CHAPTER 16

Example specifications

16.1 The Airport example

The air-traffic control of an airport keeps a record of the *planes waiting* to land and the *assignment* of planes to *gates* on the ground. There are operations to accept a plane when it *arrives* in the airport's waiting space, to *assign* a plane to a gate at the airport and to record that a plane *leaves* its gate.

16.2 The types

The types used in this formal specification are:

[PLANE]	the set of all possible, uniquely identified planes
[GATE]	the set of all gates at this airport

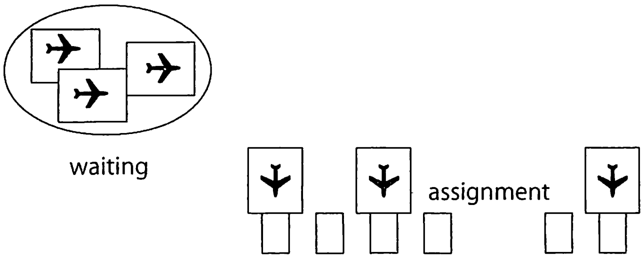
16.3 The state

The state of the *airport*, at any time, can be expressed by this Z *schema*:

```
AIRPORT
waiting:        ℙPLANE
assignment:     GATE ⤔ PLANE
─────────────────────────────
waiting ∩ ran assignment = ∅
```

Each plane is assigned to at most one gate and each gate has at most one plane assigned to it; so the assignment of planes to gates is an *injective* (one-to-one) *function* from gate to plane.

Figure 16.1



The planes waiting are a *set* of planes (they are in no particular order). No plane is both waiting and assigned to a gate.

16.4 The initialisation operation

Initially there are no planes waiting or at any gate.

```
__Init__________________
| AIRPORT'
|_______________________
| waiting' = ∅
| assignment' = ∅
|_______________________
```

The schema has access to the variables of the schema *AIRPORT*. The set *waiting* is the *empty* set. The function *assignment* is an *empty* function.

16.5 The operations

16.5.1 Arrive

The operation *Arrive* records the arrival of a plane *p?* in the airport's waiting area:

```
__Arrive______________________________
| p?:        PLANE
| ΔAIRPORT
|_____________________________________
| p? ∉ (waiting ∪ ran assignment)
| waiting' = waiting ∪ {p?}
| assignment' = assignment
|_____________________________________
```

The plane must be neither already waiting nor assigned to a gate. The schema has access to the before and after (') values of the schema *AIRPORT*. The new value of the set *waiting* is the same as before but with *p?* added. The new value of the function *assignment* is the same as the old value of *assignment*.

16.5.2 Assign

The operation *Assign* records the assignment of a plane *p?* to a free gate *g?*:

Assign

$p?: \quad PLANE$
$g?: \quad GATE$
$\Delta AIRPORT$

$p? \in waiting$
$g? \notin \mathrm{dom}\ assignment$
$assignment' = assignment \cup \{g? \mapsto p?\}$
$waiting' = waiting \setminus \{p?\}$

The plane must be waiting and the gate must be free. The pairing between gate *g?* and plane *p?* is added to *assignment*. Plane *p?* is removed from *waiting*.

16.5.3 Leave

The operation *Leave* records the plane *p?* leaving its gate:

Leave

$p?: \quad PLANE$
$\Delta AIRPORT$

$p? \in \mathrm{ran}\ assignment$
$assignment' = assignment \rhd \{p?\}$
$waiting' = waiting$

The plane *p?* must be assigned to a gate. The assignment for plane *p?* is removed. The waiting planes are unaffected.

16.6 Handling errors

So far we have indicated the preconditions for successful operations, but have not said what will happen if these preconditions are not satisfied. We can do this by using separate error-handling schemas.

Firstly we introduce a type for a result message:

$RESULT ::= OK \mid full \mid badAircraft \mid notWaiting \mid gateNotFree \mid notAtGate$

16.7 Error-handling operations

We introduce a further, output (!), parameter, *reply!*, to each error schema; the Ξ in the schema name *ΞAIRPORT* signifies that there will be no change to the state of the airport.

16.7.1 ArriveErr

This schema handles the cases where the preconditions of *Arrive* are not satisfied:

```
ArriveErr
p?:          PLANE
reply!:      RESULT
ΞAIRPORT
────────────────────────────────────────
#waiting  = limit ∧ reply! = full
∨
p? ∈ (waiting ∪ ran assignment) ∧ reply! = badAircraft
```

16.7.2 AssignErr

The operation *AssignErr* handles the cases where the preconditions of *Assign* are not satisfied:

```
AssignErr
p?:          PLANE
g?:          GATE
reply!:      RESULT
ΞAIRPORT
────────────────────────────────────────
p? ∉ waiting ∧ reply! = notWaiting
∨
g? ∈ dom assignment ∧ reply! = gateNotFree
```

16.7.3 LeaveErr

The operation *LeaveErr* handles the cases where the preconditions of *Leave* are not satisfied:

```
LeaveErr
p?: PLANE
reply!: RESULT
ΞAIRPORT
────────────────────────────────────────
p? ∉ ran assignment ∧ reply! = notAtGate
```

16.8 Fully specified operations

Finally, we use *schema calculus* to put together the schemas to give fully specified operations.

We start with a small schema *OKMessage* that simply produces the reply *OK*:

OKMessage == [reply!: RESULT | reply! = OK]

Then we combine schemas using schema operators.

16.9 The operations

16.9.1 Arrive

The operation *Arrive* records the arrival of a plane *p?* in the airport's waiting area:

$\text{Arrive} == \text{Arrive}_0 \wedge \text{OKMessage} \vee \text{ArriveErr}$

The *Arrive* operation behaves either like the *Arrive*$_0$ operation conjoined with the *OKMessage* operation, or like the *ArriveErr* operation.

16.9.2 Assign

The operation *Assign* records the assignment of a plane *p?* to a free gate *g?*:

$\text{Assign} == \text{Assign}_0 \wedge \text{OKMessage} \vee \text{AssignErr}$

The *Assign* operation behaves either like the *Assign*$_0$ operation conjoined with the *OKMessage* operation, or like the *AssignErr* operation.

16.9.3 Leave$_0$

The operation *Leave* records the plane *p?* leaving its gate:

$\text{Leave} == \text{Leave}_0 \wedge \text{OKMessage} \vee \text{LeaveErr}$

The *Leave* operation behaves either like the *Leave*$_0$ operation conjoined with the *OKMessage* operation, or like the *LeaveErr* operation.

16.10 The Library example

A library has a stock of books which may be taken out by its registered members.

16.11 Types

[BOOK]	the set of all possible uniquely identified books
[PERSON]	the set of all possible persons

16.12 State

LIB_0

$\mathit{stock}: \mathbb{P}\,\mathrm{BOOK}$

$\mathit{members}: \mathbb{P}\,\mathrm{PERSON}$

$\mathit{outTo}: \mathrm{BOOK} \rightarrowtail\!\!\!\!\!+ \ \mathrm{PERSON}$

$\mathrm{dom}\ \mathit{outTo} \subseteq \mathit{stock}$

$\mathrm{ran}\ \mathit{outTo} \subseteq \mathit{members}$

Only books which are in the library's *stock* can be recorded as *out to* a *member*. Only registered members may take out books.

16.13 Initialisation operation

Initially the library has no stock of books and no members and no books are recorded as out to members.

Init_0

$\mathrm{LIB}_0{}'$

$\mathit{stock}' = \varnothing$

$\mathit{members}' = \varnothing$

$\mathit{outTo}' = \varnothing$

16.14 Operations

16.14.1 Acquire book

The book must not already belong to the library's stock. It is added to the stock. The members remain unchanged.

$$
\begin{array}{|ll}
\multicolumn{2}{l}{\text{Acquire}_0} \\ \hline
\Delta\text{LIB}_0 & \\
\text{b?:} & \text{BOOK} \\ \hline
\multicolumn{2}{|l}{\text{b?} \notin \text{stock}} \\
\multicolumn{2}{|l}{\text{stock}' = \text{stock} \cup \{\text{b?}\}} \\
\multicolumn{2}{|l}{\text{members}' = \text{members}} \\
\multicolumn{2}{|l}{\text{outTo}' = \text{outTo}} \\ \hline
\end{array}
$$

16.14.2 Register member

The person must not already be a member. The person becomes a member. The stock remains unchanged.

$$
\begin{array}{|ll}
\multicolumn{2}{l}{\text{Register}_0} \\ \hline
\Delta\text{LIB}_0 & \\
\text{p?:} & \text{PERSON} \\ \hline
\multicolumn{2}{|l}{\text{p?} \notin \text{members}} \\
\multicolumn{2}{|l}{\text{members}' = \text{members} \cup \{\text{p?}\}} \\
\multicolumn{2}{|l}{\text{stock}' = \text{stock}} \\
\multicolumn{2}{|l}{\text{outTo}' = \text{outTo}} \\ \hline
\end{array}
$$

16.14.3 Take a book out

The person must be a member. The book must be part of the library's stock and must not be out to anyone. The book becomes recorded as out to the member. The members and stock are unchanged.

$$
\begin{array}{|ll}
\multicolumn{2}{l}{\text{TakeOut}_0} \\ \hline
\Delta\text{LIB}_0 & \\
\text{p?:} & \text{PERSON} \\
\text{b?:} & \text{BOOK} \\ \hline
\multicolumn{2}{|l}{\text{p?} \in \text{members}} \\
\multicolumn{2}{|l}{\text{b?} \in \text{stock} \setminus \text{dom outTo}} \\
\multicolumn{2}{|l}{\text{outTo}' = \text{outTo} \cup \{\text{b?} \mapsto \text{p?}\}} \\
\multicolumn{2}{|l}{\text{member}' = \text{member}} \\
\multicolumn{2}{|l}{\text{stock}' = \text{stock}} \\ \hline
\end{array}
$$

16.14.4 Bring back

The book must be recorded as out. The reference to the book being out is removed. The members and stock are unchanged.

$$
\begin{array}{|l}
\hline
\text{BringBack}_0 \\
\Delta \text{LIB}_0 \\
b?: \quad \text{BOOK} \\
\hline
b? \in \mathrm{dom}\ outTo \\
outTo' = \{b?\} \mathbin{⩤} outTo \\
member' = member \\
stock' = stock \\
\hline
\end{array}
$$

16.14.5 Dispose of a book

The book must belong to the library and not be out to a member. The book is removed from the stock. The members and the record of what books are out are unchanged.

$$
\begin{array}{|l}
\hline
\text{Dispose}_0 \\
\Delta \text{LIB}_0 \\
b?: \quad \text{BOOK} \\
\hline
b? \in stock \setminus \mathrm{dom}\ outTo \\
stock' = stock \setminus \{b?\} \\
members' = members \\
outTo' = outTo \\
\hline
\end{array}
$$

16.14.6 De-register member

The person must be a member who has no books out. The person is removed from the membership. The stock and record of books out are unchanged.

$$
\begin{array}{|l}
\hline
\text{Deregister}_0 \\
\Delta \text{LIB}_0 \\
p?: \quad \text{PERSON} \\
\hline
p? \in members \setminus \mathrm{ran}\ outTo \\
members' = members \setminus \{p?\} \\
stock' = stock \\
outTo' = outTo \\
\hline
\end{array}
$$

16.15 Extension 1: Limit on number of books out

There is a *limit* to the number of books a member may take out.

$$\begin{array}{|l}
limit: \mathbb{N} \\ \hline
limit \in 1..10
\end{array}$$

The limit lies between 1 and 10. It will be the same for all members.

16.16 The state

The state is modified to reflect the fact that no member may ever have taken out more books than the limit.

$$\begin{array}{|l}
\underline{\quad LIB_1 \qquad\qquad\qquad\qquad} \\
LIB_0 \\ \hline
\forall p: PERSON \mid p \in members \bullet \\
\#(outTo \rhd \{p\}) \leq limit \\ \hline
\end{array}$$

For each person p who is a member, the size of the *outTo* relation range restricted to the set containing just the person p (the number of books that are out to person p) must not exceed the limit.

The schema LIB_1 can be expanded to:

$$\begin{array}{|ll}
\multicolumn{2}{l}{\underline{\quad LIB_1 \qquad\qquad\qquad\qquad}} \\
stock: & \mathbb{P}BOOK \\
members: & \mathbb{P}PERSON \\
outTo: & BOOK \nrightarrow PERSON \\ \hline
\multicolumn{2}{|l}{dom\ outTo \subseteq stock} \\
\multicolumn{2}{|l}{ran\ outTo \subseteq members} \\
\multicolumn{2}{|l}{\forall p: PERSON \mid p \in members \bullet} \\
\multicolumn{2}{|l}{\quad \#(outTo \rhd \{p\}) \leq limit} \\ \hline
\end{array}$$

16.17 Initialisation operation

The new initialisation operation is identical to the previous one, except that it applies to the extended state schema.

$$Init_1 == Init_0\ [LIB_1'/LIB_0']$$

The initial state satisfies the new state invariant.
The schema *Init*$_1$ can be expanded to:

$$\begin{array}{|l}
\text{Init}_1 \\
\hline
\text{LIB}_1' \\
\hline
\text{stock}' = \varnothing \\
\text{members}' = \varnothing \\
\text{outTo}' = \varnothing \\
\hline
\end{array}$$

16.18 Operations

The operations of acquiring a book, registering a member, disposing of a book, de-registering a member and bringing back a book, are all unaffected by the new requirement:

$\text{Acquire}_1 == \text{Acquire}_0\ [\Delta\text{LIB}_1/\Delta\text{LIB}_0]$

$\text{Register}_1 == \text{Register}_0\ [\Delta\text{LIB}_1/\Delta\text{LIBO}]$

$\text{Dispose}_1 == \text{Dispose}_0\ [\Delta\text{LIB}_1/\Delta\text{LIB}_0]$

$\text{Deregister}_1 == \text{Deregister}_0\ [\Delta\text{LIB}_1/\Delta\text{LIB}_0]$

$\text{BringBack}_1 == \text{BringBack}_0\ [\Delta\text{LIB}_1/\Delta\text{LIB}_0]$

These new operations are identical to the previous ones, except that they apply to the new state schema.

16.18.1 Taking a book out

When a member attempts to take a book out, there is a check that the limit would not be exceeded.

$$\begin{array}{|l}
\text{TakeOut}_1 \\
\hline
\text{TakeOut}_0\ [\Delta\text{LIB}_1/\Delta\text{LIB}_0] \\
\hline
\#(\text{outTo} \rhd \{p?\}) < \text{limit} \\
\hline
\end{array}$$

For the person *p?* the size of the *outTo* relation range restricted to the set containing just the person *p?* (the number of books that are out to person *p?*) must not yet have reached the limit.

16.19 Extension 2: Books out for a limited period

We now consider a library where the books may be borrowed for limited number of days. There is no charge for taking out books, but a fine is payable for late return.

16.20 Types

We introduce the type *dates.* We will represent *money* as whole numbers of currency units, possibly negative:

[DATE] the set of all possible dates

$\text{MONEY} == \mathbb{Z}$

All members may take out books for the same *period* of days.

$$\begin{array}{|l}
\text{period}: \mathbb{N}
\end{array}$$

Books returned late incur the same *fine* each day.

$$\begin{array}{|l}
\text{fine}: \text{MONEY} \\ \hline
\text{fine} \geq 0
\end{array}$$

16.21 The state

The *date* that each book was taken *out* is recorded. The amount that each member owes (even if nothing) is recorded.

$$\begin{array}{|ll}
\multicolumn{2}{l}{\text{LIB}_2} \\ \hline
\text{LIB}_1 & \\
\text{dateOut}: & \text{BOOK} ⇸ \text{DATE} \\
\text{owes}: & \text{PERSON} ⇸ \text{MONEY} \\ \hline
\multicolumn{2}{|l}{\text{dom dateOut} = \text{dom outTo}} \\
\multicolumn{2}{|l}{\text{dom owes} = \text{member}} \\ \hline
\end{array}$$

16.22 Initialisation operation

The initial state is extended to show that there is no information recorded about dates that books were taken out or about money owing:

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{Init}_2} \\ \hline
\text{Init}_1\ [\text{LIB}_2/\text{LIB}_1] \\ \hline
\text{dateOut}' = \varnothing \\
\text{owes}' = \varnothing \\ \hline
\end{array}$$

16.23 Operations

The operations to acquire a book and to dispose of a book are unchanged.

$\text{Acquire}_2 == \text{Acquire}_1\ [\Delta\text{LIB}_2/\Delta\text{LIB}_1]$

$\text{Dispose}_2 == \text{Dispose}_1\ [\Delta\text{LIB}_2/\Delta\text{LIB}_1]$

16.23.1 Register member

A new member owes nothing.

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{Register}_2} \\ \hline
\text{Register}_1\ [\Delta\text{LIB}_2/\Delta\text{LIB}_1] \\ \hline
\text{owes}' = \text{owes} \cup \{p? \mapsto 0\} \\
\text{dateOut}' = \text{dateOut} \\ \hline
\end{array}$$

16.23.2 De-register member

To be de-registered the member must not owe anything or be in credit.

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{Deregister}_2} \\ \hline
\text{Deregister}_1\ [\Delta\text{LIB}_2/\Delta\text{LIB}_1] \\ \hline
\text{owes}\ p? = 0 \\
\text{dateOut}' = \text{dateOut} \\ \hline
\end{array}$$

16.23.3 Taking a book out

The date that the book is taken out is recorded.

$$
\begin{array}{l}
\text{TakeOut}_2 \\
\hline
\text{TakeOut}_1\ [\Delta\text{LIB}_2/\Delta\text{LIB}_1] \\
\text{d?:}\quad \text{DATE} \\
\hline
\text{dateOut}' = \text{dateOut} \cup \{\text{b?} \mapsto \text{d?}\} \\
\text{owes}' = \text{owes} \\
\hline
\end{array}
$$

16.23.4 Bring back

We specify that there should be a function *Diff*(*d1*, *d2*) that returns the number of working days that *d2* is later than *d1*:

$$
\text{Diff: DATE} \times \text{DATE} \rightarrow \mathbb{Z}
$$

If the book is returned within the period then there is no change in what the member owes. If the book is late then the member's debt is increased by a fixed fine for each day late. This is calcutaed only when the book is brought back:

$$
\begin{array}{l}
\text{BringBack}_2 \\
\hline
\text{BringBack}_1\ [\Delta\text{LIB}_2/\Delta\text{LIB}_1] \\
\text{today?:}\quad \text{DATE} \\
\hline
\text{dateOut}' = \{\text{b?}\} \lhd \text{dateOut} \wedge \\
(\ \text{Diff ((dateOut b?) today?)} \leq \text{period} \wedge \\
\quad \text{owes}' = \text{owes}) \\
\vee \\
(\text{Diff ((dateOut b?) today?)} > \text{period} \wedge \\
\quad \text{owes}' = \text{owes} \oplus \\
\quad \{\text{outTo b?} \mapsto \text{owes outTo b?} + \\
\qquad (\text{Diff ((dateOut b?) today?)} - \text{period}) * \text{fine}\}) \\
\hline
\end{array}
$$

The date that the book was taken out is removed from the *dateOut* function.

If the difference between the date when the book was taken out and today's date is within the period, then the amount the person owes stays the same.

If the difference between the date the book was taken out and today's date exceeds the period, then the amount owed by the person who took the book out is increased by the fine amount for each day over the period.

16.23.5 PayIn

A member may *pay in* an *amount* of money (positive) to offset current or future money owed. Everything else remains unchanged.

PayIn_2

ΔLIB_2
ΞLIB_1
p?: PERSON
amount?: MONEY

$p? \in \text{members}$
$\text{amount?} > 0$
$\text{owes}' = \text{owes} \cup \{p? \mapsto \text{owes } p? - \text{amount?}\}$
$\text{dateOut}' = \text{dateOut}$

16.24 Extension 3: Reservations

We now extend the library system to allow members to reserve titles.

16.25 Types

A member will wish to reserve a *title*, rather than a particular *book*.

[TITLE]	the set of all possible book titles

16.26 The state

The *title* is known of every book in stock. *Reservations* are allowed for every title for which there is a book in stock. Only members may reserve *titles* and they can only reserve each title once. A book which belongs to the library and which is not out may be *held for* a member (who has previously reserved it and will later collect it).

LIB_3

LIB_3
title: BOOK ⇸ TITLE
reserved: TITLE ⇸ iseq PERSON
heldFor: BOOK ⇸ PERSON

$\text{dom title} = \text{stock}$
$\text{dom reserved} = \text{ran title}$
$(\forall t: \text{TITLE} \mid t \in \text{dom reserved} \bullet$
$\quad \text{ran (reserved } t) \subseteq \text{members})$
$\text{dom heldFor} \subseteq \text{stock} \setminus \text{dom outTo}$
$\text{ran heldFor} \subseteq \text{members}$

16.27 Initialisation operation

Initially there are no titles recorded, no titles reserved and no books held for members.

$$
\begin{array}{|l}
\text{Init}_3 \\
\hline
\text{Init}_2\ [\text{LIB}_3'/\text{LIB}_2'] \\
\hline
\text{title}' = \emptyset \\
\text{reserved}' = \emptyset \\
\text{heldFor}' = \emptyset \\
\hline
\end{array}
$$

16.28 Operations

16.28.1 Acquire a book

The title of a book is recorded when it is acquired. If the title is new then an empty reservation list is set up for it, otherwise reservations are unchanged. Books held are unchanged.

$$
\begin{array}{|l}
\text{Acquire}_3 \\
\hline
\text{Acquire}_2\ [\Delta\text{LIB}_3/\Delta\text{LIB}_2] \\
\text{t?}: \quad \text{TITLE} \\
\hline
\text{title}' = \text{title} \cup \{\text{b?} \mapsto \text{t?}\} \\
(\text{t?} \notin \text{dom reserved} \wedge \\
\quad \text{reserved}' = \text{reserved} \cup \\
\{\text{t?} \mapsto \langle\rangle\}) \\
\vee \\
(\text{t?} \in \text{dom reserved} \wedge \\
\quad \text{reserved}' = \text{reserved}) \\
\text{heldFor}' = \text{heldFor} \\
\hline
\end{array}
$$

16.28.2 Reserve a title

The person reserving must be a member and must not already have reserved this title. The library must have a book of this title in stock.

Reserve_3

ΔLIB_3
ΞLIB_2
$p?: \quad \text{PERSON}$
$t?: \quad \text{TITLE}$

$p? \in \text{members}$
$p? \notin \text{ran}\ (\text{reserved}\ t?)$
$t? \in \text{ran title}$
$\text{reserved}' = \text{reserved} \oplus$
$\quad \{t? \mapsto \text{reserved}\ t? \frown \langle p? \rangle\}$
$\text{title}' = \text{title}$
$\text{heldFor}' = \text{heldFor}$