INTRDUCTION TO Z SPECIFICATIONS

Formal Methods

BCS 2213

Semester 1 Session 2014/2015

What is Z

- First off – its pronounced Zed
  - After the Zermelo-Fränkel set theory
- Z is based on the mathematical notation based on *axiomatic* set theory, lambda calculus and first order predicate logic.
- It has been developed by the Programming Research Group at the Oxford University Computing Laboratory and elsewhere since the middle of 1980.

What is Z

- Z – is a modelling notation.
- Can be used to model the behavior of a system.
- Z models a system by representing its *state* (a collection of *state variables* and their values) and *operations* that may change the state.
- Z is just a notation, it is not a method.
- Z can support many different methods.

Z is….

- Z is not software or executable code.
- Describe “what” the system must do without saying “how” it is to be done.
- Can be used to represent structure of software code: data types, procedures, functions, modules, classes, objects etc.
- Z is suitable for
  - Procedural programming language
  - Functional approach
  - Object-oriented programming. There are some object oriented languages that extend Z (Object-Z, Z++)

Z is…

- Designed for people not machines
- Z Notation is a mixture of boxes, text, Greek letters and invented pictorial symbols.
- Includes two notations
  - Notation to express ordinary discrete mathematics
  - Notation to structure mathematical text *(schema)*.
- Problems with Z notation
  - Z notation uses many non-ASCII symbols
  - The specification includes suggestions for rendering the Z notation symbols in ASCII as well as LaTeX.

Z Mathematical tool-kit

- Collection of mathematical objects and operators : definitions and laws concerning sets, tuples, relations, functions, sequences and their operators.
- Plays the same role as standard library of types and functions in programming.
- Sample of Z Mathematical tool-kit : http://staff.washington.edu/jon/z/toolkit.html

Elements of Z - Sets

A **set** is a collection of *distinct* objects

Order of objects is not significant

A set can have two or more members which are identical

{ 1, 2, 3, 4, 5, 6 }

{ 4, 5, 6, 1, 2, 3 }

{ 4, 4, 5, 5, 6, 6, 1, 2, 3 }

{ i: ▨ | 1 ≤ i ≤ 6 }

{ red, yellow, green }

$\emptyset$ The Null Set or Empty Set. This is a set with no elements

Notation. Usually we denote sets with upper-case letters, elements with lower-case

Elements of Z - Types

Every object in Z belongs to a set called its type. E.g. 1, 2, 3 etc. belong to the **basic** type $\mathbb{Z}$.

red, green belong to the **free** (user defined) type COLOR. Free types are like enumerations.

COLOR ::= red | green | blue | yellow | cyan | magenta | white | black

Every type must be introduced in a declaration.

Elements of Z – Basic types

$\mathbb{Z}$ - The set of integers

$\mathbb{N}$ - The set of natural numbers, starting with zero

$\mathbb{N}_1$ - The set of strictly positive numbers, starting with one

$\mathbb{R}$ - The set of real numbers

Definitions

$\mathbb{N} == \{\ n : \mathbb{Z} \mid n \geq 0\ \}$

$\mathbb{N}_1 == \mathbb{N} \setminus \{ 0 \}$

Elements of Z – Compound Types

Set type: $\mathbb{P}\,T$

The type of set of elements of type T

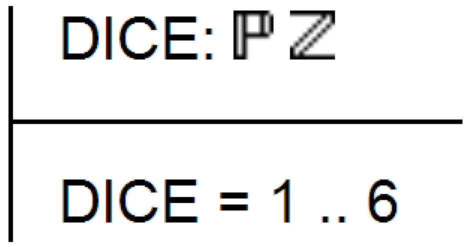
$\mathbb{P}_1$ - Non-empty sets

$\mathbb{P}_1\,X == \{\ S : \mathbb{P}\,X \mid S \neq \varnothing\ \}$

Elements of Z - Variables

A variable is a name for an object.
Variables are introduced in declarations.

$x: S$ The value of x belongs to set S
$y: \mathbb{P}\, S$ $\mathbb{P}\, S$ means set of S



$DICE: \mathbb{P}\, \mathbb{Z}$

$DICE = 1 \,..\, 6$

Elements of Z – Operations on Sets

The *size* operator # counts elements.

$\#\{ \text{red}, \text{yellow}, \text{blue}, \text{green}, \text{red} \} = 4$

The *union* operator ∪ combines sets.

$\{ 1, 2, 3 \} \cup \{ 2, 3, 4 \} = \{ 1, 2, 3, 4 \}$

The *difference* operator \ removes the elements of one set from another.

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

The *intersection* operator ∩ finds the elements common to both sets.

$\{ 1, 2, 3 \} \cap \{ 2, 3, 4 \} = \{ 2, 3 \}$

Elements of Z – Operations on Sets

Set operators work with sets of any type, but

$\{ 1, 2, 3 \} \cup \{ \text{red, green} \}$ Type error!

Involved sets must have the same type T

$$x \in S_1 \cup S_2 \Leftrightarrow (x \in S_1 \vee x \in S_2)$$

Combining Sets – Set Union

- $$A \cup B$$

- "*A* union *B*" is the set of all elements that are in *A*, or *B*, or both.

- This is similar to the logical "or" operator.

Combining Sets – Set Intersection

- $A \cap B$
- "*A* intersect *B*" is the set of all elements that are in *both* *A* and *B*.
- This is similar to the logical "and"

Set Complement

- $\overline{A}$
- "*A* complement," or "not *A*" is the set of all elements not in *A.*
- The complement operator is similar to the logical not, and is reflexive, that is,

$$\overline{\overline{A}} = A$$

Set Difference

- $A\backslash \mathrm{B}$

- The set difference “A minus B” is the set of elements that are in A, with those that are in B subtracted out. Another way of putting it is, it is the set of elements that are in A, *and* not in B, so
- $A\backslash \mathrm{B} = A \cap \bar{B}$

Examples

$$\Omega = \{1, 2, 3, 4, 5, 6\}$$

$$A = \{1, 2, 3\} \qquad B = \{3, 4, 5, 6\}$$

$$A \cap B = \{3\} \qquad A \cup B = \{1, 2, 3, 4, 5, 6\}$$

$$B \setminus A = \{4,5,6\} \qquad \bar{B} = \{1, 2\}$$

Elements of Z – Notation for Sets

Set comprehension

$$\{x : T \mid pred(x) \bullet expr(x)\}$$

The set of all elements that result from evaluating $expr(x)$ **for all** x **of type** T **for which** $pred(x)$ **holds**

Example

$$\{x : \mathbb{Z} \mid prime(x) \bullet x * x\}$$

The set of all squares of integer primes i.e. {1, 4, 9, 25 …}

Elements of Z – Notation for Sets

Abbreviation

$\{x : T \mid pred(x)\}$ **for** $\{x : T \mid pred(x) \bullet x\}$

Example

$\mathrm{N} = \{x : \mathrm{Z} \mid x \geq 0\}$

The empty set

$\emptyset = \{x : T \mid \mathbf{false}\}$

Note:

$\emptyset = \emptyset[T]$ **is typed**

Elements of Z – Predicates on Sets

$x \in S$ Set membership, x is a member of S.

$x \notin S$ - Non-membership: x is not an element of S

$S \subseteq T$ - Subset: all elements of S belong to T

$S \subset T$ - Proper subset: S is a subset of T and S is not equal to T

Predicates are not expressions. They do not have values, they can be *true* or *false*.

Membership Relationships

- *Definition.* Subset.

$A \subseteq B$ “A is a subset of B”

We say “A is a subset of B” if $x \in A \Rightarrow x \in B$, i.e., all the members of A are also members of B. The notation for subset is very similar to the notation for “less than or equal to,” and means, in terms of the sets, “included in or equal to.”

Membership Relationships

- *Definition.* Proper Subset.

  $A \subset B$ “A is a proper subset of B”

  We say “A is a proper subset of B” if all the members of A are also members of B, but in addition there exists at least one element $c \in B$ such that $c \notin A$ but

The notation for subset is very similar to the notation for “less than,” and means, in terms of the sets, “included in but not equal to.”

Elements of Z – Predicates on Sets

$\subseteq$ subset relation

S_1 and S_2 must have the same type

$$S_1 \subseteq S_2 \Leftrightarrow (\forall x : S_1 \mid x \in S_2)$$

$\times$ Cartesian product

The *Cartesian product* of two sets *A* and *B*, denoted by *A* × *B* is the set of all ordered pairs (*a*, *b*) such that *a* is a member of *A* and *b* is a member of *B*.

$$(x_1, \ldots, x_n) \in S_1 \times \ldots \times S_n \Leftrightarrow (x_1 \in S_1 \land \ldots \land x_n \in S_n)$$

Elements of Z – Arithmetic Operation

Operations allow to build larger expressions from smaller ones.

m+n	Addition
m-n	Subtraction
m*n	Multiplication
m **div** n	Division
m **mod** n	Remainder (modulus)
m ≤ n	Less than or equal
m .. n	Number range (up to)
min A	Minimum of a set of numbers
max A	Maximum of a set of numbers

Definition of a Function

A function is a correspondence between two sets X and Y that assings to each element x of set X exactly one element y of Y.

Domain and Range

For each element x in X, the corresponding element y in Y is called the value of the function at x. The set X is called the domain of the function, and the set of all function values, Y, is called the range of the function.

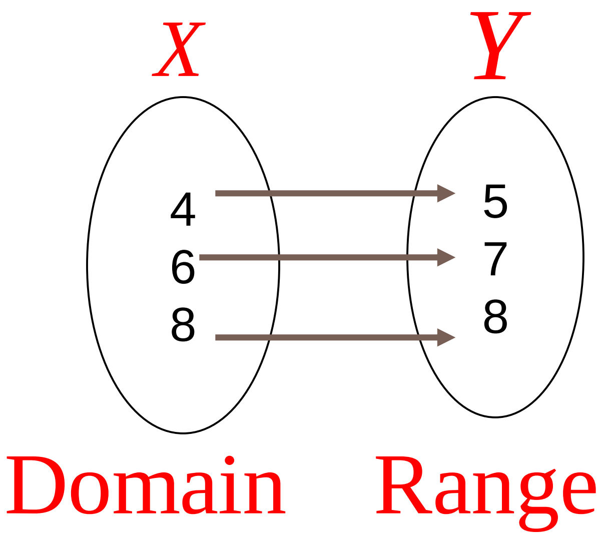
Ex 1: Determine whether each relation is a function.

a. $\{(4,5),(6,7),(8,8)\}$

b. $\{(5,6),(4,7),(6,6),(6,7)\}$

Solution We begin by making a figure for each relation that shows set X, the domain, and set Y, the range.

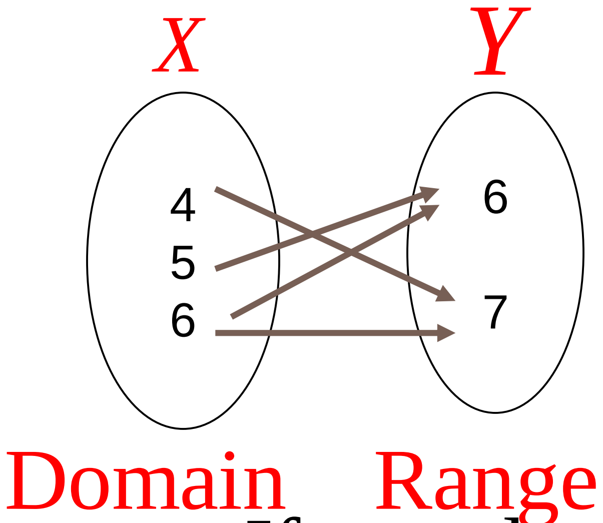
Solution for part (a)



The figure shows that every element in the domain corresponds to exactly one element in the range.

No two ordered pairs in the given relation have the same first component different second components. Thus, the relation is a function

Solution for part (b)



The figure shows that 6 corresponds to both 6 and 7.

If any element in the domain corresponds to more than one element in the range, the relation is not a function, Thus, the relation is not a function.

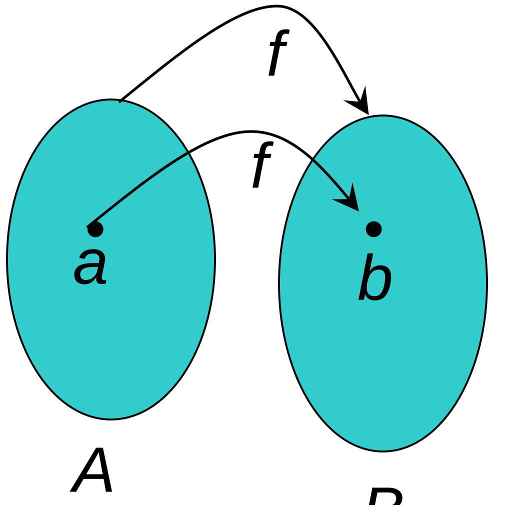
Definition of Functions

- Given any sets *A*, *B*, a function *f* from *(or "mapping") A to B* ($f{:}A\rightarrow B$) is an assignment of **exactly one** element $\boldsymbol{f(x)\in B}$

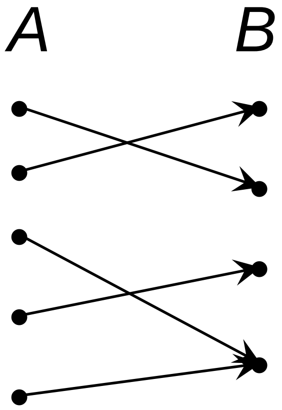
  to each element $\boldsymbol{x\in A}$.

Graphical Representations

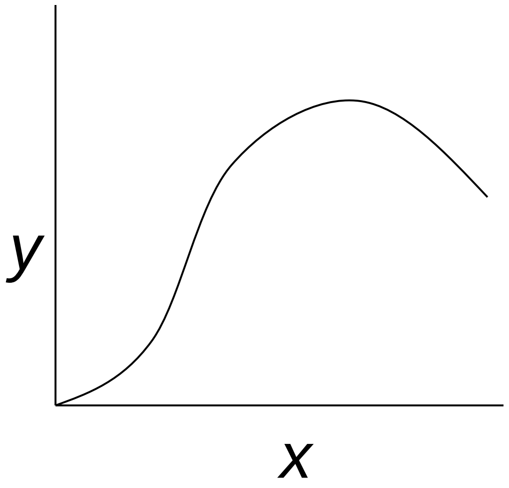
- Functions can be represented graphically in several ways:



Like Venn diagrams



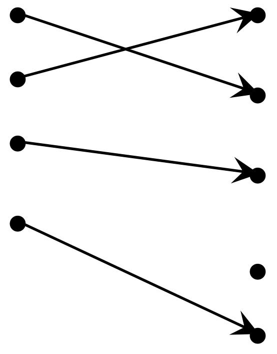
Graph



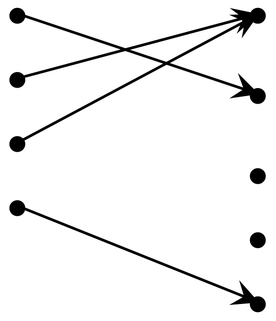
Plot

One-to-One Functions

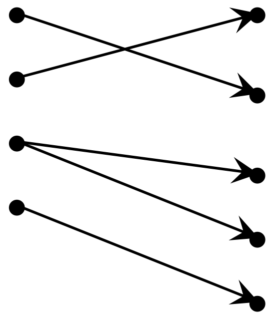
- A function is *one-to-one (1-1)*, or *injective*, or *an injection*, iff only <u>one</u> element of the domain is mapped <u>to</u> any given <u>one</u> element of the range.



One-to-one

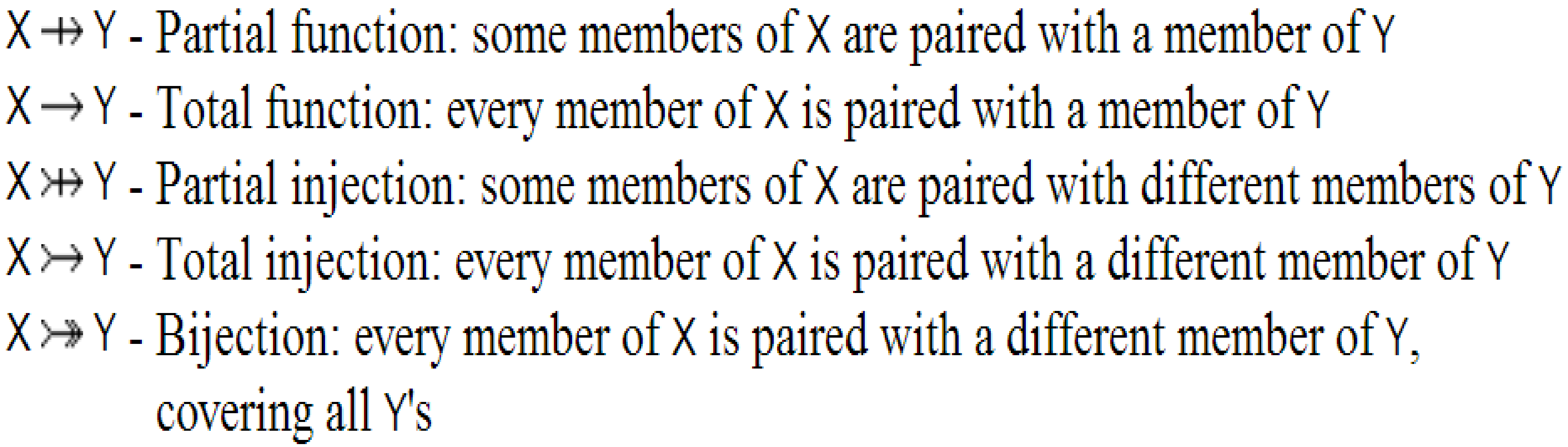


Not one-to-one



Not even a function!

Elements of Z – Functions



$X \nrightarrow Y$ - Partial function: some members of X are paired with a member of Y

$X \rightarrow Y$ - Total function: every member of X is paired with a member of Y

$X ⤔ Y$ - Partial injection: some members of X are paired with different members of Y

$X \rightarrowtail Y$ - Total injection: every member of X is paired with a different member of Y

$X ⤖ Y$ - Bijection: every member of X is paired with a different member of Y, covering all Y's

Elements of Z – Pairs and binary relations

(x,y)	- The pair x,y
$x \mapsto y$	- Maplet: x maps to y, same as (x,y)
first p	- First element of pair p
second p	- Second element of pair p
$X \leftrightarrow Y$	- Binary relation
dom R	- Domain: the set of first elements of all pairs in R
ran R	- Range: the set of second elements of all pairs in R

Elements of Z – Logical Operations

$\neg$	negation	
$\wedge$	conjunction	
$\vee$	disjunction	
$\Rightarrow$	implication	(note: not $\rightarrow$)
$\Leftrightarrow$	equivalence	(note: not $\leftrightarrow$)

How To Model A System

- Z explicitly describes behavior in terms of mathematical types (set, sequences, relations, functions) & defined operations
- Z specification includes
  - types – mathematical syntax of object being specified
  - invariants - properties of modeled object
  - operations – expressed in terms of pre/post conditions
- Z decomposes specifications into manageably sized module's, called schemas. Schemas are divided into 3 parts:
    - A State
    - A collection of *state variables* and their values
    - *Operations* that can change the state

Static and Dynamic Aspects of Behaviour

- Static aspects
  - The *states* that a system can occupy.
  - The *invariant relationships* that are maintained as the system moves from state to state.
- Dynamic aspects
  - The possible *operations*.
  - The *relationship* between their inputs and outputs.
  - The *changes* of state that happen.

How to Specify Static Aspects?

Use *schemas* - math in a box with a name - to describe the state space (state variables and invariants).

```
┌──── Name of schema ─────────────────────
│
│ Declaration of state variables.
│
├─────────────────────────
│ Invariant relationship between values of the variables
└──────────────────────────────────────────
```

Example of a State Schema

- Simple text editor with limited memory
- Editor state modeled by two state variables, the texts to the **left** and **right** of the cursor
- [CHAR] – given set (user defined type)

```
[CHAR]

TEXT == seq CHAR

│ maxsize: ℕ
├──────────────
│ maxsize ≤ 65535

┌─ Editor ──────────────────────────────
│ left, right: TEXT
├────────────────
│ # (left ⁀ right) ≤ maxsize
└───────────────────────────────────────
```

The Birthday Book Example

- This specification will allow you to do 3 things:
  - Add a person’s name and birthday
  - Store that information
  - Find a birthday by name
  - Find a name by given date of birthday
- For example, lets define a state of the system, which has three people in the set **known**, with their birthdays recorded by the function **birthday**:

$$
\begin{aligned}
\mathit{known} &= \{\text{John, Mike, Susan}\} \\
\mathit{birthday} &= \{\text{John} \mapsto \text{25–Mar}, \\
&\qquad \text{Mike} \mapsto \text{20–Dec}, \\
&\qquad \text{Susan} \mapsto \text{20–Dec}\}
\end{aligned}
$$

Example: Birthday Book

- BirthdayBook schema for recording people’s birthdays
  - [NAME; DATE] are basic types of the specification
  - known: set of names
  - birthday: function from names to birthdays

BirthdayBook

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

$$birthday : NAME \rightarrow DATE$$

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

Q: What does the **invariant** say?

Example: Birthday Book

- One possible state

$$
\begin{aligned}
known &= \{\text{John, Mike, Susan}\} \\
birthday &= \{\text{John} \mapsto \text{25–Mar}, \\
&\qquad \text{Mike} \mapsto \text{20–Dec}, \\
&\qquad \text{Susan} \mapsto \text{20–Dec}\}
\end{aligned}
$$

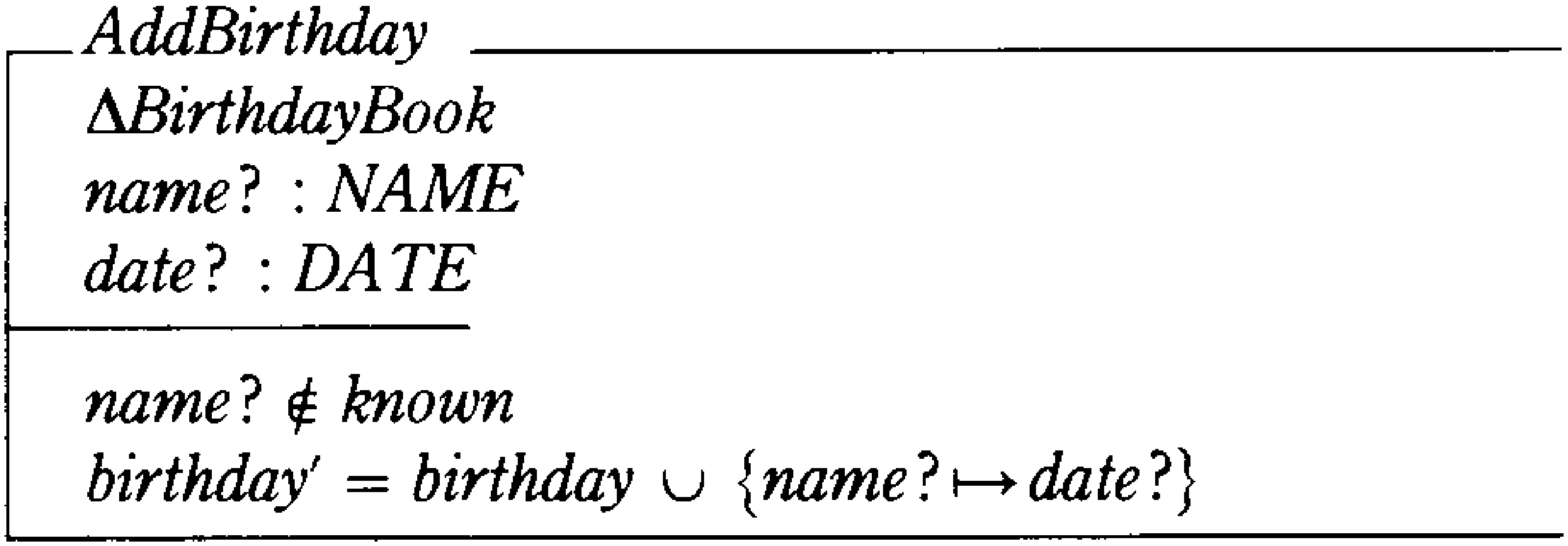
$$
\begin{array}{|l}
\hline
\textit{BirthdayBook} \\
known : \mathbb{P}\ NAME \\
birthday : NAME \rightarrow DATE \\
\hline
known = \mathrm{dom}\ birthday \\
\hline
\end{array}
$$

- Stated properties
  - No limit on the number of birthdays recorded
  - No premature decision about the format of names and dates
  - Q: How many birthday can a person have?
  - Q: Does everyone have a birthday?
  - Q: Can two persons share the same birthday?

How to Specify Dynamic Aspects?

- describe schema of *operation*
- *relationship* between **inputs (?)** and **outputs (!)** of the operation
- *changes* of the state that happen.

- Example: AddBirthday operation
  - ***name?*** and ***date?*** are inputs
  - **birthday'** – is a next state of the book



AddBirthday

$\Delta BirthdayBook$

$name? : NAME$

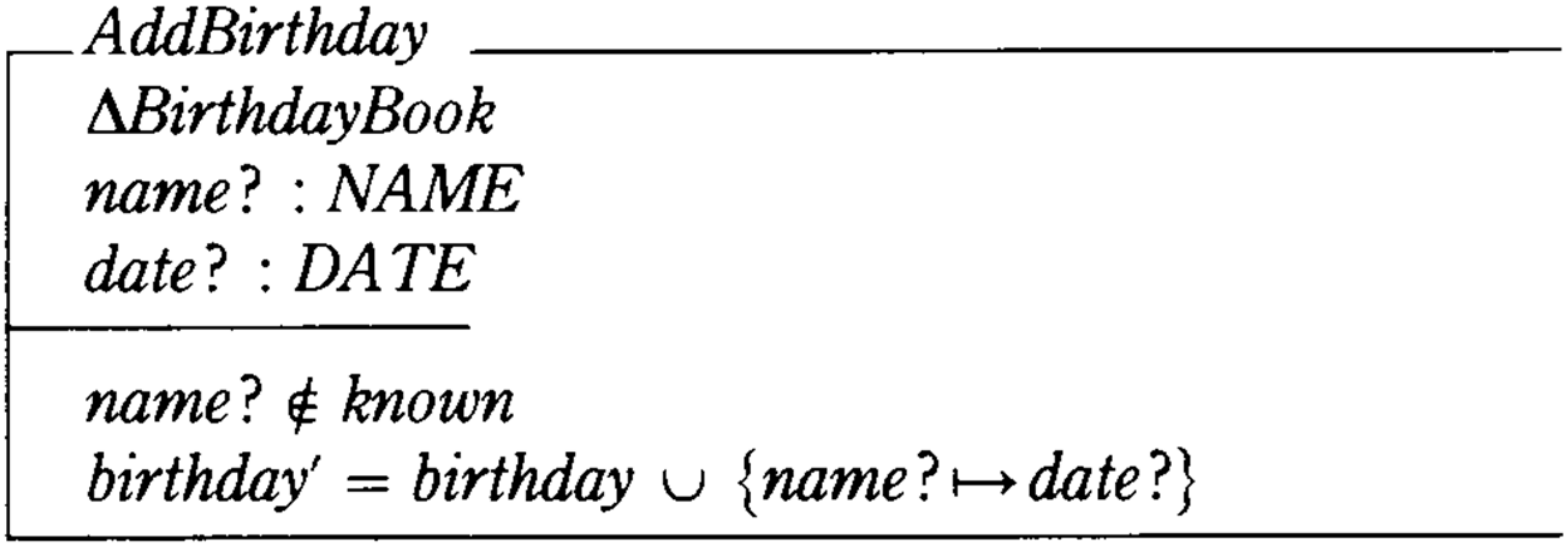
$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

Δ (Delta) And Ξ (Xi) Notations

- Δ *BirthdayBook* state change of *BirthdayBook*
- Ξ *BirthdayBook* no state change of *BirthdayBook*



AddBirthday

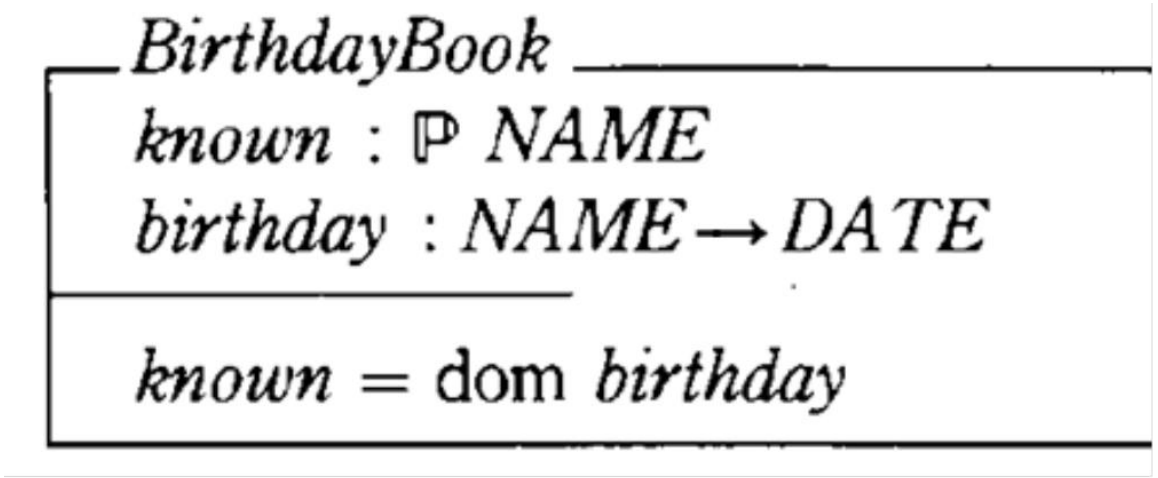
$\Delta BirthdayBook$

$name? : NAME$

$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$



BirthdayBook

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

$birthday : NAME \rightarrow DATE$

$known = \mathrm{dom}\ birthday$

More operations: *FindBirthday*

- Use Ξ notation
- No state change – we not use prime ‘ symbol

```
┌─ FindBirthday ──────────────────────────────
│ ΞBirthdayBook
│ name? : NAME
│ date! : DATE
├──────────────────
│ name? ∈ known
│ date! = birthday(name?)
└─────────────────────────────────────────────
```

More operations: *Remind*

- Use of set comprehension notation
  - Selection (|) operation – “Such that”
- Q: What is output of the *Remind* operation?

Remind

$\Xi BirthdayBook$

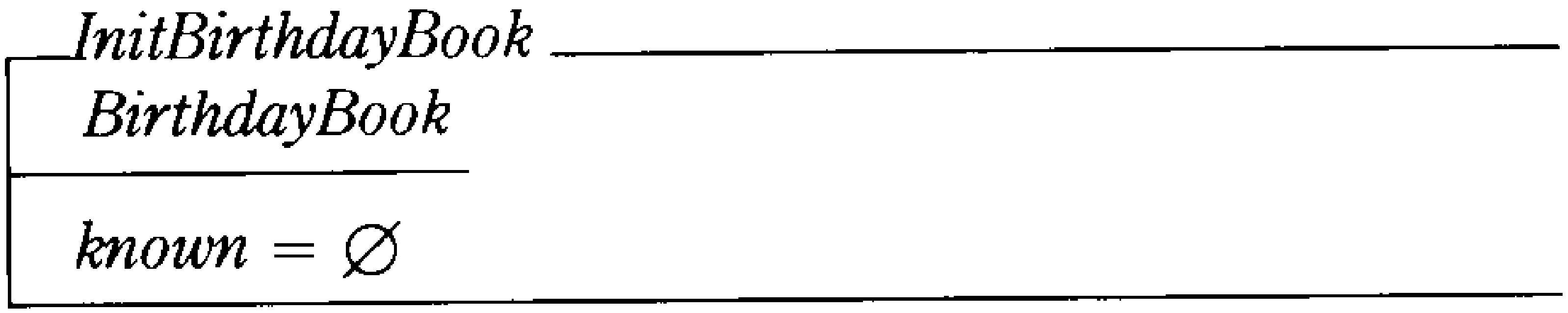
$today? : DATE$

$cards! : \mathbb{P}\ NAME$

$$cards! = \{\, n : known \mid birthday(n) = today? \,\}$$

More Examples: *InitBirthdayBook*

- Describes the *initial state* of the system
- By convention, we use Init as a prefix



InitBirthdayBook

BirthdayBook

$known = \varnothing$

Proving Properties

- E.g., known' = known ∪ {name?}

$$
\begin{aligned}
&\mathit{known}' \\
&\quad = \operatorname{dom} \mathit{birthday}' && \text{(invariant after)} \\
&\quad = \operatorname{dom}(\mathit{birthday} \cup \{\mathit{name?} \mapsto \mathit{date?}\}) \\
& && \text{(specification of } \mathit{AddBirthday}\text{)} \\
&\quad = \operatorname{dom} \mathit{birthday} \cup \operatorname{dom} \{\mathit{name?} \mapsto \mathit{date?}\} \\
& && \text{(fact about `dom')} \\
&\quad = \operatorname{dom} \mathit{birthday} \cup \{\mathit{name?}\} && \text{(fact about `dom')} \\
&\quad = \mathit{known} \cup \{\mathit{name?}\} && \text{(invariant before)}
\end{aligned}
$$

BirthdayBook

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

$\mathit{birthday} : \mathit{NAME} \rightarrow \mathit{DATE}$

$\mathit{known} = \operatorname{dom} \mathit{birthday}$

AddBirthday

$\Delta \mathit{BirthdayBook}$

$\mathit{name?} : \mathit{NAME}$

$\mathit{date?} : \mathit{DATE}$

$\mathit{name?} \notin \mathit{known}$

$\mathit{birthday}' = \mathit{birthday} \cup \{\mathit{name?} \mapsto \mathit{date?}\}$

Please ask your questions!