

**COMP4100 Lectures in 2007**

by Malcolm Newey

**Review of Elementary Z**

FM Lecture 8 ..... March 14, 2007

Source of material: Grassman & Tremblay, Chapter 8  
 Slides 3 to 24 are a subset of the slides used in COMP2600.

**What is This Lecture About?**

The main purpose of this lecture is to remind the many of us that did COMP2600 about the elements of the Z notation.

It is also to skim over these elements for the few of us who have never seen Z schemas. The slides will not form an adequate introduction, however, so students who are unfamiliar with the ideas should consult a textbook.

The important ideas to be reviewed are:

- Overview, history, references
- Types, declarations, entities, form of a Z document
- State schemas,  $\Delta$  schemas,  $\Xi$  schemas
- Specifying error handling
- Relations

**What is Z?**

**Z:** *A mathematical language, partly graphical, based on set theory and logic, that is used to formally specify systems.*

- Note the use of the words system.
- The name comes from the first initial of Zermelo-Frankel set theory.
- Not a programming language although there are similarities.
- It is *declarative* and *strongly typed*.

**Language Elements**

- Some fragments look like declarations
- Other parts look like mathematical formulae
- Schemas are usually presented graphically

Identifiers

- Basic identifiers are composed of letters, digits and underscores (starting with a letter)
- Case-sensitive
- Prefixes -  $\Delta$  and  $\Xi$
- Decorations - question mark, exclamation mark, prime

## Types in Z vs Types in a Prog. Lang.

In a programming language types are synthetic:

- `Date` would typically have fields for day, month, year.
- `Receipt` would have some structure to model the relevant attributes of receipts in the application.

In Z,

- What constitutes a date or a receipt is vague.
- What will be specified will be certain operations, not its structure or its representation.

## Declarations and Decorations

Declarations

- Just like in a programming language
- Examples:
  - boss: Person
  - mine: Car

Decorations

- $v?$  indicates  $v$  is an input variable;
- $v!$  indicates  $v$  is an output variable;
- $v$  and  $v'$  give values of a variable before and after an operation

## Entities

Z focuses on 3 kinds of entities:

**States** The collection of values that constitute the mathematical structure that models the system at some point in time.

**Events** Occurrences, Operations, Transitions

**Observations** Examination of variables or aspects of the state before or after an event. A predicate on states.

## The Enrolment Example

*Class*

$enrolled : \mathbb{P} Student$

$\#enrolled \leq classLimit$

*Initial*

*Class*

$enrolled = \emptyset$

*Class'*

$enrolled' : \mathbb{P} Student$

$\#enrolled' \leq classLimit$

### $\Delta$ Schemas

The prefix  $\Delta$  before a class name indicates it is concerned with states both before and after an operation.

Given schemas for  $Class$  and  $Class'$  above, the schema  $\Delta Class$  will be defined as

$\Delta Class$
$Class$
$Class'$

and is equivalent to:

$\Delta Class$
$enrolled : \mathbb{P} Student; \quad enrolled' : \mathbb{P} Student$
$\#enrolled \leq classLimit; \quad \#enrolled' \leq classLimit$

### Operations on the Class

$AddClass_o$
$\Delta Class$
$s? : Student$

$s? \notin enrolled$
$\#enrolled < classLimit$
$enrolled' = enrolled \cup \{s?\}$

$DropClass_o$
$\Delta Class$
$s? : Student$

$s? \in enrolled$
$enrolled' = enrolled \setminus \{s?\}$

### $\Xi$ Schemas

The prefix  $\Xi$  before a class name indicates it can be used in the specifications of enquiries (no state change).

Given schemas for  $Class$  and  $Class'$  above, the schema  $\Xi Class$  will be equivalent to:

$\Xi Class$
$enrolled : \mathbb{P} Student; \quad enrolled' : \mathbb{P} Student$
$\#enrolled \leq classLimit; \quad \#enrolled' \leq classLimit$
$enrolled' = enrolled$

### Queries on Class Enrolments

$StudentInClass$
$\Xi Class$
$response! : Reply$
$s? : Student$

$(s? \in enrolled \wedge response! = yes)$
$\vee (s? \notin enrolled \wedge response! = no)$

### Operations with Errors - I

If there are errors associated with an operation then a result should indicate so.

If there are no errors the result should do likewise.

```

OkMessage _____
output! : PossibleMessages
output! = ok
  
```

This schema relies on the prior type declaration:

```

PossibleMessages
 ::= ok | already_in_class | full | not_in_class | two_errors
  
```

### Operations with Errors - II

```

AddClassError _____
∃ Class
s? : Student
output! : PossibleMessages

(s? ∉ enrolled ∧ #enrolled = classLimit ∧
 output! = full)
(s? ∈ enrolled ∧ #enrolled < classLimit ∧
 output! = already_in_class)
(s? ∈ enrolled ∧ #enrolled = classLimit ∧
 output! = two_errors)
  
```

### Operations with Errors - III

The complete specification of adding a class is the schema *AddClass* which is defined in the non-graphical way:

$$AddClass \hat{=} (AddClass_o \wedge OkMessage) \vee AddClassError$$

This illustrates:

- The schema definition operator,  $\hat{=}$
- The fact that schemas can be connected by logical operations - conjunction, disjunction and implication.

### Relations

More Z notation:

- The notation  $X \leftrightarrow Y$  indicates the type of all possible relations on  $X$  and  $Y$ .
- $X \leftrightarrow Y$  is the same type as  $\mathbb{P}(X \times Y)$ .
- The character  $\mapsto$  is called the *map symbol*.
- $x \mapsto y$  denotes the ordered pair  $(x, y)$  and called a *maplet*. A binary relation consists of a set of maplets.

## A System-defining Schema

*Parking*

$staff : \mathbb{P} Person$   
 $available, occupied : \mathbb{P} PSpot$   
 $assignedTo : PSpot \leftrightarrow Person$

$ran\ assignedTo \subseteq staff$   
 $available \cap occupied = \emptyset$

- *staff* is a set of people. *staff* is subject to change while *Person* is not.
- The keyword 'ran' denotes the range operator.
- There are two system invariants.

## The Spot Assignment Operation

$AssignSpot_0$

$\Delta Parking$   
 $name? : Person$   
 $s? : PSpot$

$name? \in staff$   
 $(s? \mapsto name?) \notin assignedTo$   
 $available' = available \setminus \{s?\}$   
 $occupied' = occupied \cup \{s?\}$   
 $staff' = staff$   
 $assignedTo' = assignedTo \cup \{s? \mapsto name?\}$

Assignment of a spot to multiple staff members allowed.