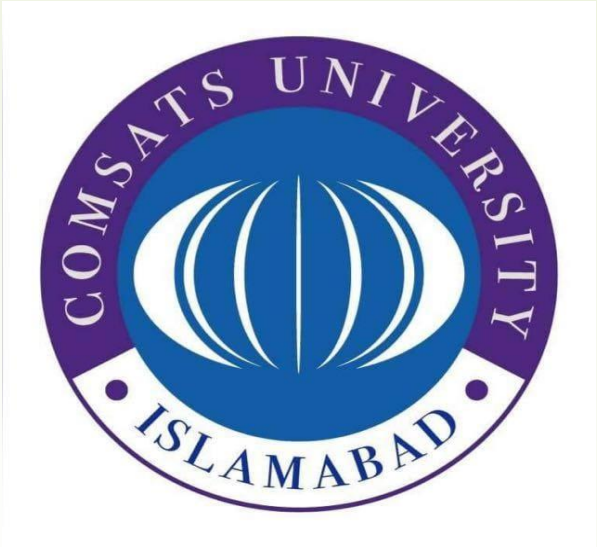



# Department of Computer Science, CUI Lahore Campus

Formal Methods

By

Farooq Ahmad

# Software Specification using Sets

Sets as systems' state variable, state Schemas, operation schemas

# Topics covered

- Case study: class management system
- Set data type
- Set as system state variable
- Free type definition
- Error report schemas

# 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* ]

# class system as two sets of students

**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.

$$maxClassSize : \mathbb{Z}$$

---

$$maxClassSize = 20$$

# class system as two sets of students

**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*.

*Class*

$$enrolled : \mathbb{F}\ STUDENT$$
$$passed : \mathbb{F}\ STUDENT$$

---

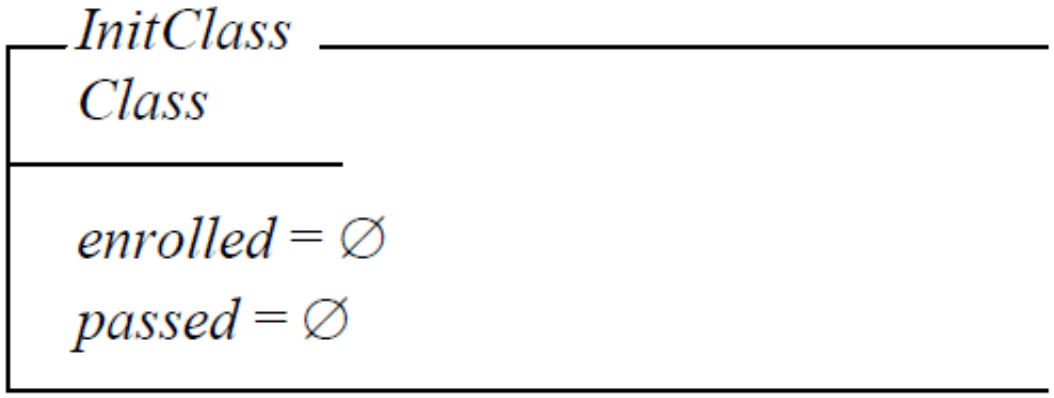
$$\#enrolled \leqslant maxClassSize$$
$$passed \subseteq enrolled$$

F: Set of finite subsets

# class system as two sets of students

**The Initial State**

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



# Z decoration

- Variable names that end in ? are understood to model inputs.
- The question mark (?) is not an operator; it is an example of a Z decoration: a character that we put at the end of a variable name to indicate some conventional meaning.
- The prime ' is another decoration. It tells us that enrolled' and passed' describe the state after the operation.

# Δ and Ξ decoration

- 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;
- The Ξ symbol indicates an operation where the state does not change (Ξ, the capital Greek letter xi, suggests an equal sign). We use Ξ when we only need to read state variables without changing their values, or where the operation exists only to consume inputs or produce outputs.

# Operation schemas; Operation Schema Enroll with Δ decoration

**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.

*Enrol*

$\Delta Class$

$student? : STUDENT$

---

$\#enrolled < maxClassSize$

$student? \notin enrolled$

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

$passed' = passed$

The ? in *student*? means input.

# Use case Complete

**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*

$\Delta Class$

$student? : STUDENT$

---

$student? \subseteq enrolled$

$student? \notin passed$

$passed' = passed \cup student?$

$enrolled' = enrolled$

*student? ∈ enrolled, Passed' = passed ∪ {student?}*

# Use case; Leave the class with certificate

**Leave With Certificate**

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

*LeaveWithCertificate*

$\Delta Class$

$student? : STUDENT$

---

$student? \in passed$

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

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

# Schemas for Error scenarios

- 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.

# Free type definition

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.

# 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*

$\Xi Class$

$report! : REPORT$

---

$\#enrolled \geqslant maxClassSize$

$report! = classFull$

# Schema modeling error report

**Already Enrolled**

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

*AlreadyEnrolled*

$\Xi Class$

$student? : STUDENT$

$report! : REPORT$

---

$student? \in enrolled$

$report! = alreadyEnrolled$

# Schema modeling error report

**Not Enrolled**

If a student is not enrolled they cannot be passed.

*NotEnrolled*

$\Xi Class$

$student? : STUDENT$

$report! : REPORT$

---

$student? \notin enrolled$

$report! = notEnrolled$

# Schema modeling error report

**Already Passed**

The same student cannot be passed twice.

*AlreadyPassed*

$\Xi Class$

$student? : STUDENT$

$report! : REPORT$

---

$student? \in passed$

$report! = alreadyPassed$

# Schema modeling error report

**Not Passed**

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

*NotPassed*

$\Xi Class$

$student? : STUDENT$

$report! : REPORT$

---

$student? \notin passed$

$report! = notPassed$

# 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. If pre-conditions are violated:

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

# A class manager's assistant

- It provides support for a class manager to admit students to a class, and to record who has done the midweek exercises.
- A computerized class manager's assistant is required to keep track of students enrolled on a class, and to record which of them have done the midweek exercises.
- The operations required are as follows:
- **Enroll, Test, Leave, Enquire**

# The operations of the system

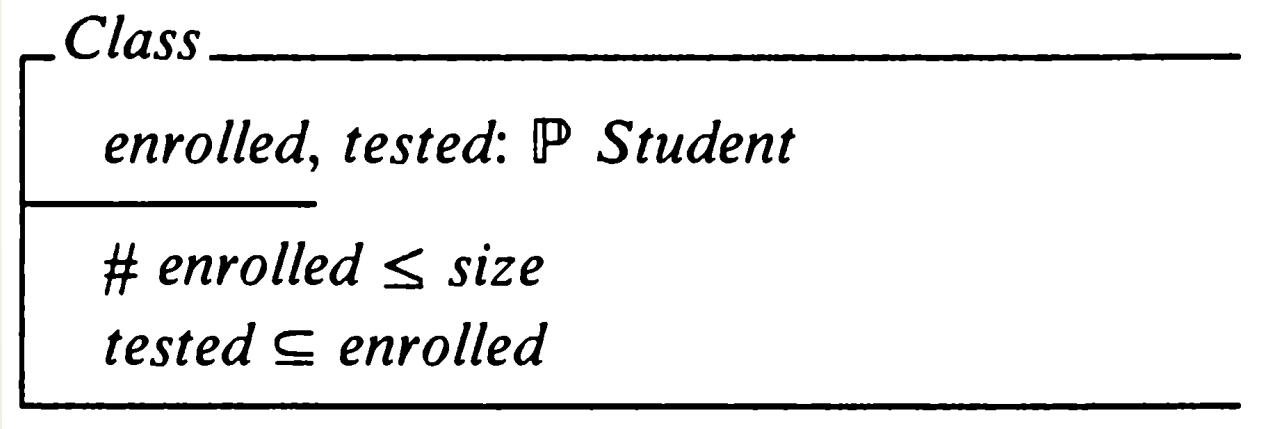
| | |
|---|---|
| **Enrol** | This operation enrols a student on a class, or issues a warning if the class is full, or if the student is already enrolled. |
| **Test** | This operation records that a student has done the exercises, or warns if the student is not enrolled, or has already done the exercises. |
| **Leave** | This operation removes a student from the class with an indication of whether the student is entitled to a completion certificate. Only students who have done the exercises are entitled to a certificate. If the student is not enrolled, a warning is given. |
| **Enquire** | This operation shows whether a student is enrolled, and whether the student has done the exercises. |

# System state schema: Class

- [Student]
- | size: N

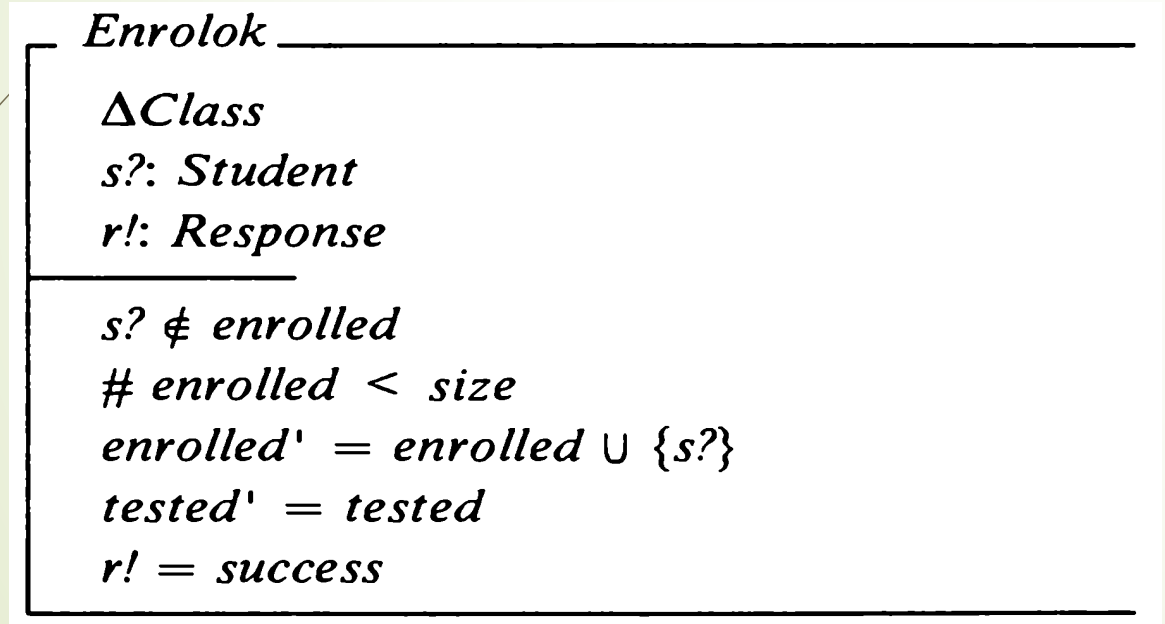
$$
\begin{aligned}
Response ::= \ & success \\
& \mid notenrolled \\
& \mid nocert \\
& \mid cert \\
& \mid alreadyenrolled \\
& \mid alreadytested \\
& \mid noroom
\end{aligned}
$$

# System state schema



# Enrolling a student: Use case

To record the enrolling of a student, the class changes. The student s? must be supplied as input, and an output response r! will be generated.




***Response ::= success | notenrolled | nocert | cert | alreadyenrolled | alreadytested | noroom***

# Testing a student

- To record that a student has done the exercises, the class changes.
- The student s? must be supplied as input, and an output response will be generated.

*Testok*

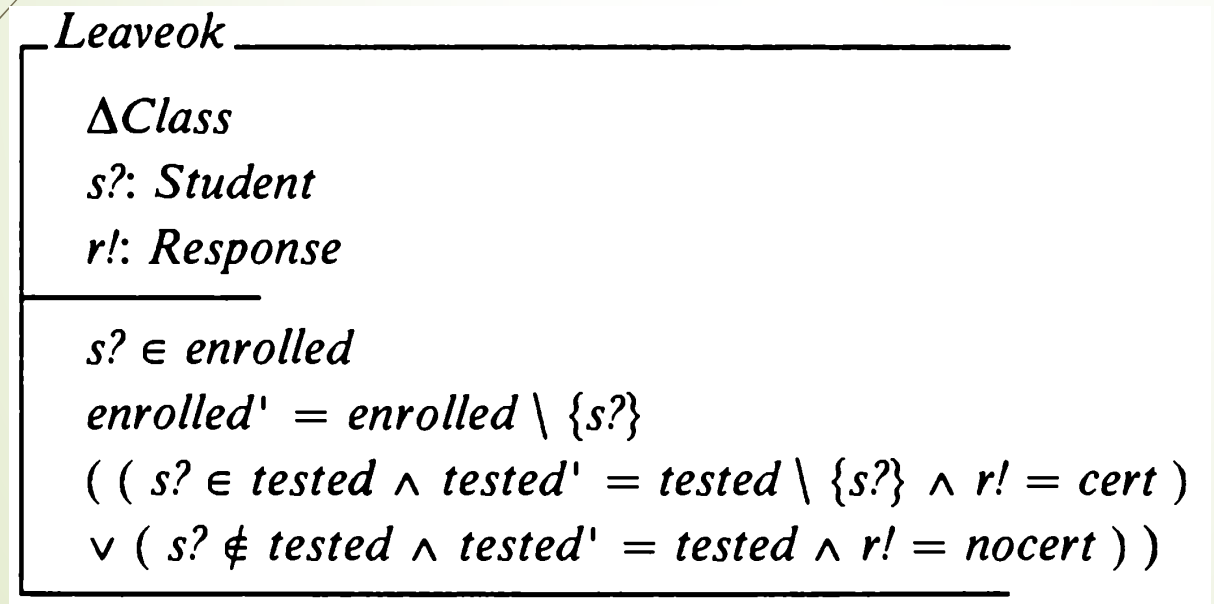
$$\Delta Class$$
$$s?: Student$$
$$r!: Response$$

---

$$s? \in enrolled$$
$$s? \notin tested$$
$$tested' = tested \cup \{s?\}$$
$$enrolled' = enrolled$$
$$r! = success$$

# Discharging a student

- To record that a student is leaving the class, the class changes. The student s? must be supplied as input, and an output response r! will be generated.




*noCert:* (no certificate) The student provided as input to the Leave operation had been enrolled, but had not done the exercises. The operation to discharge the student has been completed successfully.

# Enquiries

- There is an enquiry operation to report whether a student is enrolled, and to indicate whether the student has done the exercises. This operation does not change the class, it merely reports on it. The student s? must be supplied as input. An output response r! will be generated.

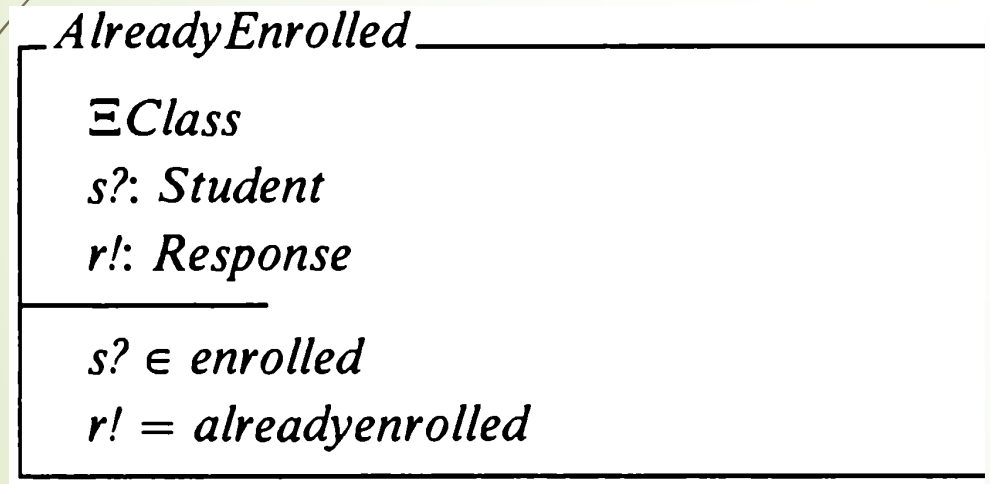
*Enquire*

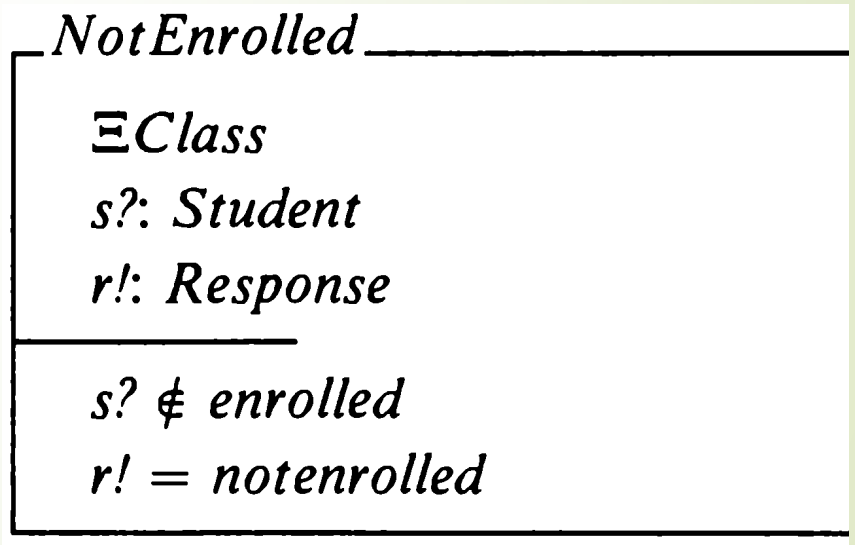
$$\Xi Class$$
$$s? : Student$$
$$r! : Response$$

$$\begin{aligned}&( ( s? \notin enrolled \land r! = notenrolled )\\ &\lor ( s? \in ( enrolled \setminus tested ) \land r! = alreadyenrolled )\\ &\lor ( s? \in tested \land r! = alreadytested ) )\end{aligned}$$

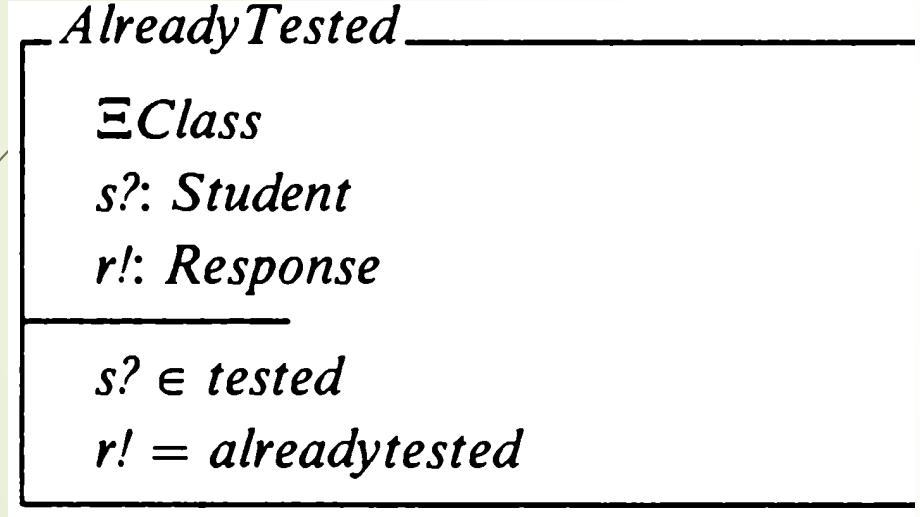
# Errors

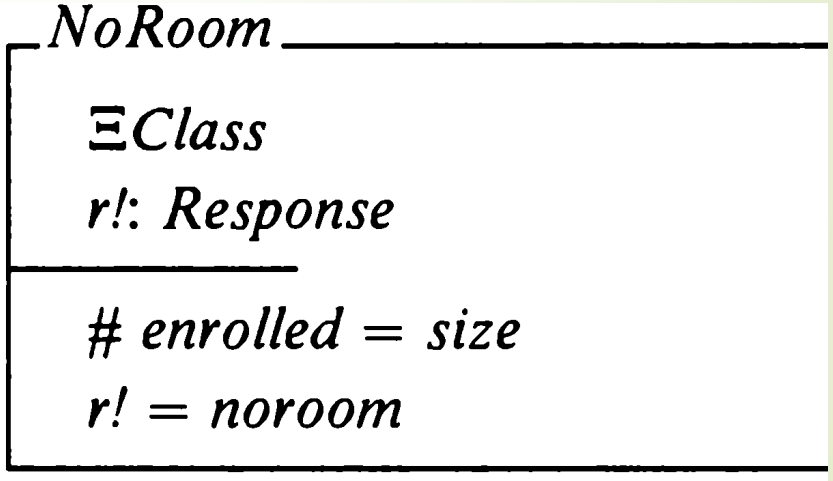
- Sometimes the user will attempt to enroll a student who is already enrolled. The class must not change as a result of this behavior. The student s? must be supplied as input. An output response r! will be generated.



# Schemas modeling errors



# Reference and reading material

- Chapter 2
- Chapter 5: Section 5.2 of the book “Software Development with Z”