CHAPTER 11

A seat allocation system

11.1 Introduction

This example concerns recording the allocation of seats to passengers on an aircraft.

11.2 The types

The types involved here are the set of *all possible* persons, called *PERSON*, and the set of all seats *on this aircraft*, *SEAT*.

[PERSON]	the set of all possible uniquely identified persons
[SEAT]	the set of all seats on this aircraft

11.3 The state

The state of the system is given by the relation between the set of seats and the set of persons:

```
┌─Seating───────────────────────────
│ bookedTo:        SEAT ⇸ PERSON
└───────────────────────────────────
```

Note that a seat may be booked to only one person, but a person may book many seats.

11.4 Initialisation operation

There must be an initial state for the system. The obvious one is where the aircraft is entirely unbooked. An initialisation operation is:

```
┌─Init──────────────────
│ Seating'
├───────────────────
│ bookedTo' = Ø
└──────────────────────
```

11.5 Operations

11.5.1 Booking

There is an operation to allow a person *p?*, to book the seat *s?*. A first attempt is:

```
Book0
ΔSeating
p?:     PERSON
s?:     SEAT
--------------------------
s? ∉ dom bookedTo
bookedTo' = bookedTo ∪ {s? ↦ p?}
```

11.5.2 Cancel

It is also necessary to have an operation to allow a person *p?* to cancel a booking for a seat *s?*:

```
Cancel0
ΔSeating
p?:     PERSON
s?:     SEAT
--------------------------
s? ↦ p? ∈ bookedTo
bookedTo' = bookedTo \ { s? ↦p?}
```

11.6 Enquiry operations

These operations leave the state unchanged.

11.6.1 Whose seat

In addition to operations which change the state of the system it is necessary to have an operation to discover the owner of a seat:

REPLY ::= yes | no

WhoseSeat

ΞSeating
s?: SEAT
taken!: REPLY
who!: PERSON

(s? ∈ dom bookedTo ∧
taken! = yes ∧
who! = bookedTo s?)
∨
(s? ∉ dom bookedTo ∧
taken! = no)

11.7 Dealing with errors

The schemas $Book_0$ and $Cancel_0$ do not state what happens if their preconditions are not satisfied. The schema calculus of Z allows these schemas to be extended, firstly to give a message in the event of success:

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

11.7.1 Booking

A schema to handle errors on making a booking is defined:

BookError

ΞSeating
s?: SEAT
p?: PERSON
rep!: RESPONSE

---

s? ∈ bookedTo
rep! = alreadyBooked

Finally, *Book* can be defined:

Book == ($Book_0$ ∧ OKMessage) ∨ BookError

11.7.2 Cancel

A schema to handle errors on cancelling a booking is defined:

```
__CancelError______________
| ΞSeating
| s?:      SEAT
| p?:      PERSON
| rep!:    RESPONSE
|__________________________
| s? ↦ p? ∉ bookedTo
| rep! = notYours
|__________________________
```

Finally *Cancel* can be defined:

$$\text{Cancel} == (\text{Cancel}_0 \wedge \text{OKMessage}) \vee \text{CancelError}$$

EXERCISES

By using schema inclusion where possible, extend your specification of the computer system from Question 1, Chapter 2, and onwards, to include security passwords. Each registered user must have a password. On being registered a user is given a dummy password. Use the declarations:

PASSWORD the set of all possible passwords
dummy: PASSWORD

1. Give a schema to define the state of this system.
2. Give an initialisation operation for the system.
3. Give a schema for the operation to register a new user.
4. Give a schema for the operation to log in. The user must give the correct password. Consider only the case where the user gives the correct password.
5. Give a schema for a logged-in user to change the password. The user must supply the new and the old passwords. Define only the case where the conditions are met.