main Success Scenario

1 Receptionist inputs room
2 System confirms room is now occupied
3 Exit success

Exceptions

2a Room not in the system
2a1 Exit failure
2b Room already occupied
2b1 Exit failure

Use Case	**Vacate**
Goal	To inform the system that the room is now vacant
Pre-condition	The room is in the Accommodation System and the room is currently occupied
Initiating Actor	Receptionist

Main Success Scenario

1 Receptionist inputs room
2 System confirms room is now vacant
3 Exit success

Exceptions

2a Room not in system
  2a1 Exit failure
2b Room already vacant
  2b1 Exit failure

Use Case	**Query Occupied Rooms**
Goal	To count the number of rooms that are currently occupied
Pre-condition	
Initiating Actor	Receptionist

Main Success Scenario

1 Receptionist requests count of occupied rooms
2 System shows count
3 Exit success

Use Case	**Decommission**
Goal	To remove a room from the Accommodation System
Pre-condition	The room is in the Accommodation System and the room is vacant
Initiating Actor	Accommodation Manager

Main Success Scenario

1 Accommodation Manager inputs room
2 System confirms room is now removed from the Accommodation System
3 Exit success

Exceptions

2a Room not in system
  2a1 Exit failure
2b Room occupied
  2b1 Exit failure

A use case (as shown above) describes a sequence of interactions between a system and its users. The users are known as actors. Exceptions are the error scenarios - what can go wrong. The pre-conditions of a use case describe the set of states for which the successful outcome is defined. For example, the pre-condition for the add a new room use case, *Commission*, is that the room has not already been added.

REVIEW

We defined a class of students system in terms of two sets, *enrolled* and *passed*, where *passed* is a subset of *enrolled*. We looked at the error cases that could occur and expressed them in a table of pre-conditions and in schemas. We combined schemas using conjunction and disjunction to form larger, more complex schemas.

Next, we look binary relations.

BIBLIOGRAPHY

JONES and WORDSWORTH, both cited in BARDEN et al, described the classic two sets pattern used in the Class system example.

BARDEN R., STEPNEY S. & COOPER D 1994 Z *In Practice Prentice* Hall pp 126, 166
JACKY J. 1997 *The Way of Z* Cambridge University Press pp 122
JONES C.B. 1980 *Software Development: a Rigorous Approach* Prentice Hall
NORCLIFFE A & SLATER G 1991 *Mathematics of Software Construction* Ellis Horwood pp 50
WORDSWORTH J.B. 1992 *Software Development with Z* Addison-Wesley