1.1 Functions in Z

Function

- A function is special case of a relation in which there is at most one element in the range for each element in the domain. A function with a finite domain is also called as a mapping.
- In Z a function f from the typed set X to the typed set Y is declared by ‘f: $X \nrightarrow Y$’ and is pronounced ‘the function f, from X to Y’.
- $f: X \nrightarrow Y == f: X \rightarrow Y$ such that $\exists_1 y: Y \bullet x\ f\ y$ if $x \in \operatorname{dom} f$

Function application

- The application of the function f to the value X (called its argument) is written ‘f x’ and pronounced ‘ f of x’ or ‘f applied to x’.

1.2 Functions in Z

- **Partial functions**
  - In general, a function may be partial, i.e., there may be values of the source which are not in the domain of the function.
- **Total functions**
  - A total function is one where there is a value for every possible value of the source, so f x is always defined. The domain is the whole of the source.
  - A total function is declared by ‘f: X → Y’ and is pronounced ‘f is a total function from X to Y’. It is equivalent to the partial function X t Y with the restriction that dom f = X.

1.3 Functions in Z

- **Injection**
  - An injection, or injective function, or one to one, written as ‘f: X >+⟶ Y’, is a function which maps different values of the source on to different values of the target. An injection may be partial, f: X >+⟶ Y’, or total, f: X >⟶ Y’.
- **Surjection**
  - A surjection, or surjective function, or on-to, written as ‘f: X +⟶> Y’, is a function for which its range is the whole of its target. A surjection may be partial, ‘f: X +⟶> Y’, or total, f: ‘f: X ⟶> Y’.

1.5 Functions in Z

- **Bijection**
  - A bijection, or bijective function(one-to-one correspondence), written as ‘f: X >⟶> Y’, is a function which maps every element if the source on to every element of the target in a one-to-one relationship. Therefore, it is injective, surjective, and total.

1.6 Functions in Z

Overriding

- A function can be modified so that for a particular set of values of the domain it has new values in the range. This is called overriding.
- For the function $f: X \mathrel{+\!\!\longrightarrow} Y$ and function $g: X \mathrel{+\!\!\longrightarrow} Y$, f overridden by g is a function, written as $f \oplus g$, is defined as:

  $f \oplus g == (\text{dom } g <\!\!+ f) \cup g$

  if $x \in \text{dom } f \wedge x \notin \text{dom } g$ then $f \oplus g\ x = f\ x$

  if $x \in \text{dom } g$ then $f \oplus g\ x = g\ x$
- Note: if $\text{dom } f \cap \text{dom } g = \Phi$ then $f \oplus g = f \cup g$

1.7 Example: Using Functions to Describe stock Control

[ITEM] the set of all kinds of items (item codes)

Warehouse

carried: $\mathbb{P}$ITEM

level: ITEM $\mathrel{+\!\!\!\longrightarrow} \mathbb{N}$

---

dom level = carried

Init

Warehouse’

---

carried’ = Φ

dom level’ = Φ

1.7 Example: Using Functions to Describe stock Control

CarryNewItem

ΔWarehouse
$i? : ITEM$

$i? \notin carried$
$level' = level \cup \{(i?, 0)\}$
$carried' = carried \cup \{i?\}$

Deliver

ΔWarehouse
$i? : ITEM$
$qty?: \mathbb{N}_1$

$i? \in carried$
$level' = level \oplus \{(i?, (level\ i? + qty?))\}$
$carried' = carried$

1.7 Example: Using Functions to Describe stock Control

Withdraw

ΔWarehouse
i? : ITEM
qty?: N_1

i? $\in$ **carried**
level i?≥ qty?
level' = level $\oplus$ **{(i?, (level i? - qty?))}**
carried' = carried

1.7 Example: Using Functions to Describe stock Control

DiscontinueItem

ΔWarehouse
$i? : ITEM$

$i? \in carried$
$level\ i? = 0$
$carried' = carried \setminus \{i?\}$
$level' = \{i?\} \lessdot level$

1.8 Example: A Seat Allocation System

[PERSON] **the set of all possible uniquely identified persons**

[SEAT] **the set of all seats on the aircraft**

REPLY ::= yes | no

RESPONSE ::= OK | alreadyBooked | notYours

Seating

bookedTo: SEAT +⟶ PERSON

Init

Seating'

---

bookedTo' = $\boldsymbol{\Phi}$

1.8 Example: A Seat Allocation System

$Book_0$

Δ Seating
p? : PERSON
s?: SEAT

$s? \notin \operatorname{dom} bookedTo$
$bookedTo' = bookedTo \cup \{(s?, p?)\}$

$Cancel_0$

ΔSeating
p? : PERSON
s?: SEAT

$(s?, p?) \in bookedTo$
$bookedTo' = bookedTo \setminus \{(s?, p?)\}$

1.8 Example: A Seat Allocation System

WhoseSeat

Ξ Seating
s?: SEAT
taken!: REPLY
who!: PERSON

$(s? \in \operatorname{dom} bookedTo \land taken! = yes \land who! = bookedTo (\!| s? |\!))$
$\lor$
$(s? \notin \operatorname{dom} bookedTo \land taken! = no)$

1.8 Example: A Seat Allocation System

OKMessage == [rep!: RESPONSE | rep! = OK]

BookError
Ξ Seating
s?: SEAT
p?: PERSON
rep!: RESPONSE

(s? $\in$ dom bookedTo
rep! = alreadyBooked

Book == (Book$_0$ $\wedge$ OKMessage) $\vee$ BookError

1.8 Example: A Seat Allocation System

CancelError

Ξ Seating
s?: SEAT
p?: PERSON
rep!: RESPONSE

$(s?, p?) \notin bookedTo$
rep! = notYours

$$Cancel == (Cancel_0 \wedge OKMessage) \vee CancelError$$