CHAPTER 3
Z Structure

Chapter Outline

- Z Structure
  - Tuples
  - Relations
  - Functions
- Birthday Book Example

Z Structure

Z Structure

- Consists of:
  - Tuples (records)
  - Relations (tables, linked data structures)
  - Functions (lookup tables, trees and lists)
  - Sequences (lists, arrays)
  - Bags

Will be discussed in other Chapter

Tuples

Tuples

- Tuples are used to represent **records** such as ordered pairs, triples, quadruples etc.

- Example:
  - CountriesAndCapitals == (Malaysia, KL) - Pair
  - StudentStudyYear == (Lim, 2012, 2016) - Triple

Tuples

- To represent tuples, **Cartesian Product** will be used.
- Cartesian Product or Cross Product, or just Product are constructed from a set of all ordered pairs/triples/quadruples (tuples) etc. of those sets.
- If S and T are two sets, it is written S **x** T
- First elements are drawn from S, and second elements are drawn from T.
- Example:

  {1, 2, 3} x {4, 5} = {(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3,5)}

Tuples

- Example if we want to declare a tuple of triples data in a record
- First declare types for each component, such as:

  [NAME]

  ID == $\mathbb{N}$

  DEPT ::= admin | manufacturing | research

- Then, you can define the Cartesian product type such as EMPLOYEE

  EMPLOYEE == ID x NAME x DEPT

Tuples

staff: EMPLOYEE

→ staff == (0019, Frank, admin)

$$\begin{array}{|l}
staff : EMPLOYEE \\
\hline
staff == \{(0019, Frank, admin)\}
\end{array}$$

- But, for this declaration, we have one and only one record

Tuples

- Assume we have these data in our table (many records):

ID	NAME	DEPT
0019	Frank	admin
0305	Jane	research
0611	Mike	manufacturing
0899	Anne	admin
…	…	…

Tuples

- Before, we have the Cartesian product defines as:

EMPLOYEE == ID x NAME x DEPT

$$employee : \mathbb{P}\ EMPLOYEE$$

$$employee == \{(0019, Frank, admin), (0305, Jane, research), (0611, Mike, manufacturing), \ldots\}$$

Tuples

- Or, we can expressed:

$$employee : \mathbb{P}\,(ID \times NAME \times DEPT)$$

$$employee == \{(0019, Frank, admin), (0305, Jane, research), (0611, Mike, manufacturing), \ldots\}$$

Relation

Relation

- A relation is a set of tuples. They can resemble tables or databases. Express links between elements of objects.
- **Many-to-many relationship**
- A relation in Z is expressed as a **binary** relation. A binary relation is just a set of **pairs**.
- $\leftrightarrow$ is a *binary relation operator*. It associates from **left to right**.

$$\mathbb{P}\ (NAME \times PHONE)$$

or

$$NAME \leftrightarrow PHONE$$

Relation

- Each element in a binary relation is as pair of objects

  E.g. (Eric, Suzy)

- (Eric, Suzy) is **NOT** the same pair as (Suzy, Eric).
- The ordered pair (Eric, Suzy) may be written as

  Eric $\mapsto$ Suzy

  - $\mapsto$ this is known as ***maplet notation***
  - (Eric, Suzy) = Eric $\mapsto$ Suzy
  - The **maplet** notation provides alternate syntax without parentheses.

Relation

- Binary relations can model lookup tables

NAME	PHONE
Frank	1019
Philip	1107
Doug	2136
Anne	1107
Mike	3110
Jane	2300
Philip	2140
…	…

- In Z, we can expressed:

$$phone : NAME \leftrightarrow PHONE$$

$$phone == \{Frank \mapsto 1019, Philip \mapsto 1107, Doug \mapsto 2136, Anne \mapsto 1107, Mike \mapsto 3110, Jane \mapsto 2300, Philip \mapsto 2140, \ldots\}$$

Relation

- Assume, we have basic type:

  [DATE, PERSON]

- We can define *appointments* as a binary relation from DATE to PERSON

$$
\begin{array}{|l}
appointments : DATE \leftrightarrow PERSON \\
\hline
appointments == \{nov7 \mapsto tom, nov7 \mapsto anne, nov8 \mapsto jerry, \\
\qquad nov12 \mapsto tom, \ldots\}
\end{array}
$$

Relation

- Example:

  $lineColour: \mathbb{N} \leftrightarrow COLOUR$

  $lineColour == \{2 \mapsto red, 5 \mapsto blue, 3 \mapsto red, \ldots\}$

  $birthday: PERSON \leftrightarrow MONTH$

  $birthday == \{Dave \mapsto June, Mary \mapsto Aug, Bill \mapsto Feb, \ldots\}$

First and Second

- *Coordinate notation* is used to represent a pair by itself

  aPair = (Eric, Suzy)

- **first** and **second** split an ordered pair into its first and second coordinates

  first (Eric, Suzy) = Eric

  second(Eric, Suzy) = Suzy

- **first** and **second** are known as the *projection operations* for ordered pairs.

Domain and Range

- We have this binary notation:

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$$

- The set form by all the *first* coordinates/elements is known as **domain**.

$$\mathrm{dom}\ \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \{1, 2, 3\}$$

- The set form by all the *second* coordinates /elements is known as **range**.

$$\mathrm{ran}\ \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \{a, b, c\}$$

Domain and Range

- Example:

  $A == \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 2\}$

  dom $A = \{1, 2\}$ ran $A = \{1, 2\}$

  $B == \{2 \mapsto red, 5 \mapsto blue, 3 \mapsto red\}$

  dom $B = \{2, 3, 5\}$ ran $B = \{red, blue\}$

  $C == \{David \mapsto Jun, Mary \mapsto Aug, Bill \mapsto Feb\}$

  dom $C = \{David, Mary, Bill\}$ ran $C = \{Jun, Aug, Feb\}$

Source and Target

- We have this binary notation:

$$\{1 \mapsto 23, 2 \mapsto 29, 3 \mapsto 31, 4 \mapsto 37, 5 \mapsto 41\}$$

- Its domain is 1..5, a subset of $\mathbb{Z}$
- A domain is a subset of its **source**.

$$dom\ \{ 1 \mapsto 23, 2 \mapsto 29, 3 \mapsto 31, 4 \mapsto 37, 5 \mapsto 41 \} \subseteq \mathbb{Z}$$

source

Source and Target

- Its range is {23, 29, 31, 37, 41}, also a subset of $\mathbb{Z}$
- The range is a subset of its **target**.

$$\mathit{ran}\ \{\ 1 \mapsto 23,\ 2 \mapsto 29,\ 3 \mapsto 31,\ 4 \mapsto 37,\ 5 \mapsto 41\ \} \subseteq \underset{\uparrow}{\mathbb{Z}}$$

target

Set Operators

- We can use the usual set operations on binary pairs:

$$\{1 \mapsto a, 2 \mapsto b\} \cup \{2 \mapsto b, 3 \mapsto c\} = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$$

$$\{1 \mapsto a, 2 \mapsto b\} \cap \{2 \mapsto b, 3 \mapsto c\} = \{2 \mapsto b\}$$

$$\{1 \mapsto a, 2 \mapsto b\} \setminus \{2 \mapsto b, 3 \mapsto c\} = \{1 \mapsto a\}$$

$$\# \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = 3$$

Domain Restriction

- Works like a database query.
- If we have this binary notation:

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$$

- Its domain is {1, 2, 3}
- If we restrict the binary relation so that its domain is {2}, we get $\{2 \mapsto b\}$
- We write:

$$\{2\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \{2 \mapsto b\}$$

$\lhd$ is known as *domain restriction operator*.

Domain Restriction

- Domain restriction selects pairs based on their **first component.**
- E.g.:

$\{4\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \emptyset$

$\{1, 3\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \{1 \mapsto a, 3 \mapsto c\}$

$\{Doug, Philip\} \lhd phone = \{Philip \mapsto 1107, Doug \mapsto 2136, Philip \mapsto 2140\}$

$\{2\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 2 \mapsto d\} = \{2 \mapsto b, 2 \mapsto d\}$

$ran(\{2\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}) = \{b\}$

$ran(\{2\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 2 \mapsto d\}) = \{b, d\}$

Range Restriction

- If we have this binary notation:

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$$

- Its range is {a, b, c}
- If we restrict the binary relation so that its range is {b}, we get $\{2 \mapsto b\}$
- We write

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} \rhd \{b\} = \{2 \mapsto b\}$$

$\rhd$ is known as *range restriction operator*.

Range Restriction

- Range restriction selects according to the **second element**.
- E.g.:

$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} \rhd \{d\} = \emptyset$

$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} \rhd \{a, c\} = \{1 \mapsto a, 3 \mapsto c\}$

$phone \rhd \{1107, 3110\} = \{Philip \mapsto 1107, Anne \mapsto 1107, Mike \mapsto 3110\}$

$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 3 \mapsto b\} \rhd \{b\} = \{2 \mapsto b, 3 \mapsto b\}$

$dom\ (\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} \rhd \{b\}) = \{2\}$

$dom\ (\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 3 \mapsto b\} \rhd \{b\}) = \{2, 3\}$

Domain Anti-Restriction

- A.k.a domain subtraction
- Removes elements from domain
- If we have this binary notation:

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$$

- We write:

$$\{2\} \lhd\!\!\!- \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \{1 \mapsto a, 3 \mapsto c\}$$

$\lhd\!\!\!-$ is known as *domain anti-restriction operator*

Range Anti-Restriction

- A.k.a range subtraction
- Removes elements from range
- If we have this binary notation:

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$$

- We write

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} \rhd\!\!\!\!- \{b\} = \{1 \mapsto a, 3 \mapsto c\}$$

$\rhd\!\!\!\!-$ is known as *range anti-restriction operator*

Relational Image

- If we have this binary notation:

  $\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$

- The second coordinate of the pair whose first coordinate is 2 is b.
- We write:

  $\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} (\!|\{2\}|\!) = \{b\}$

  $(\!|\ |\!)$ is the *relational image operator*

Relational Image

- Relational image can model table lookup.
- E.g.:

$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} (\!| \{4\} |\!) = \emptyset$

$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} (\!| \{1,3\} |\!) = \{a, c\}$

$phone (\!| \{Doug, Philip\} |\!) = \{1107, 2136, 2140\}$

$ran(\{2\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}) = \{b\}$ OR

$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} (\!| \{2\} |\!) = \{b\}$

Inverse

- Inverse reverses domain and range by exchanging the components of each pair.
- Meaning, inverse swaps between domain and range (i.e. domain becomes range, and range becomes domain).

Inverse

- Assume that we have DRINK as a given set for all drinks. We define *costs* as a binary relation from drink to price.

$$costs : DRINK \leftrightarrow \mathbb{Z}$$

$$costs = \{ tea \mapsto 50, coffee \mapsto 75, hotChocolate \mapsto 75, soup \mapsto 75 \}$$

Inverse

- If we reversed the binary relation, for example from price to drink and called it *buys*, so 50 cents *buys* tea.

$$buys : \mathbb{Z} \leftrightarrow DRINK$$

$$buys = \{ 50 \mapsto tea, 75 \mapsto coffee, 75 \mapsto hotChocolate, 75 \mapsto soup \}$$

Inverse

- Then *buys* is the **inverse** of *costs*. We write

  costs~ = buys

  ~ is the *inverse operator*.

- From previous example, the inverse for phone:

  phone~ = { 1019 ↦ Frank, 1107 ↦ Philip, 2136 ↦ Doug, 1107 ↦ Anne, 3110 ↦ Mike, 2300 ↦ Jane, 2140 ↦ Philip, …}

Inverse

- E.g.:

  $\mathrm{dom}\ (\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 3 \mapsto b\} \rhd \{b\}) = \{2, 3\}$

  $\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 3 \mapsto b\}^{\sim} (\!| \{b\} |\!) = \{2, 3\}$

  $\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 3 \mapsto b\}^{\sim} = \{a \mapsto 1, b \mapsto 2, c \mapsto 3, b \mapsto 3\}$

Overriding

- Overriding can model database updates.
- $\oplus$ is the *override operator*.
- Example:

$$\text{phone} \oplus \{\text{Anne} \mapsto 1108\} =$$

$$\{ \text{Frank} \mapsto 1019, \text{Philip} \mapsto 1107, \text{Doug} \mapsto 2136, \text{Anne} \mapsto 1108, \text{Mike} \mapsto 3110, \text{Jane} \mapsto 2300, \text{Philip} \mapsto 2140, \ldots \}$$

Composition

- Composition merges two relations by combining pairs that **share a matching component**.
- Given the types for the set of all persons and rooms respectively:

  [PERSON, ROOM]

Composition

We have two binary relations, *hasPhone* and *phoneInRoom*

$$hasPhone : PERSON \leftrightarrow \mathbb{Z}$$

$$hasPhone = \{ roy \mapsto 317, tom \mapsto 208, tom \mapsto 209, jim \mapsto 326, lee \mapsto 225 \}$$

$$phoneInRoom : \mathbb{Z} \leftrightarrow ROOM$$

$$phoneInRoom = \{ 317 \mapsto A306, 208 \mapsto A39, 209 \mapsto A39, 326 \mapsto A306, 225 \mapsto A39 \}$$

Composition

- Referring to the binary relation:

  roy has phone *317* that is in room *A306.*

- Therefore,

  roy is in room *A306.*

- The composition of the two binary relations where the range of one binary is a subset of the domain of the other.

  ran hasPhone ⊆ dom phoneInRoom

Composition

- We write:

  hasPhone ⨾ phoneInRoom =

  {roy ↦A306, tom↦A39, jim↦A306, lee↦A39}

  ⨾ is the *composition operator*.

Composition

- Another example:

  We have variable *phone*, and another variable *dept*:

$$\begin{array}{|l}
phone : NAME \leftrightarrow PHONE \\
\hline
phone = \{\, Frank \mapsto 1019,\ Philip \mapsto 1107, \\
\quad Doug \mapsto 2136,\ Anne \mapsto 1107, \\
\quad Mike \mapsto 3110,\ Jane \mapsto 2300, \\
\quad Philip \mapsto 2140, \ldots \}
\end{array}$$

$$\begin{array}{|l}
dept : PHONE \leftrightarrow DEPT \\
\hline
dept = \{1000 \mapsto admin, \ldots, 1999 \mapsto admin, \\
\quad 2000 \mapsto research, \ldots, 2999 \mapsto research, \\
\quad 3000 \mapsto manufacturing, \ldots, \\
\quad 3999 \mapsto manufacturing, \ldots \}
\end{array}$$

Composition

- Composing the *phone* and *dept* relations:

  phone ⨾ dept =

  { Frank ↦ admin, Philip ↦ admin,

  Doug ↦ research, Anne ↦ admin,

  Mike ↦ manufacturing, Jane ↦ research,

  Philip ↦ research, …}

Exercise

Write the result for each of the following expressions:

a) $\{1 \mapsto mon, 2 \mapsto tue, 3 \mapsto wed, 4 \mapsto thu, 5 \mapsto fri\} (\!| \{3\} |\!)$

b) $\{1013 \mapsto PSP, 2073 \mapsto SDA, 2083 \mapsto FM, 3283 \mapsto SC\} \setminus \{3293 \mapsto SQAT\}$

c) $\{Suzy \mapsto 25, Adam \mapsto 27, Lim \mapsto 24\} \oplus \{Adam \mapsto 23, Cindy \mapsto 22\}$

d) $\{060 \mapsto Malaysia, 061 \mapsto Australia, 091 \mapsto India, 064 \mapsto Singapore\}^{\sim}$

e) $\{Jan \mapsto 31, Feb \mapsto 28, Mar \mapsto 31, Apr \mapsto 30, May \mapsto 31, Jun \mapsto 30, Jul \mapsto 31, Aug \mapsto 31, Sep \mapsto 30, Oct \mapsto 31, Nov \mapsto 30, Dec \mapsto 31\} \rhd \{30\}$

f) $\{064, 061\} \lhd \{061 \mapsto Australia, 091 \mapsto India, 064 \mapsto NewZealand, 060 \mapsto Malaysia\}$

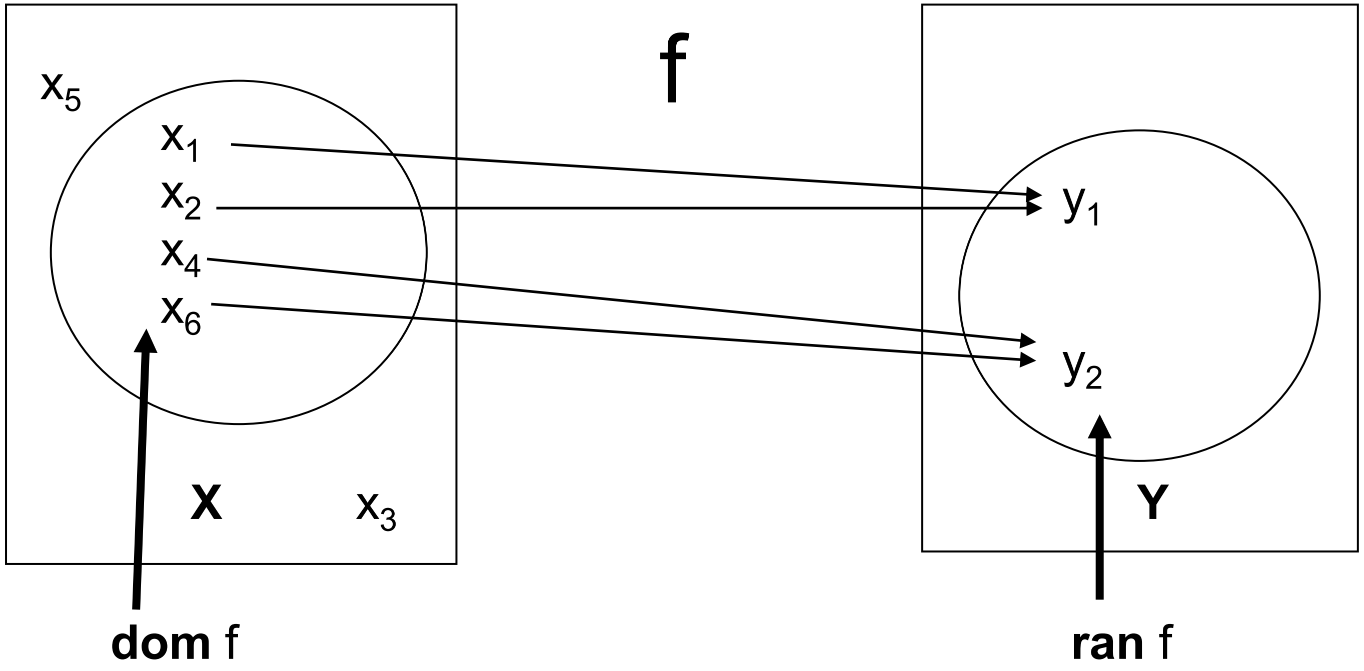
Function

Function

- A function is a special case of a relation in which there is **AT MOST ONE** value in the range for each value in the domain.
- Each element in the **domain** appears just **ONCE**.
- There is **NO** TWO distinct pairs contain the SAME **FIRST** element (All the **first** components are different)
- Each domain element is a unique key.
- Mostly will be shown as **many-to-one relationship.** Can also be **one-to-one relationship.**

Function

- Allows **at most one** element in the **domain** to point to **any element** in the **range**.



Function

- Example of a function:

  $\{1 \mapsto a, 2 \mapsto b, 3 \mapsto a\}$

- Whereas,

  $\{1 \mapsto a, 2 \mapsto b, 1 \mapsto c\}$

  is **NOT** a function because the first component, 1, appears twice.

Function

- Other example:

  lineColour == {2 ↦ red, 5 ↦ blue, 3 ↦ red}

  birthday == {David↦ June, Mary↦ Aug, Bill↦ Feb}

BUT Not

  numbers == {1↦1, 1↦2, 2↦2}

Examples of Function

registered : STUDENTID $\leftrightarrow$ STUDENT

- Because no two students have the same ID

car : REGISTRATION $\leftrightarrow$ VEHICLE

- Because no two vehicles have the same registration number

As these examples do not have duplicate domains, the symbol $\leftrightarrow$ must be changed to function symbol (e.g. $\nrightarrow$).

Function Application

- All the concepts which pertain to relations apply to functions. In addition a function may be **applied**.
- Function application associates a domain element with its unique range element.
- Since there will be **at most only one value in the range** for a given x it is possible to designate that value directly.

Function Application

- The application of the function to the value x (the argument) is written:

  fx or f(x)

- It is important to check that the value of x is in the domain of the function f before attempting to apply f to x.
  - The application is **undefined** if the value of x is NOT in the domain of f.

Function Application

- The **input** is an element from the set of all first components (domain).
- The **output** is the corresponding second component (range).
- A function always has just **ONE** output value for any input value.
- Examples of function:

  phone (Jane) = 2300

  phone Jane = 2300

Function Application

- Example: if ***day*** is a function as defined below:

$$\begin{array}{|l} day : \mathbb{N} \rightarrowtail\!\!\!\!\!\!+ \; DAY \\ \hline day == \{1 \mapsto mon, 2 \mapsto tue, 3 \mapsto wed\} \end{array}$$

- We may write:

  day (2) = tue OR day 2 = tue

  day (4) is undefined because 4 is not in the domain of the function.

Function Declaration

- Assume that we have two basic types:

  [ID] - the set of all students' ids

  [NAME] - the set of all students' names

- We intend to have:

  NO two students in a class have the same identity number

Function Declaration

- We define *class* as a function from ID to NAME:

  class : ID ⇸ NAME

  Example of a function using ⇸ (partial function)

- The name of a function is usually a singular and chosen to reflect the **range**.

Function Declaration

- Assume that we want to have maximum students in a *class* to be 25 students only. Then, we can apply the *cardinality* operator, **#**

$$
\begin{array}{|l}
\hline
\textit{Classroom} \\
\hline
class : ID \rightarrowtail\!\!\!\!\!\!+ NAME \\
\hline
\#class \geqslant 0 \\
\#class \geqslant 25 \\
\hline
\end{array}
$$

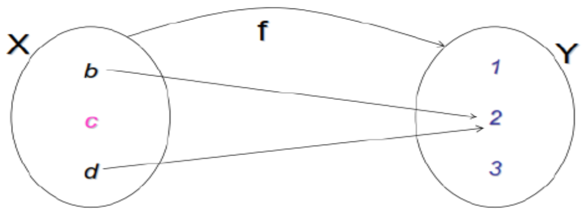
Partial & Total Functions

Partial & Total Functions

- Basically, the **domain** of a function can be represented in two different ways:
  - Partial function ($\nrightarrow$)
  - Total function ($\rightarrow$)

Partial Function (⇸)

- A special kind of relation in which each domain element has **at most one range** element associated with it.
- Suppose f: X ⟷ Y
  - f is a "partial" function defined for **some** values of X, written as X ⇸ Y
  - Some members of X are paired with a member of Y.
- A partial function is only partially works where some function applications will be undefined.



Partial Function($\nrightarrow$)

- A lot of what we call *functions* in software are actually **partial functions** because they're not defined everywhere.
- One is the application of a partial function such as the 'div' operator.
- Thus, the function:

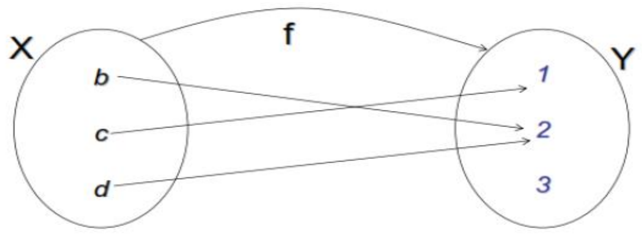
  $\{(1,3), (4,9), (8,3)\}$

over $\mathbb{N}$ x $\mathbb{N}$ is a partial function because its domain, $\{1,4,8\}$, is a proper subset of the natural numbers.

Total Function (→)

- Total function is a function whose domain is equal to the set from which the first elements of its pairs are taken.
- Suppose f: X ↔ Y
  - f is a "total" function defined for **all** values of X,

    written as X → Y
  - Every member of X is paired with a member of Y.



- A total function defined for **every element of its domain** and always return answers.
- Function application can always be used with a total function.

Function as Binary Relation

- We can use the usual binary relation operations but with care.

domain:	$\mathrm{dom}\,\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \{ 1, 2, 3 \}$
range:	$\mathrm{ran}\,\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \{ \mathrm{a}, \mathrm{b} \}$
domain restriction:	$\{ 2 \} \lhd \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \{ 2 \mapsto b \}$
range restriction:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \rhd \{ \mathrm{a} \} = \{ 1 \mapsto a, 3 \mapsto a \}$
relational image:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} (\!| \{ 2 \} |\!) = \{ b \}$
override:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \oplus \{ 3 \mapsto c, 4 \mapsto d \} = \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 4 \mapsto d \}$
composition:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \mathbin{⨾} \{ a \mapsto X, b \mapsto Y \} = \{ 1 \mapsto X, 2 \mapsto Y, 3 \mapsto X \}$
intersection:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \cap \{ 1 \mapsto a, 3 \mapsto a \} = \{ 1 \mapsto a, 3 \mapsto a \}$
difference:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \setminus \{ 1 \mapsto a, 3 \mapsto a \} = \{ 2 \mapsto b \}$

Function as Binary Relation

- The inverse of a function is NOT necessarily another function.

$$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \}^{\sim} = \{ a \mapsto 1, b \mapsto 2, a \mapsto 3 \}$$

Is NOT a function because **a** appears twice in the result domain.

$$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto c \}^{\sim} = \{ a \mapsto 1, b \mapsto 2, c \mapsto 3 \}$$

Is a function because each first component in the result domain is unique.

Function as Binary Relation

$name : ID \nrightarrow NAME$

$name == \{A001 \mapsto Lim, A002 \mapsto Siti, A003 \mapsto Lim, A004 \mapsto Kumar, \ldots\}$

$name\ A003 = \{Lim\}$

$name^{\sim} == \{Lim \mapsto A001, Siti \mapsto A002, Lim \mapsto A003, Kumar \mapsto A004, \ldots\}$

$name^{\sim} (\!| \{Lim\} |\!) = \{A001, A003\}$

Function as Binary Relation

- The unions of two functions is NOT necessarily another function.

$$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \cup \{ 2 \mapsto c \} = \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a, 2 \mapsto c \}$$

Is NOT a function because 2 appears twice in the result domain.

$$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \cup \{ 4 \mapsto c \} = \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a, 4 \mapsto c \}$$

Is a function because each first component in the result domain is unique.

Properties of Functions

- Basically, the **range** of a function can be represented in three different ways or properties:
  - Injection (one-to-one)
  - Surjection (onto)
  - Bijection

Injection (One-to-One)

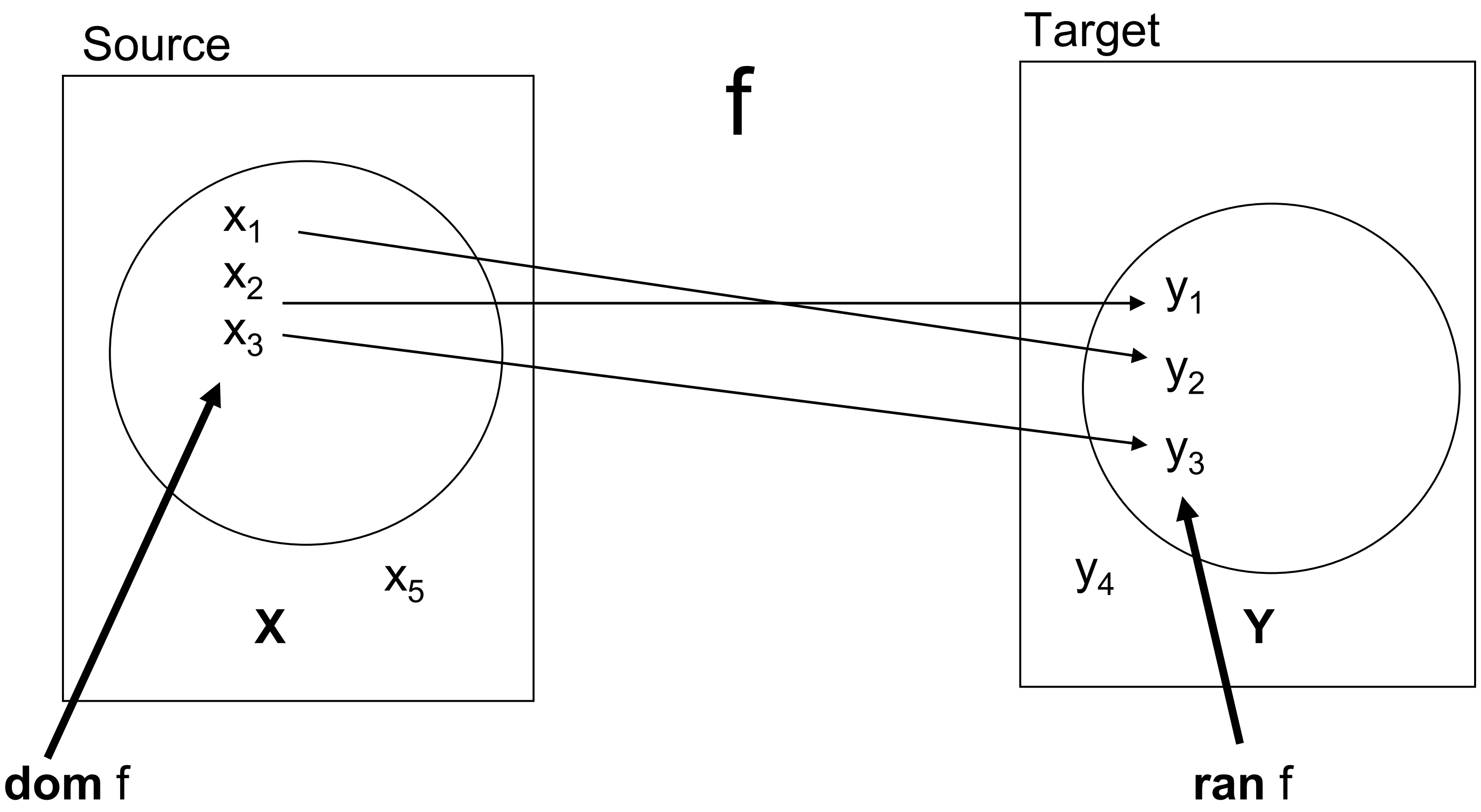
- An injection or (injective function) is a function which maps different values of the domain (source) on to different values of the range (target).
- Example:

$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$$

whereas, the following is NOT:

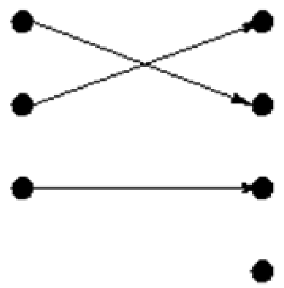
$$\{1 \mapsto a, 2 \mapsto b, 3 \mapsto a\}$$

Injection (One-to-One)



Partial (⤔) and Total (↣) Injections

- Suppose f: A ↔ B
  - f is a "one-to-one" which no element in ran(f) is associated with more that one element in dom(f), written as A⤔B or A↣B
- Partial injections (⤔):
  - Some members of A are paired with different members of B.
- Total injections (↣):
  - Every member of A is paired with a different member of B.
  - Inverse is also a function.

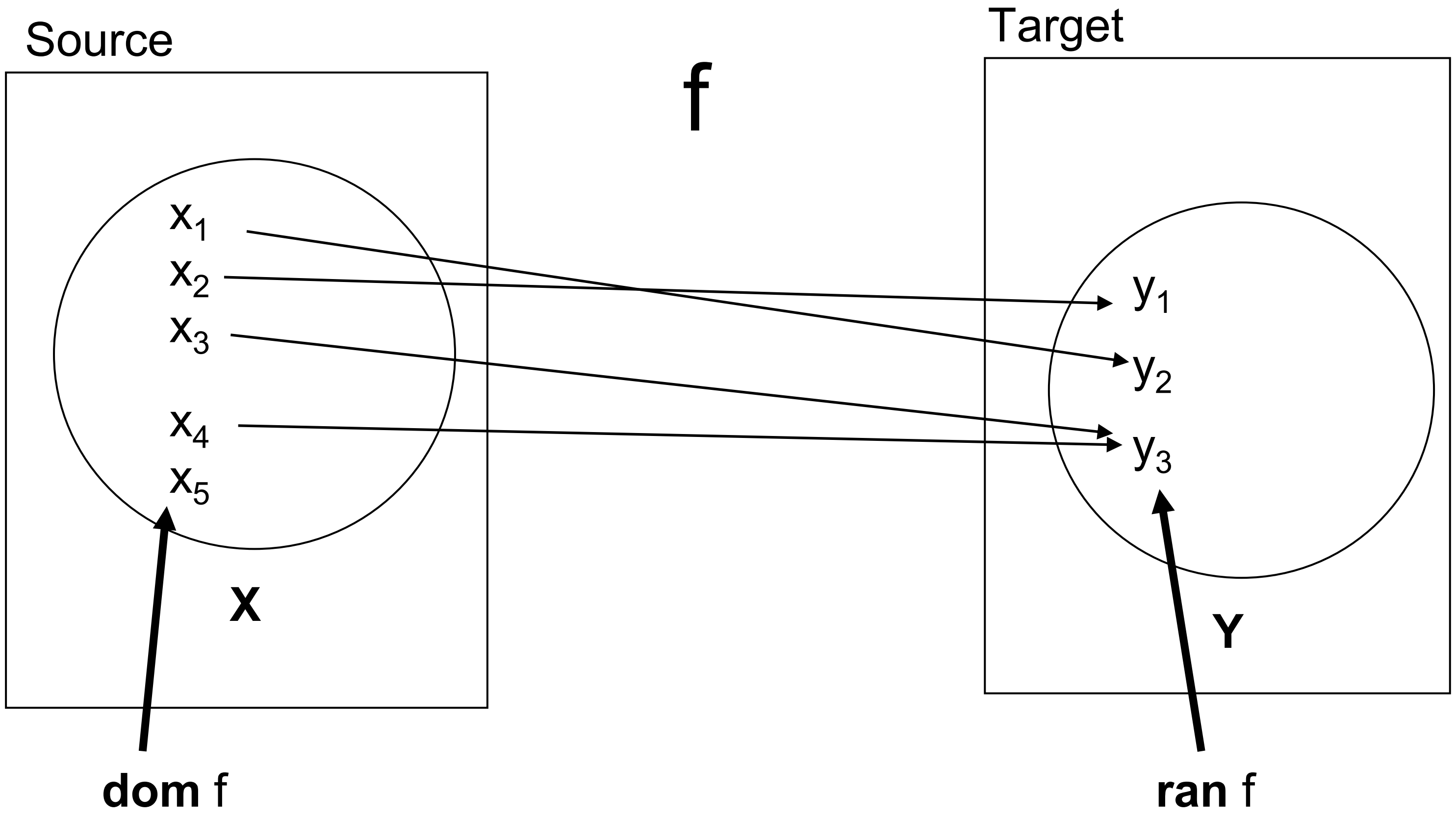


A one-to-one function
(Not onto)

Surjection (Onto)

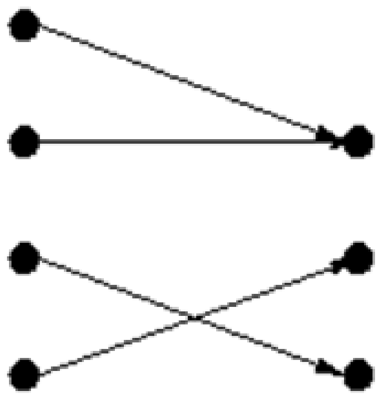
- The function $f: X \leftrightarrow Y$ is surjective (or onto) because range of f is all of Y.
- The **range** of the function is the **whole of the target**.
- A function f is **onto** iff every possible element $y \in \text{ran } f$ has some corresponding value $x \in \text{dom } f$.

Surjection (Onto)



Partial (⤀) and Total (↠) Surjections

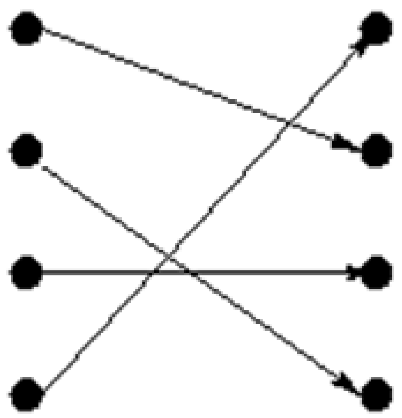
- f is an "onto" or "surjective" function for whose range is B, written as A ↠ B
- Partial surjections (⤀):
  - Some members of A are paired with a whole members of B.
- Total surjections (↠):
  - Every member of A is paired with a whole members of B.



An onto function
(Not one-to-one)

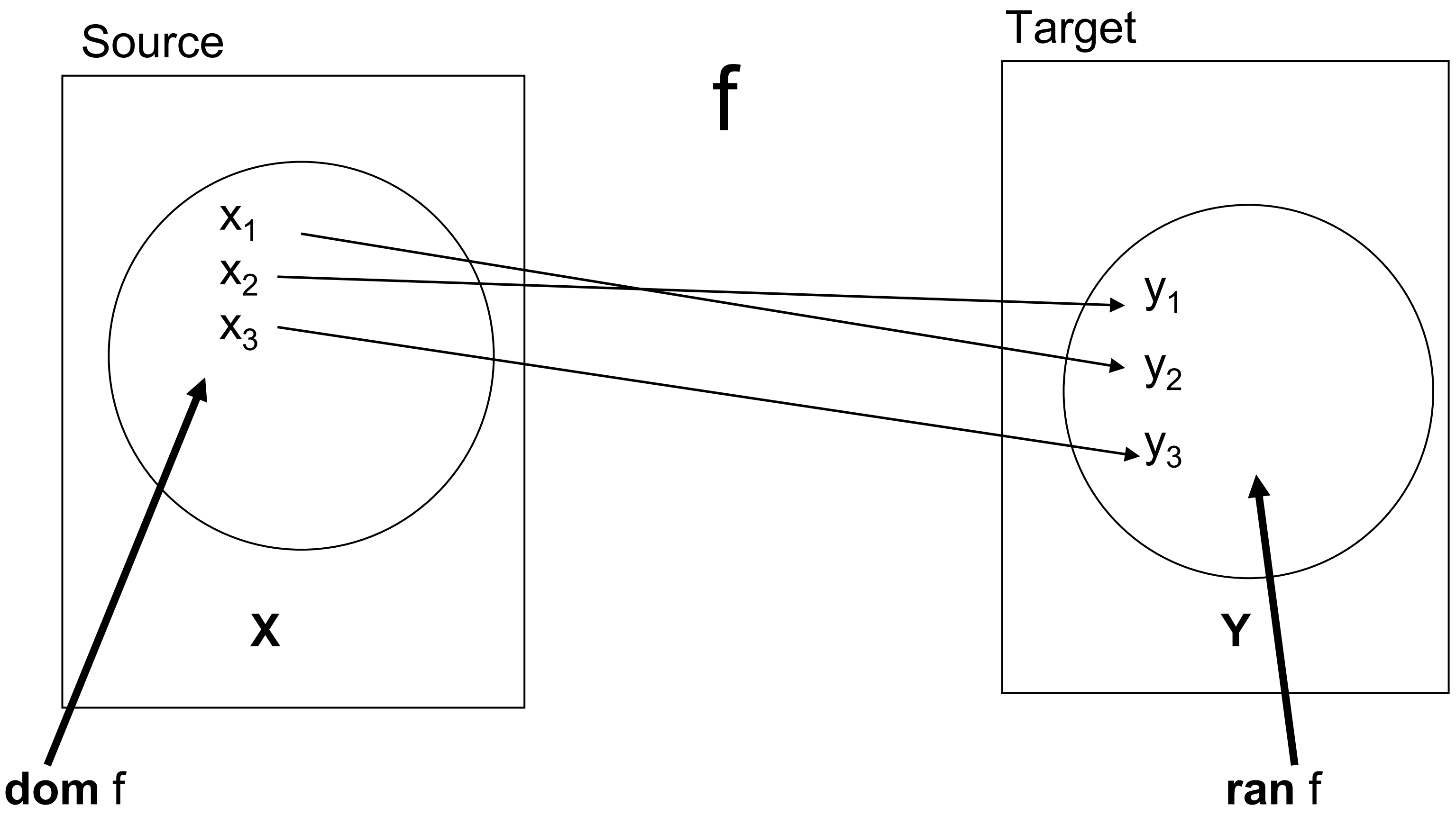
Bijection

- If a function is both an injection and a surjection
- Suppose f: A ↔ B
  - f is both one-to-one and onto or a "bijective" function, written as A ⤖ B
  - Every member of A is paired with a different member of B, covering all B's.
- A bijective function has an inverse.
- Normally will be Total Bijection function.



A bijection

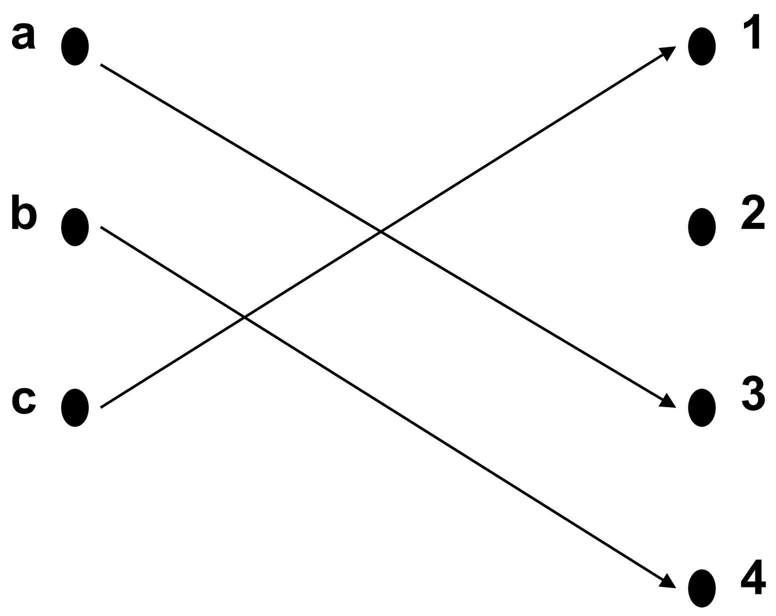
Bijection



Different Types of Correspondence

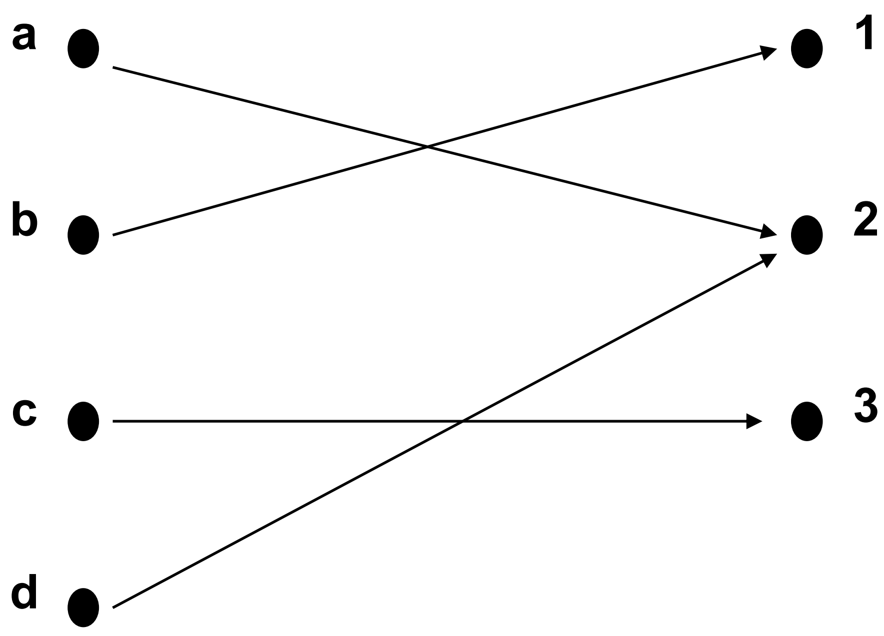
(a) One-to-one, not onto

(Total Injection ↣)



(b) Onto, not one-to-one

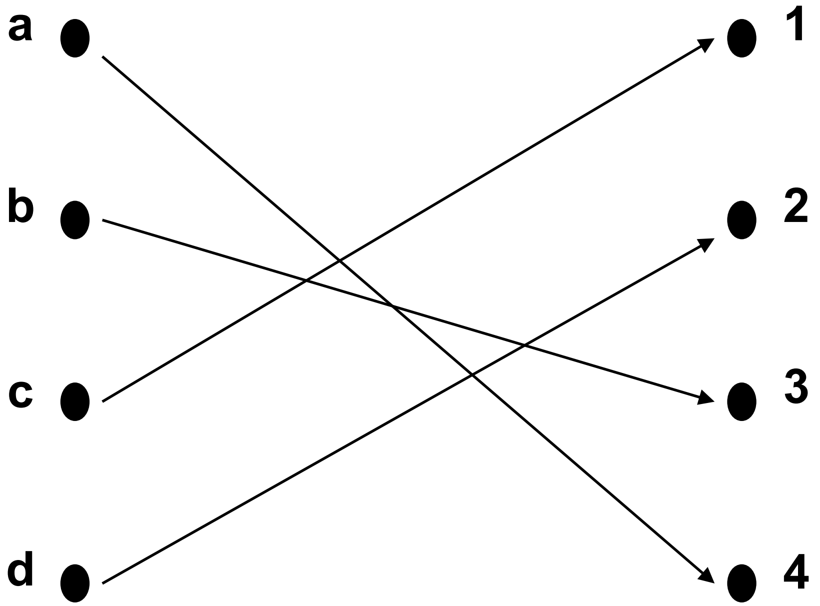
(Total Surjection ↠)



Different Types of Correspondence

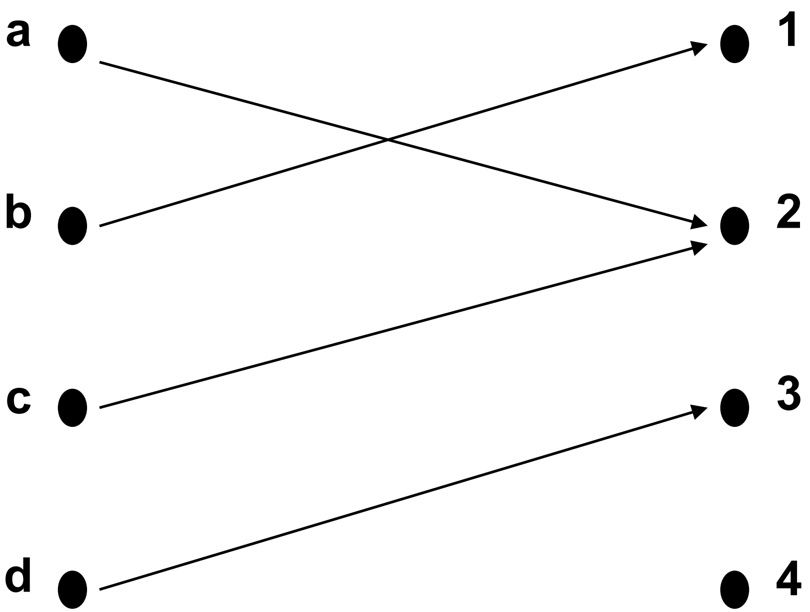
(c) One-to-one, and onto

(Bijection ⤖)



(d) Neither one-to-one nor onto

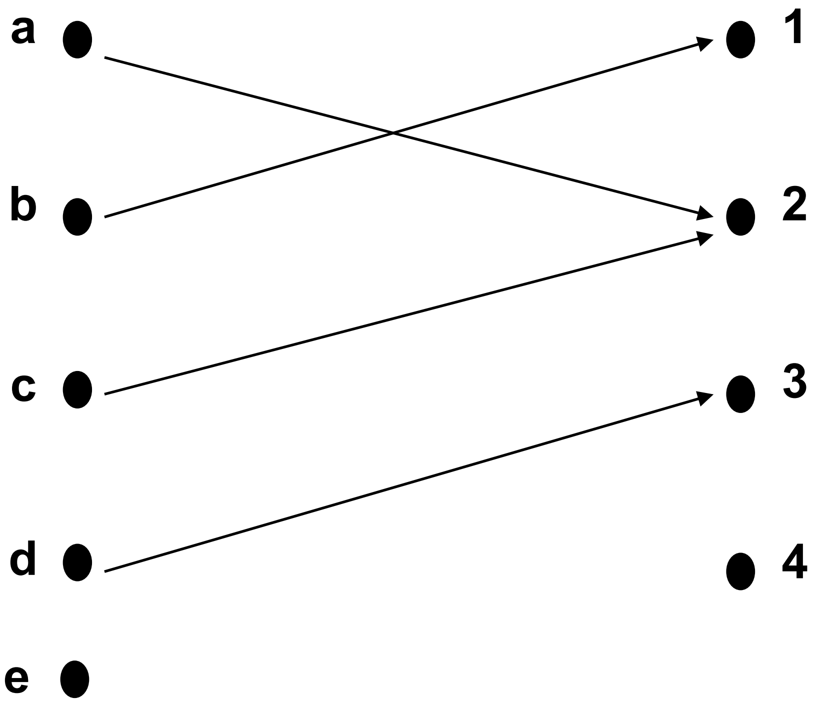
(Total function ⟶)



Different Types of Correspondence

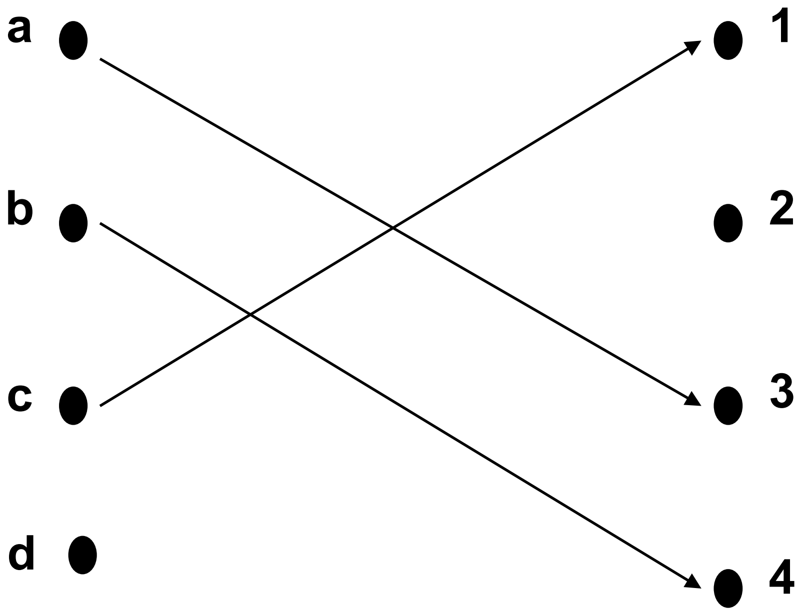
(e) Neither one-to-one nor onto

(Partial function ⇸)



(f) One-to-one, not onto

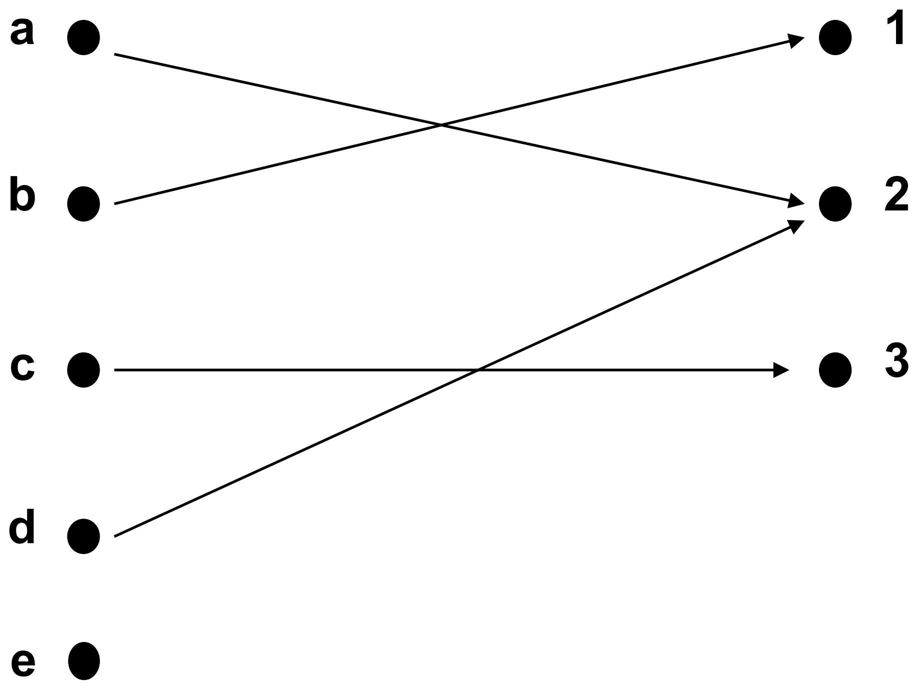
(Partial Injection ⤔)



Different Types of Correspondence

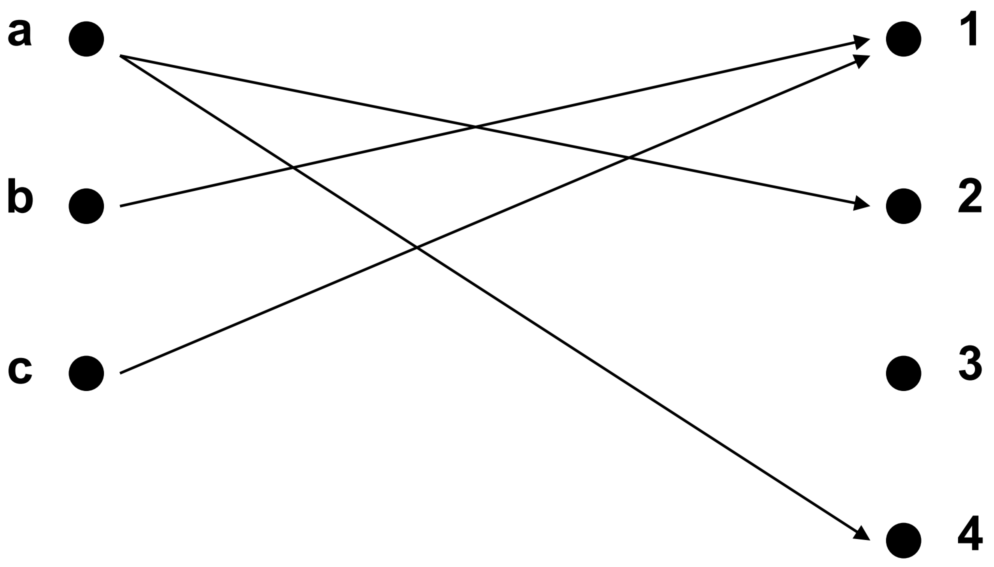
(g) Onto, not one-to-one

(Partial Surjection ⤀)



(h) Not a function

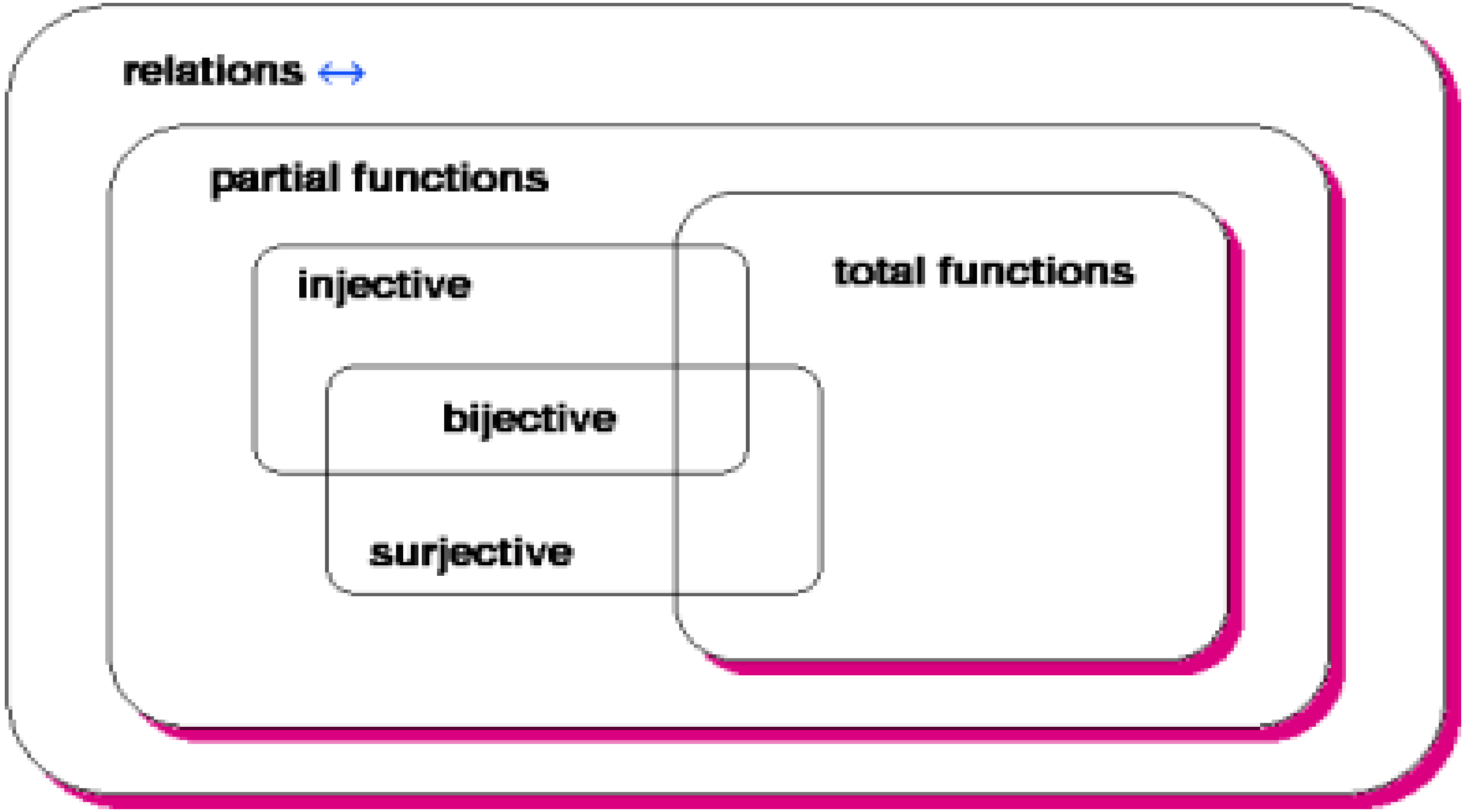
(Relation ↔)



Function Notations

Constructor	Returns
$\rightarrow$	Total functions
⇸	Partial functions
$\rightarrowtail$	Total injections
⤔	Partial injections
$\twoheadrightarrow$	Total surjections
⤀	Partial surjections
⤖	Bijections

Function Notations



Function Overriding

- Make some changes to the set
- Use symbol $\oplus$
- Suppose we have:

$$phone = \{peter \mapsto 1531, paul \mapsto 1488, mary \mapsto 1777\}$$

then

$$phone \oplus \{peter \mapsto 1555\} =$$

$$\{peter \mapsto 1555, paul \mapsto 1488, mary \mapsto 1777\}$$

Exercise

Given $rooms = \{1 \mapsto occupied, 2 \mapsto vacant, 3 \mapsto occupied, 5 \mapsto vacant, 6 \mapsto occupied\}$. Write the result for each of the following expressions:

a) $rooms \oplus \{1 \mapsto vacant\}$

b) $rooms(5)$

c) $rooms \setminus \{5 \mapsto rooms(5)\}$

Birthday Book Example

Birthday Book

- A birthday book is a system which:
  - Records people’s birthdays
  - Find a person’s birthday
  - Able to issue a reminder when the day comes around.
- So, we need to deal with people’s names and dates.

Birthday Book

- So we introduce **basic types** of the specification NAME as the set of all names and DATE as the set of all dates.

  [NAME, DATE]

- We are able to name the sets without saying what kind of objects they contain.

Birthday Book

- The first aspect of the system to describe is its **state space**, and we do this with a schema.

$$\underline{\quad BirthdayBook \quad}$$

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

$$birthday : NAME \rightarrowtail\!\!\!\!\!\!+\ DATE$$

$$\text{dom } birthday \subseteq known \qquad Or \qquad \text{dom } birthday = known$$

Birthday Book

- Where are known objects are:
  - known is the set of names with birthdays recorded;
  - birthday is a function which, when applied to certain names, gives the birthdays associated with them.

$$known == \{John, Mike, Susan, \ldots\}$$

$$birthday == \{John \mapsto 25Mar, Mike \mapsto 20Dec,$$
$$\{Susan \mapsto 20Dec, \ldots\}$$

Birthday Book

- When the state starts, the function birthday is empty.
- So, we describes an initial state of a birthday book in which the set known is empty.

InitBirthdayBook

InitBirthdayBook
$BirthdayBook$
$known = \varnothing$

InitBirthdayBook
$BirthdayBook$
$known = \varnothing$ $birthday = \varnothing$

Birthday Book

- To add a new birthday, we describe the operation with a schema.

AddBirthday

$\Delta\ BirthdayBook$
$name? : NAME$
$date? : DATE$

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

Δ BirthdayBook indicates that the schema is describing a *state change*

pre-condition for the success of the operation

updates of the operation

Birthday Book

- Another operation might be to find the birthday of a person known to the system.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\mathit{FindBirthday}}} \\
\Xi\ \mathit{BirthdayBook} \\
name? : NAME \\
date! : DATE \\
\hline
name? \in known \\
date! = birthday\,(name?) \\
known' = known \\
birthday' = birthday \\
\hline
\end{array}
$$

Ξ BirthdayBook indicates that the schema is describing a *state does not change*

Function application

Remain unchanged variables

Birthday Book

- To remind who has birthday on the particular day so that birthday cards should be sent.
- The Remind schema does not change the BirthdayBook.

$$
\begin{array}{|l}
\textit{Remind} \\
\hline
\Xi\ \textit{BirthdayBook} \\
\textit{today}? : \textit{DATE} \\
\textit{cards}! : \mathbb{P}\ \textit{NAME} \\
\hline
\textit{cards}! = \{n : \textit{known} \mid \textit{birthday}\ (n) = \textit{today}?\} \\
\textit{known}' = \textit{known} \\
\textit{birthday}' = \textit{birthday} \\
\hline
\end{array}
$$

Birthday Book

- To strengthen the specification, extra states can be specified especially to detect error.

$$REPORT ::= ok \mid alreadyKnown \mid notKnown$$

$$\begin{array}{|l}
\hline
\textit{Success} \\
result! : REPORT \\
\hline
result! = ok \\
\hline
\end{array}$$

Birthday Book

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{___ } AlreadyKnown \text{ ___}} \\
\Xi BirthdayBook \\
name? : NAME \\
result! : REPORT \\
\hline
name? \in known \\
result! = alreadyKnown \\
\hline
\end{array}$$

$$TotalAddBirthday \mathrel{\widehat=} (AddBirthday \land Success) \lor AlreadyKnown$$

Birthday Book

$$
\begin{array}{|l}
\hline
\textit{NotKnown} \\
\hline
\Xi BirthdayBook \\
name? : NAME \\
result! : REPORT \\
\hline
name? \notin known \\
result! = notKnown \\
\hline
\end{array}
$$

$$TotalFindBirthday \mathrel{\widehat{=}} (FindBirthday \land Success) \lor NotKnown$$

Summary

- Z structure such as tuples, relations and functions are explained in this lecture.
- At the end of the lecture, Birthday Book example is explained to grasp the concepts of Z specification.

THANK YOU!!