

# Formal Specification using Z

CSE 308: Software Engineering  
SUNY at Stony Brook

# What is Z?

- Z (pronounced “zed”) is:
  - a form of notation
  - named after Zermelo-Fraenkel\* set theory
  - that provides a concise way to describe the functions of a software system
- Z describes *how* a task should be accomplished, not *what* should be done

\* No, I don't know exactly what that is, either

# Z Specifications

- A Z specification is a mixture of
  - formal mathematical statements
    - to precisely describe a system
  - informal explanatory text
    - to aid reading and comprehension

# Set Theory

- Set theory organizes objects in the world into sets
  - set: an unordered collection of similar elements
- Z expands on set theory by adding
  - types (i.e., sets do not overlap)
  - schemas: named groups of elements

# Set Basics

- Sets are described by lists of elements, in curly braces
  - $EU == \{ Belgium, France, Germany, Italy \}$
  - Use  $\emptyset$  to represent the empty set
- Use the  $\in$  operator to test for membership
  - $Belgium \in EU$  (“Belgium is a member of the EU”)
  - $Japan \notin EU$  (“Japan is not a member of the EU”)

# Set Operators

- Set union ( $\cup$ ) — includes all elements that are members of one or both operand sets
  - $\{ A, B, C \} \cup \{ C, D, E \} = \{ A, B, C, D, E \}$
- Set intersection ( $\cap$ ) — only includes elements that are members of both operand sets
  - $\{ A, B, C, D \} \cap \{ C, D, E, F \} = \{ C, D \}$
- Set difference ( $\setminus$ ) — includes members of the first set that are not members of the second set
  - $\{ A, B, C \} \setminus \{ B, A \} = \{ C \}$

# Set Comparisons

- $\subseteq$  (subset) — true if the set on the right contains all the members of the set on the left
- $\subset$  (proper subset) — the set on the right contains all the members of the set on the left, but has at least one additional member
- $\neq$  (inequality) — at least one of the sets has a member that is not found in the other set

# Propositions and Predicates

- A *proposition* is a definite statement about some member of a set
  - e.g., “the country France has more than 40 million inhabitants”
  - A proposition may be true or false
- A *predicate* is a template for a class of propositions
  - e.g., “the country  $c$  has more than 40 million inhabitants”

# Connectives and Quantifiers

- Use  $\neg$  (not) to negate a proposition
- Propositions can be combined via logical operators:

$\wedge$  (and),  $\vee$  (or),  $\Rightarrow$  (implies),  $\Leftrightarrow$  (if and only if)

- Quantifiers can turn predicates into propositions
  - $\exists o : \textit{opera} \bullet \textit{Beethoven composed } o$  (existential)
  - $\forall p : \textit{project} \bullet p \textit{ has multiple milestones}$  (universal)

# Specification Example

- Goal: create a simple list that pairs English words with their foreign language translations
- We will use tuples of the form:
  - (Native, Foreign)

# First Steps

- Start by describing the sets that are involved

Given sets (general types):

[ Native, Foreign ]

Subsets (properly-formed words):

OrthoNative : set of Native

OrthoForeign: set of Foreign

# More Complex Sets

Set description:

WellFormedVocabs ==

$$\{V : \text{set of (Native, Foreign)} \mid \forall n : \text{Native}, f : \text{Foreign} \bullet \\ (n, f) \in V \Rightarrow n \in \text{OrthoNative} \wedge f \in \text{OrthoForeign} \}$$

Translation: “WellFormedVocabs is a set of (Native, Foreign) pairs such that, for all Native words  $n$  and Foreign words  $f$ , if  $(n, f)$  is in  $V$ , then  $n$  is in OrthoNative and  $f$  is in OrthoForeign”

# Defining Initial Conditions

- We describe the initial state of a system by defining a set of acceptable starting configurations

$\text{InitVocab} == \{ \text{Vocab}' : \text{set of (Native, Foreign)} \mid \text{Vocab}' = \emptyset \}$

# Adding Operations

- To define an operation, we must describe the relationship between the inputs and outputs
  - The operation is defined as a set of tuples
- General notation:
  - input objects have names ending in ‘?’
  - output objects have names ending in ‘!’
  - An “after” object has the same name as the “before” object with a prime (‘’) added

$\text{AddPair} ::= \{ \text{Vocab}, \text{Vocab}' : \text{set of (Native, Foreign)}; n? : \text{Native}; f? : \text{Foreign} \mid \text{Vocab} \in \text{WellFormedVocabs} \wedge n? \in \text{OrthoNative} \wedge f? \in \text{OrthoForeign} \wedge \text{Vocab}' = \text{Vocab} \cup \{ (n?, f?) \} \}$

- Translation:
  - $\text{Vocab}$  and  $\text{Vocab}'$  are sets of (Native, Foreign) elements
  - $n?$  is of type Native and  $f?$  is of type Foreign
  - $\text{Vocab}$  is an element of  $\text{WellFormedVocabs}$
  - $n?$  is an element of  $\text{OrthoNative}$
  - $f?$  is an element of  $\text{OrthoForeign}$
  - $\text{Vocab}'$  is equal to the union of  $\text{Vocab}$  with  $(n?, f?)$

# Summarizing Operations

- Unfortunately, operation descriptions like the previous one can be hard to decipher
- Use a table to summarize operations:

Operation	Inputs/Outputs	Preconditions
AddPair	n? : Native f? : Foreign	n? ∈ OrthoNative f? ∈ OrthoForeign

# Error-Handling

- A specification must be complete
  - It must describe every possible scenario
- What happens if one of the words is incorrect?
  - We should notify the user, and leave Vocab unchanged
- Solution: create two operation definitions (success and failure) and treat them as parts of the same (total) operation

Message == { OK, ErrorInInput }

AddPairError ==

{ Vocab, Vocab' : set of (Native, Foreign);

n? : Native;

f? : Foreign;

rep! : Message |

Vocab ∈ WellFormedVocabs ∧

(n? ∉ OrthoNative ∨ f? ∉ OrthoForeign) ∧

Vocab' = Vocab ∧ rep! = ErrorInInput }

TotalAddPair == AddPair ∪ AddPairError

# Schemas

- A schema is the basic unit of a formal description
  - It describes the admissible (abstract) states and operations of a system
- Schemas can be represented in text or via “box notation”
  - The “box” separates declarations and constraints

# Schema Example

Library

stock : Copy  $\mapsto$  Book

issued : Copy  $\mapsto$  Reader

shelved : **F** Copy

readers : **F** Reader

shelved  $\cup$  dom issued = dom stock

shelved  $\cap$  dom issued =  $\emptyset$

ran issued  $\subseteq$  readers

$\forall r : \text{readers} \cdot \#(\text{issued} \triangleright \{ r \}) \leq \text{maxloans}$

Notes on notation:  
**F** = “finite subset”  
 $\mapsto$  = partial function  
ran = range  
dom = domain  
# = set cardinality  
 $\triangleright$  = set restriction

# Nesting Schemas

- We can define a schema by listing two or more other (subsidiary) schemas in its declaration section
- The new schema includes all of the declarations and predicates of the included schema(s)
- For example, Library could simply be defined as a combination of a LibraryDB schema and a LibraryLoans schema

# Operation Schemas

- We can also use schemas to describe operations
- The top (declaration) section lists the input, output, and internal “variables”
- The bottom (constraints) section describes all of the conditions that must be true before and after the operation

# So What's the Point?

- Z specifications are a compact way to describe the required behavior of a system
  - Think of them as a variant of UML
- Z specifications can be checked for consistency
  - ex. Z/Eves is a system that verifies Z specifications and can perform some theorem-proving

# Next Time

- Theorem-Proving
- PVS (the Prototype Verification System)