CHAPTER 3

Using sets to describe a system – a simple example

3.1 Introduction

With only the mathematics covered so far it is possible to describe a very simple system.

This example concerns recording the passengers aboard an aircraft. There are no seat numbers. Passengers are allowed aboard on a first-come-first-served basis. The aircraft has a fixed capacity which must never be exceeded.

The only *basic type* involved here is the set of all possible persons, which we will call *PERSON*.

[PERSON] the set of all possible uniquely identified persons

Normally, people are identified by name and the possibility of two or more persons having the same name poses difficulties, so we assume that people are identified *uniquely*; for example, by identity-card number or passport number.

The capacity of the aircraft is a natural number which we will call *capacity*. Its actual value is not relevant to the specification; it could even be zero:

capacity: $\mathbb{N}$ the seating capacity of the aircraft

3.2 State-based approach

We describe a system by firstly defining the variables that give its *state* and any invariant properties relating those variables. We also define an operation to set the values of the variables to some suitable *initial state* that satisfies the invariant requirement. We then define *operations* that *change* that state while maintaining the *invariant* properties. We can also define *enquiries* that obtain information about the system without changing its state. Finding the invariant properties that exist between

components of a system is a very important early stage of formal specification.

3.3 The state

The state of the aircraft system is given by the set of people on board the aircraft. This can be described by a set of persons, for which we use the variable *onboard*. Its type is a subset of the set *PERSON*:

$$onboard: \mathbb{P} PERSON$$

The number of persons on board must never exceed the capacity:

$$\#onboard \leq capacity$$

This is an *invariant* property of the state of the system. No operation will be permitted to lead the system into a state for which it does not hold.

3.4 Initialisation operation

There must be an initial state for the system. The obvious one is where the aircraft is empty. The value of a variable *after* an operation is denoted by its name 'decorated' with a prime sign. For example, the value of the variable *onboard* after an operation is designated by *onboard'* (pronounced 'onboard prime'). The 'undecorated' name (*onboard*) refers to the value *before* the operation.

We will define an initialising operation to states that the new value of the set *onboard* is the empty set:

$$onboard' = \emptyset$$

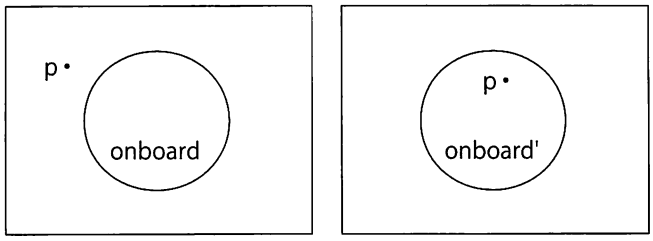
The initialised state must satisfy the invariant property. This it clearly does, since the size of the empty set is zero, which is less than or equal to all natural numbers, and so to all possible values of *capacity*.

3.5 Operations

3.5.1 Boarding

There must be an operation to allow a person, *p*, to board the aircraft. This changes the value of *onboard*.

Figure 3.1



onboard' = onboard ∪ {p}

Note that $\{p\}$ is the singleton set containing just the value p. The new value of the variable *onboard* is the same as the union of its old value and the singleton set containing p.

Each normal boarding operation increases the size of the set *onboard* by one, so eventually the size of *onboard* would exceed *capacity*, thus violating the invariant condition. Therefore, it is a necessary *precondition* of this operation that the size of *onboard* before the operation is strictly less than *capacity*:

#onboard < capacity

It would clearly be an error to record the boarding of a person who is already recorded as being on board, so a further precondition is:

p ∉ onboard

To summarise:

p: PERSON

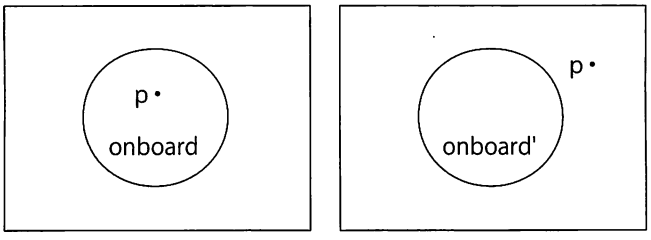
p ∉ onboard
#onboard < capacity
onboard' = onboard ∪ {p}

Decisions regarding the behaviour of the system when the preconditions are not satisfied are best deferred. The Z language offers a convenient notation for adding to a specification at a later stage.

3.5.2 Disembark

It is also necessary to have an operation to allow a person to disembark from the aircraft. The effect on *onboard* is:

Figure 3.2



onboard' = onboard \ {p}

The new value of the variable *onboard* is the same as the old value of *onboard* with the set containing just *p* removed. The precondition of this operation is that the person *p* must be onboard

$$p \in onboard$$

To summarise:

$$p: PERSON$$

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

3.6 Enquiries

3.6.1 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: *numOnboard*. This has no precondition; it always works. It leaves the value of *onboard* unchanged:

$$numOnboard: \mathbb{N}$$
$$numOnboard = \#onboard$$
$$onboard' = onboard$$

Note that it is necessary to state *explicitly* that *onboard* does not change. To say nothing about *onboard'* would imply that it could have *any* value after the operation.

3.6.2 Person on board

Finally, a useful enquiry is to discover whether or not a given person, *p*, is on board. The reply will be a value of the free type *RESPONSE*, which we declare:

$$RESPONSE ::= yes \mid no$$

There is no precondition and the state remains unchanged:

$$p: PERSON$$
$$reply: RESPONSE$$

$$((p \in onboard \land reply = yes)$$
$$\lor$$
$$(p \notin onboard \land reply = no))$$

$$onboard' = onboard$$

The logical operators *and* and *or* will be covered in a later chapter. This section can be read: 'either p is onboard and the reply is *yes* or p is not onboard and the reply is *no*. The state is unchanged.'

3.7 Conclusion

Although the system described here is very simple, it has been described using the concepts of set theory. The description is not complete since treatment of violation of preconditions has been deferred, but the description will be augmented later. The purpose of this example has been to give a first taste of the use of mathematics to describe a system.

EXERCISES

For a computer system as described in Question 1, Chapter 2:

1. Discover any invariant properties.
2. Define a suitable initialisation operation for this system.
3. Define an operation to register a person as a new user, who is initially not logged-in.
4. Define an operation to remove a user's registration, when the user is not logged-in.
5. Define operations for a user to log in and to log out.