CHAPTER 7

Examples of Z specification documents

7.1 Introduction

We now re-express the aircraft example of Chapter 3 as a Z specification using schemas. This specification concerns recording the passengers aboard an aircraft. There are no seat numbers; passengers are allowed aboard on a first-come-first-served basis.

7.2 The types

The only *basic type* involved here is the set of all possible persons, *PERSON*:

[PERSON] the set of all possible uniquely identified persons

The aircraft has a fixed capacity:

capacity: $\mathbb{N}$

7.3 The state

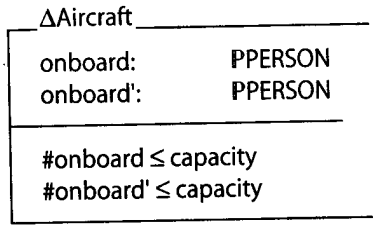
The state of the system is given by the set of persons on board the aircraft. The number of persons on board must never exceed the capacity. This is the state's *invariant* property.

Aircraft

$onboard : \mathbb{P} PERSON$

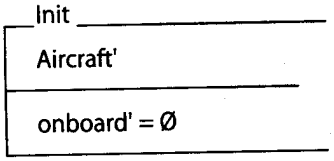
$\#onboard \leq capacity$

The state before and after an operation is described by the schema $\Delta Aircraft$, which has its conventional meaning:



7.4 Initialisation operation

There must be an initial state for the system. The obvious one is where the aircraft is empty. A suitable initialisation operation sets a new value to the variable:

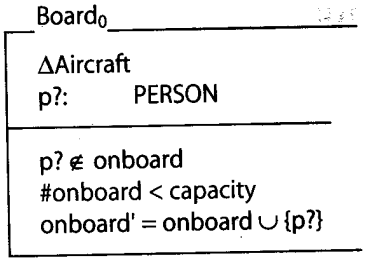


The initialised state must satisfy the state's invariant property. This it clearly does, since the size of the empty set is zero, which is less than or equal to all natural numbers and so to all possible values of *capacity*.

7.5 Operations

7.5.1 Boarding

There must be an operation to allow a person *p?* to board the aircraft. A first version of this is called $Board_0$:



7.5.2 Disembarking

It is also necessary to have an operation to allow a person *p?* to disembark from the aircraft. A first version of this is $Disembark_0$:

Disembark$_0$

ΔAircraft
p?: PERSON

---

p? $\in$ onboard
onboard' = onboard \ {p?}

7.6 Enquiry operations

These operations leave the state unchanged and therefore use the schema:

ΞAircraft

which has its (automatic) conventional meaning:

ΞAircraft

onboard: PPERSON
onboard': PPERSON

#onboard $\leq$ capacity
#onboard' $\leq$ capacity
onboard = onboard'

7.6.1 Number on board

In addition to operations which change the state of the system it is necessary to have an operation to discover the number of persons on board:

Number

ΞAircraft
numOnboard!: $\mathbb{N}$

---

numOnboard! = #onboard

7.6.2 Person on board

Furthermore, a useful enquiry is to discover whether or not a given person *p* ? is on board. The data type *YESORNO* is defined to provide suitable values for the reply and is used in the schema *OnBoard*:

YESORNO ::= yes | no

OnBoard

ΞAircraft
p?: PERSON
reply!: YESORNO

(p? ∈ onboard ∧ reply! = yes)
∨
(p? ∉ onboard ∧ reply! = no)

7.7 Dealing with errors

The schemas $Board_0$ and $Disembark_0$ do not state what happens if their preconditions are not satisfied. The schema calculus of Z allows these schemas to be extended. First we define a small schema *OKMessage* to give the reply *OK* in the event of success:

RESPONSE ::=
OK | twoErrors | onBoard | full | notOnBoard
OKMessage == [rep!: RESPONSE | rep! = OK]

7.7.1 Boarding

A schema to handle errors *BoardError* is defined. It causes no change to the value of *onboard*, so the schema *ΞAircraft* is used:

BoardError

ΞAircraft
p?: PERSON
rep!: RESPONSE

(p? ∈ onboard ∧
#onboard = capacity ∧
rep! = twoErrors)
∨
(p? ∈ onboard ∧
#onboard < capacity ∧
rep! = onBoard)
∨
(p? ∉ onboard
∧
#onboard = capacity ∧
rep! = full)

Finally, *Board* can be defined:

$$Board == (Board_0 \land OKMessage) \lor BoardError$$

7.7.2 Disembark

DisembarkError

$\Xi Aircraft$
$p?: \quad PERSON$
$rep!: \quad RESPONSE$

$p? \notin onboard \land rep! = notOnBoard$

Finally *Disembark* can be defined:

$$Disembark == (Disembark_0 \land OKMessage) \lor DisembarkError$$

7.8 Example of schemas: Student Programme of Modules

7.8.1 Introduction

This specification concerns a student on a modular course. The student chooses modules from those offered and constructs a *programme* by *adding* and *deleting modules*. The programme is *viable* if it fulfils certain conditions. At least one viable programme must exist.

7.8.2 Types

[MODULE] the set of all possible modules (module identifications)

7.8.3 Sets

$offered, advanced, basic,$
$field1acc, field2acc: \mathbb{P} MODULE$

$advanced \cap basic = \varnothing$
$advanced \cup basic = offered$
$field1acc \subseteq offered$
$field2acc \subseteq offered$
$\#offered \geq 18$
$\#(field1acc \cap advanced) \geq 7$
$\#(field2acc \cap advanced) \geq 7$
$\#((field1acc \cup field2acc) \cap advanced) \geq 16$