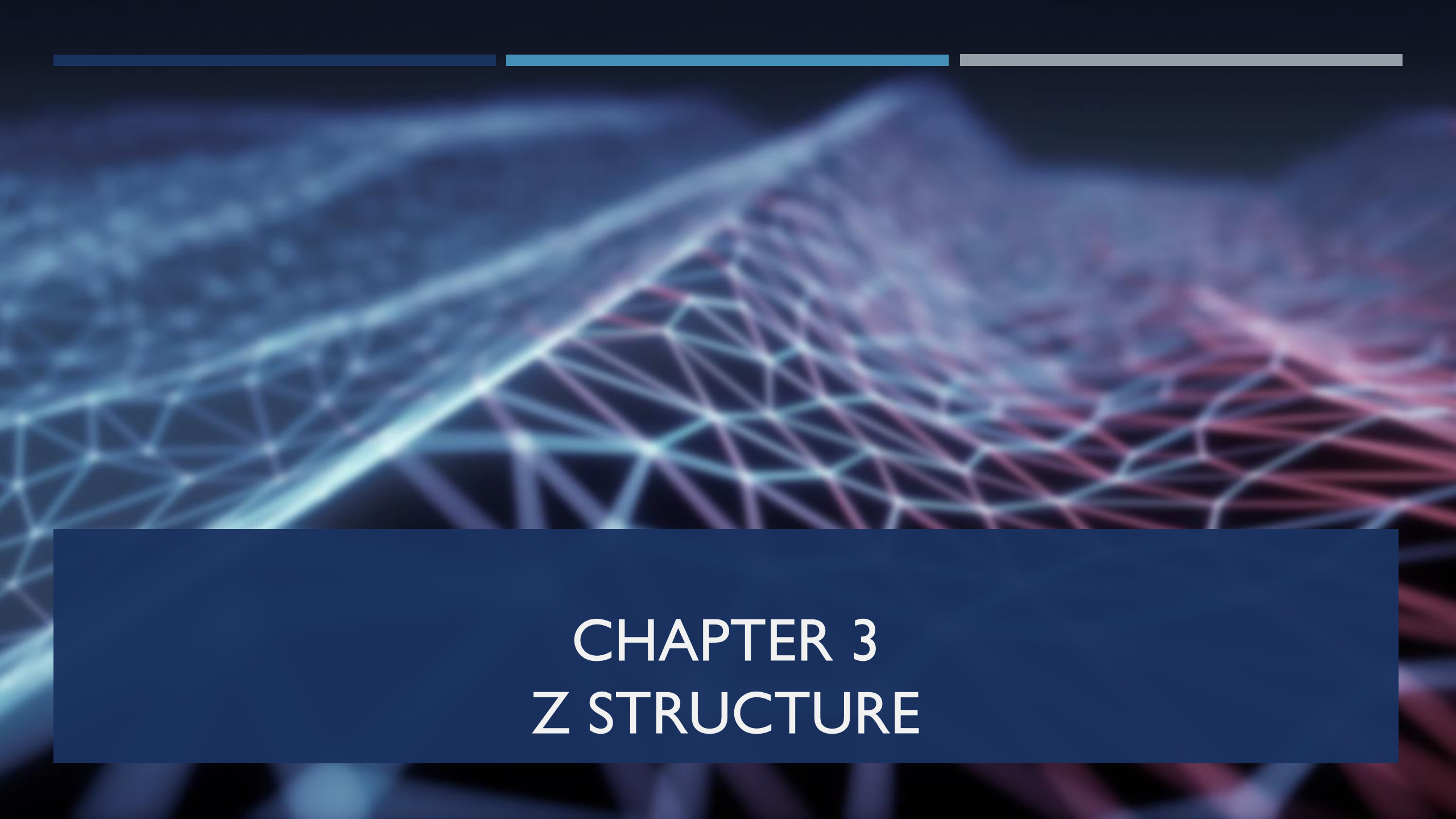


CHAPTER 3
Z STRUCTURE

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 later Chapter

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

- Can be described using variables.
- 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 (specific format)

  • month can be FREE TYPE

TUPLES

- Declare tuples which are instances of the type

  staff: EMPLOYEE

  → staff == (0019, Frank, admin)

$$\frac{staff : EMPLOYEE}{staff == (0019, Frank, admin)}$$

Cartesian Products

- **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)}

Cartesian Products

- Assume we have these data in our table:

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

Cartesian Products

- In Z, we can expressed:

EMPLOYEE == ID x NAME x DEPT - fixed, position can't exchange

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

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

RELATION

- A relation is a set of tuples. They can resemble tables or databases.
- The set of all relations over sets S, T is indicate by $S \leftrightarrow T$
  - Note **$S \leftrightarrow T$** is equivalent to **$\mathbb{P}(S \times T)$** <– x nid use this anymore
- **Many-to-many relationship**
- Example:

  A: $\mathbb{N} \leftrightarrow \mathbb{N}$ B: $\mathbb{N} \leftrightarrow$ Color C: Person $\leftrightarrow$ Month

  A == {1↦1, 1↦ 2, 2↦2}

  B == {2↦red, 5↦blue, 3↦red}

  C == {David↦Jun, Mary↦Aug, Bill↦Feb}

BINARY RELATION

- A relation in Z is expressed as a binary relation.
- A binary relation is just a set of pairs.

↔ is a *binary relation operator*. It associates from **left to right**.

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

or

NAME ↔ PHONE

BINARY 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:

$$phone : NAME \leftrightarrow PHONE$$

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

BINARY RELATION

- We have given types for the set of all dates and the set of all people

  [DATE, PERSON]

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

$$appointments : DATE \leftrightarrow PERSON$$

$$appointments == \{nov7 \mapsto tom, nov7 \mapsto anne, nov8 \mapsto jerry, nov12 \mapsto tom, \ldots\}$$

BINARY RELATION

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

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

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

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

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

FIRST AND SECOND

- Each element in a binary relation is a 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

  1st coordinate ↖ ↗ 2nd coordinate

  Eric ↦ Suzy : Eric map to Suzy, position exchange different meaning
  - ↦ this is known as *maplet notation*
  - (Eric, Suzy) = Eric ↦ Suzy
  - The **maplet** notation provides alternate syntax without parentheses.

FIRST AND SECOND

- An element in a binary relation is represented using *maplet notation*

  partners = {Eric ↦ Suzy, Mike ↦ Anne}

- *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 is known as **domain**.

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

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

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

DOMAIN AND RANGE

- Example:

A == {1↦1, 1↦ 2, 2↦ 2}

dom A = {1, 2}

ran A = {1, 2}

- data in set dh duplication of data

B == {2↦ red, 5↦ blue, 3↦ red}

dom B = {2, 3, 5}

ran B = {red, blue}

C == {David↦ Jun, Mary↦ Aug, Bill ↦ 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**. (not keyword)

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

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**. → not keyword

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

target

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:

search domain data & display (both domain & range)

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

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

DOMAIN RESTRICTION

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

  $\{4\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \emptyset$ -> can't find relevant domain data

  $\{1, 3\} \lhd \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \{1 \mapsto a, 3 \mapsto c\}$ -> can do searching > 1

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

DOMAIN RESTRICTION

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

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

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

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

search range data & display (both domain & range)

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

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

RANGE RESTRICTION

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

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

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

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

RANGE RESTRICTION

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

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

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

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

- Used to search domain data on

- 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.

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

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

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

RELATIONAL IMAGE

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

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

INVERSE

- The binary relation symbol, $\leftrightarrow$, associates from left to right.
- Inverse reverses domain and range by exchanging the components of each pair.
- 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

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

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

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

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

  costs~ = buys

- ~ is the *inverse operator*.

INVERSE

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

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

INVERSE

- The inverse for phone:

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

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*:

phone : NAME ↔ PHONE

if wan combine, make sure same type

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

dept : PHONE ↔ DEPT

dept = {1000 ↦ admin,…, 1999 ↦ admin,
2000 ↦ research, …, 2999 ↦ research,
3000 ↦ manufacturing, …,
3999 ↦ manufacturing, …. }

COMPOSITION

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

$$\text{phone} \fatsemi \text{dept} = \{ \text{Frank} \mapsto \text{admin}, \text{Philip} \mapsto \text{admin}, \text{Doug} \mapsto \text{research}, \text{Anne} \mapsto \text{admin}, \text{Mike} \mapsto \text{manufacturing}, \text{Jane} \mapsto \text{research}, \text{Philip} \mapsto \text{research}, \ldots \}$$

EXERCISE

Write the result for each of the following expressions:

a) {1 ↦ mon, 2 ↦ tue, 3 ↦ wed, 4 ↦ thu, 5 ↦ fri} ⦇{3}⦈ {wed}

b) {1013 ↦ PSP, 2073 ↦ SDA, 2083 ↦ FM, 3283 ↦ SC} \ {3293 ↦ SQAT} (ans is remain all)

c) {Suzy ↦ 25, Adam ↦ 27, Lim ↦ 24} ⊕ {Adam ↦ 23, Cindy ↦ 22} 23 → add as 4th record

d) {060 ↦ Malaysia, 061 ↦ Australia, 091 ↦ India, 064 ↦ Singapore}~ exchange dom & ran

e) {Jan ↦ 31, Feb ↦ 28, Mar ↦ 31, Apr ↦ 30, May ↦ 31, Jun ↦ 30, Jul ↦ 31, Aug ↦ 31, Sep ↦ 30, Oct ↦ 31, Nov ↦ 30, Dec ↦ 31} ▷ {30}

f) {064, 061} ⩤ {061 ↦ Australia, 091 ↦ India, 064 ↦ NewZealand, 060 ↦ Malaysia}

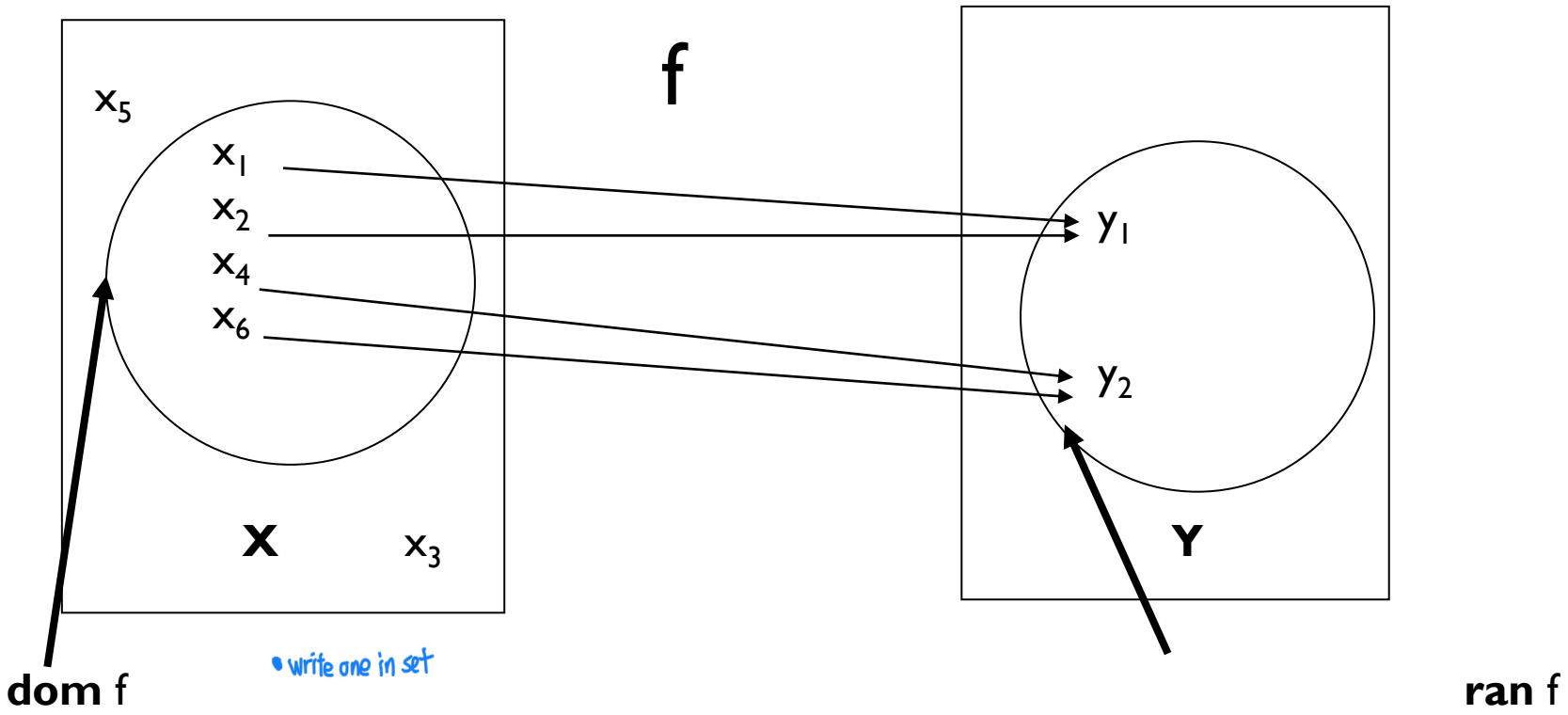
remain these two only

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.
- A function with a finite domain is also known as a mapping.
- 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.

• OK the data not in order

FUNCTION

- Other example:

  B == {2 ↦ red), 5 ↦ blue, 3 ↦ red}

  C == {David↦ Jun, Mary↦ Aug, Bill↦ Feb}

  BUT Not

  A == {1↦1, 1↦2, 2↦2}

EXAMPLES OF FUNCTION

registered : STUDENTID ↔ STUDENT

- Because no two students have the same ID

car : REGISTRATION ↔ VEHICLE

- Because no two vehicles have the same registration number

account : BANKACCOUNT ↔ OWNER

- Because each bank account is unique. An owner could be a person, a couple or a company. An owner could have several accounts, but bank account belongs to just one owner.

FUNCTION DECLARATION

- We introduce a given type STUDENT, the set of all students.
- We intend to have:

  no two students in a class have the same identity number

- We define *class* as a function from $\mathbb{Z}$ to STUDENT

  class : $\mathbb{Z} \pfun$ STUDENT (map only a student, only one result)

- $\pfun$ is a *finite function* – not the whole of $\mathbb{Z}$ is used and the function is finite – we can count the number of members in a class.

FUNCTION DECLARATION

- As the function is **finite**, we can apply the *cardinality* operator, **#**, and restrict the size of class, such as between 0 and 20 student.

$$class : \mathbb{Z} \pfun STUDENT$$

$$\#class \geqslant 0$$

$$\#class \leqslant 20$$

- Example: class = {1↦peter, 2↦paul, 3↦mary}
- The name of a function is usually a singular and chosen to reflect the **range**.

FUNCTION APPLICATION

- Get the exact data

- All the concepts which pertain to relations apply to functions. In addition a function may be **applied**.
- Since there will be at most only one value in the range for a given x it is possible to designate that value directly.
- The value of f applied to x is the value in the range of function of the function f corresponding to the value of x in its domain.
- The application is undefined if the value of x is NOT in the domain of f.

FUNCTION APPLICATION

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

  fx or f(x)

- We can check the applicability of f by writing:

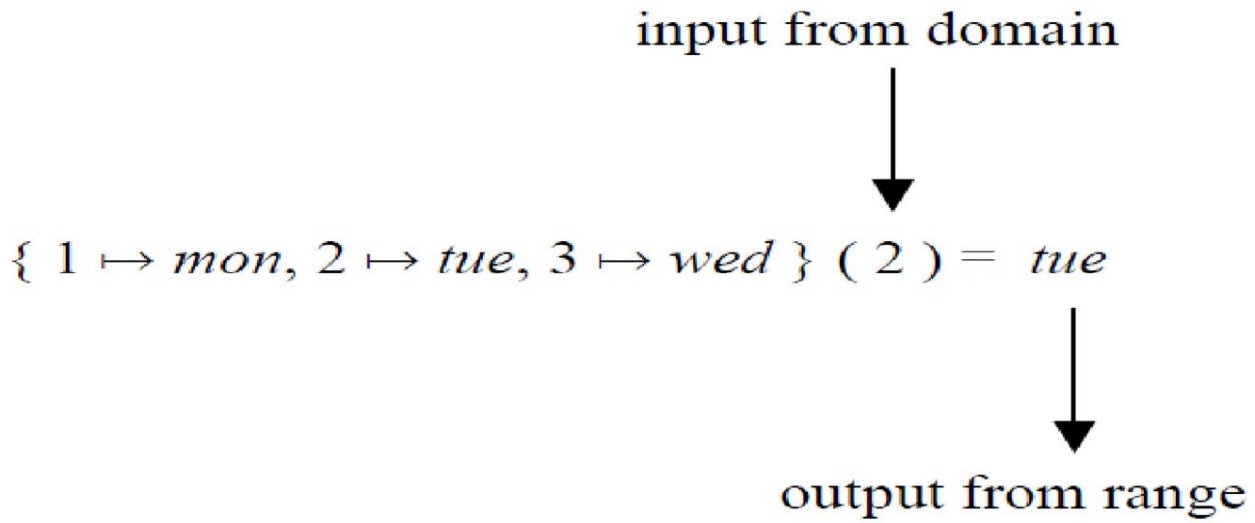
  $x \in \text{dom } f$

  $f(x) = y$

- These predicates must be true if f(x) = y is a function.

FUNCTION APPLICATION

- Because the *relational image* on a function must always give a set with just one element, so we can use *function application* notation.
- Example: {1 ↦ mon, 2 ↦ tue, 3 ↦ wed} (2) = tue



FUNCTION APPLICATION

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

FUNCTION APPLICATION

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

$$day : \mathbb{Z} \pfun DAY$$

$$day = \{ 1 \mapsto mon, 2 \mapsto tue, 3 \mapsto wed \}$$

- We may write:

same thing

day (2) = tue OR day 2 = tue

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

(do checking to avoid

FUNCTION APPLICATION

- Examples of function:

  phone (Jane) = 2300

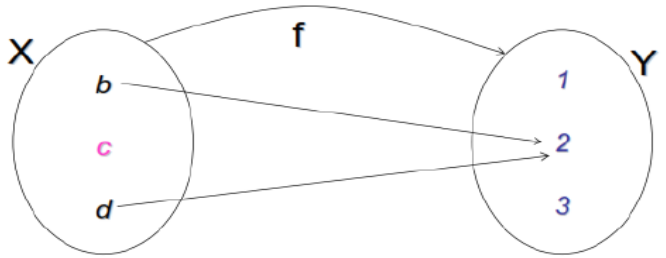
  phone Jane = 2300

FUNCTION APPLICATION

- Basically, the **domain** of a function can be represented in two different ways:
  - Partial function ($\nrightarrow$) - Eg, finite function
  - 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. (talking about domain data, x)



PARTIAL FUNCTION (⇸)

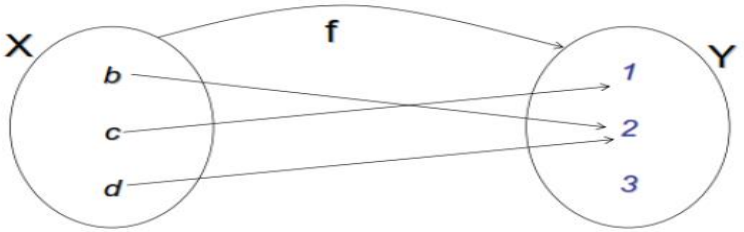
- A lot of what we call *functions* in software are actually **partial functions** because they're not defined everywhere. One is the application of a partial function such as the 'div' operator.
- Thus, the function:

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

domain restriction: $\{ 2 \} \lhd \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \{ 2 \mapsto b \}$

range restriction: $\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \rhd \{ a \} = \{ 1 \mapsto a, 3 \mapsto a \}$

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: $\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \fatsemi \{ a \mapsto X, b \mapsto Y \} = \{ 1 \mapsto X, 2 \mapsto Y, 3 \mapsto X \}$

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

- Injection (one-to-one)
- Surjection (onto)
- Bijection

INJECTION (ONE-TO-ONE)

- might have undefined answer

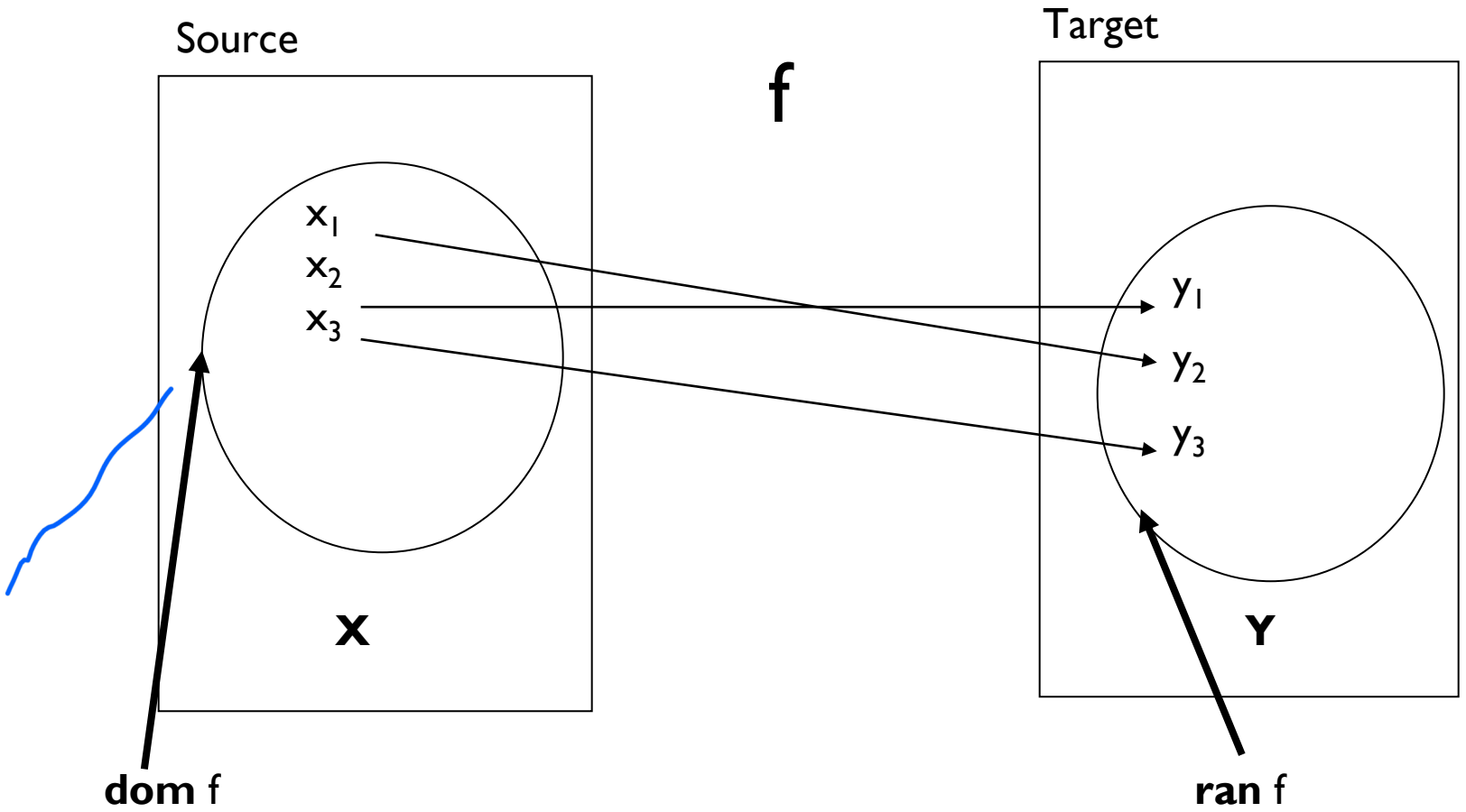
- An injection or (injective function) is a function which maps different values of the source on to different values of the target.
- Example: • range must be diff

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

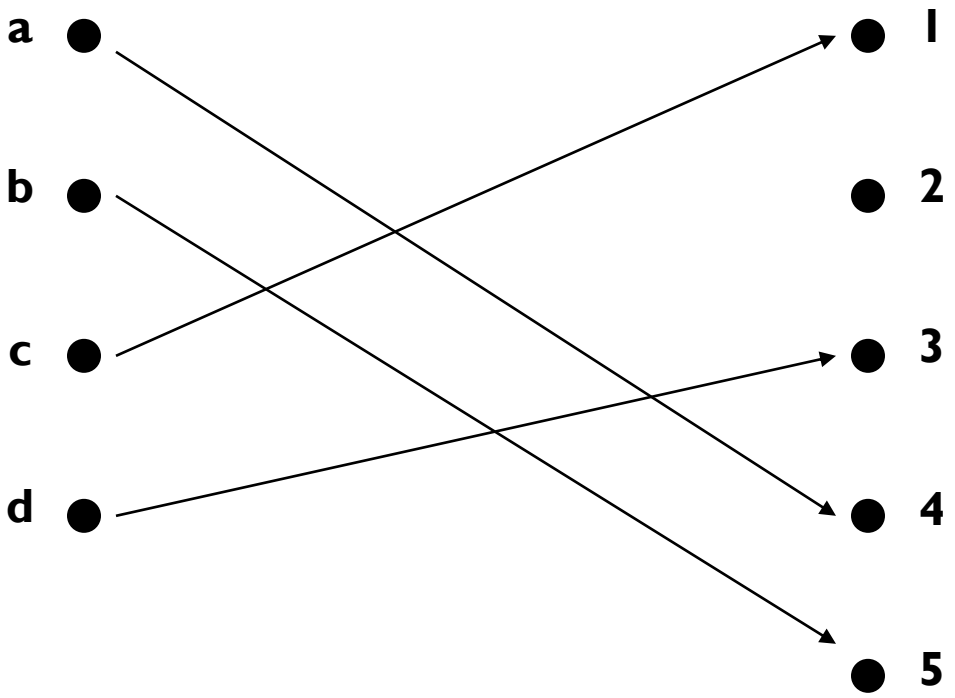
whereas, the following is NOT: (cuz domain x duplicated, range duplicated)

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

INJECTION (ONE-TO-ONE)



INJECTION (ONE-TO-ONE)

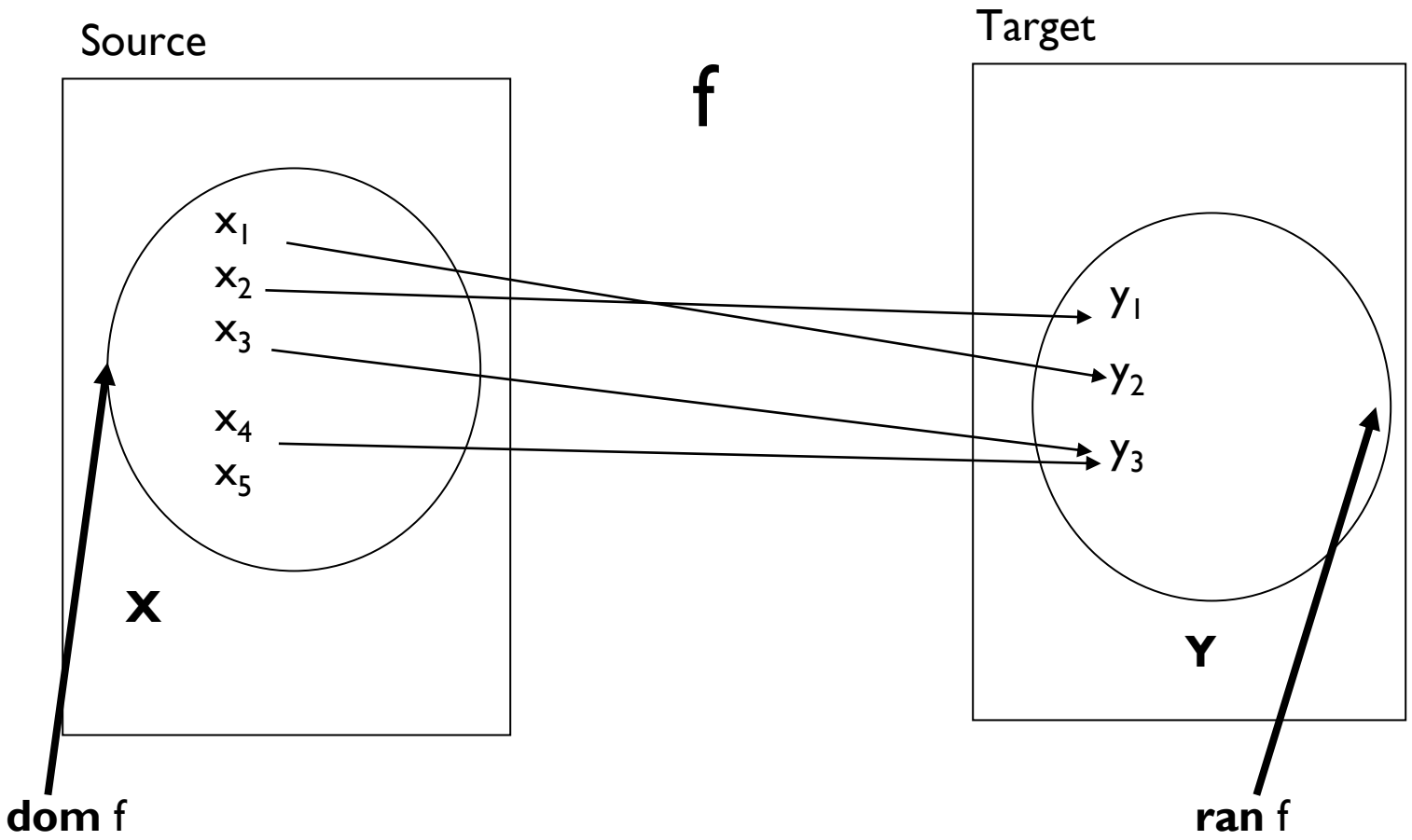


SURJECTION (ONTO)

- Result might not we want
- might not a function

- The function f: X ↔Y is surjective (or onto) because range of f is all of Y. The range is the whole of the target.
- A function f is **onto** iff every possible element y ∈ ran f has some corresponding value x ∈ dom f.
  - all range data must at least map to a domain data (x necessary 1 to 1 relationship)

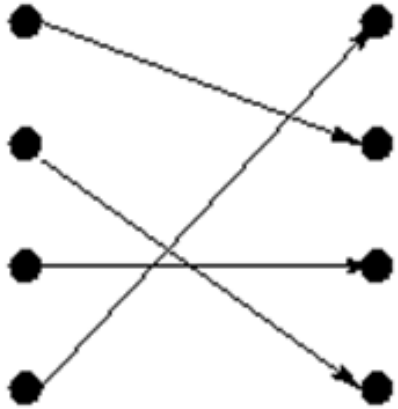
SURJECTION (ONTO)



BIJECTION

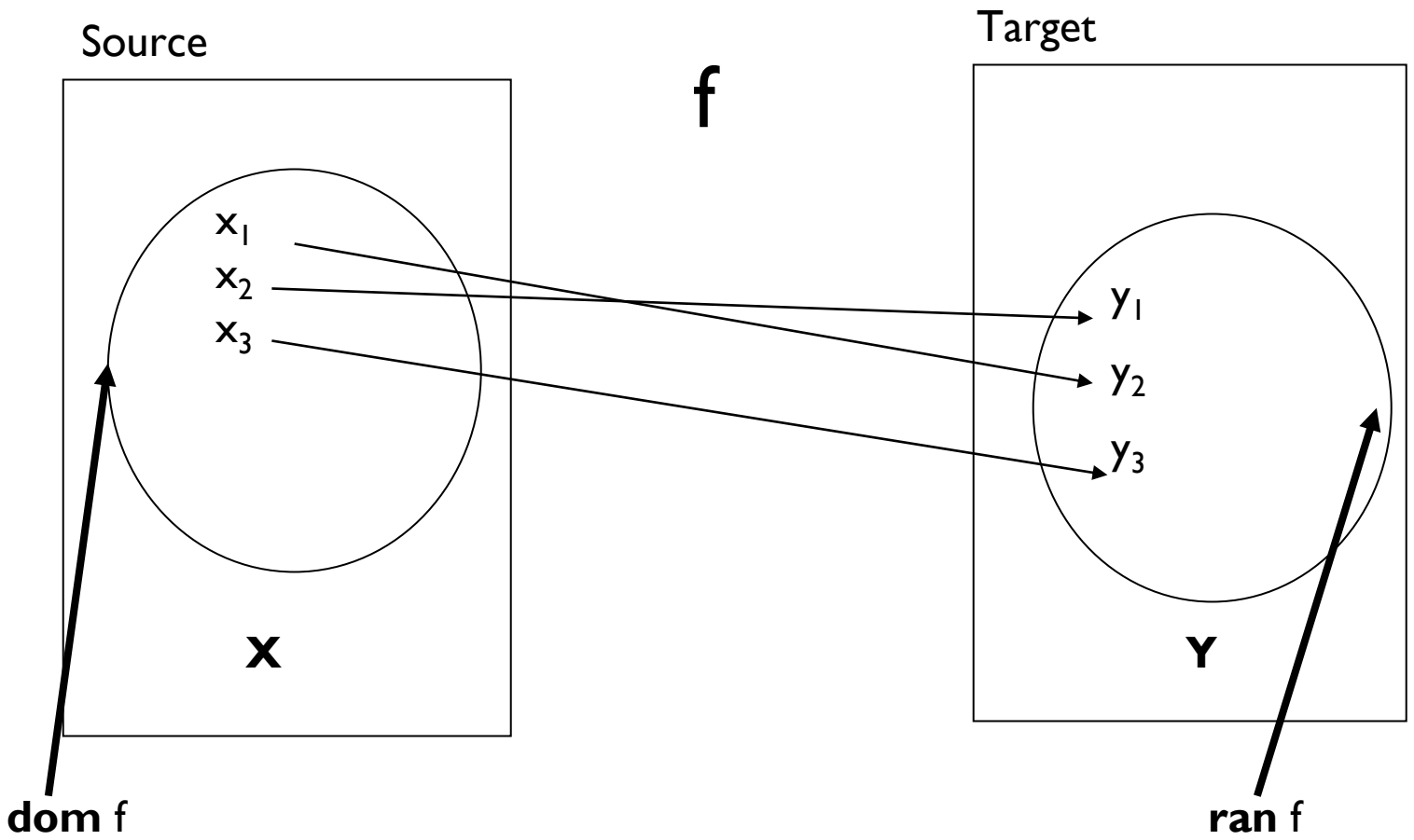
- same like surjection but 1 to 1 relationship

- 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.



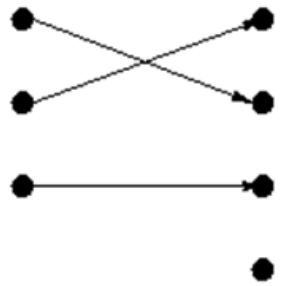
A bijection

BIJECTION



PARTIAL (⤔) AND TOTAL (↣) INJECTIONS

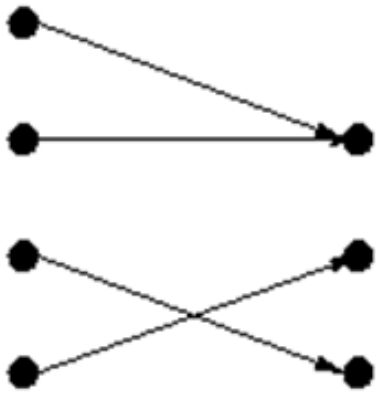
- Suppose f: A ↔ B
  - f is a "one-to-one" which no element in ran(f) is associated with more that one element in dom(f), written as A⤔B or A↣B
- Partial injections : (software development goes here)
  - Some members of A are paired with different members of B.
- Total injections :
  - Every member of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



A one-to-one function
(Not onto)

PARTIAL (↠) AND TOTAL (→) SURJECTIONS

- f is an "onto" or "surjective" function for whose range is B, written as A ↠ B
  - all range data will map to domain

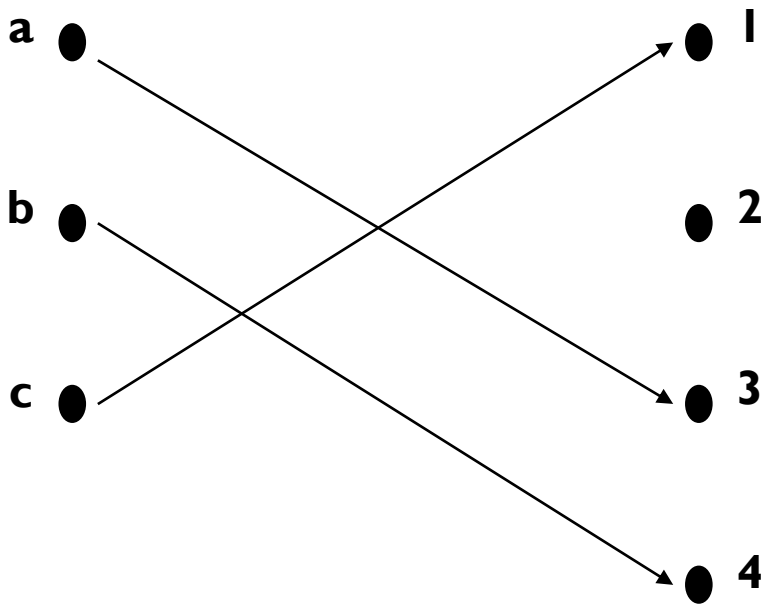


An onto function
(Not one-to-one)

DIFFERENT TYPES OF CORRESPONDENCE

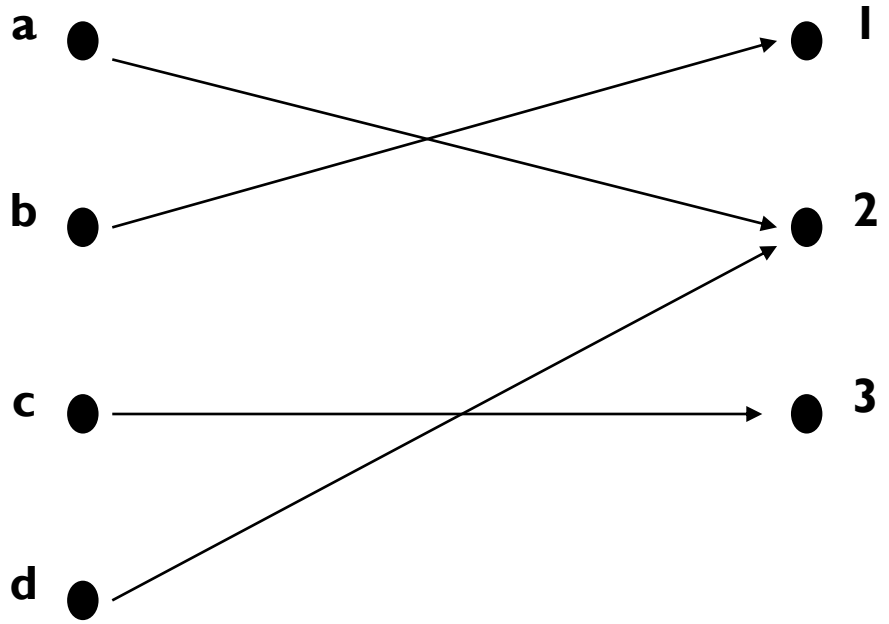
injection

(a) One-to-one, not onto



surjection

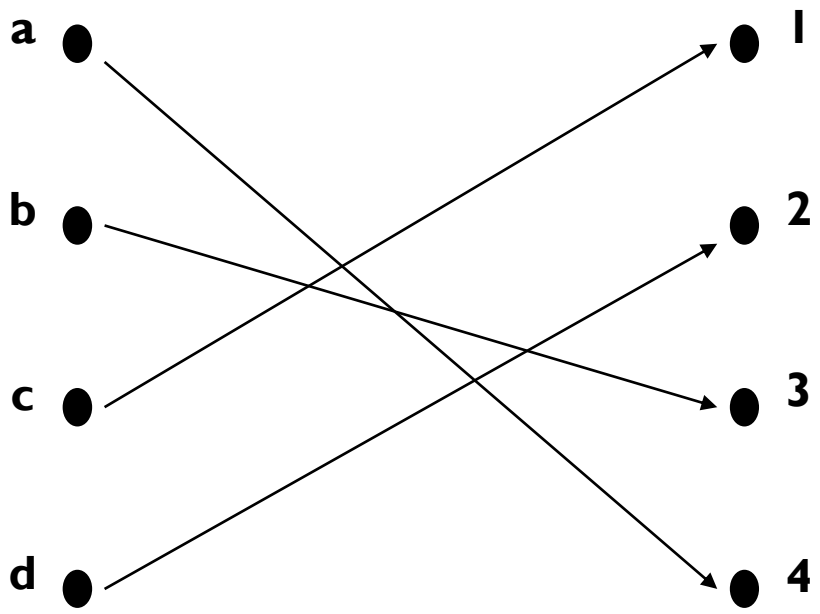
(b) Onto, not one-to-one



DIFFERENT TYPES OF CORRESPONDENCE

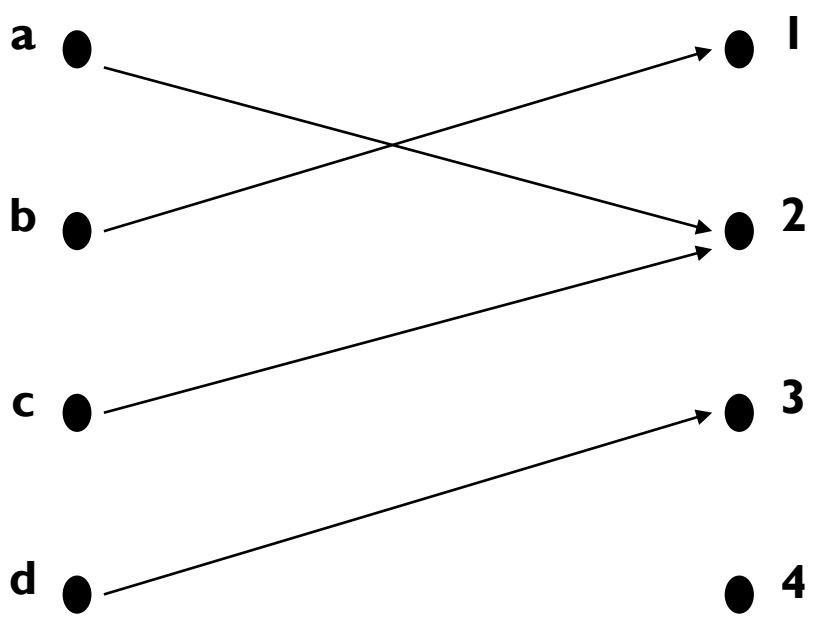
bijection

(c) One-to-one, and onto



normal partial function

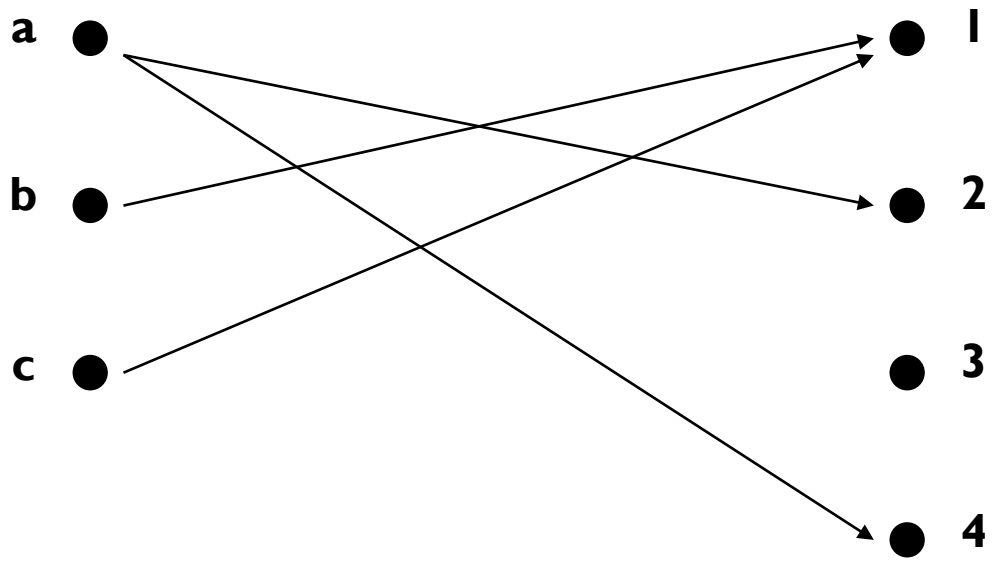
(d) Neither one-to-one nor onto



DIFFERENT TYPES OF CORRESPONDENCE

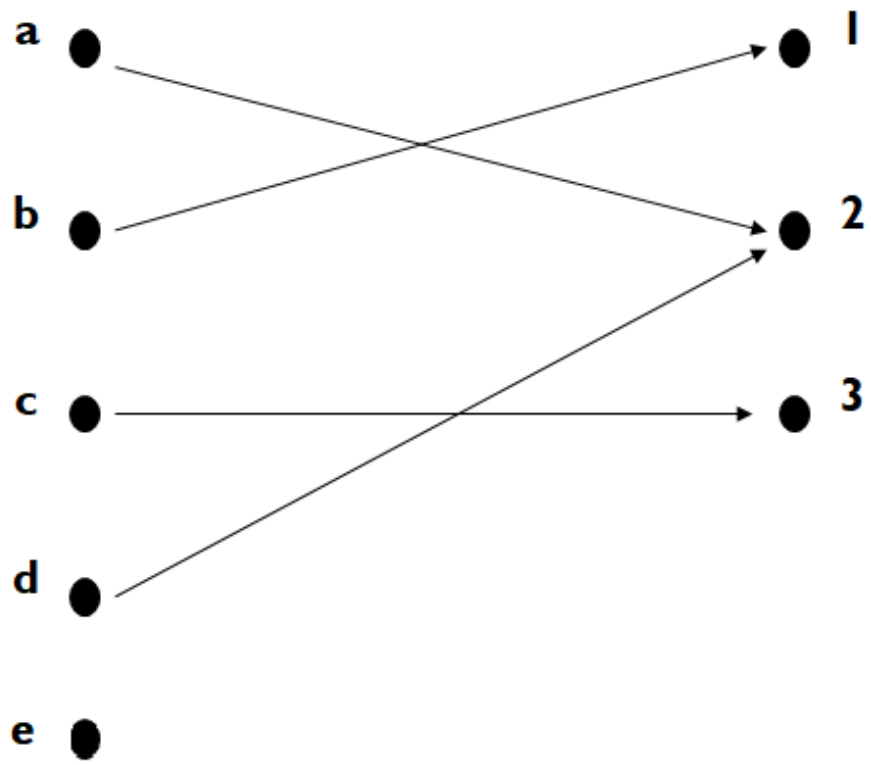
binary relation cuz many-to-many relationship

(e) Not a function

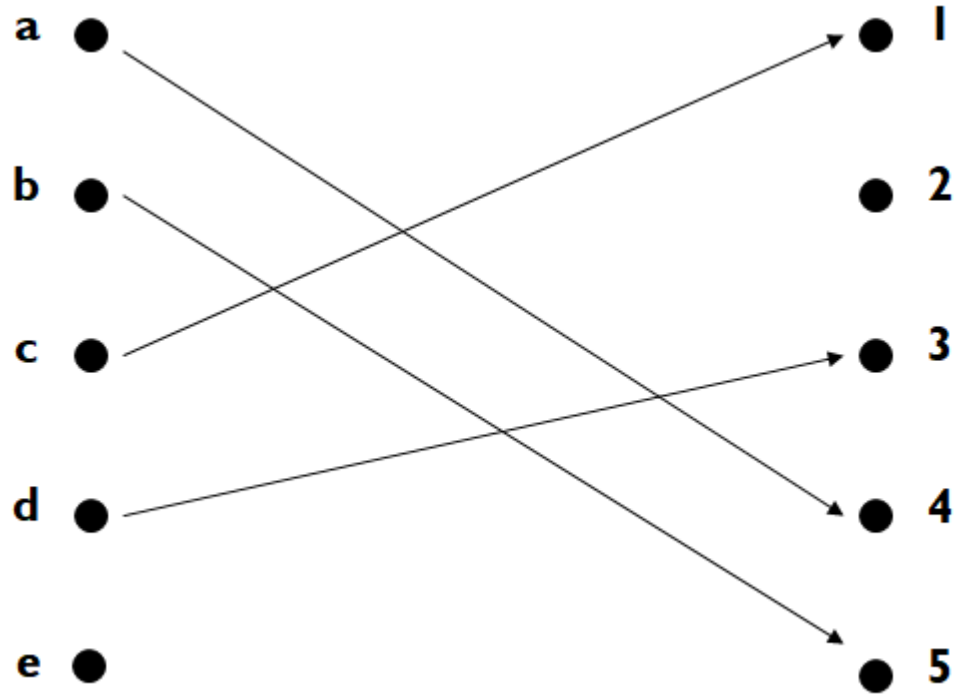


DIFFERENT TYPES OF CORRESPONDENCE

(f) Onto partial surjection



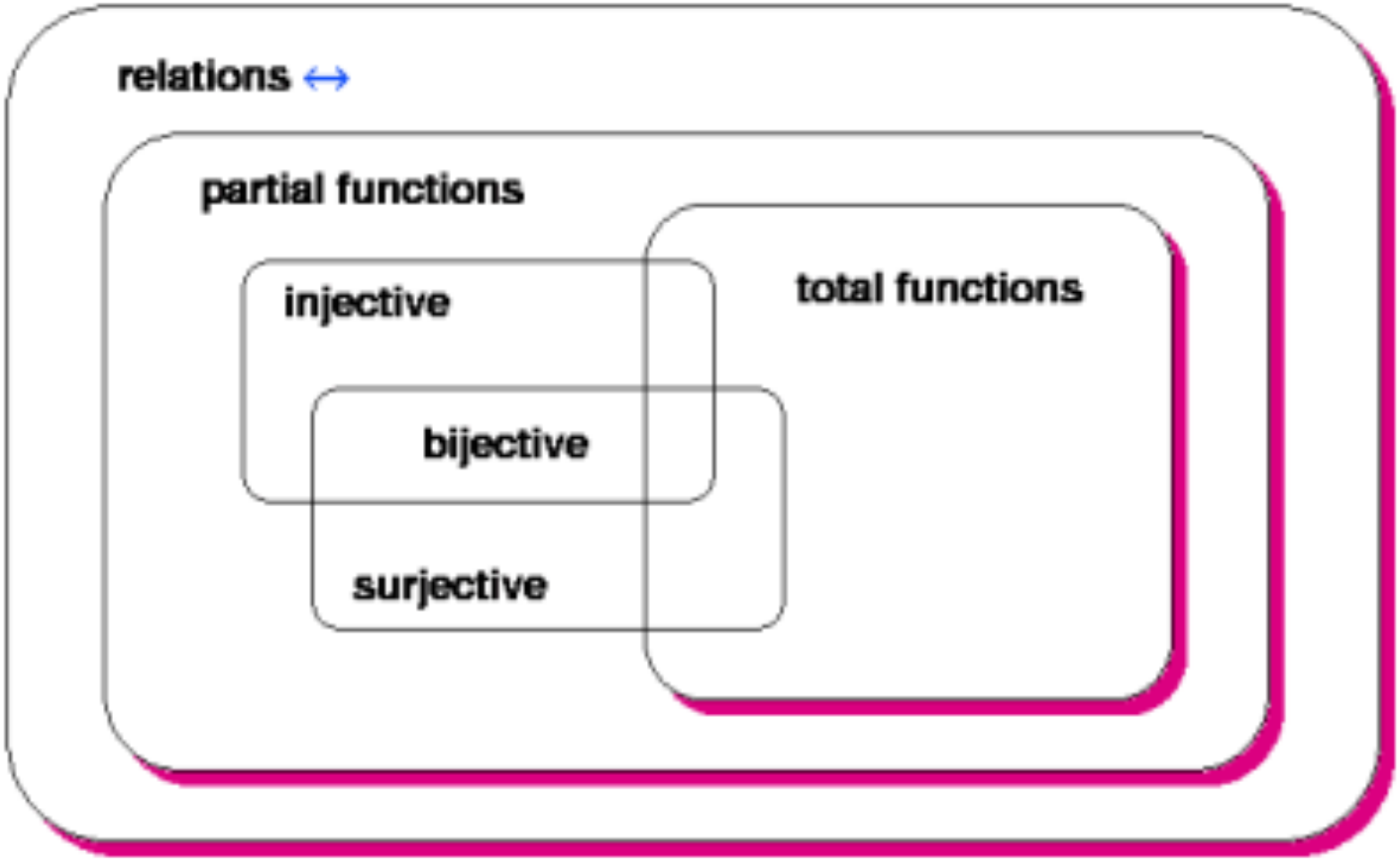
(g) One-to one partial injection



FUNCTION NOTATIONS

Constructor	Returns
$\rightarrow$	Total functions
$\nrightarrow$	Partial functions
$\rightarrowtail$	Total injections
$\nrightarrow$	Partial injections
$\twoheadrightarrow$	Total surjections
$\nrightarrow$	Partial surjections
$\rightarrowtail\!\!\!\!\twoheadrightarrow$ $\nrightarrow\!\!\!\!\twoheadrightarrow$	Bijections (total) (partial)

FUNCTION NOTATIONS



FUNCTION OVERRIDING

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

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

then

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

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

EXERCISE

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

a) rooms ⊕ {1 ↦ vacant} = {1 ↦ vacant, 2 ↦ vacant, 3 ↦ occupied, 5 ↦ vacant, 6 ↦ occupied}

b) rooms(5) = vacant

c) rooms \ {5 ↦ rooms(5)} = {1 ↦ occupied, 2 ↦ vacant, 3 ↦ occupied, 6 ↦ occupied}

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.

$$
\begin{array}{|l}
\textit{BirthdayBook} \\
\hline
known : \mathbb{P}\, NAME \\
birthday : NAME \pfun DATE \quad \text{(many to one)} \\
\hline
known = dom\ birthday \\
\hline
\end{array}
$$

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.

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

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

$BirthdayBook$

$known = \varnothing$

InitBirthdayBook

$BirthdayBook$

$known = \varnothing$

$birthday = \varnothing$

BIRTHDAY BOOK

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

AddBirthday

$\Delta\, BirthdayBook$

$name? : NAME$

$date? : DATE$

$name? \notin known$

$known' = known \cup \{name?\}$ -> actual data nid put inside {}

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

BIRTHDAY BOOK

- The declaration Δ BirthdayBook indicates that the schema is describing a *state change*.
- The part of the schema below the line gives a pre-condition for the success of the operation.

BIRTHDAY BOOK

- Another operation might be to find the birthday of a person known to the system. Again we describe the operation with a schema.

FindBirthday

$\Xi BirthdayBook$

$name? : NAME$

$date! : DATE$

$name? \in known$ $\land$ name? $\in$ dom birthday (optional)

$date! = birthday\ (name?)$

$known' = known$

$birthday' = birthday$

BIRTHDAY BOOK

- The declaration Ξ BirthdayBook indicates that this is an operation in which the *state does not change* where all values are derived from BirthdayBook.

BIRTHDAY BOOK

- To remind who has birthday on the particular day so that birthday cards should be sent, remind state schema is specified:

Remind

$$\Xi\, BirthdayBook$$
$$today? : DATE$$
$$cards! : \mathbb{P}\, NAME$$

---

temp variable

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

- The Remind schema does not change the BirthdayBook.

BIRTHDAY BOOK

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

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

Success

$$result! : REPORT$$

$$result! = ok$$

AlreadyKnown

$$\Xi BirthdayBook$$
$$name? : NAME$$
$$result! : REPORT$$

$$name? \in known$$
$$result! = alreadyKnown$$

BIRTHDAY BOOK

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

NotKnown

$\Xi BirthdayBook$
$name? : NAME$
$result! : REPORT$

$name? \notin known$
$result! = notKnown$

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

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