CHAPTER 14

Extending a specification

14.1 Limitations of specifications given so far

The most complex specification given so far in this book concerns booking of seats on an aircraft. This example is unrealistic in that it only considers one flight of one aircraft. It is wise to start a specification in this way by simplifying the problem, but naturally it would be useful to extend this specification to consider many flights.

14.2 The type of a flight

The new type *FLIGHT* will be introduced for this extended specification.

[FLIGHT] the set of all flight identifications

If the flight identification is composed of a date and a flight number it could be declared as:

[DATE] the set of all dates
[FLIGHTNO] the set of all flight numbers

$$FLIGHT == DATE \times FLIGHTNO$$

The twin equal signs mean abbreviation definition.

14.3 A function to a function

The behaviour of the extended system of seat bookings across a fleet of aircraft is to be the same for each flight, so the new seat booking information can conveniently be represented by a function from flight to seat bookings for that flight. The seat bookings for a flight will be, as before, a function from seat to person.

[SEAT] the set of all seat numbers
[PERSON] the set of all persons

The state schema will use the function *booked*:

$$booked: FLIGHT \pfun (SEAT \pfun PERSON)$$

Note that for any flight f

$$booked\ f$$

is a function from *SEAT* to *PERSON*, as in the previous, simpler, version of this specification and, for any seat s

$$booked\ f\ s$$

is the application of the function from *SEAT* to *PERSON* yielded by the application of the function *booked* to the flight f, and is thus the person to whom the seat s on flight f is booked.

14.3.1 Seats and aircraft

It is no longer sufficient to consider the set *SEAT* to be the set of seats on this aircraft, since there are now several aircraft involved and the available seat numbers will vary according to the aircraft type. For this reason *SEAT* has been declared as the set of *all* seat *numbers* and a new function *hasSeat* is used to discover what seat numbers a flight has assigned to it.

The relating of a seat to a flight rather than to a particular aircraft reflects the fact that the same seat number on a given aircraft can be booked many times in its lifetime, but only once for a given flight of that aircraft.

14.4 The state

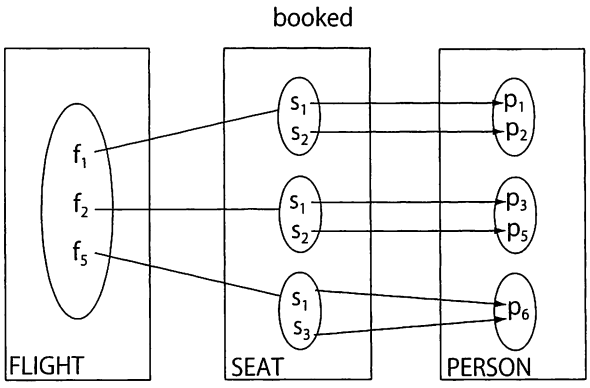
Only the seat numbers assigned to a flight may be booked.

FleetSeatAllocation

$$booked: FLIGHT \pfun (SEAT \pfun PERSON)$$
$$\_hasSeat\_: FLIGHT \rel SEAT$$

$$dom\ booked = dom\ hasSeat$$
$$\forall f: FLIGHT \mid f \in dom\ booked \bullet$$
$$dom\ (booked\ f) \subseteq hasSeat\ (\{f\})$$

Figure 14.1



14.5 Initialisation operation

A possible initial state is where there are no bookings for any seats on any flights.

Init

FleetSeatAllocation'

$$hasSeat' = \emptyset$$
$$booked' = \emptyset$$

14.6 Operations

14.6.1 Creating a new flight

An operation to create a new flight is

$NewFlight_0$

$\Delta FleetSeatAllocation$

f?: FLIGHT

available?: $\mathbb{P}$SEAT

$$f? \notin \mathrm{dom}\ booked$$
$$available? \neq \emptyset$$
$$hasSeat' = hasSeat \cup \{s: SEAT \mid s \in available? \bullet f? \mapsto s\}$$
$$booked' = booked \cup \{f? \mapsto \emptyset\}$$

The flight must not already be known. The set of seats available to that flight must not be empty. The seat bookings for the new flight are initially empty.

14.6.2 Booking a seat on a flight

An operation to book a seat on a particular flight needs to have flight, seat number and person as input.

$Book_0$

$\Delta FleetSeatAllocation$
$f?: \quad FLIGHT$
$s?: \quad SEAT$
$p?: \quad PERSON$

$f? \in dom\ booked$
$f?\ hasSeat\ s?$
$s? \notin dom\ (booked\ f?)$
$booked' = booked \oplus \{f? \mapsto (booked\ f? \cup \{s? \mapsto p?\})\}$
$hasSeat' = hasSeat$

The flight must be known, the seat must be one that is assigned to this flight and the seat must not already be booked.

14.6.3 Cancelling a booking

An operation to cancel a booking needs to have flight, seat and person as input.

$Cancel_0$

$\Delta FleetSeatAllocation$
$f?: \quad FLIGHT$
$s?: \quad SEAT$
$p?: \quad PERSON$

$f? \in dom\ booked$
$f?\ hasSeat\ s?$
$s? \mapsto p? \in booked\ f?$
$booked' = booked \oplus \{f? \mapsto (booked\ f? \setminus \{s? \mapsto p?\})\}$
$hasSeat' = hasSeat$

The flight must be known, the seat must be assigned to this flight and the seat must be booked to the person.

14.7 Enquiry operations

14.7.1 Enquiry about a booking

An operation to enquire about a booking needs to have flight and seat as input:

REPLY ::= yes | no

Enquire

ΞFleetSeatAllocation
f?: FLIGHT
s?: SEAT
taken!: REPLY
who!: PERSON

---

f? ∈ dom booked
f? hasSeat s?
((s? ∈ dom booked f? ∧ taken! = yes ∧ who! = booked f? s?)
∨
(s? ∉ dom booked f? ∧ taken! = no))

The flight must be known and the seat must be assigned to this flight.

14.8 Error conditions

This extended example has made preconditions of operations explicit. Error handling schemas can be defined to handle the violation of pre-conditions by returning a result value, as in previous examples. This part has been omitted here since it introduces no new ideas.

14.9 Conclusions

This more realistic specification is much more complex than the earlier examples and is nonetheless for a fairly modest system. It is wise to specify a simple version of any problem first, before taking on the entire complexity of any problem.

EXERCISES

A company lets its apartments to known customers for fixed time-slots. At any given time an apartment may only be booked to one person. A system is required to handle bookings. From time-to-time apartments are acquired and given up.

For a formal specification of this system give the following:

1. Schemas to describe the state of the system and an initialisation operation.
2. A schema for the operation to add a new customer.
3. A schema for the operation to acquire a new apartment.
4. A schema for the operation to make a booking of a given apartment to a given customer at a given time.

5. A schema for an enquiry operation to show all apartments which are free at a given time.
6. The Swiss Air Rescue organisation, REGA, has a fleet of helicopters that operate within Switzerland. Part of a formal specification for a control system for the helicopters is given here:

[AIRCRAFT] — the set of all aircraft
[POINT] — the set of all coordinates

helis:	$\mathbb{P}$ AIRCRAFT
Switzerland:	$\mathbb{P}$ POINT
base:	AIRCRAFT $\rightarrow\!\!\!\!\!\!+$ POINT

dom base = helis
ran base $\subseteq$ Switzerland

The set *helis* is the set of aircraft (helicopters) operated by REGA. *Switzerland* is the set of points that are on Swiss territory. The function *base* gives the point that is the normal base for a given helicopter. Every REGA helicopter has a base and all REGA bases are in Switzerland.

The state of REGA's helicopters at any given time is given by the following: The functions *position, minsLeft* and *flightMins* respectively give the helicopter's current position, its remaining flying time in minutes and its flight time to a given point in minutes.

REGA

position:	AIRCRAFT $\rightarrow\!\!\!\!\!\!+$ POINT
minsLeft:	AIRCRAFT $\rightarrow\!\!\!\!\!\!+$ $\mathbb{N}$
flightMins:	AIRCRAFT $\rightarrow\!\!\!\!\!\!+$ (POINT $\rightarrow\!\!\!\!\!\!+$ $\mathbb{N}$)

dom position = dom flightMins = dom minsLeft = helis

Use the variables from the schema *REGA* in answering the following questions.

(a) Explain the meaning of each component of the predicate of the schema *REGA*.
(b) Write an expression that states that all REGA's helicopters are currently within Switzerland.
(c) Write an expression to state that any point in Switzerland can be reached by a REGA helicopter within 15 minutes.
(d) Write an expression to state that each REGA helicopter is currently at a REGA base.
(e) Write an expression to state that each REGA helicopter is currently at its own base.
(f) Write a expression to state that every REGA helicopter has enough flying time left to reach a REGA base.