U O W

# Software Requirements, Specifications and Formal Methods

A/Prof. Lei Niu

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

# Fundamentals of Z notation

# Objectives

- To introduce the three basic constructs in Z notation:
  - Declarations
  - Expressions
  - Predicates
- To introduce the basic data structures in Z notation:
  - Tuples
  - Relations
  - Functions

# Declarations: sets, types and variables

- Z uses sets to represent everything, sets are central in Z
- Sets can be represented by using
  - A list to enumerate all its members, or
  - A set name
- A traffic light: {red, yellow, green}
  - No order
  - No duplicate elements
  - Set without element is the empty set $\phi$

# Declarations: sets, types and variables

$\emptyset$ The *empty set* – also denoted as $\{\}$

$\{Jane, Alice, Emma\}$ Three (type-compatible!) elements

$\{0,1,2\} = \{0,0,1,0,2,1\} = \{1,2,0\}$ Same three element set of numbers (no ordering)

$\{0,1,2,3,\ldots\}$ Set of *natural numbers* – infinite number of elements, denoted $\mathbb{N}$ for brevity

$\{\emptyset\} \neq \emptyset$ Set containing the empty set is *not* empty

Set elements drawn from some *basic type* (or *given set*) in Z (even the empty set).

$\emptyset[T]$ indicates empty set drawn from type $T$.

# Declarations: sets, types and variables

## Set membership

$x$ is a member of $S$: $x \in S$

Examples:

$0 \in \{0, 1, 2\}$

$\emptyset \in \{\emptyset\}$

$0 \notin \emptyset$ Alternative to '$\neg\ (0 \in \emptyset)$'

In fact for any $x$: $x \notin \emptyset$

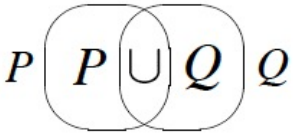
$William \notin \{Jonathan, Jane, Alice, Emma\}$

# Declarations: sets, types and variables

**Operations on sets**

$P \cup Q$ '*P union Q*' – union.

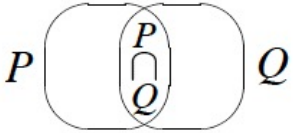
All elements in either $P$ or $Q$ (or both).




$\{a, b, c, \ldots\}$ is shorthand for $\{a\} \cup \{b\} \cup \{c\} \cup \ldots$

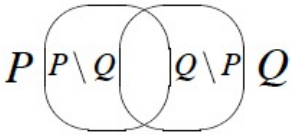
$P \cap Q$ '*P intersect Q*' – intersection.

Elements in both $P$ and $Q$.




$P \setminus Q$ '*P minus Q*' – difference.

Elements in $P$, but NOT in Q



# Declarations: sets, types and variables

$P = Q$ '$P$ *equals* $Q$' – identity of sets.

$P$ and $Q$ have the same elements.

≀≀≀ No repeated elements and no order:

$$\{Alice\} = \{Alice, Alice, Alice\}$$
$$\{Alice, Emma\} = \{Emma, Alice\}$$

$$(P \setminus Q) \cup (P \cap Q) \cup (Q \setminus P) = P \cup Q$$

Negation of $P = Q$

$P$ and $Q$ are *not* the same.

'$P \neq Q$' (equivalent to '$\neg\ (P = Q)$').

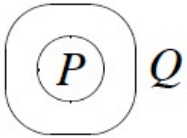
# Declarations: sets, types and variables

$P \subseteq Q$ '*P contained in Q*' – subset.

Elements of $P$ are in $Q$. (Perhaps $P = Q$.)

*Strict subset*: '$P \subset Q$'

Shorthand for '$P \subseteq Q \land P \neq Q$'.

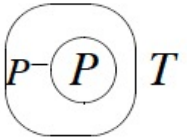



$P^-$ '*complement P*' – complementation.

All the elements *not* in a set.

For a type $T$ where $P \subseteq T$:

$$P^- = T \setminus P$$



# Declarations: sets, types and variables

- Sets names are used when we can't list all members

- Z has some predefined standard sets names
  - Z: set of integers (including negative numbers)
  - N: set of natural numbers (beginning with zero)
  - N1: set of natural numbers (beginning with 1)

- We can also make our own sets names
  - DICE: {1,2,3,4,5,6} or {1..6} for short
  - LAMP: {red, yellow, green}

# Declarations: sets, types and variables

- In Z, we can only create sets from objects that are similar in some way, i.e., *elements in the same set must have the same type*

- Sets in Z are typed
  - Composed of colours, or numbers, but not both
  - This is NOT Z: {2, 4, red, yellow, 6}

- In Z, every type has a carrier set that contains all of the objects of that type
  - {…,-2,-1,0,1,2,…} is the carrier set for integer type Z

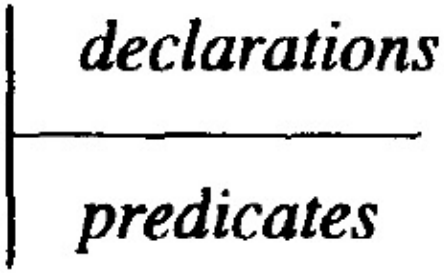
# Declarations: sets, types and variables

- If we want to represent a particular object in Z, we have to *declare it as a variable*
- Declarations introduce variables and tell to which set each variable belongs.

| | |
|---|---|
| – i: Z | [i is an integer] |
| – d1, d2: DICE | [d1 and d2 are two numbers in the range 1..6] |
| – signal: LAMP | [signal is one of the colors red, yellow, green] |
| – e: EVEN | [e is an even number] |
| – p: PRIME | [p is a prime number] |

# Constraining variables

- Global variables can be used anywhere in Z. It is called declaration before use.




- Predicates are optional, they are formulas that constrains the variables

- d1, d2: 1..6

$$d_1, d_2 : DICE$$

# Constraining variables

- Two numbers in DICE set, and add up to 7

$$d_1, d_2 : DICE$$
$$\hline$$
$$d_1 + d_2 = 7$$

d1=1, d2=6, or d1=2, d2=5, or d1=3, d2=4, or d1=4, d2=3, or d1=5, d2=2, or d1=6, d2=1

- Two numbers in DICE, d1<d2 and add up to 7

$$d_1, d_2 : DICE$$
$$\hline$$
$$d_1 + d_2 = 7$$
$$d_1 < d_2$$

d1=1, d2=6, or d1=2, d2=5, or d1=3, d2=4

# Constants, abbreviation definitions

- A variable can only take on one value is called constant

$$size : \mathbb{N}$$
$$size = 2048$$

- An abbreviation definition in Z is “size == 2048”.

$$DICE == 1 \,..\, 6$$
$$LAMP == \{red, yellow, green\}$$

# Constants, abbreviation definitions

- A definition where the types are explicitly spelled out in this way is said to be normalized

$$e, o, p : \mathbb{Z}$$

---

$$e \in EVEN$$
$$o \in ODD$$
$$p \in PRIME$$

- EVEN, ODD, PRIME are declarations
- Z is a signature

# Set variables

- Z uses the symbol P (power set) to indicate it

Example: S={x,y,z}, then PS={ϕ,{x},{y},{z},{x,y},{y,z},{x,z},{x,y,z}}

The following definition declares a set of natural numbers named PRIME and a set of numbers in the range 1..6 named toss:

$$PRIME : \mathbb{P}\,\mathbb{N}$$
$$toss : \mathbb{P}\,DICE$$

- In Z notation, every set has a type. The type of a set whose elements have type T is denoted as PT. So the type of PRIME and toss are both PZ
- PZ means “the set of all subsets of the integers”.

# Set variables

**Power set**

For a set $S$, $\mathbb{P}S$ denotes the set of all subsets (*power set*) of $S$.

$$X \in \mathbb{P}S \Leftrightarrow X \subseteq S$$

$\emptyset \in \mathbb{P}S$, so $\mathbb{P}S \neq \emptyset$.

Examples:

$\mathbb{P}\emptyset = \{\emptyset\}$

$\mathbb{P}\{a\} = \{\emptyset, \{a\}\}$

$\mathbb{P}\mathbb{N}$ Sets of natural numbers

$\mathbb{P}(\mathbb{N} \times \mathbb{N})$ Sets of pairs of numbers

# Defining new types

- In Z, we can define new types via
  - the **free type definition**
  - the **basic type definition**

- Free types are similar to the enumerated types.

For example, the declaration for the free type COLOR is:

$$COLOR ::= red \mid green \mid blue \mid yellow \mid cyan \mid magenta \mid white \mid black$$

# Defining new types

- We use a basic type when we do not want to list all elements in advance.
- This is very common when sets are large
- *To define a basic type, simply mention its name, enclosed in brackets*

For example, the declaration for the type whose elements are all the names that might appear in a telephone directory is:

$$[NAME]$$

- The free type and basic type define new types, but not just sets
- The basic type can be used *whenever the internal elements can be ignored* in advance

# Expressions and operators

- In Z, expressions describe the values that variables might have
- *Expressions are formulas where variables and literal values appear together with operators*
- Expressions are also called terms
- We already introduced the definition of variables
- Operators shall also be defined and they are type-related

# Expressions and operators

- Arithmetic expressions
  - +, -, *, div and mod
  - (1+2)*3 div (5-6)

- Set expressions
  - The union operator (U)

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

  - The difference operator (\)

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

  - The intersection operator ($\cap$)

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

# Expressions and types

In Z, every expression has a type.

- Every arithmetic expression has type $\mathbb{Z}$ (integers)
- Every set expression has type PT, where T is the type of the set elements

In Z, we use the size (or cardinality) operator # to counts the elements of a set.

$$\#DICE = 6$$

$$\#\{red, yellow, green\} = 3$$

# Predicates

- In Z, predicates express constraints that determine which values the variables actually do have
  - Equations predicates (size = 2048)
  - Inequalities predicates (size > 640)
  - Membership predicates (e ∈ EVEN)

- Predicates are NOT expressions because they do not denote values.
- Predicates can be freely to put wherever they will help the reader.

# Equations

- Equation

In Z, an equation is a predicate where two expression are joined by an equal (=) sign, such as e1=e2, size = 2048, {a}U{b}={a,b}

- An equation in NOT an expression, because it does not denote a value

- In Z, equations are also not assignment statements

- Equality is symmetric, and we can reverse the order of the expressions, such as 2048 = size (which is wrong in a programming language)

# Laws

- Predicates are not just for constraining variables, they are also used to describe the operators.
- Predicates used in this way are called laws.

For example, the following law defines we will get the same result when we divide after factoring out common factors from the numerator and divisor:

$$(c * n) \text{ div } (c * d) = n \text{ div } d$$

- A law must be held no matter what values are assigned to its variables

# Tuples and records

- Sets collect whole groups, but we often need particular individuals
- *All elements in a set must have the same type*, but we want to create *structures can have different things*
- Elements in a set are not ordered, but sometimes order is also important
- In Z, we can use tuples, which associate with particular elements of any type, in a fixed order
  - (May, 1, 2020)

# Tuples and records

- First, we define the sets for DAY, MONTH, and YEAR

$$DAY == 1 \,..\, 31;\ MONTH == 1 \,..\, 12;\ YEAR == \mathbb{Z}$$

- Then, we decide an order for the three components
  - (day, month, year)

- Third, we define a tuple, which is an instances of Cartesian product types (also called cross product types)

$$DATE == DAY \times MONTH \times YEAR$$

  - Because DAY, MONTH, and YEAR all have type Z, so the set DATE has the product type P(Z*Z*Z)

# Tuples and records

**Cartesian product**

$$T \times U$$

denotes the type of ordered pairs $(t, u)$ with $t : T$ and $u : U$.
If $P \subseteq T$ and $Q \subseteq U$ then

$$P \times Q = \{p : T;\ q : U \mid p \in P \wedge q \in Q\}$$

Generalizable to an ordered *n-tuple*:

$$E_1 \times E_2 \times \ldots \times E_n$$

Examples:

$$(Z, Jonathan) \in Methods \times People$$

$$(Oxford, Cambridge) \in Places \times Places$$

$$(Jonathan, Oxford, 1956) \in People \times Places \times \mathbb{N}$$

# Tuples and records

- After we define the tuple, we can declare variables and assign the values

$$
\begin{array}{|l}
landing, opening : DATE \\
\hline
landing = (20, 7, 1969) \\
opening = (9, 11, 1989)
\end{array}
$$

- We can also define a tuple with different types

$$[NAME]$$

$$ID == \mathbb{N}$$

$$DEPARTMENT ::= administration \mid manufacturing \mid research$$

# Tuples and records

- Then we can define the tuple

$$EMPLOYEE == ID \times NAME \times DEPARTMENT$$

- And also define variables of the tuple

$$\begin{array}{|l} Frank, Aki : EMPLOYEE \\ \hline Frank = (0019, frank, administration) \\ Aki = (7408, aki, research) \end{array}$$

# Relations, tables and database

- In Z, a set of tuples is called a relation.
- Relation can help to model tables and databases.

| ID | Name | Department |
|---|---|---|
| 0019 | Frank | Administration |
| 0308 | Philip | Research |
| 6302 | Frank | Manufacturing |
| 7408 | Aki | Research |
| 0517 | Doug | Research |
| 0038 | Philip | Administration |
| ⋮ | ⋮ | ⋮ |

# Relations, tables and database

- This is how we notate it in Z

$$
\begin{array}{|l}
Employee : \mathbb{P}\, EMPLOYEE \\
\hline
Employee = \{ \\
\quad \vdots \\
\quad (0019, frank, administration), \\
\quad (0308, philip, research), \\
\quad (6302, frank, manufacturing), \\
\quad (7408, aki, research), \\
\quad (0517, doug, research), \\
\quad (0038, philip, administration), \\
\quad \vdots \\
\quad \} \\
\end{array}
$$

- A relation is a set of tuples, so it is unordered

# Pairs and binary relations

- In Z, we call a tuple just with two components as the pair.
- For short, in Z, we use the maplet arrow ↦ to emphasize the asymmetry between the two components.
  - The pair (aki, 4117) can also be written aki ↦ 4117
- Z also provides the first() and second() operators for extracting each component from a pair
  - first(aki, 4117) = aki
  - second(aki, 4117) = 4117

# Pairs and binary relations

- In general, Z uses P (T*T) to declare a binary relations, or T $\leftrightarrow$ T
- So, binary relations can be many-to-many relations
- For example, we have a phone directory

| Name | Phone |
|---|---|
| Aki | 4117 |
| Philip | 4107 |
| Doug | 4107 |
| Doug | 4136 |
| Philip | 0113 |
| Frank | 0110 |
| Frank | 6190 |
| ⋮ | ⋮ |

# Pairs and binary relations

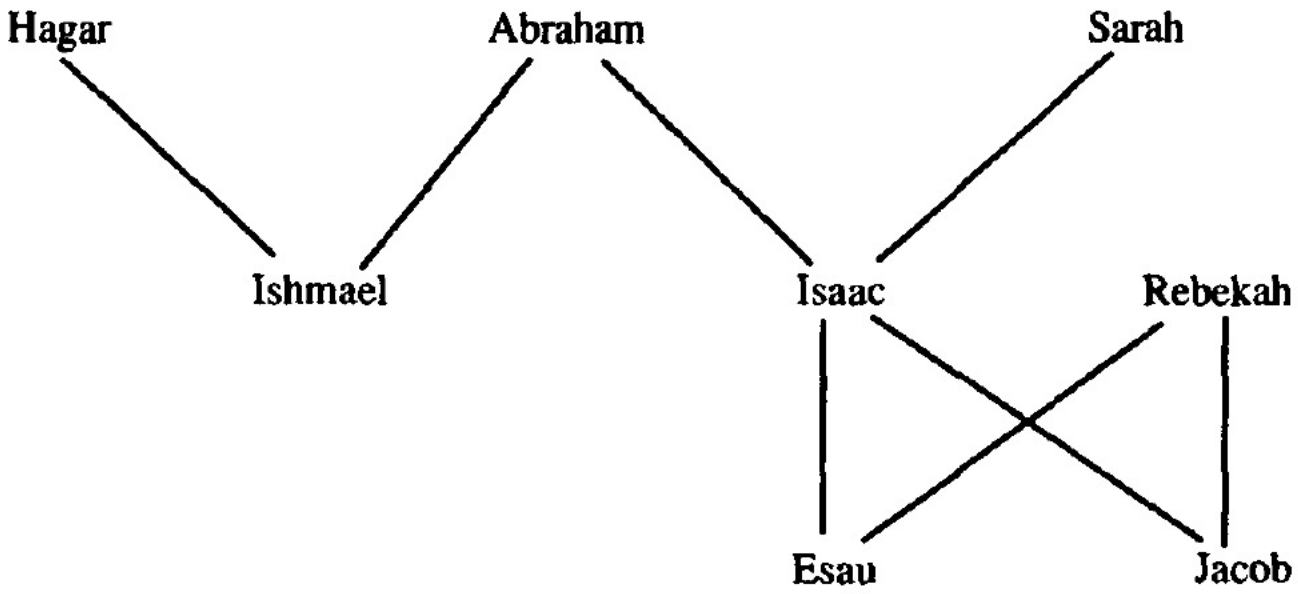
- This is how we notate it in Z

[Name]

PHONE == 0..9999

$$
\begin{array}{|l}
phone : NAME \leftrightarrow PHONE \\
\hline
phone = \{ \\
\quad \vdots \\
\quad aki \mapsto 4117, \\
\quad philip \mapsto 4107, \\
\quad doug \mapsto 4107, \\
\quad doug \mapsto 4136, \\
\quad philip \mapsto 0113, \\
\quad frank \mapsto 0110, \\
\quad frank \mapsto 6190, \\
\quad \vdots \\
\quad \}
\end{array}
$$

# Pairs and binary relations




$$PERSON ::= hagar \mid abraham \mid sarah \mid ishmael \mid isaac \mid rebekah \mid esau \mid jacob$$

$$child == \{hagar \mapsto ishmael, abraham \mapsto ishmael, abraham \mapsto isaac, sarah \mapsto isaac, isaac \mapsto esau, isaac \mapsto jacob, rebekah \mapsto esau, rebekah \mapsto jacob\}$$

# Domain and range

- Domain (abbreviation dom): the set formed by *all the first components of all the pairs in a binary relation*
- Range (abbreviation ran): the set formed by *all the second components of all the pairs in a binary relation*

$$\mathrm{dom}\, phone = \{\ldots, aki, philip, doug, frank, \ldots\}$$

$$\mathrm{ran}\, phone = \{\ldots, 4117, 4107, 4136, 0113, 0110, 6190, \ldots\}$$

- The sets appeared in a relation are called the *source set* and the *target set*, and the domain and range are just subsets of the source and target set
  - We declared $phone : NAME \leftrightarrow PHONE$
  - Not all names in a company appear in dom phone because some people may not have phones

# Operators for relations

- Relations are very important in Z
- Z provides *a rich collection of operators* for binary relations
- Many Z relational operators behave like typical database operations
- Operators:
  - lookup,
  - query,
  - update and
  - inverse

# Operators for relations

Lookup operation

- Z uses the relational image operator ($(\!|\ldots|\!)$) to perform *lookup*.
- The argument is a set of the first elements (domain) in a relation
- The operator returns a set of its second elements (range) in the relation

$$phone(\!|\{doug, philip\}|\!) = \{4107, 4136, 0113\}$$

# Operators for relations

## Domain restriction operation

- In Z, it also defines another two operators to select tuples based on domain and range elements

- Domain restriction operator ( $\lhd$ ) returns the tuple set based on the given element set *from the domain set of a relation*

$$\{doug, philip\} \lhd phone =$$

$$\{philip \mapsto 4107,$$
$$doug \mapsto 4107,$$
$$doug \mapsto 4136,$$
$$philip \mapsto 0113\}$$

# Operators for relations

## Range restriction operation

- Range restriction operator ( ▷ ) returns the tuple set based on the given element set *from the range set of a relation*

$$
\begin{aligned}
&\mathit{phone} \rhd (4000 \mathinner{..} 4999) = \{ \\
&\quad \vdots \\
&\quad \mathit{aki} \mapsto 4117, \\
&\quad \mathit{philip} \mapsto 4107, \\
&\quad \mathit{doug} \mapsto 4107, \\
&\quad \mathit{doug} \mapsto 4136, \\
&\quad \vdots \\
&\quad \}
\end{aligned}
$$

# Operators for relations

- We can combine domain and range restriction.

$$\{doug, philip\} \lhd phone \rhd (4000 \,..\, 4999) =$$

$$\{philip \mapsto 4107,$$
$$doug \mapsto 4107,$$
$$doug \mapsto 4136\}$$

# Operators for relations

## Domain and Range anti-restriction operations

- There are also domain and range ( $\lhd$ and $\rhd$ ) anti-restriction operators
- $S \lhd R$ selects the tuple set from relation R, except *without the pairs whose first elements is in S*
- $R \rhd T$ selects the tuple set from the relation R, except *without the pairs whose second elements is in T*
- Anti-restriction operators can also be combined to use

# Operators and relations

- What is the result of {doug} ⩤ phone ?

$$aki \mapsto 4117,\ philip \mapsto 4107,\ \cancel{doug \mapsto 4107,}\ \cancel{doug \mapsto 4136,}\ philip \mapsto 0113,\ frank \mapsto 0110,\ frank \mapsto 6190,$$

- What is the result of phone ⩥ {4107} ?

$$aki \mapsto 4117,\ \cancel{philip \mapsto 4107,}\ \cancel{doug \mapsto 4107,}\ doug \mapsto 4136,\ philip \mapsto 0113,\ frank \mapsto 0110,\ frank \mapsto 6190,$$

- What is the result of {doug} ⩤ phone ⩥ {4107}

$$aki \mapsto 4117,\ \cancel{philip \mapsto 4107,}\ \cancel{doug \mapsto 4107,}\ \cancel{doug \mapsto 4136,}\ philip \mapsto 0113,\ frank \mapsto 0110,\ frank \mapsto 6190,$$

# Operators for relations

Override operation

- In Z, it uses the override operator ( $\oplus$ ) to update a relation.

- Override operator return a new relation that contains the tuples from both argument sets, except the tuples in the second argument replace any tuples from the first argument that have the same first component

$$
\begin{aligned}
&phone \oplus \{heather \mapsto 4026, aki \mapsto 4026\} = \{ \\
&\quad \vdots \\
&\quad aki \mapsto 4026, \\
&\quad philip \mapsto 4107, \\
&\quad doug \mapsto 4107, \\
&\quad doug \mapsto 4136, \\
&\quad philip \mapsto 0113, \\
&\quad frank \mapsto 0110, \\
&\quad frank \mapsto 6190, \\
&\quad heather \mapsto 4026, \\
&\quad \vdots \\
&\quad \}
\end{aligned}
$$

# Operators for relations

## Inverse operation

- The inverse operator (~) *reverses the direction of a binary relation by exchanging the first and second components of each pair*.
- Inverse operator is a postfix unary operator

$$
\begin{aligned}
&\mathit{phone}^{\sim} = \{\\
&\quad \vdots\\
&\quad 4117 \mapsto \mathit{aki},\\
&\quad 4107 \mapsto \mathit{philip},\\
&\quad 4107 \mapsto \mathit{doug},\\
&\quad 4136 \mapsto \mathit{doug},\\
&\quad 0013 \mapsto \mathit{philip},\\
&\quad 0110 \mapsto \mathit{frank},\\
&\quad 6190 \mapsto \mathit{frank},\\
&\quad \vdots\\
&\quad \}
\end{aligned}
$$

# Composing relations

- If several relations describes the same collection of objects, we can make inferences by *forming chains of associations from different relations*

- Relational composition ($\fatsemi$) *merges two relations into one by combining pairs* that share a matching component

# Composing relations

$$phone : NAME \leftrightarrow PHONE$$

$$\begin{aligned} phone = \{ & \\ & \vdots \\ & aki \mapsto 4117, \\ & philip \mapsto 4107, \\ & doug \mapsto 4107, \\ & doug \mapsto 4136, \\ & philip \mapsto 0113, \\ & frank \mapsto 0110, \\ & frank \mapsto 6190, \\ & \vdots \\ & \} \end{aligned}$$

$$dept : PHONE \leftrightarrow DEPARTMENT$$

$$\begin{aligned} dept = \{ & \\ & 0000 \mapsto administration, \\ & \vdots \\ & 0999 \mapsto administration, \\ & 4000 \mapsto research, \\ & \vdots \\ & 4999 \mapsto research, \\ & 6000 \mapsto manufacturing, \\ & \vdots \\ & 6999 \mapsto manufacturing\} \end{aligned}$$

# Composing relations

- Then the relational composition notates this operation

$$
\begin{aligned}
&phone \fatsemi dept = \{ \\
&\quad \vdots \\
&\quad aki \mapsto research, \\
&\quad philip \mapsto research, \\
&\quad doug \mapsto research, \\
&\quad philip \mapsto administration, \\
&\quad frank \mapsto administration, \\
&\quad frank \mapsto manufacturing, \\
&\quad \vdots \\
&\quad \}
\end{aligned}
$$

- The relational composition operator can be used many times

# Functions

- A function is *a special kind of relation* which associates a single item with each element in a set
- A function is *a mapping which can only be many-to-one or one-to-one relations*
- Suppose we try to update the phone relation, i.e., each employee is allowed only one telephone number
- In Z, the function arrow ($\pfun$) defines a asymmetry of the functional relation

# Functions

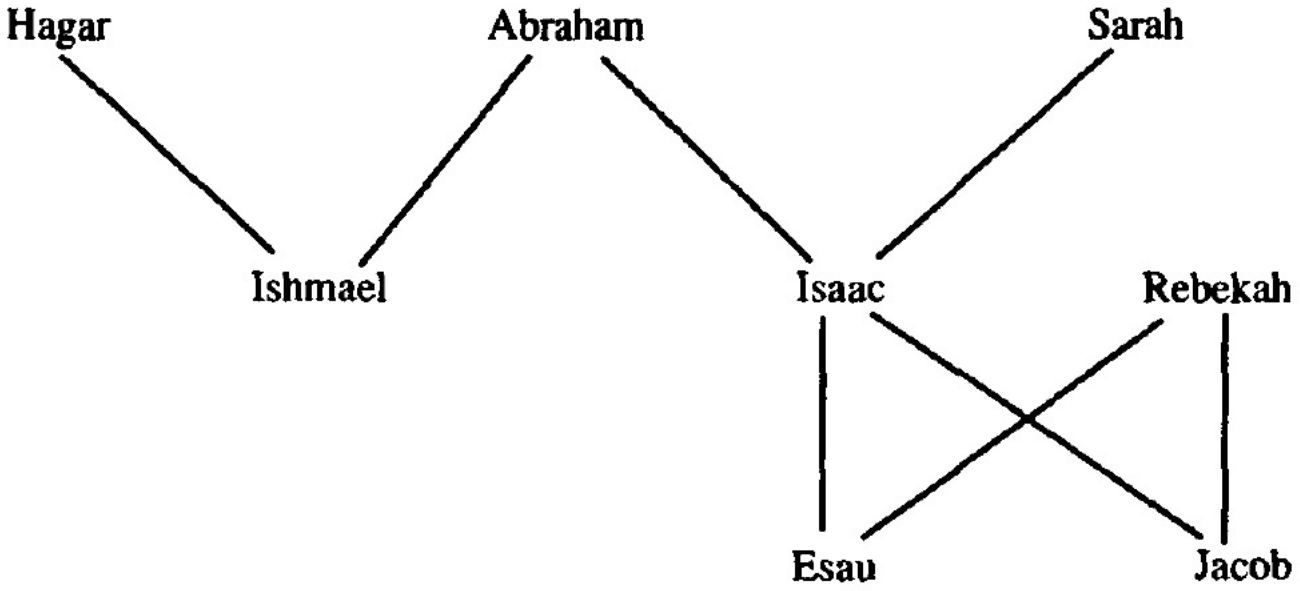
- Each employee can only have one phone number, but several employees might have the same number

$$phone_F : NAME \twoheadrightarrow PHONE$$

$$phone_F = \{$$
$$\vdots$$
$$aki \mapsto 4117,$$
$$philip \mapsto 4107,$$
$$doug \mapsto 4107,$$
$$frank \mapsto 6190,$$
$$\vdots$$
$$\}$$

# Functions

- Define a mother function based on the following relation
- Hint: mother function shall indicate each person only can have one mother, and several people might have the same mother.




$$mother : PERSON \pfun PERSON$$

$$mother = \{ishmael \mapsto hagar, isaac \mapsto sarah, esau \mapsto rebekah, jacob \mapsto rebekah\}$$

# Function application

- Finding the single item associated with an element by a function is called function application

For example, to find Jacob's mother, we can use the *relational image operator* as follows:

$$mother(\!|\{jacob\}|\!) = \{rebekah\}$$

But we know one person can only have one mother, so we don't need to use sets, but can use function application as

$$mother(jacob) = rebekah \quad \text{or}$$

$$mother\ jacob = rebekah$$

# Partial functions and total functions

- In the above mother function, we didn't name every person's mother. Abraham, Sarah, and Rebekah do not appear in the domain of the function mother
- The expression "mother rebekah" does not denote anything, because it has no value
- So we call mother function is a *partial function*
- *The domain of a partial function might not include every element of the source set*
- *If every element of the source set can map a value in the range set, we call the function a total function*
  - For example, the integer square function is a total function because every nature number has an integer square

# Injections

- Injections are special functions that associate each element in their domain with a different element in their range
- So *injection is a one-to-one relation*
- In Z, we declare the injection with the injection arrow ( $\rightarrowtail$ )

$$
\begin{array}{|l}
phone_1 : NAME \rightarrowtail PHONE \\
\hline
phone_1 = \{ \\
\quad \vdots \\
\quad aki \mapsto 4117, \\
\quad philip \mapsto 4107, \\
\quad doug \mapsto 4200, \\
\quad frank \mapsto 6190, \\
\quad \vdots \\
\quad \}
\end{array}
$$

# Operators

- Most of operators are just functions that use infix syntax and have symbolic name, such as 2+3 = 5, which is another format of (_+_)(2, 3) = 5
- When we mention an infix function name without the arguments, we must surround its name with underscores to show where the arguments should go.
- A formal definition of “+” operator is

$$\_ + \_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$$

---

**. . . definition omitted . . .**

- So in Z, the “+” is actually the name of a set

$$(\_ + \_) = \{ \ldots, (1, 1) \mapsto 2, (1, 2) \mapsto 3, (1, 3) \mapsto 4, \ldots \}$$

# Operators

## Arithmetic

| | | |
|---|---|---|
| addition | $\_+\_$ | $2+2=4$ |
| subtraction | $\_-\_$ | $4-2=2$ |
| multiplication | $\_*\_$ | $2*2=4$ |
| division | $\_\,\mathsf{div}\,\_$ | $5\,\mathsf{div}\,2=2$ |
| modulo arithmetic | $\_\,\mathsf{mod}\,\_$ | $5\,\mathsf{mod}\,2=1$ |
| negation | $-$ | $-(-2)=2$ |

Brackets used for grouping; e.g., $2*(4+5)=18$.

# Successor function

$$succ = \{0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 3, \ldots\}$$

$$\operatorname{ran} succ = \mathbb{N}_1.$$

Inverse 'predecessor' function: $pred == succ^{\sim}$

Laws:

$$\begin{aligned}
succ &= \mathbb{N} \lhd (_ + 1) \\
&= (_ + 1) \rhd \mathbb{N}_1 \\
pred &= \mathbb{N}_1 \lhd (_ - 1) \\
&= (_ - 1) \rhd \mathbb{N} \\
succ \fatsemi succ &= \mathbb{N} \lhd (_ + 2) \\
&= (_ + 2) \rhd (\mathbb{N}_1 \setminus \{1\}) \\
succ \fatsemi pred &= \mathrm{id}\, \mathbb{N} \\
pred \fatsemi succ &= \mathrm{id}\, \mathbb{N}_1
\end{aligned}$$

# Iteration

$$R^n = R \fatsemi R \fatsemi \ldots \fatsemi R \quad k \text{ times}$$

Example, using $succ : \mathbb{N} \to \mathbb{N}$

$$succ^0 = \mathrm{id}\,\mathbb{N}$$

$$succ^1 = succ$$

$$succ^2 = succ \fatsemi succ$$

$$succ^n = \mathbb{N} \lhd (\_ + n) = (\_ + n) \rhd \{i : \mathbb{N} \mid i \geq n\}$$

for any $n : \mathbb{N}$

$$succ^{-1} = succ^{\sim}$$