CHAPTER 8

A second specification: The video shop

Aims

To apply the concepts of the previous chapter to the development of a simple specification.

Learning objectives

When you have completed this chapter, you should be able to:

- create specifications which use functions;
- use functions in combination with other structures within your specifications.

8.1 Introduction and basic types

A video rental shop keeps one or more copies of each of a set of video titles. To rent a copy, a person must be registered with the shop as one of its members. For simplicity, we will say that a given member can only have one copy of any given title on rental at any given time. We identify the basic types in our specification as

$[PERSON]$	the set of all people
$[TITLE]$	the set of all video titles

Note that a title is an abstract idea; each physical video cassette is not a title, but a *copy* of a title.

8.2 The system state

The state of the system must contain all information relevant to our (albeit simplified) video shop: who the members are, which titles are stocked and how many copies of each, and which titles are currently rented out to which members.

We can capture this information in the following state schema:

$$\begin{array}{|l}
\hline
\textit{VideoShop} \\
members : \mathbb{P}\ PERSON \\
rented : PERSON \leftrightarrow TITLE \\
stockLevel : TITLE \nrightarrow \mathbb{N}_1 \\
\hline
\mathrm{dom}\ rented \subseteq members \\
\mathrm{ran}\ rented \subseteq \mathrm{dom}\ stockLevel \\
\forall t : \mathrm{ran}\ rented \bullet \#\ rented \rhd \{t\} \leqslant stockLevel\ t \\
\hline
\end{array}$$

there are titles for which we do not have amount data.

integrity constraint: you cannot rent more than you have of a title

Here:

members is the set of all registered members.

$p \mapsto t \in rented$ iff person p currently has a copy of title t out on loan. *rented* is a relation, which captures the fact that each member can have copies of many video titles on loan (but only *one* copy of each title), and copies of each video title can be on loan to many people.

$stockLevel\ t$ is the number of copies of title t stocked by the shop. *stockLevel* is a function because each title in dom *stockLevel* is associated with precisely one stock-level figure, and *stockLevel* is a partial function, because the shop does not necessarily stock all the titles in the world! The target of *stockLevel* is $\mathbb{N}_1$, which states that the stock level for any title in stock cannot be zero.

dom *stockLevel* is the set of titles stocked by the shop.

The predicate

$$\mathrm{dom}\ rented \subseteq members$$

captures the requirement that only members may rent videos.

The predicate

$$\mathrm{ran}\ rented \subseteq \mathrm{dom}\ stockLevel$$

captures the requirement that a video can be rented iff it is in stock.

The predicate

$$\forall t : \mathrm{ran}\ rented \bullet \#\ rented \rhd \{t\} \leqslant stockLevel\ t$$

captures the requirement that the shop cannot rent out more copies of a given title than it has in stock.

8.3 The initial state

In the initial state, there are no members and no stock.

$$\begin{array}{|l}
\textit{InitVideoShop} \\
\hline
\textit{VideoShop}' \\
\hline
\textit{members}' = \{\,\} \\
\textit{stockLevel}' = \{\,\} \\
\hline
\end{array}$$

The only possible value of *rented'* is therefore the empty set, and the invariant

$$\forall t : \operatorname{ran} \textit{rented}' \bullet \#\, \textit{rented}' \rhd \{t\} \leqslant \textit{stockLevel}\ t$$

is trivially true, as ran *rented'* is the empty set. The initial state is therefore a valid state for the system.

8.4 Operations

We now specify the successful cases for operations to rent out a video, change the stock level of a given title, and remove a given title from stock. Each of these successful cases will cause a change in the system state. We then specify the successful cases of some query operations which interrogate but do not change the state. These comprise an operation to return the set of all titles rented by a given person, an operation to return the number of copies of a given title which are out on rental, and an operation to return the number of copies of a given title which are in the shop.

Renting out a video

This operation will change the state, and therefore includes Δ *VideoShop*. The inputs required are a person to rent the video and the title to be rented. There are no explicit outputs, only a change of state. The preconditions are as follows:

1. The person *p?* is a member, and the title *t?* is in stock.

$$\begin{array}{l}
p? \in \textit{members} \\
t? \in \operatorname{dom} \textit{stockLevel}
\end{array}$$

2. At least one copy of title *t?* is available for renting.

$$\textit{stockLevel}\ t? > \#\, \textit{rented} \rhd \{t?\}$$

3. The person does not already have a copy of title *t?* on rental.

$$p? \mapsto t? \notin \textit{rented}$$

The postcondition simply adds the required maplet to the relation *rented*.

$$\textit{rented}' = \textit{rented} \cup \{p? \mapsto t?\}$$

The operation schema is as follows:

$$\begin{array}{|l}
\textit{RentVideo} \\
\hline
\Delta\ \textit{VideoShop} \\
p? : \textit{PERSON} \\
t? : \textit{TITLE} \\
\hline
p? \in \textit{members} \\
t? \in \operatorname{dom} \textit{stockLevel} \\
\textit{stockLevel}\ t? > \#\, \textit{rented} \rhd \{t?\} \\
p? \mapsto t? \notin \textit{rented} \\
\textit{rented}' = \textit{rented} \cup \{p? \mapsto t?\} \\
\textit{stockLevel}' = \textit{stockLevel} \\
\textit{members}' = \textit{members} \\
\hline
\end{array}$$

stock level of a title =
# rented out +
# in the shop

Increasing or decreasing the stock level of a given title

This operation requires the title and the required change in stock level (which may be positive or negative) as inputs. The preconditions are as follows:

1. The title *t?* must be in stock.

$$t? \in \operatorname{dom} \textit{stockLevel}$$

2. The potential change must leave a positive number of copies of the title in stock.

$$\textit{stockLevel}\ t? + \textit{change}? > 0$$

3. The number of copies of the title in stock after the operation must not be less than the number of copies out on rental.

$$\textit{stockLevel}\ t? + \textit{change}? \geqslant \#\, \textit{rented} \rhd \{t?\}$$

The postcondition uses the overriding operator $\oplus$ to modify a single pair from *stockLevel* using a singleton function.

$$stockLevel' = stockLevel \oplus \{t? \mapsto stockLevel\ t? + change?\}$$

Title $t?$ is mapped by *stockLevel'* to the new value for its stock level. This is a very common pattern of usage for $\oplus$.

You should note that preconditions 2 and 3 are not strictly necessary. Precondition 2 is implicit in the postcondition, as the target of *stockLevel* is $\mathbb{N}_1$. Precondition 3 is implicit in the 'after' state invariant

$$\forall t : \operatorname{ran} rented' \bullet \#\, rented' \rhd \{t\} \leqslant stockLevel'\ t$$

together with the postcondition. However, we include them in order to construct exception handling schemas for each of these conditions.

The operation schema is as follows:

ChangeStockLevel

$\Delta\, VideoShop$
$t? : TITLE$
$change? : \mathbb{Z}$

$t? \in \operatorname{dom} stockLevel$
$stockLevel\ t? + change? > 0$ (stock level ≥ 1 at all times)
$stockLevel\ t? + change? \geqslant \#\, rented \rhd \{t?\}$ (cannot remove rented copies from stock)
$stockLevel' = stockLevel \oplus \{t? \mapsto stockLevel\ t? + change?\}$
$rented' = rented$
$members' = members$

Removing a title from stock

The title $t?$ must be in stock, and there must be no copies of the title out on rental.

$t? \notin \operatorname{ran} rented$
$t? \in \operatorname{dom} stockLevel$

We must remove the pair containing this title from the *stockLevel* function.

$$stockLevel' = \{t?\} \lhd stockLevel$$

This time, instead of using overriding to map $t?$ to a new range element, we have removed the pair from the function altogether, using the domain anti-restriction operator $\lhd$.

The operation schema is as follows:

DeleteTitle

$\Delta\, VideoShop$
$t? : TITLE$

$t? \notin \operatorname{ran} rented$
$t? \in \operatorname{dom} stockLevel$
$stockLevel' = \{t?\} \lhd stockLevel$
$members' = members$
$rented' = rented$

Note that the predicate

$$t? \in \operatorname{dom} stockLevel$$

is not strictly necessary, because if $t?$ is not in the domain of *stockLevel*, the predicate

$$stockLevel' = \{t?\} \lhd stockLevel$$

represents no change in *stockLevel*. However, in the implementation of the system, we will want to pick this up as an exception to be reported to the user, and it is therefore included in the specification with an appropriate exception handling schema (see below) to specify the generation of a message when the title is not in stock.

Finding out the titles currently rented out by a given person

This operation will not change the state, and therefore includes $\Xi\, VideoShop$. The person must be a member. The required set of titles is the relational image in *rented* of the set containing only this person.

TitlesOut

$\Xi\, VideoShop$
$p? : PERSON$
$titles! : \mathbb{P}\ TITLE$

$p? \in members$
$titles! = rented(\!|\{p?\}|\!)$

The number of copies of a given title currently out on rental

The title must be one stocked by the shop. The required output is the number of pairs in *rented* which have this title as their second element.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CopiesRentedOut}} \\ \hline
\Xi\, \textit{VideoShop} \\
t? : \textit{TITLE} \\
\textit{copiesOut}! : \mathbb{N} \\ \hline
t? \in \operatorname{dom} \textit{stockLevel} \\
\textit{copiesOut}! = \#\, \textit{rented} \rhd \{t?\} \\ \hline
\end{array}
$$

The number of copies of a given title currently in the shop

The title must be one that is stocked by the shop. The required output is the number of copies stocked by the shop minus the number of copies currently out on rental. We can access the latter by including the *CopiesRentedOut* schema with appropriate renaming of *copiesOut*! so that it is not an output from the operation. Note that this also brings the declaration of the title *t*? and Ξ *VideoShop* into scope.

becomes a local variable

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CopiesInShop}} \\ \hline
\textit{CopiesRentedOut}[\textit{copiesOut} / \textit{copiesOut}!] \\
\textit{copiesIn}! : \mathbb{N} \\ \hline
t? \in \operatorname{dom} \textit{stockLevel} \\
\textit{copiesIn}! = \textit{stockLevel}\ t? - \textit{copiesOut} \\ \hline
\end{array}
$$

already existed in CopiesRentedOut, so no need here!

8.5 Error handling schemas

The following free type represents the set of output messages required to construct total versions of the above operations:

$$
\begin{aligned}
\textit{MESSAGE} ::= {} & \textit{success} \mid \textit{notMember} \mid \textit{notInStock} \mid \textit{allCopiesOut} \\
& \mid \textit{alreadyRented} \mid \textit{nonPosStockLevel} \mid \textit{tooManyRented} \\
& \mid \textit{stillRented}
\end{aligned}
$$

The *success* message is used to indicate that an operation has been successfully completed, using the following schema:

$$\textit{SuccessMessage} \mathrel{\widehat{=}} [\textit{outcome}! : \textit{MESSAGE} \mid \textit{outcome}! = \textit{success}]$$

Renting out a video

The precondition exceptions and the schemas to handle them are as follows:

1. The person *p*? is not a member.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NotMember}} \\ \hline
\Xi\, \textit{VideoShop} \\
p? : \textit{PERSON} \\
\textit{outcome}! : \textit{MESSAGE} \\ \hline
p? \notin \textit{members} \\
\textit{outcome}! = \textit{notMember} \\ \hline
\end{array}
$$

2. The title *t*? is not in stock.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NotInStock}} \\ \hline
\Xi\, \textit{VideoShop} \\
t? : \textit{TITLE} \\
\textit{outcome}! : \textit{MESSAGE} \\ \hline
t? \notin \operatorname{dom} \textit{stockLevel} \\
\textit{outcome}! = \textit{notInStock} \\ \hline
\end{array}
$$

3. No copy of title *t*? is available.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AllCopiesOut}} \\ \hline
\Xi\, \textit{VideoShop} \\
t? : \textit{TITLE} \\
\textit{outcome}! : \textit{MESSAGE} \\ \hline
\textit{stockLevel}\ t? = \#\, \textit{rented} \rhd \{t?\} \\
\textit{outcome}! = \textit{allCopiesOut} \\ \hline
\end{array}
$$

(all copies rented out)

4. The person already has a copy on rental.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AlreadyRented}} \\ \hline
\Xi\, \textit{VideoShop} \\
p? : \textit{PERSON} \\
t? : \textit{TITLE} \\
\textit{outcome}! : \textit{MESSAGE} \\ \hline
p? \mapsto t? \in \textit{rented} \\
\textit{outcome}! = \textit{alreadyRented} \\ \hline
\end{array}
$$

Increasing or decreasing the stock level of a given title

The precondition exceptions and the schemas to handle them are as follows:

1. The title $t?$ is not in stock. This is handled by the *NotInStock* schema above.
2. The potential change would not leave a positive number of copies of the title in stock.

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{---}\ NonPosStockLevel\ \text{---}} \\
\Xi\, VideoShop \\
t? : TITLE \\
change? : \mathbb{Z} \\
outcome! : MESSAGE \\
\hline
stockLevel\ t? + change? \leqslant 0 \\
outcome! = nonPosStockLevel \\
\hline
\end{array}$$

3. The number of copies of the title in stock after the operation would be less than the number of copies out on rental.

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{---}\ TooManyRented\ \text{---}} \\
\Xi\, VideoShop \\
t? : TITLE \\
change? : \mathbb{Z} \\
outcome! : MESSAGE \\
\hline
stockLevel\ t? + change? < \#\ rented \rhd \{t?\} \\
outcome! = tooManyRented \\
\hline
\end{array}$$

Removing a title from stock

The precondition exceptions and the schemas to handle them are as follows:

1. The title $t?$ is not in stock. This is handled by the *NotInStock* schema above.
2. There is at least one copy of the title out on rental.

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{---}\ StillRented\ \text{---}} \\
\Xi\, VideoShop \\
t? : TITLE \\
outcome! : MESSAGE \\
\hline
t? \in \operatorname{ran}\ rented \\
outcome! = stillRented \\
\hline
\end{array}$$

Finding out the titles currently rented out by a given person

The precondition exception occurs when the person is not a member. This is handled by the *NotMember* schema above.

The number of copies of a given title currently out on rental

The precondition exception occurs when the title $t?$ is not in stock. This is handled by the *NotInStock* schema above.

The number of copies of a given title currently in the shop

The precondition exception occurs when the title $t?$ is not in stock. This is handled by the *NotInStock* schema above.

8.6 Total operation schemas

The total versions of the operation schemas are now defined using the above exception handling schemas.

$$\begin{aligned}
TotalRentVideo \mathrel{\widehat{=}} {} & (RentVideo \wedge SuccessMessage) \\
& \vee NotMember \\
& \vee NotInStock \\
& \vee AllCopiesOut \\
& \vee AlreadyRented
\end{aligned}$$

$$\begin{aligned}
TotalChangeStockLevel \mathrel{\widehat{=}} {} & (ChangeStockLevel \wedge SuccessMessage) \\
& \vee NonPosStockLevel \\
& \vee TooManyRented
\end{aligned}$$

$$\begin{aligned}
TotalDeleteTitle \mathrel{\widehat{=}} {} & (DeleteTitle \wedge SuccessMessage) \\
& \vee NotInStock \\
& \vee StillRented
\end{aligned}$$

$$\begin{aligned}
TotalTitlesOut \mathrel{\widehat{=}} {} & (TitlesOut \wedge SuccessMessage) \\
& \vee NotMember
\end{aligned}$$

$$\begin{aligned}
TotalCopiesRentedOut \mathrel{\widehat{=}} {} & (CopiesRentedOut \wedge SuccessMessage) \\
& \vee NotInStock
\end{aligned}$$

$$\begin{aligned}
TotalCopiesInShop \mathrel{\widehat{=}} {} & (CopiesInShop \wedge SuccessMessage) \\
& \vee NotInStock
\end{aligned}$$

Exercises 8.1

1. Write schemas for operations to add and remove a member to or from the video shop.
2. Write a schema for the operation whereby a member returns a video to the shop.
3. How could you modify the state schema to allow a maximum of *n* videos to be rented by any given member?
4. Write a schema for an operation to output the set of all people who have a given title on rental.
5. Write a schema *SimilarTastes* to output the set of all people who have on rental at least one of the titles currently rented out to a given person.
6. Write exception handling schemas to totalise (make robust) the above operations.
7. Given the schema *AddTitle*, which adds a new title to the stock,

AddTitle

$\Delta\, VideoShop$
$t?: TITLE$
$level?: \mathbb{N}_1$

$stockLevel' = stockLevel \cup \{t? \mapsto level?\}$
$members' = members$
$rented' = rented$

what implicit precondition is present in this schema?

8. How could you modify the state schema *VideoShop* to allow any given member to rent more than one copy of a given title at the same time?

CHAPTER 9

Sequences

Aims

To introduce the concept of the sequence as a specialised sort of function, to introduce some sequence operators and to demonstrate the application of sequences in writing specifications.

Learning objectives

When you have completed this chapter, you should be able to:

- understand the kinds of system which may be modelled using sequences, and the styles of specification commonly used with sequences;
- understand the effect of relation, function and sequence operators when applied to sequences, and how to construct sequence-valued expressions using them;
- understand how the Z language may be extended by adding generic axiomatic definitions to specifications, and appreciate when it is appropriate to do so.

9.1 Introduction

Sequences embody the idea of the members of a set being arranged in a particular *order*. Examples from everyday life are situations such as a supermarket checkout queue (a sequence of people), a phone directory (a sequence of names arranged alphabetically, each paired up with the corresponding phone number) or a queue at traffic lights (a sequence of vehicles).

Sequences allow us to model the common linear abstract data types of computer science, for example lists, stacks and queues. As artefacts in a specification for a computer program, sequences may naturally be implemented in the target language as arrays, arrangements of pointers and records/structures,