CHAPTER 2
Z Specification Language

Chapter Outline

- Z Language
- Z Fundamental Concepts
- Z Types Declaration
- Z Schema

Z Language

Z Language

- Pronounced as "**Zed**".
- A set of conventions for presenting mathematical text.
- Based on set theory and first-order predicate logic.
- Semi-graphical notation for writing formal specifications based on typed sets, relations, and functions to express:
  - What are the functionalities of the system
  - And what the desired results are
  - But without stating the "how" part.

Z Language

- It is a **model-based notation**.
- Usually model a system by representing its **state** **(**a collection of state variables and their values) and some **operations/functions** that can change its state.
- Organised behaviour by:
  - Describing possible states
  - Describing initial states
  - Describing states changes
  - Describing states queries

Z Language

- Makes use of a graphical construction known as a **schema**
  - provide an effective low level structuring facility
  - are useful as specification building blocks
  - can be understood fairly easily
  - Allow modularity
  - Improve readability
- The most widely-used formal specification language.

Z Fundamental Concepts

Sets

- Formal Z is based on Zermelo-Frankel set theory.
- The consequence is that everything in Z is a **set**.
- A set to be any well-defined collection of distinct objects.
    - There is **no ambiguity** in deciding whether or not a given object belongs to a set.
    - The objects in a set must be **distinguishable** from each other.
- The objects in a set are called its **elements** or **members**.

Sets

- Example: Four oceans of the world can be defined by :

  Oceans == {Atlantic, Arctic, Indian, Pacific}

  == means “defined as”

- Other examples:

  Odds == {1, 3, 5, 7,…}

  Colors == {red, green, blue, yellow}

  Vowels == {a, e, i, o, u}

Sets

- Members of set
  - If x is a set and s is a member of x, then we write $s \in x$.
  - If x is a set and s is not a member of x, then we write $s \notin x$.

- Empty set (a.k.a null set)
  - A set that has no elements, $\emptyset$
  - on_loan = $\emptyset$

Sets

- Singleton set
  - Contain **only one** single element/member
  - For example: {a}, with brackets, is a singleton set
  - a, without brackets, is an element of the set {a}
  - Note the subtlety in $\emptyset \neq \{\emptyset\}$
    - The left-hand side is the empty set
    - The right hand-side is a singleton set, and a set containing a set

Sets

- Subsets
  - $\subseteq$
  - A is said to be a **subset** of B, and we write $A \subseteq B$, if and only if every element of A is also an element of B.
  - That is, we have the equivalence:

$$A \subseteq B \Leftrightarrow \forall x\ (x \in A \Rightarrow x \in B)$$

    - E.g.

$$\{1,2\} \subseteq \{1, 2, 3, 4\}$$

$$\{1, 2, 3, 4\} \subseteq \{1, 2, 3, 4\}$$

Sets

- Proper subsets
  - $\subset$
  - A set A that is a subset of a set B is called a **proper subset** if $A \neq B$.
    - We write: $A \subset B$
    - E.g.

$\{1,2\} \subset \{1, 2, 3, 4\}$

$\{4\} \subset \{1, 2, 3, 4\}$

Sets

- Power sets
  - $\mathbb{P}$
  - Considering all possible combinations of elements of a set
  - Given a set A, the **power set** of A, denoted $\mathbb{P}(A)$, is the set of all subsets of A.
  - Example:
    - Let A = {x, y, z}

$$\mathbb{P}(A) = \{\emptyset, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$$

  - Note: the empty set $\emptyset$ and the set itself are always elements of the power set.

Sets

- Finite sets
  - $\mathbb{F}$
  - If there are exactly n distinct elements in a set S, with **n** a nonnegative integer, we say that S is a finite set.
- Cardinality
  - #
  - The **number of elements** in the set. Use with $\mathbb{P}$ and $\mathbb{F}$
  - #S = n
  - #{1, 2, 4} = 3

Sets

- Infinite sets
  - A set that is not finite is said to be infinite.
  - The sets $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ are all infinite
  - $\mathbb{N}$, Natural numbers = {0, 1 ,2, 3, …}
  - $\mathbb{Z}$, Integers = {…,-2, -1, 0, 1, 2, …}
  - $\mathbb{Q}$, Rational numbers = $\{p/q \mid p \in Z, q \in Z, q \neq 0\}$
  - $\mathbb{R}$, Real numbers

Sets

- Equality of sets
  - Two sets, A and B, are **equal** is they contain the same elements. We write $A = B$
  - $\{2,3,5,7\} = \{3,2,7,5\}$
  - $\{2,3,5,7\} \neq \{2,3\}$
  - $\{2,3,5,7\} = \{2,2,3,5,3,7\}$ because a set contains unique elements

Set Operations

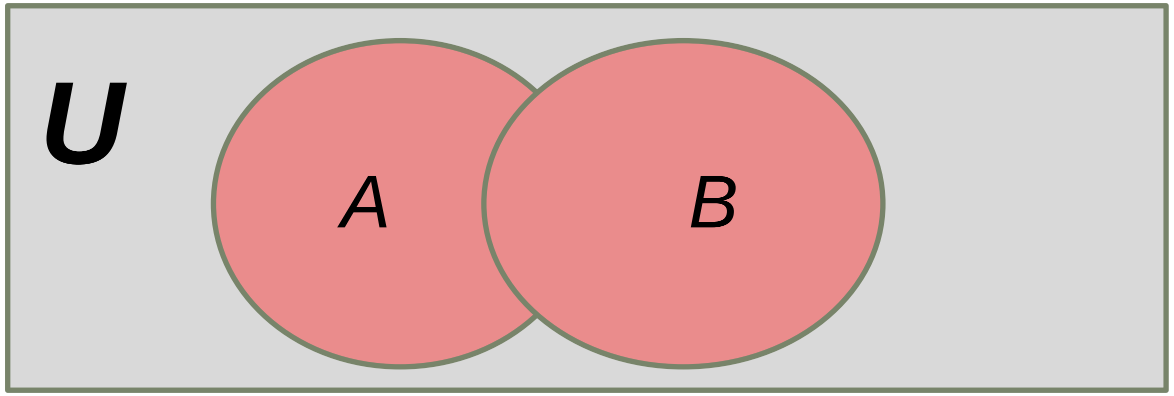
- Arithmetic operators (+, - , *, mod, div) can be used on pairs of numbers to give us new numbers.
- Similarly, set operators exist and act on two sets to give us new sets
  - Union, $\cup$
  - Intersection, $\cap$
  - Set difference, $\setminus$
  - Set complement, $\overline{}$

Union

- The **union** of two sets A and B is the set that contains all elements in A, B, or both.
- We write: $A \cup B = \{ x \mid (a \in A) \vee (b \in B) \}$

$A = \{a, b\}, B = \{b, c, d\}$

$A \cup B = \{a, b, c, d\}$

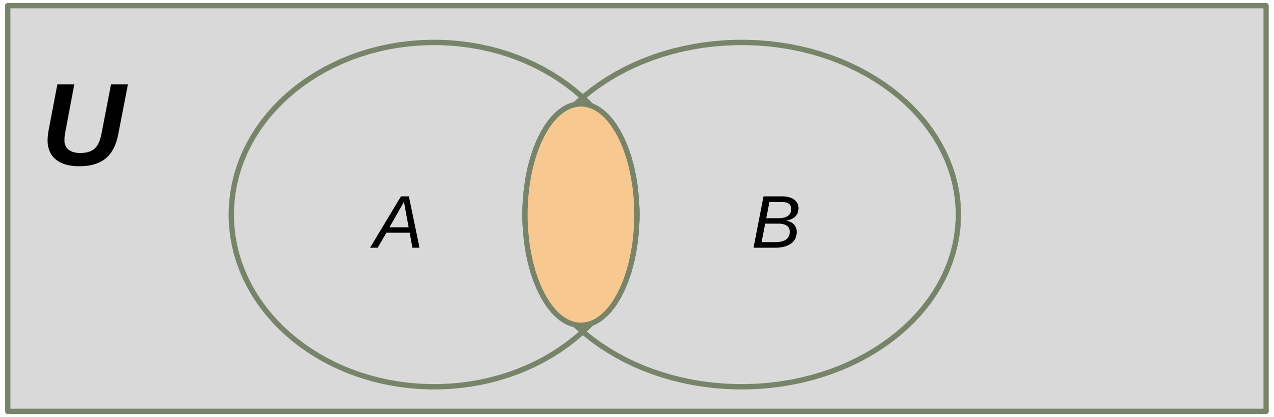


Intersection

- The **intersection** of two sets A and B is the set that contains all elements that are element of both A and B.
- We write: $A \cap B = \{ x \mid (a \in A) \wedge (b \in B) \}$

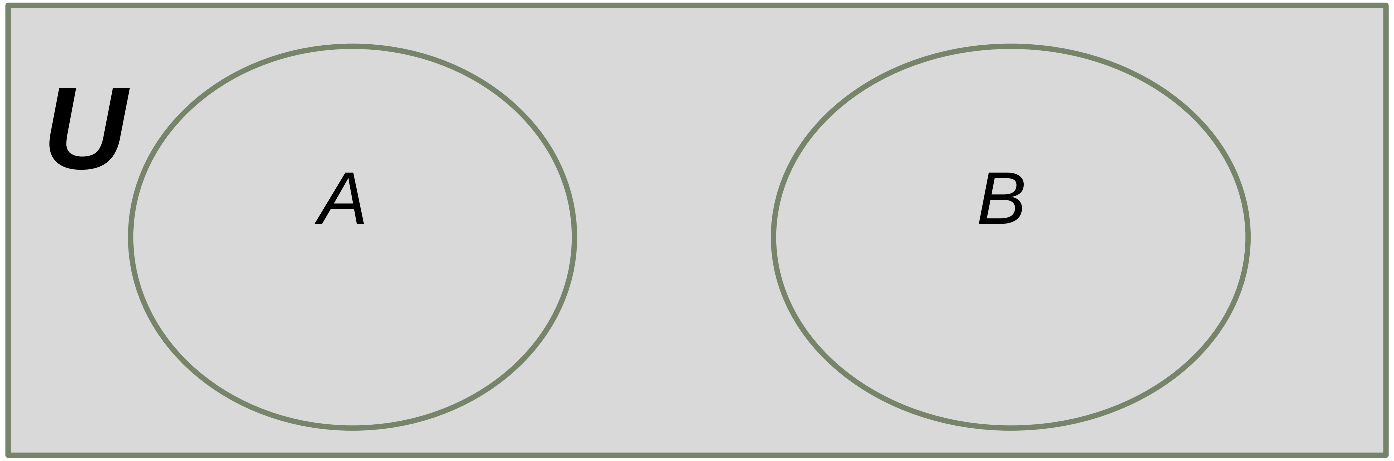
$A = \{a, b\}$, $B = \{b, c, d\}$

$A \cap B = \{b\}$



Disjoint Sets

- Two sets are said to be **disjoint** if their intersection is the empty set:
- $A \cap B = \varnothing$

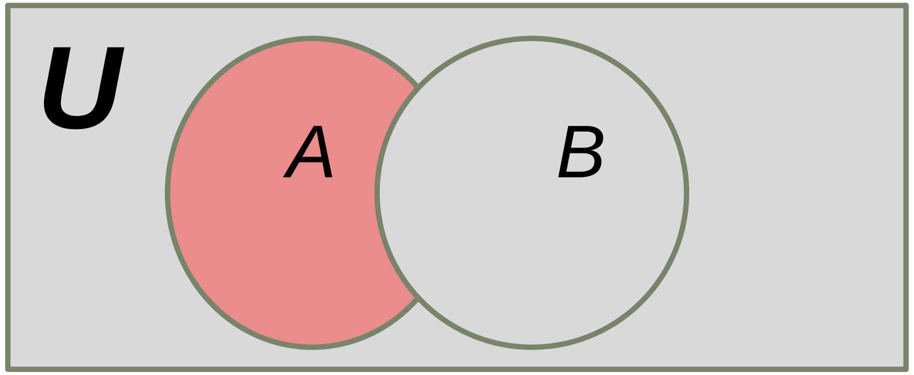


Set Difference (\)

- The **difference** of two sets A and B, denoted A\B or A–B, is the set containing those elements that are in A but not in B.
- It has two arguments both of the same type. It forms the set which is its first argument with elements of the second argument removed.
- E.g.

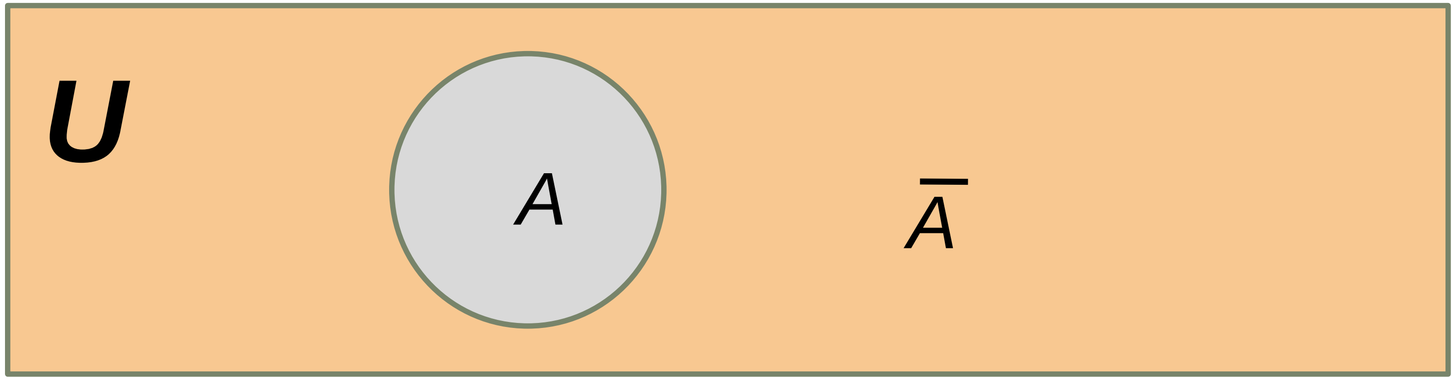
  $\{1, 2, 3, 4, 8, 9\} \setminus \{1, 2, 3\} = \{4, 8, 9\}$

  $\{1, 2, 3\} \setminus \{1, 2, 3\} = \varnothing$



Set Complement

- The **complement** of a set A, denoted $\overline{A}$, consists of all elements **not** in A. That is the difference of the universal set and U: U\A



$$A= A^{C} = \{x \mid x \notin A \}$$

Set Comprehension

- Enumerating all of the elements of a set is NOT always possible
- Simple form of set comprehension:

$$\{ x : S \mid P(x) \}$$

"The set of all x in S that satisfy P(x)", OR

"The set of all x in S such that P(x)"

- Example:

"Natural number less that 20"

$$\{x : \mathbb{N} \mid x < 20\}$$

Set Comprehension

- Sometimes it is helpful to specify a pattern for the elements
- We will use the form:

$$\{x : S \bullet f(x)\}$$

where f is some function defined on elements of S

Use "•" instead of "|"

- Example:

squares : $\{x : \mathbb{N} \bullet x^2\}$

Set Comprehension

- Most general form combines the two forms:

  {x: S | P(s) • f(x) }

  ➔ {set : range | condition • function}

- In notational form (aka comprehensive specification):

  {Signature | Predicate • Term}

$$\{x : X \mid P(x) \cdot E(x)\}$$

Set Comprehension

- Examples:
  - Squares of integers less than 20

    $\{ x: \mathbb{N} \mid x < 20 \bullet x^2 \}$
  - Squares of even numbers

    $\{ x : \mathbb{Z} \mid (\exists y : \mathbb{Z} \bullet x = 2y) \bullet x^2 \}$
  - Squares of multiples of 4 (excluding zero)

    $\{ x : \mathbb{Z} \mid (x \bmod 4 = 0) \land (x > 0) \bullet x * x \}$
  - Alternate even numbers = {0,4,8,12,16,..}

    $\{ x : \mathbb{N} \mid x \bmod 2 = 0 \bullet 2 * x \}$

Quantifiers

- We use quantifiers to express general truths
- For example:
  - To assert 'for all x, x + 1 > 1', we use a universal or for all quantifier : ∀

$$\forall x : \mathbb{N} \bullet (x + 1 > 1)$$

  - To assert 'there exists an x, x + 1> 1', we use an existential or there exists quantifier : ∃

$$\exists x : \mathbb{N} \bullet (x + 1 > 1)$$

Quantifiers

- The syntax :

$$\forall \text{ (name) : (type) [| (constraint)] } \bullet \text{ (predicate)]}$$

This is read as:

"**For all** (name) of type (type) [**such that** (constraint)], it is true that (predicate)."

| is read as "such that"

Quantifiers

- The syntax :

  $\exists$ (name) : (type) [| (constraint)] • (predicate)]

  This is read as:

  "**There exists a** (name) of type (type) [**such that** (constraint)], for which it is true that (predicate)."

Quantifiers

Universal Quantification	Existential Quantification
$\forall n : \mathbb{N} \mid n \leq 10 \bullet n^2 \leq 100$ $\forall n : \mathbb{N} \bullet (n \leq 10 \Rightarrow n^2 \leq 100)$ $\forall n : \mathbb{Z} \mid n \geq 0 \bullet (n \leq 10 \Rightarrow n^2 \leq 100)$ $\forall n : \mathbb{Z} \bullet ((n \geq 0 \land n \leq 10) \Rightarrow n^2 \leq 100)$ $\forall n : \mathbb{Z} \mid n \leq 10 \bullet (n \geq 0 \Rightarrow n^2 \leq 100)$	$\exists n : \mathbb{N} \mid n \leq 10 \bullet n^2 = 64$ $\exists n : \mathbb{N} \bullet (n \leq 10 \land n^2 = 64)$ $\exists n : \mathbb{Z} \mid n \geq 0 \bullet (n \leq 10 \land n^2 = 64)$ $\exists n : \mathbb{Z} \bullet (n \geq 0 \land n \leq 10 \land n^2 = 64)$ $\exists n : \mathbb{Z} \mid n \leq 10 \bullet (n \geq 0 \land n^2 = 64)$ $\exists n : \mathbb{Z} \mid (n \leq 10 \land n \geq 0) \bullet n^2 = 64$ $\exists n : \mathbb{Z} \mid n^2 = 64 \bullet (n \leq 10 \land n \geq 0)$

Existential Quantifier

- $\exists_1$ means 'there exists a unique'

  $$\exists_1 x : \mathbb{N} \mid x < 10 \bullet x + 9 > 12$$

  may be read as "There exists precisely one natural number x such that x is less than 10, for which it is true that x + 9 > 12"

- Example:
  - $\exists_1 x : \mathbb{N} \bullet x = 25$ is true
  - $\exists_1 x : \mathbb{Z} \bullet x^2 = 25$ is false

Z Types Declaration

Z Types

- When people use set theory to specify software systems, they often include some notion of types.
- Every object belongs to a set called its ***type***.
- Z is “strongly” typed - that is , every identifier must be declared.
- Z has 3 types:
  - **Built-in**
  - **Basic**
  - **Free type**

Built-in Types

- In Z, the **TWO** built-in types used are:
  - The set of all whole numbers (integers), $\mathbb{Z}$ (…,-3,-2,-1,0,1,2,3,…)
  - The set of all natural numbers, $\mathbb{N}$ (0, 1, 2, 3, …)
- Example:
  - $\mathbb{N} == \{n : \mathbb{Z} \mid n \geq 0\}$ (positive integers)
  - $\mathbb{N}_1 == \{n : \mathbb{Z} \mid n > 0\}$ (positive, non-zero, integers)
  - $\{1, 2, 3\} \in \mathbb{P}\,\mathbb{Z}$

Basic Types

- A.k.a "given sets", is a basic type is a set whose internal structure is invisible.
- We may introduce elements of such a set, and associate properties with them, but we can assume nothing about the set itself.
- To represent global variables (sets) for the system without having to specify the details of those sets.
- May be written in bracket, [ ]

Basic Types

- Can include indefinitely many elements.
- Basic types are constructed from characters with no length restriction.
  - Characters must be ALL capital letters
  - NO "_", digits, space and other symbols are allowed.
  - Use singular word, if more than one word are used, combine as one
- Example:
  - [NAME]
  - [STUDENTNAME, STUDENTID]

Free Types

- May be written as enumerations.
- Use the symbol
  - ::= , data type definition symbol
  - | , branch separator
- Example:
  - STUDENT ::= william | shilpa | harish | carolyn | amed | joel
  - COLOUR ::= violet | indigo | blue | green | yellow | orange | red
  - POSITION ::= off | on

Z Identifiers

- Z identifiers are constructed from letters, digits, and the "_" characters.
  - Upper and lower case are distinct
  - No length restriction
- Rules for Z identifiers:
  - First word must be lower case letters. If combine more than one word, the rest of the words must start with upper case letters.
  - Can separate more than one words with underscores.
  - Cannot start with a digit
  - No other symbols can be used other than underscore.

Z Identifiers

Valid Z Identifiers	Invalid Z Identifiers
student	Student
studentName	STUDENT
student_name	999student
student_Name	student@
student999	StudentName

Z Identifiers

- Identifiers followed by a prime ’ indicate the values of **objects after the action** has taken place.
- Identifiers followed by a question mark **?** indicate the **input** values identifiers.
- Identifiers followed by a exclamation **!** indicate the **output** values identifiers.

Type Decoration

Z Identifiers

Input	Output	Update
student?	price!	login’
name1?	total!	enrolled’
icNo?	phone!	student’
emailAdd?	emailAdd!	price’
phone?	student!	phone'

Axiomatic Definition

- In Z, the starting point of a specification is to define the basic types (given sets)
- The next step will be to define:
  - Free types
  - General rules (axiom) (called Axiomatic definition) related to the system to be specified.

Axiomatic Definition

- Define any global variables for the whole specification and can include optional constraints.
- Consists of two parts:

$$
\begin{array}{|l}
\mathit{Declaration} \\
\hline
\mathit{Predicate}
\end{array}
$$

- Example:

$$
\begin{array}{|l}
maxStudent : \mathbb{N} \\
\hline
maxStudent = 100
\end{array}
$$

Axiomatic Definition

- If there is **NO** constraining predicate:

$$\left|\, \textit{Declaration} \right.$$

- Example:

$$\left|\, \text{maxQuantity} : \mathbb{N} \right.$$

Axiomatic Definition

- Also can be used to declare **global constant**.
- Constants are variables that are constrained to one value.

$$
\begin{array}{|l}
minimum: \mathbb{N} \\
\hline
minimum = 0
\end{array}
$$

- Global constant also can be declared with the abbreviation definition "==" symbol:

$$minimum == 0$$

Z Declaration

- Every type in Z must be introduced in a **declaration**.
- A name (variable) is assigned a type when it is declared.
- A variable is a name for an object: its value.
- Example:
  - stud1 : STUDENT
  - b1, b2 : BOOK
  - jane : PERSON
  - studName : NAME

Z Declaration

[NAME]	- the set of all staff names in the system
[ID]	- the set of all staff IDs in the system
POSITION	::= admin | staff | customer

name: NAME	(basic type – one record)	→ name: $\mathbb{P}$ NAME
id: ID	(basic type – one record)	→ id: $\mathbb{P}$ ID
pos: POSITION	(free type – one record)	→ pos: $\mathbb{P}$ POSITION
price: $\mathbb{N}$	(built-in type)	→ price: $\mathbb{N}$

Z Schema

Z Schema

- Z specifies a system using
  1) Mathematically defined data types (**given sets**), to model the data in a system.
  2) Decompositions of a specification into small pieces called **schemas** (a boxed notation or graphical representation).
- **Schema** is manageably sized module where it allows the specification of:
  a) Data that shows the representation of data in the system
  b) Operations that access that data

Z Schema

- A schema describes both **static** and **dynamic** aspects of a system.
- **Static** aspects include:
  - The *states* (or variables and constants) that a system can occupy (Individual states) - model a state as an assignment of *values* to a collection of named *variables*.
  - The *invariant /constraint relationships* (requirements) that are maintained as the system moves from state to state.
  - In Z, static aspects is normally shown through *State space schema*.

Z Schema

- **Dynamic** aspects include:
  - The *operations (functions/methods)* that are possible (Events)
  - The *relationship* between their inputs and outputs (Transformations)
  - The *changes* of state that happen (State Transitions) in terms of "pre"(assumptions) and "post" conditions (results/goals).
  - In Z, dynamic aspects is normally shown through *Operation schema.*

Z Schema

- Z schema consists of:
  1) Abstract state schema describing the major components (state space schema).
  2) Schemas for initialisation
  3) Operation schema describing aspects of the normal operations of the system.
- Three basic structures/elements of Z Schemas:
  - Declarations introduce variables.
  - Expressions describe values that variables can assume.
  - Predicates place constraints on the values that variables do assume.

Z Schema

- Each schema will be divided into 3 parts:
  - **Schema Name** – always starts with **Capital letter**
  - **Declaration**/Signature part
    - A collection of *state variables* and their values (declarations)
  - **Predicate**/Logical part
    - Some *operations* that can change its state (predicates)

Z Schema

- The Z schema can be displayed in two ways:
  - **Text form**

    SchemaName ≙ [Declarations | Predicates]

    ≙ schema definition

  - **Graphical form**

    SchemaName

    Declarations

    Predicates

Schema Name

- Always starts with **Capital letter**
- If two words or more are used, combine these words as one word but every word must start with a capital letter
- No symbols, no digit, no space, no underscore is allowed

Valid Schema Name	Invalid Schema Name
AddCustomer	AddCustomer999
Customer	ADDCUSTOMER
	Add_Customer

Schema Declaration / Signature

- Introduces the identifiers (or variables) and assigns them the set type
- Each line statement is "assumed" to be terminated with ;
- Contains a list of variable declarations; as well as references to other schemas (this is called schema inclusion /state transitions).
- Example:
  - student : STUDENT
  - price : $\mathbb{N}$

Schema Predicate / Logical

- Refers to the identifiers in the declaration part or some global identifier in other schemas
- The predicates, when there is more than one, are logical "conjunctions", $\wedge$, of the predicates.
- Relationship between the values of the variables
- Schema predicates are **always TRUE**.

State Schema

- A.k.a. State Space Schema
- The starting state where the first aspect of the system to describe is its state space.
- Refers to a minimum set of variables, known as state variables, that fully describe the system and its response to any given set of inputs.
- Describes the logic of the overall state of the system.

Initial State

- What state the system is in when it is first started.
- This is the initial state of the system, and it also is specified by a schema.
- Normally shows the initiliased values for all variables from the state space.

Z Example

- Assume we are going to model a simple system called **Counter** where **Counter** is used to count the number of customers entering a shop for one whole day.
- The is a limit of customers who can enter the shop where the value is represented as *maximum*. This can be represented as a constant value.
- A **Counter** has one variable named as *total*. The variable *total* can be equal, but never exceed the *maximum* value.

Z Example - Counter

- **Step 1**: As the **Counter** has a constant value, we can define using Axiomatic definition. Assume that the initial value for *maximum* is 100.

$$
\begin{array}{|l}
maximum : \mathbb{N} \\
\hline
maximum = 100
\end{array}
$$

Z Example - Counter

- **Step 2** : Then, create the Z **state space** schema for the **Counter** system.

$Counter$ Schema Name

$total : \mathbb{N}$ Declaration

$total \leqslant maximum$ Predicate

Z Example - Counter

- **Step 3** : Create the **Initial state** of the **Counter** schema called e.g. *InitCounter* where the initial value for *total* is 0.

$InitCounter$

Schema Name

$Counter$

State space schema

$total = 0$

Initialise value

Operation Schema

- Shows the transition/change of state that happen in terms of "pre"(assumptions) and "post" conditions (results/goals).
- Also shows remain unchanged states.
- Can be:
  - **Query State** operation schema
  - **Change State** operation schema

Query State

- Provides information about the state of the system, without changing the state.
- Shows the output of the current value (that stored data is not affected). **All values** remain **unchanged**.
- Use "Xi", denoted by the Greek literal (Ξ)

Change State

- Used to extend the schema components to indicate update operations, i.e. changes in state variables.
- **At least one value** will have an **update/change** in this operation schema.
- Use "Delta", denoted by the Greek literal (Δ)

Pre- and Post- Conditions

- Pre-conditions are statements that must be true for the operation to be successful and post-conditions specify the result of the operation.
- A pre-condition is a predicate describing the state **before**.
- A post-condition is a predicate describing the state **after**.
- These conditions are predicates over the inputs (?) and outputs (!) of a function.

Pre- and Post- Conditions

- Assume that we want to express changes to the state in one of our operation schemas.
- In order to express the changes, we need represent a state change by making two copies of the variables.
- Variables **without** prime symbol (’) to indicate **before**.
- Variables **with** prime symbol (’) to indicate **after**.

Pre- and Post- Conditions

- Predicates include:
  - operators (such as =, >, <, not, and, or),
  - the universal and existential quantifiers, and
  - the operator in which is used to select the range over which the quantifier applies.

Z Example - Counter

- Referring to **Counter** system
- **Step 4** : Continue with the operation schema
  - **Query state** of the **Counter** schema called *QueryTotal* that produce the **output** of the current value of *total.*

Xi notation – No changes in the system values

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ QueryTotal\ \hspace{6cm}}} \\
\Xi\, Counter \quad \text{State space name} \\
output! : \mathbb{N} \quad \text{An output variable declaration - local} \\
\hline
output! = total \quad \text{Produce output of the total value} \\
\hline
\end{array}
$$

Z Example - Counter

- **Change state** of the **Counter** schema called *AddCounter* and *RemoveCounter* that update the *total* state variable.

Delta notation – There is at least one value in the system that will change

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{AddCounter} \\
\hline
\Delta\ Counter \\
\hline
total < maximum \\
total' = total + 1 \\
\hline
\end{array}
$$

State space name

Pre-condition

Post-condition

Z Example - Counter

- **Change state** of the **Counter** schema called *AddCounter* and *RemoveCounter* that update the *total* state variable.

$$\textit{RemoveCounter}$$

$$\Delta\ \textit{Counter}$$

$$\textit{total}' = \textit{total} - 1$$

Delta notation – There is at least one value in the system that will change

State space name

Post-condition

Z Exercise

Question 1

- Now, let's change the **Counter** system so that the system **can keep the customer information** rather than to only count the number of customers entering a shop for one whole day.
- Let's rename the **Counter** state space to **CountCustomer** and basic type [CUSTOMER] was defined to be used in the system.
- A limit of customers who can enter the shop represented as *maximum* will still remain in the system.

Question 1

- Step 1: Basic type

  [CUSTOMER]

- Step 2: Axiomatic Definition

$$
\begin{array}{|l}
maximum : \mathbb{N} \\
\hline
maximum = 100
\end{array}
$$

Question 1

- Step 3: State Space schema called **CountCustomer**

$$\mathit{CountCustomer}$$

$$\mathit{customer} : \mathbb{P}\ \mathit{CUSTOMER}$$

Many customers can be stored in the system

$$\#\mathit{customer} \leqslant \mathit{maximum}$$

Total customers (#) cannot exceed the maximum value

Question 1

- Step 4 : **Initial state** of the **CountCustomer** schema called *InitCountCustomer* where the *customer* is empty.

$$
\begin{array}{|l}
\hline
\textit{InitCountCustomer} \\
\hline
CountCustomer \\
\hline
customer = \varnothing \\
\hline
\end{array}
$$

State space name

Customer record is empty

Question 1

- Step 5 : Operation Schema
  - **Query state** of the **CountCustomer** schema called *QueryCustomer* that produce the **output** of the current total of *customers in the shop*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;QueryCustomer\;}} \\
\Xi CountCustomer \\
total! : \mathbb{N} \\
\hline
total! = \#customer
\end{array}
$$

Total customers (#) in the system

Question 1

- **Change state** of the **CountCustomer** schema called
  - *AddCustomer*
  - *UpdateCustomer*
  - *RemoveCustomer*

that update the *CountCustomer* system state variable.

Question 1

$$
\begin{array}{|l}
\hline
\textit{AddCustomer} \\
\Delta CountCustomer \\
cust? : CUSTOMER \\
\hline
cust? \notin customer \\
\#customer < maximum \\
customer' = customer \cup \{cust?\} \\
\hline
\end{array}
$$

Input of the new customer

The new customer must not exist in the system

Total customers (#) must be less than the maximum value

Add the new customer into the system

Question 1

$$\textit{UpdateCustomer}$$

$$\Delta CountCustomer$$

$$cust? : CUSTOMER$$

Input of the customer

$$cust? \in customer$$

The customer must exist in the system

$$customer' = customer \oplus \{cust?\}$$

Update the record of the customer

Question 1

$$
\begin{array}{|l}
\hline
\textit{RemoveCustomer} \\
\hline
\Delta CountCustomer \\
cust? : CUSTOMER \\
\hline
cust? \in customer \\
customer' = customer \setminus \{cust?\} \\
\hline
\end{array}
$$

Input of the customer

The customer must exist in the system

Remove the record of the customer

Question 2

Consider a scenario concerning recording the passengers boarding an aircraft. There are NO seat numbers allocated and passengers are allowed to board on a first-come-first-served basis. The only basic type involved is the set of all possible passengers, *PERSON*:

[PERSON]

The aircraft has a fixed capacity:

$| \; capacity: \quad \mathbb{N}$

Question 2

Given to you the state of the set of passengers on board the aircraft where the number of passengers on board must never exceed the capacity.

State space schema:

$$
\begin{array}{|l}
\hline
\textit{Aircraft} \\
\hline
onboard : \mathbb{P}\ PERSON \\
\hline
\#onboard \leqslant capacity \\
\hline
\end{array}
$$

Question 2

- Write the initial state for the system where the aircraft is empty.

Question 2

- Write the boarding state for the system called *BoardAircraft* to allow a passenger *p?* to board the aircraft where *p?* is of type *PERSON.*

Question 2

- Write the disembarking state called *DisembarkAircraft* to allow a passenger *p?* to disembark from the aircraft where *p?* is of type *PERSON*.

Question 2

- Write a query state for the system called *TotalOnboard* to discover the total number of passengers who are on board.

Question 3

- Consider a specification of a system used to record staff members who go inside and outside of a building.
- We are given a basic type called STAFF.

  [STAFF]

- There is a limit of staff who can be in the building, where this will be represented as *capacity*.

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

Question 3

- The state space schema called *LogStaff* consists of three variables:
  - *user* - the set of all staff members in the system
  - *in* - the set of staff members who are currently inside the building
  - *out* - the set of staff members who are currently outside the building

- The predicate part (invariants) of the system where:
  - No staff member can be simultaneously inside and outside the building
  - The set of users of the system is exactly the union of those who are inside and those who are outside.
  - Those who are inside the building cannot exceed the capacity.

Question 3

State space schema:

$$
\begin{array}{|l}
\hline
\textit{LogStaff} \\
\hline
user, in, out : \mathbb{P}\, STAFF \\
\hline
in \cap out = \varnothing \\
in \cup out = user \\
\#in \leqslant capacity \\
\hline
\end{array}
$$

Question 3

- The system would be initialised so that all sets are empty.

InitLogStaff

$$LogStaff$$

$$user = \emptyset$$
$$in = \emptyset$$
$$out = \emptyset$$

InitLogStaff

$$LogStaff$$

$$user = \emptyset$$

Question 3

- Write an operation called *ReportIn* to report a staff member goes inside the building where:
  - The staff member to be reported in must be currently outside.
  - The total number of staff inside the building must be lesser that the limit.
  - The staff member will be added to the set *in*
  - The staff member will be removed from the set *out*
  - The overall set of users remains unchanged

Question 3

- Write an operation called *ReportOut* to report a staff member goes outside the building where:
  - The staff member to be reported out must be currently inside the building.
  - The staff member will be added to the set *out*
  - The staff member will be removed from the set *in*
  - The overall set of users remains unchanged

Summary

- In conclusion, this lecture described about Z language and the fundamental concepts of Z using sets.
- Z identifiers also was explained in this lecture.
- Furthermore, Z specification including the Z declarations, Z predicates and Z schemas was discussed in detailed.

THANK YOU!!