Week-6-7
Specifications Languages and Characteristics

Specification Languages and Characteristics

- **Introduction**
- Z Specification Language
  - What is Z?
  - An Example in Z
- **Elements of Z:**
  - Sets and Types, declarations and variables
  - Expressions and operators
  - Identifiers and Layout
  - Introduction to Schema: Examples

Introduction

The **formal methods** notation is used for ***formal specification*** of a software system:

- **As a process:** translation of a non-mathematical description into a formal language
- **As a product:** concise description of the properties of a system in a language with well defined semantics and formal deduction support

A **formal software specification** is a statement expressed in a **language** whose ***vocabulary***, *syntax*, and ***semantics*** are formally defined.

The need for a **formal** semantic definition means that the **specification languages** cannot be based on natural **language**; it must be based on mathematics.

Types of Formal Specification Languages

Formal Specification languages consists of the followings:

- **Model Based Languages:** This is expressed as a system state model which is constructed using well understood mathematical entities such as sets, relations, sequences and functions.

The most widely used notations for developing model based languages are *Vienna Development Method* (VDM) *Zed* (Z) and B.

- **Algebraic Specification Languages:** The algebraic specification languages describe the ***behavior of a system*** in terms of ***axioms*** that characterize its ***desired properties.***

Examples include: OBJ and the Common Algebraic Specification Language (CASL), ACT1,CLEAR, Larch

- **Process oriented Languages:** In these languages processes are denoted and built up by *expressions* and *elementary expressions*, respectively, which describe particularly simple processes.

Example:CSP. CCS, LOTOS

Concurrent systems are described using process oriented formal specification language

❑ Hybrid Languages

Many systems are built with a combination of analog and digital components.

In order to specify and verify such systems it is necessary to use a specification language that encompasses both ***discrete*** and ***continuous mathematics***.

There has been recent interest in these hybrid languages, such as CHARON.

A simple example of a nonlinear hybrid system is that of a temperature controller. The temperature of a room is controlled through a thermostat which continuously senses the temperature and turns the heater on and off.

Characteristics of Specifications Language

A specification language should

- Be a precise thinking tools for designers
- Enhance communication between
  - Designers
  - Designers and programmers
  - Designers and client
- Enable formal(rigorous) analysis
- Lead to separation of concerns
- Allow abstraction, non-determinism
- Encourage a sound design style
- Have a formal syntax and semantics
- Be usable

Z Specification Language

What is Z?
The Z language is a model oriented, formal specification language that was proposed by *Jean-Raymond Abrail, Steve Schuman* and *Betrand Meyer* in 1977 and it was later further developed at the programming research group at Oxford University.

Z specifications can be used to model a system's state *which consist of* ***a collection of state variables and their values*** and some ***operations*** that can change the system's state.

Z specifications describes a similar concepts to object-oriented programming in that the Z state variables are similar to instance variables, and the operations are like methods; furthermore Z demonstrates a form of inheritance..

The components of a Z text can be seen to correspond to code structure such as *modules*, *data types*, *procedures*, *functions*, *classes, objects*.

Z is not a programming language and so cannot be compiled or interpreted into a running program, Z was designed for people, not machines.

Essentially, Z specifications are useful for

- ❑ Requirements elicitations,
- ❑ programs implementations in order to satisfy those requirements,
- ❑ those who test the consequences,
- ❑ and those who write instruction manuals for the sys

Two languages used in Z notations:

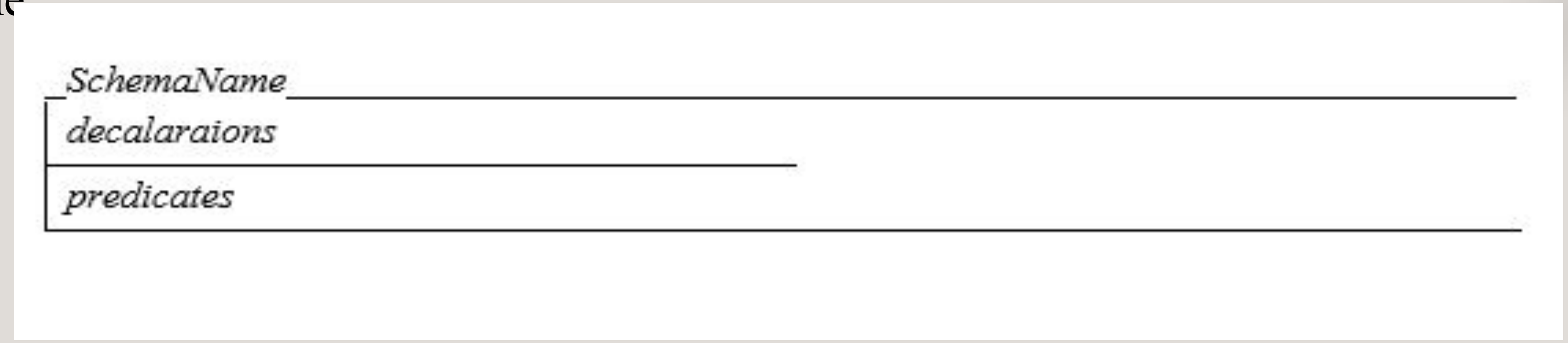
- **Mathematical Language:** This is used to describe various aspects of a design: objects and the relationships between them by using propositional logic, predicate logic, sets, relation and functions.

- **Schema Language: This** is used to structure and compose descriptions: collecting pieces of information, encapsulating them, and naming them for reuse.

Structure for **Z** Specification

Schemas are box like structure that describes **variables** and **specifies** the relationship between the variables.

In a schema as shown below: All ***declarations*** are made above the central line and ***predicates*** are defined below the central line



Declarations: This part contains a

- list of variable declarations;
- references to other schemas (this is called schema inclusion).

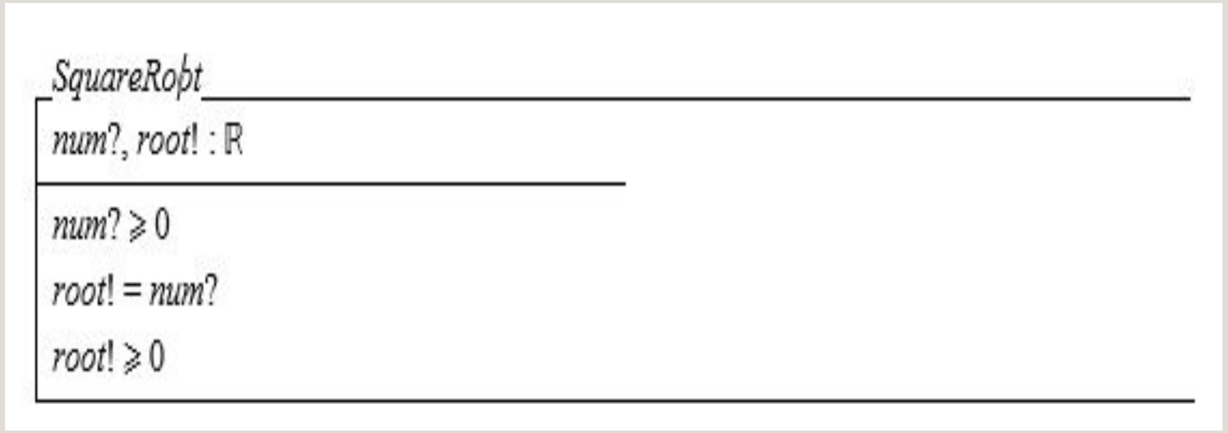
Predicates:. This part of a schema contains:

- a list of predicates, separated either by semi-colons or new lines.

Values of variables are constrained below the central line.

Some Examples in Z

Example 1:
The simple schema below is the specification of the positive square root of a real number



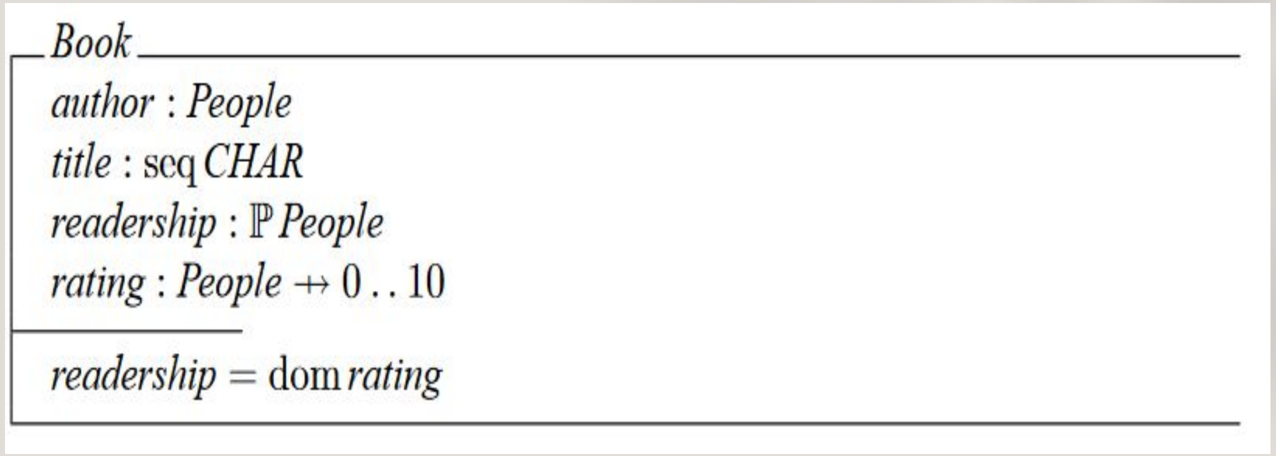
SquareRoot

$num?, root! : \mathbb{R}$

$num? \geqslant 0$

$root! = num?$

$root! \geqslant 0$



Book

$author : People$

$title : \mathrm{seq}\, CHAR$

$readership : \mathbb{P}\, People$

$rating : People \nrightarrow 0\,..\,10$

$readership = \mathrm{dom}\, rating$

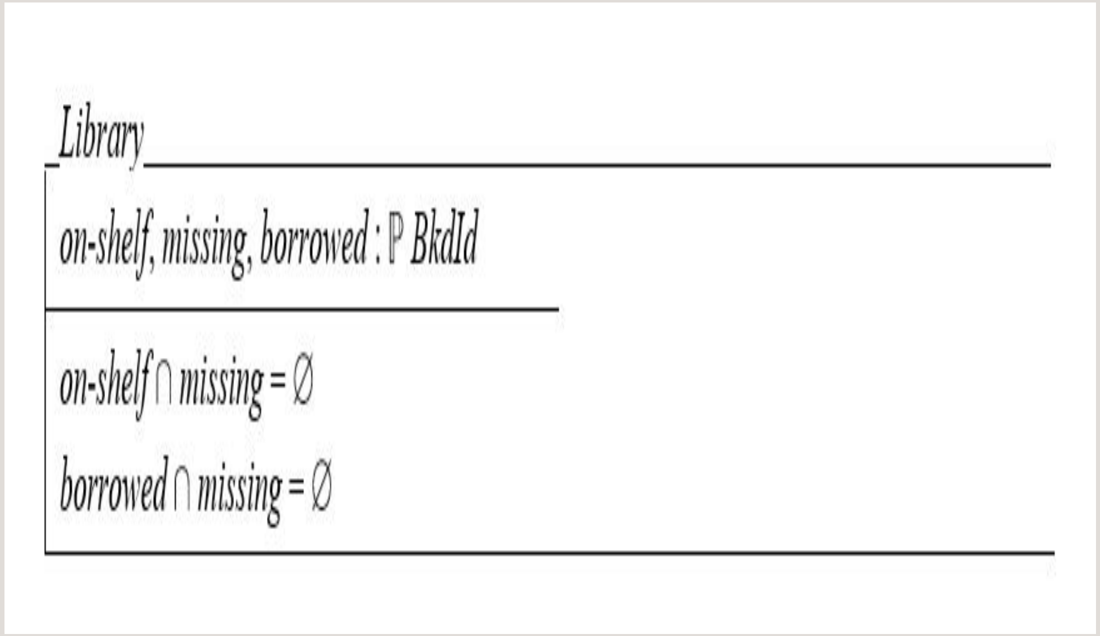
Show in Z/Word and Z/EVES

For zword downloads ;http://sourceforge.net/projects/zwordtools

Some examples in Z

Example2 : The following is a Z specification to borrow a book from a library system. The library is made up of books that are on the shelf; books that are borrowed and books that are missing. The specification models a library

Specification of a library



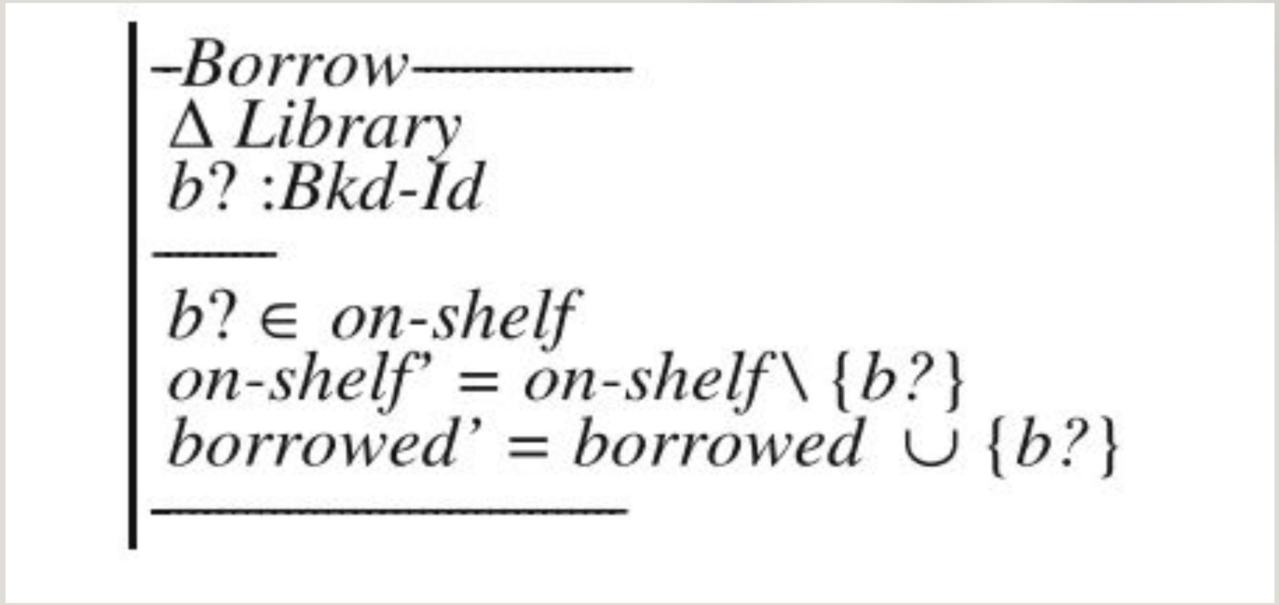
Library

$on\text{-}shelf, missing, borrowed : \mathbb{P}\ BkdId$

$on\text{-}shelf \cap missing = \emptyset$

$borrowed \cap missing = \emptyset$

Specification of a borrow



Borrow

$\Delta\ Library$

$b? : Bkd\text{-}Id$

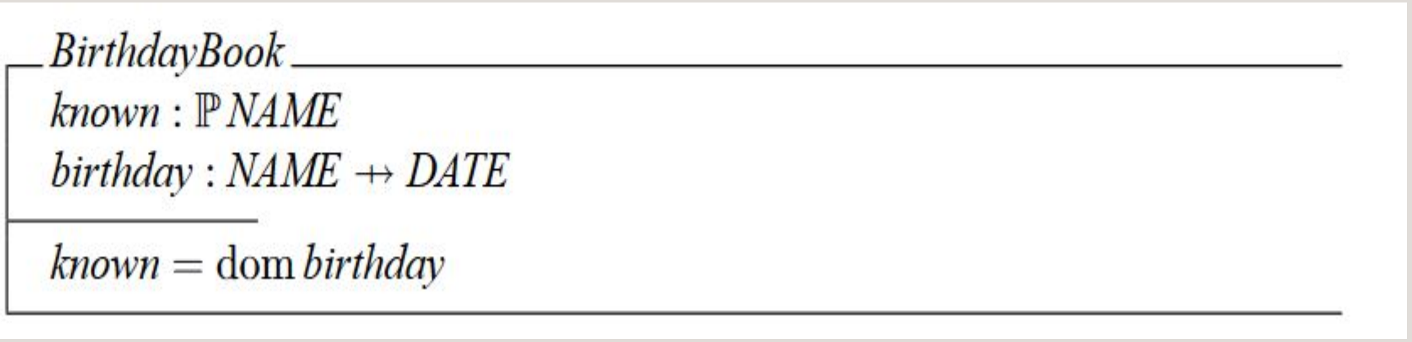
$b? \in on\text{-}shelf$

$on\text{-}shelf' = on\text{-}shelf \setminus \{b?\}$

$borrowed' = borrowed \cup \{b?\}$

More examples on Z

Birthday book as a system for recording birthdays.
Consider the basic types: [NAME, DATE]. The state space of the Birthday Book is specified by:



BirthdayBook

$known : \mathbb{P}\, NAME$

$birthday : NAME ⇸ DATE$

$known = \mathrm{dom}\, birthday$

The ***state variables*** are known (a number of people's names) and birthday (unique dates associated with each known person's name).

Elements of Z

This section presents further interests in set and the three *basic constructs* that appear everywhere in Z texts:

- ✔ **Declarations:** Introduce variables
- ✔ **Expressions:** Describe the values that variables might have
- ✔ **Predicates**: This express constraints that determine which values the variables actually do have

Sets and types, declarations, and variables

We have discussed some basic concepts in sets earlier in the class, but here we present the use of sets in Z.

Example1: The set of even *natural* numbers *less than 10* is given as

$$\{\underbrace{n : \mathbb{N}}_{\text{signature}} | n \neq 0 \wedge n < 10\ mod\ 2 = 0 \cdot n\}$$

Example2:The power set of a set X is the set of all subsets of X and is denoted by ℙX. The set of non-empty subsets of X is denoted by $\mathbb{P}_1 X$ $defined\ as$ $\mathbb{P}_1 X == \{U : \mathbb{P}X | U \neq \emptyset\}$

Example3: A finite set of elements of type X (denoted by $\mathbb{F}$ X) is a subset of X that cannot be put into a one to one correspondence with a proper subset of itself.. That is

$$\mathbb{F}\, X == \{U : \mathbb{P}\, X : \neg \exists\, V : \mathbb{P}\, U \cdot V \neq U \wedge (\exists f : V \rightarrowtail U)\}$$

Note: The expression f:V $\rightarrowtail$ U denotes that f is a bijection from U to V

Types and Bags

Types: $\mathcal{Z}$ is strongly typed, every expression is given a type. Any set can be used as a type. The following are equivalent

(x,y) : A x B

x : A; y : B

x,y : A (when B = A)

That is, every variable in $\mathcal{Z}$ has a particular type (i.e., set from which it is drawn) associated with it which must match appropriately when it is combined with other variables.

Types are used to differentiate the various forms of data present in a specification.

Advantages of Using Types

- **help to structure specifications by differentiating objects;**

We can specify **the set of possible values from which a variable can be drawn**, and we **can then further constrain the variable using a predicate if required**

Example: x is a path of least cost from A to B. We give x a type as: x : Path

and put the constrain predicate as: *least cost x*

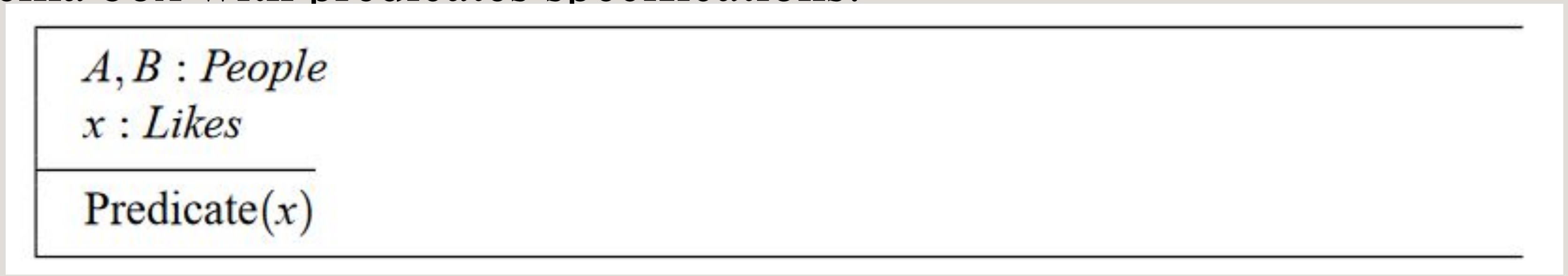
- **help to prevent errors by not allowing us to write meaningless things;**

The use of types means ***the specification can more easily be checked for consistency***, either manually by human inspection, or automatically using the machine assistance of a type-checker.

Example: For example, given Z : Methods and Jonathan : People. It would be meaningless to say

Z = Jonathan

Using a schema box with predicates specifications:



$A, B : People$
$x : Likes$

Predicate(x)

- **they can be checked by computer.**

The declaration x:T says that x is of type T, where T is a set. This is like saying x $\in$ T

$X : \mathbb{N}$
$Z : \mathbb{R}$
Unix : {windows, unix, mac}
$7 : \mathbb{N}$
$(-7 + 5) : \mathbb{Z}$

Questions: What are the types of the following expressions
Mac, log y, sin $(\pi/2)$

A new basic type T is introduced to a specification by putting its name in square brackets .i.e. [T].

This allows us to name the types of a specification without saying what kind of objects they contain. For example, a specification of an *address book* might introduce the basic types ***Name*** and ***Address*** without worrying about the structure of these types:

That is: [Name, Address]

If we know the exact values of a type we use an ***enumerated type*** declaration:

Direction==north|south|east|west

Sets have types too. The type of the set {3,4,5} is "set of ℕ". More precisely, this is written: $\{3,4,5\} : \mathbb{P}\,\mathbb{N}$

Assume S and T have type ℙ M. What are the types of: $S \cup T$ *and* $S \cap T$?

Examples of incorrectly typed expressions:

{4,6,unix} or {windows,mac} ∪ {bmw,rover,ford}

Bags: A **bag** is similar to a set except that there may be **multiple occurrences** of each element in the bag. A bag of elements of type X is defined as a partial function from the type of the elements of the bag to positive whole numbers. The definition of a bag of type X is

bag X == X ⇸ $\mathbb{N}_1$

Example: For example, a bag of marbles may contain 3 blue marbles, 2 red marbles and 1 green marble. This is denoted by B = ⟦b,b,b,g,r,r⟧ . The bag of marbles is thus denoted by:

bag Marble == Marble ⇸ $\mathbb{N}_1$

The function ***count*** determines the number of occurrences of an element in a bag. For the example above, ***count Marble b = 3***, and ***count Marble y = 0*** since there are no yellow marbles in the bag. This is defined formally as:

count bag X y = 0   y ∉ bag X

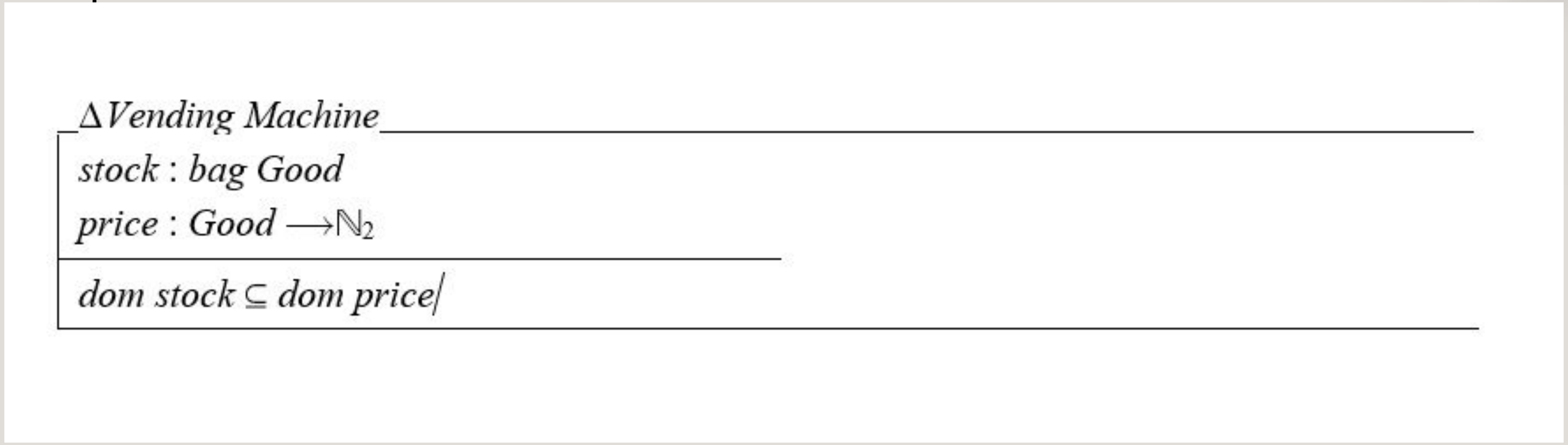
count bag X y = (bag X)(y)   y ∈ bag X

An element y is in bag X if and only if y is in the domain of bag X

y is in bag X ⇔ y ∈ $\boldsymbol{dom(bag\ X)}$

Notes: A bag may be used to record the number of occurrences of each product in a warehouse as part of an inventory system.

For instance, it may model the number of items remaining for each product in a vending machine as shown in the specification below



ΔVending Machine

$stock : \text{bag } Good$

$price : Good \longrightarrow \mathbb{N}_2$

$\text{dom } stock \subseteq \text{dom } price$

The operation of a vending machine would require other operations such as ***identifying the set of acceptable coins***, ***checking that the customer has entered sufficient coins to cover the cost of the good***, *returning change to the customer and updating the quantity on hand of each good after a purchase.[Check Dil, 1990]*

Declarations and Variables

Declarations introduce variables and indicate which set each variable belongs. The simplest way to define an object is to declare it.

Example: i : Z [i is an integer.]

d_1, d_2 : DICE or d_1, d_2:1..6 d_1, d_2 are two numbers in the range 1 .. 6.

signal: {red, yellow ,green}

Example: A hotel switchboard uses a software package to maintain a record of call charges to current guests. A formal specification of this system could include the following declaration**:**

[Guest, Room]

This declare two basic types to represent the ***set of all guests*** and ***the set of all rooms***.

A variable of the type Guest is introduced by the following declaration:

x : Guest

There are many other kinds of constraints besides set membership, which in Z variables are called axiomatic definitions

Examples:

$$\begin{array}{|l}
d_1, d_2 : DICE \\
\hline
d_1 + d_2 = 7
\end{array}$$

$$\begin{array}{|l}
d_1, d_2 : DICE \\
\hline
d_1 + d_2 = 7 \\
d_1 < d_2
\end{array}$$

(a)

$$\begin{array}{|l}
double : \mathbb{N} \rightarrow \mathbb{N} \\
\hline
\forall\, n : \mathbb{N} \bullet double(n) = 2n
\end{array}$$

(b)

$$\begin{array}{|l}
halve : \mathbb{N} \nrightarrow \mathbb{N} \\
\hline
\mathrm{dom}\, halve = \{n : \mathbb{N} \mid \exists\, m : \mathbb{N} \bullet 2m = n\} \\
\forall\, n : \mathrm{dom}\, halve \bullet 2 * halve(n) = n
\end{array}$$

(e) Let *People* be the set of all living people.

$$\begin{array}{|l}
birth : People \rightarrow \mathbb{N} \\
\hline
\forall\, p : People \bullet birth(p) \text{ is the year of } p\text{'s birth}
\end{array}$$

Expressions and Operators

Expressions describe the ***values*** that variables might have. Expressions enable us to describe values in terms of **names** and **literals** we have already defined. Expressions are formulas where names and literal values appear together with operators

Arithmetic expressions: The Z mathematical tool-kit defines the usual arithmetic operators addition, subtraction, and multiplication +, —, and *.

Set expressions: Sets are central in Z, and the tool-kit defines many operators for them. Here are a few of the most important ones. The union operator U combines sets

i). {1, 2, 3} U {2, 3,4} = {1,2, 3, 4}
ii) The **difference operator** \ removes the elements of one set from another. {1,2,3,4}\{2,3} = {1,4}
iii) The **intersection operator** n finds the elements common to both sets. {1, 2, 3}∩{2, 3,4} = {2, 3}

Identifiers and Layout

Identifiers may be composed of *upper* and *lower* case letters, *digits*, and the *underscore character*; must *begin with a letter*

Identifiers may have the following suffixes:

– ? means an ***input*** variable

– ! means an ***output*** variable

– $'$ means a ***new value*** (i.e., the after-operation value)

Case study

Schema identifiers may have prefixes:

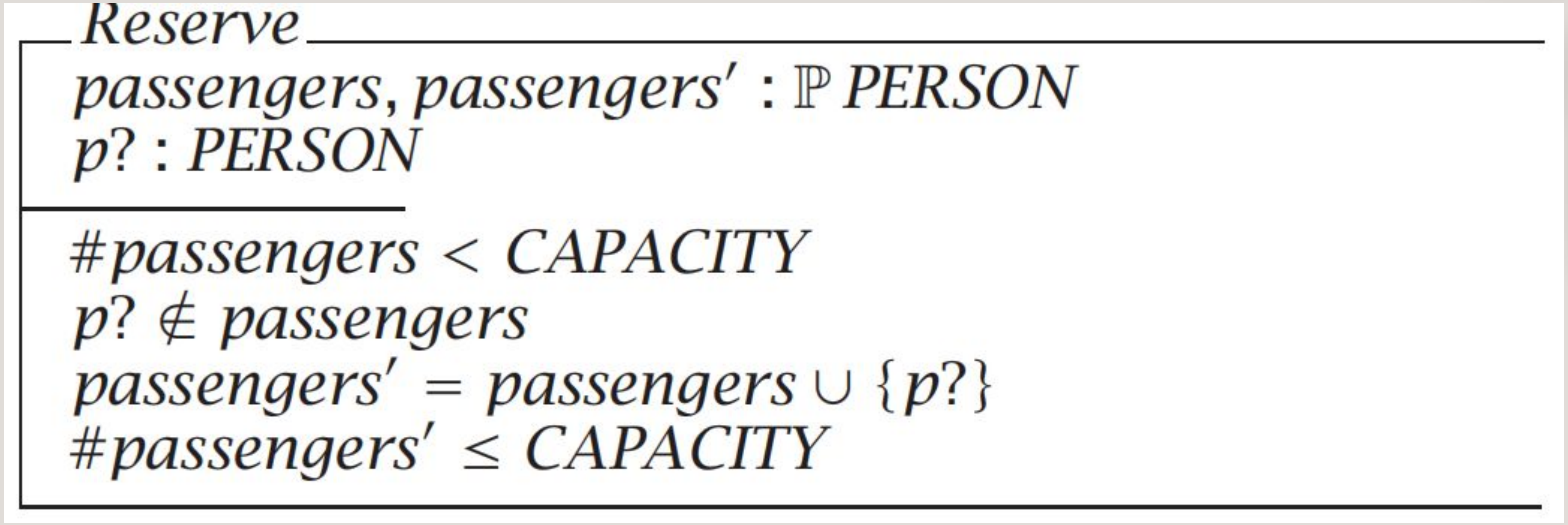
– delta(Δ) means the ***state has changed*** (described later)

- XI(Ξ) means ***no change in the state*** (described later)

If S is a schema, then ΔS is the union of S and S' and ΞS is the schema ΔS together with v = v' for all declared variables v of S.

Reservation System

Consider operation *Reserve* again:



Reserve

$passengers, passengers' : \mathbb{P}\, PERSON$

$p? : PERSON$

$\# passengers < CAPACITY$

$p? \notin passengers$

$passengers' = passengers \cup \{p?\}$

$\# passengers' \leq CAPACITY$

Conventionally, two schemas are given to define the state of data, one describing state before an operation, and one for describing state after the operation.
For a simple reservation system we may have:

ResSyst

$passengers : \mathbb{P}\, PERSON$

$\#passengers \leq CAPACITY$

ResSyst′

$passengers' : \mathbb{P}\, PERSON$

$\#passengers' \leq CAPACITY$

Reserve operation can now be specified as:

Reserve

ResSyst
ResSyst′
$p? : PERSON$

$\#passengers < CAPACITY$
$p? \notin passengers$
$passengers' = passengers \cup \{p?\}$

Z-Specifications: Examples

Example-1: The following example illustrates a specification of a system used to ***check staff members in and out of a building.*** Since we will be dealing with elements of type staff, we introduce the type Staff as a basic type:

[Staff]

The state of the system can be described by the following schema

$$
\begin{array}{|l}
\textit{Log} \\
\hline
users, in, out : \mathbb{P}\, Staff \\
\hline
in \cap out = \{\} \wedge \\
in \cup out = users \\
\hline
\end{array}
$$

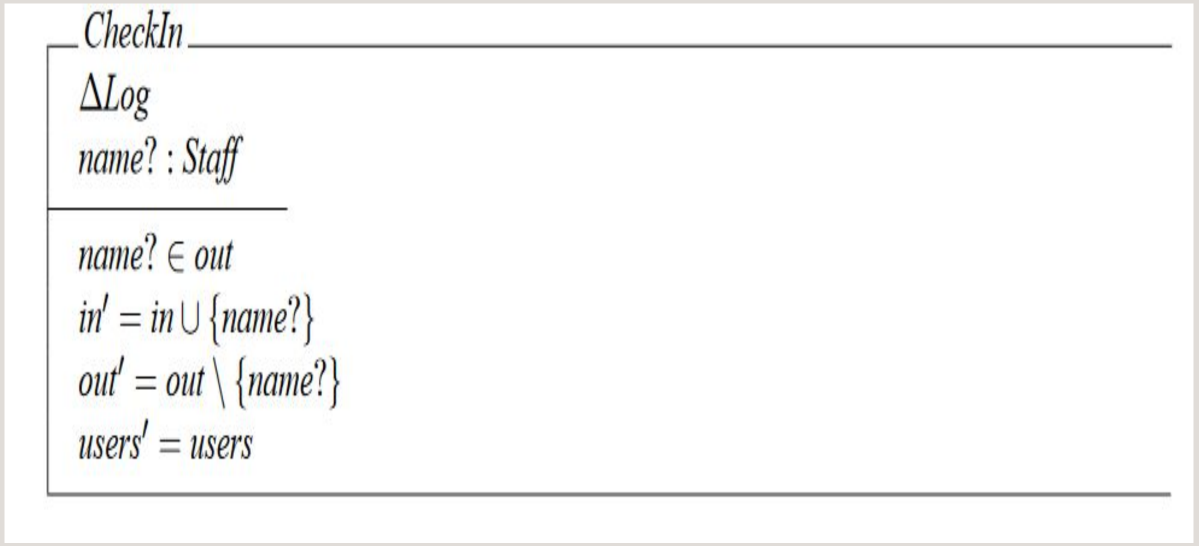
The state consists of three components modelling

- •the set of users of the system,
- •the set of staff members who are currently in and
- •the set of staff members who are currently out.

The **predicate** part of the **state schema** describes an Invariant of the system. The invariant says that

- No staff member is simultaneously in and out.
- The set of users of the system is exactly the union of those who are in and those who are out.

An operation to check a staff member into the building is specified as follows



CheckIn

ΔLog

$name? : Staff$

$name? \in out$

$in' = in \cup \{name?\}$

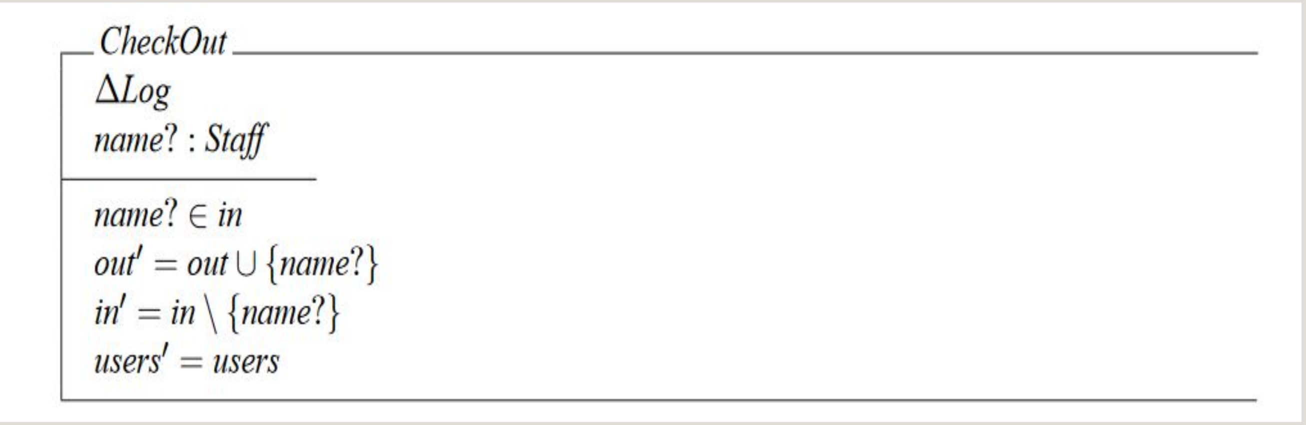
$out' = out \setminus \{name?\}$

$users' = users$

This has an input parameter representing the member of staff to be checked in. The predicate part says that:

- The staff member to be checked in must currently be out. This is a pre-condition on the operation.
- The staff member is added to the set in.
- The staff member is removed from the setout.
- The overall set of users remains unchanged

Similarly, an operation to check a staff member out of the building may be specified as follows



CheckOut

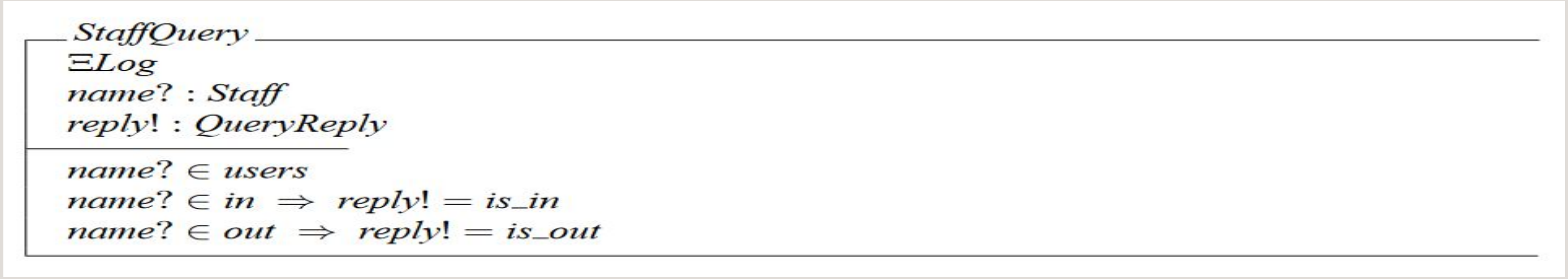
ΔLog
$name? : Staff$

$name? \in in$
$out' = out \cup \{name?\}$
$in' = in \setminus \{name?\}$
$users' = users$

A query operation to check whether a particular member of staff is ***in or out*** will give an output parameter of the following ty

QueryReply==is_in|is_out

The operation is then specified as:



StaffQuery

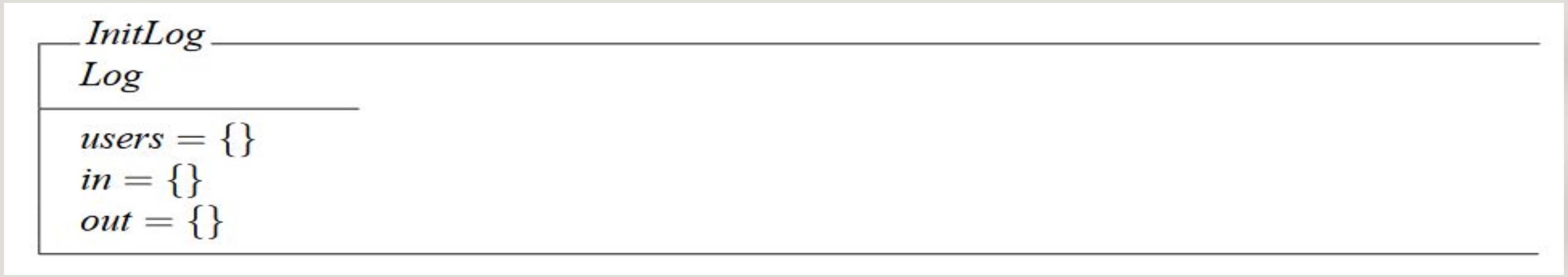
ΞLog
$name? : Staff$
$reply! : QueryReply$

$name? \in users$
$name? \in in \Rightarrow reply! = is\_in$
$name? \in out \Rightarrow reply! = is\_out$

The declaration Ξlog says that the operation makes no change to the state of the system

Initialization

Typically the system would be initialized so that all sets are empty.

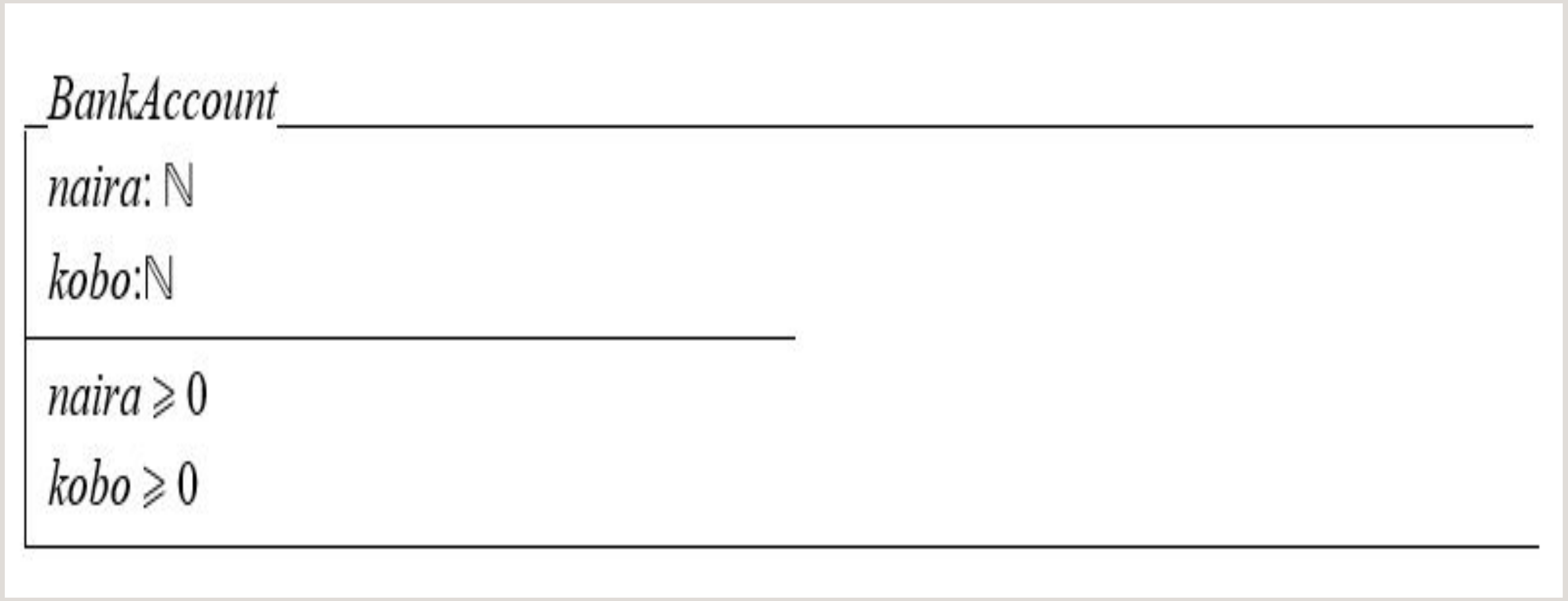


```
─ InitLog ──────────────────────────────
│ Log
├──────────────
│ users = {}
│ in = {}
│ out = {}
└───────────────────────────────────────
```

Recap of the specification:

- **State Schema**: Components/Objects of system.
- **Invariant:** Static relationship between state components
- **Operation Schemas:–**Condition on Input parameters.–Relationship between before-and after-states.–Output parameters.
- **Initialization**

Bank System



BankAccount

$naira : \mathbb{N}$

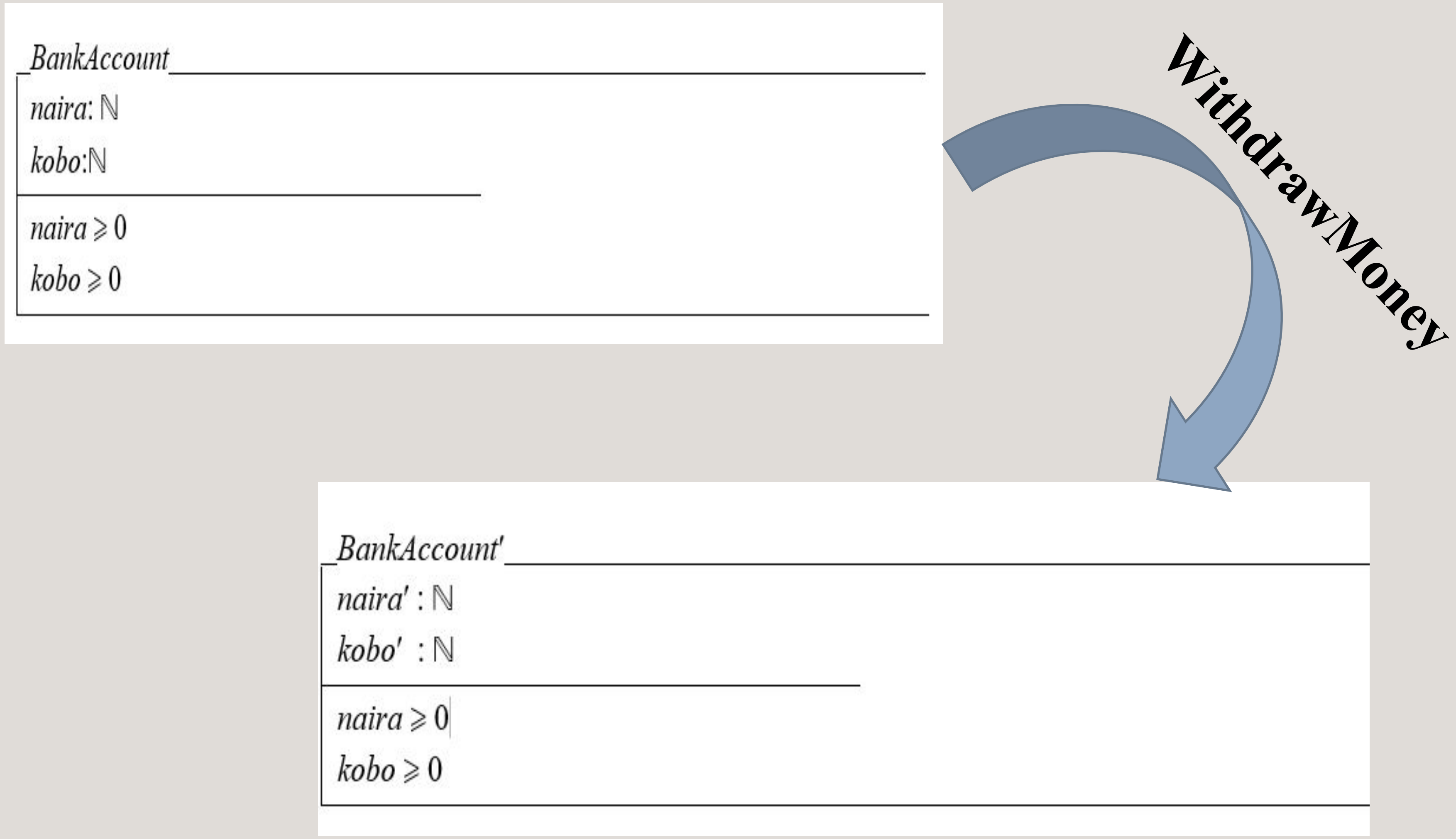
$kobo : \mathbb{N}$

$naira \geqslant 0$

$kobo \geqslant 0$

Banking System

Withdraw Money



BankAccount

$naira : \mathbb{N}$

$kobo : \mathbb{N}$

$naira \geqslant 0$

$kobo \geqslant 0$

BankAccount'

$naira' : \mathbb{N}$

$kobo' : \mathbb{N}$

$naira \geqslant 0$

$kobo \geqslant 0$

Case Study I:Phone Directory for a University

Objective:
Create a telephone directory system (called *PhoneDir*), for a university to maintain a record of faculty and their telephone numbers

Software Requirements:

- A faculty may have one or more telephone numbers
- Some faculty may not have a telephone number yet
- A number may be shared by two or more faculty
- Must be able to add new faculty and/or new entries
- Must be able to remove faculty and/or existing entries
- Must be able to query the system for a faculty or number

Output messages:

MESSAGE ::= ‘OK

|‘Faculty already exists’

| ‘No such faculty’

| ‘Faculty has no number’

| ‘Invalid number’

| ‘Invalid entry’

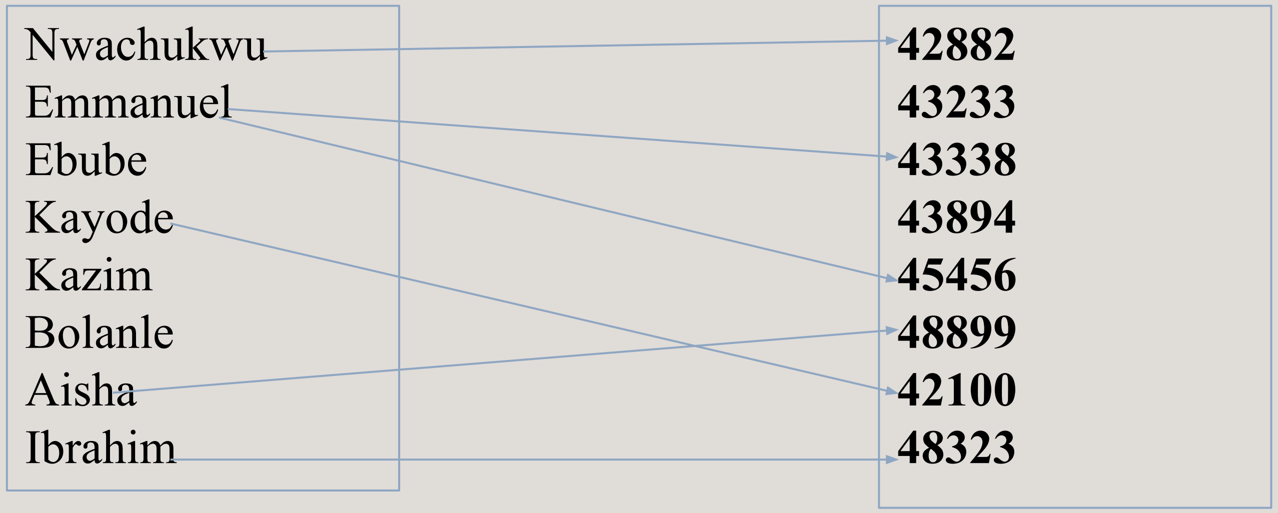
| ‘Entry already exists

Abstract State of *PhoneDir* System

- A set of type *PERSON* representing the faculty: *faculty* : **P** *PERSON*
- Abstract representation of an instance of *faculty*:

Nwachukwu
Emmanuel
Ebube
Kayode
Kazim
Bolanle
Aisha

- Abstract representation of an instance of *directory*:



- *directory* is a subset of Cartesian product of *PERSON* × *PHONE*

In Z, we represent the above as a “relation”

- A relation is a set of connections between two sets:

$directory = \{(nwachukwu, 42882), (emmanuel, 43338), (emmanuel, 45456), \ldots\}$

- Alternatively, use $\rightarrow$symbol to show connections:

$directory = \{nwachukwu \rightarrow 42882, emmanuel \rightarrow 43338, emmanuel \rightarrow 45456, \ldots\}$

- Declaring a variable of type relation: $directory : PERSON \leftrightarrow PHONE$

Partial list of operations available on a relation:

- domain : dom $directory = \{nwachukwu, emmanuel, aisha, ibrahim\}$
- range: ran $directory = \{42882, 43338, 45456, 42100\}$
- inverse $directory^{\sim} = \{42882 \rightarrow nwachukwu, 43338 \rightarrow emmanuel, \ldots\}$

Phone Directory Specifications

Using relations, we specify a phone directory which relates people to their phone numbers. We assume the following basic types:

[Person, Phone]

The state schema of the directory is given by the following schema:

Directory

$dir : Person \leftrightarrow Phone$

Initially the directory is empty:

InitDirectory

Directory

$dir = \{\}$

Phone Directory Contd

We add an entry to the directory with the following operation:

AddEntry

$\Delta Directory$

$name? : Person$

$number? : Phone$

$dir' = dir \cup \{name? \mapsto number?\}$

An operation to get all the numbers associated with a name is specified as:

GetNumbers

$\Xi Directory$

$name? : Person$

$numbers! : \mathbb{P}\, Phone$

$numbers! = \{ n : Phone \mid (name? \mapsto n) \in dir\}$

Phone Directory contd

Equally we could specify an operation to get the names associated with a number:

GetNames

$$\Xi Directory$$

$$number? : Phone$$

$$names! : \mathbb{P}\, Person$$

$$names! = \{ p : Person \mid (p \mapsto number?) \in dir \}$$

The RemoveEntry operation removes an entry from the directory:

RemoveEntry

$$\Delta Directory$$

$$name? : Person$$

$$number? : Phone$$

$$dir' = dir \setminus \{ name? \mapsto number? \}$$

Students Enrolment System

Problem Statement: Specifications for student’s enrolment management system

Objectives: The specification is describe operations which include the following
Add student’s details such Matric Number, Name, Level, program and address into school database.

Note:A specification to convert requirements written in natural language to Z formal specification language method is described as shown

Enrolment Specs

Assignment: More Operations

The following is an outline of an operation to register a new staff member and to check which staff are already check-in. Fill in the gaps

Register

ΔLog

$name? : Staff$

QueryIn

ΞLog

$names! : \mathbb{P}\, Staff$