CHAPTER 13

An example of sequences – the aircraft example again

13.1 Introduction

A system can be specified in terms of sequences. This chapter shows the aircraft example done in this fashion. Usually it is harder to use sequences; they are much less abstract than sets since order must be taken into account. A process called *refinement* concerns producing a more concrete specification from an abstract one. In so doing each transformation can be shown to be a correct implementation of its (more abstract) specification.

13.2 The state

The passengers' identifications are held in a sequence, which does not contain any name more than once:

SeqAircraft

$$\text{passengers} : \text{iseq PERSON}$$

$$\#\text{passengers} \leq \text{capacity}$$

The relationship (the *abstraction function, ABS*) between the abstract specification *Aircraft* and the more concrete *SeqAircraft* is given by:

ABS

Aircraft
SeqAircraft

$$\text{onboard} = \text{ran passengers}$$

13.3 Initialisation operation

The sequence is empty:

SeqInit

SeqAircraft'

passengers' = ⟨ ⟩

13.4 Operations

13.4.1 Boarding

The new person is appended to (the end of) the sequence:

SeqBoard

p?: PERSON
ΔSeqAircraft

p? ∉ ran passengers
#passengers < capacity
passengers' = passengers ⌢ ⟨p?⟩

13.4.2 Disembark

SeqDisembark

p?: PERSON
ΔSeqAircraft

p? ∈ ran passengers
(∃ before, after: iseq PERSON •
passengers = before ⌢ ⟨p?⟩ ⌢ after ∧
passengers' = before ⌢ after)

Note how much more complex this is than the set-based version; the element must be removed from the right place in the sequence.

13.5 Enquiry operations

13.5.1 Number on board

SeqNumber

numOnboard!: ℕ
ΞSeqAircraft

numOnboard! = #passengers

The number on board is the length of the sequence, since there are no duplicates.

13.5.2 Person on board

REPLY ::= yes | no

SeqOnBoard

p?: PERSON
rep!: REPLY
ΞSeqAircraft

(p? ∈ ran passengers ∧
rep = yes)
∨
(p? ∉ ran passengers ∧
rep = no)

13.6 Dealing with errors

For the sake of simplicity the sections on dealing with errors have been omitted here.

13.7 Implementation

Since a sequence is easily modelled in a programming language, either by the language's sequence type in the case of an applicative language, or by files or arrays in a procedural language, a sequence is closer to being implementable than, say, a set or a relation, and is thus regarded as being more concrete. In a complex specification it is best to start with a simpler, more abstract specification, in terms of sets, functions and so on, and to

refine this later to a more concrete, equivalent specification. This process of refinement is itself the subject of books.

EXERCISES

1. Define a schema for the state of a system which will maintain a *file* which is a sequence of *bytes*.
2. Give a schema for a suitable initialisation operation.
3. Give a schema for an operation to *insert* a sequence of bytes *after* a given position in the file.
4. Give a schema to *delete* the sequence of bytes within the file, given suitable starting and ending positions.
5. Give a schema to *copy* a sequence of bytes within the file, given the starting and ending positions, into an output *buffer*.