Contents

1 Objectives of this lecture 2

2 A Small library development 2

3 What is a book? 2
   3.1 A Book TYPE machine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   3.2 LibraryBook context machine . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   3.3 Book Services . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

4 User Services 5
   4.1 The UserServices machine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   4.2 Proof obligation problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   4.3 Explanation of the problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

5 Book reservation 8
   5.1 Modelling the reservation queue . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   5.2 Modelling books for collection . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   5.3 The Reservation machine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

6 Browsing Books in the Library 13
   6.1 Putting it all together . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
   6.2 The Library machine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

7 Preparing for a Robust machine 16
   7.1 The Bool_TYPE machine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
1 Objectives of this lecture

To illustrate, through a small case study, the use of structuring in developing a model.

2 A Small library development

To illustrate the structuring constructs, we will model some of the operations and activities that occur in a library. These include:

- book identity,
- book acquisition,
- registering of library users,
- borrowing of books,
- returning of books,
- maintenance of information on disposition of books,
- reservation of books,
- browsing books in the library,
- books exiting the library.

3 What is a book?

A book such as The B Book by Jean-Raymond Abrial is a single entity. The identity for this entity could be implemented by an ISBN, for example.

Multiple copies of the book are printed and acquired by individuals or a library. Thus, copies of books each have their own identity. For a book in a library this identity could be implemented by a shelf number.

Thus, we will need two identities for books in a library:

- a unique global identity of THE BOOK;
- a local identity for copies of the book acquired by the library.
3.1 A Book Context machine

We will use the set \( BOOK \) to model the global identity of each unique book. This set is specified in the \( Book_ctx \) machine that is seen (using \( SEES \)) by other machines in the development.

\( BOOK \) models an individual book, not the published copies of a book. An implementation would be the set of ISBN numbers.

\[
MACHINE \ Book_ctx \\
SETS \ BOOK \\
END
\]

3.2 LibraryBook context machine

We will use a set \( LIBRARYBOOK \) to model the identity of copies of books acquired by the library.

The identifiers are not global, but local to the library.

\( LIBRARYBOOK \) models unique book identifiers that are assigned each book in the library’s collection.

\[
MACHINE \ LibraryBook_ctx \\
SETS \ LIBRARYBOOK \\
END
\]

3.3 Book Services

The \( BookServices \) machine models the acquisition of books by the library. 

Requirements:

- to acquire new copies of books from the set \( BOOK \) for the library;
- to assign to each new book a unique book identifier from the set \( LIBRARYBOOK \);
- to provide service operations for adding books to and removing books from the shelves of the library.
- to build a catalogue that binds books in the library to their unique global book identifier from the set \( BOOK \);
- to provide an operation that reports the library identifier for all copies of a book in the library.

\[
MACHINE \ BookServices \ (\ maxbooks \ ) \\
CONSTRANTS \ maxbooks \in \mathbb{N}_1 \\
SEES \ Book_ctx , LibraryBook_ctx \\
VARIABLES \\
librarybooks , \\
books , \\
book_copies , \\
books_on_shelf
\]
**INVARIANT**

*librarybooks* models all copies of books acquired by the library; *books_on_shelf* models those books currently on the shelves of the library. *book_copies* maps books owned by the library to the identity of the book in the set *BOOK*.

\[
\text{librarybooks} \subseteq \text{LIBRARYBOOK} \land \\
\text{card} ( \text{librarybooks} ) \leq \text{maxbooks} \land \\
\text{books} \subseteq \text{BOOK} \land \\
\text{book_copies} \in \text{librarybooks} \rightarrow \text{books} \land \\
\text{books_on_shelf} \subseteq \text{librarybooks}
\]

**INITIALISATION**

\[
\text{librarybooks}, \text{books} := \{\}, \{\} \parallel \\
\text{books_on_shelf}, \text{book_copies} := \{\}, \{\}
\]

**OPERATIONS**

*AddNewBook*(book) Requirements:

- to add new books to the library’s acquisitions.
- As well as adding the book to the acquisitions, the book is also added to the library shelves.

*AddNewBook* (book) \(\triangleright\)

\[
\text{pre} \quad \text{book} \in \text{BOOK} \land \text{card} ( \text{librarybooks} ) \neq \text{maxbooks} \quad \text{then}
\]

\[
\text{any} \quad \text{bookid} \\
\text{where} \quad \text{bookid} \in \text{LIBRARYBOOK} - \text{librarybooks} \quad \text{then}
\]

\[
\text{librarybooks} := \text{librarybooks} \cup \{ \text{bookid} \} \parallel \\
\text{books} := \text{books} \cup \{ \text{book} \} \parallel \\
\text{books_on_shelf} := \text{books_on_shelf} \cup \{ \text{bookid} \} \parallel \\
\text{book_copies} ( \text{bookid} ) := \text{book}
\]

end

end ;

\[
\text{books} \leftarrow \text{LocateBook}(\text{book}) \text{ returns the set of library book identifiers for } \text{book} \in \text{BOOK}
\]

\[
\text{bookids} \leftarrow \text{LocateBook} ( \text{book} ) \triangleright
\]

\[
\text{pre} \quad \text{book} \in \text{BOOK} \quad \text{then}
\]

\[
\text{bookids} := \text{book_copies}^{-1} [ \{ \text{book} \} ]
\]

end ;


These operations enable other machines to modify *books_on_shelf*. They will not finally be exported to the interface.
AddBookToShelf (book) \equiv \\
\text{pre} \quad \text{book} \in LIBRARYBOOK \land \text{book} \in librarybooks \text{ then} \\
\text{books}	extunderscore on	extunderscore shelf := \text{books}	extunderscore on	extunderscore shelf \cup \{ \text{book} \} \\
\text{end} ; \\
RemoveBookFromShelf (book) \equiv \\
\text{pre} \quad \text{book} \in LIBRARYBOOK \land \text{book} \in librarybooks \text{ then} \\
\text{books}	extunderscore on	extunderscore shelf := \text{books}	extunderscore on	extunderscore shelf \setminus \{ \text{book} \} \\
\text{end} \\
\text{END}

4 User Services

The UserServices machine will model the operations of

1. registering a user of the library, and
2. borrowing and returning library books.

To model the identifiers issued to registered users we will use a deferred set LIBRARYUSER. The conventional practice of separating context (sets and constants) is followed by creating a separate machine, User\textunderscore ctx, to contain the set and a constant anyuser, an arbitrary member of LIBRARYUSER.

MACHINE User\textunderscore ctx
SETS LIBRARYUSER

OPERATIONS
user ← anyuser \equiv \\
user : \in LIBRARYUSER
\text{END}

4.1 The UserServices machine

The purpose of the UserServices machine is to:

- register users of the library; only registered users of the library may borrow books.
- control the borrowing capability of the library; A registered use may borrow any number of books, but may not borrow more than one copy of the same book.
- maintain information on books on loan.

The machine is specified as an extension of the BookServices machine.

MACHINE UserServices (maxuser, maxlibrary)
CONSTRAINTS

\[ \text{maxuser} \in \mathbb{N}_1 \land \text{maxlibrary} \in \mathbb{N}_1 \]

SEES

User\_ctx, 
Book\_ctx, LibraryBook\_ctx

EXTENDS BookServices (maxlibrary)

VARIABLES

libraryusers, 
librarybooks\_onloan

INARIANT

We will model the set of registered users by the set libraryusers

\[ \text{libraryusers} \subseteq \text{LIBRARYUSER} \land \]
\[ \text{card} (\text{libraryusers}) \leq \text{maxuser} \land \]

each library book is on loan to at most one library user

\[ \text{librarybooks\_onloan} \in \text{librarybooks} \leftrightarrow \text{libraryusers} \land \]
a book cannot be on loan and on the shelf at the same time

\[ \text{dom} (\text{librarybooks\_onloan}) \cap \text{books\_onshelf} = \{\} \land \]
the number of copies of books on loan to a user is the same as the number of global books on loan to
that user

\[ \forall \text{user} \cdot (\text{user} \in \text{ran} (\text{librarybooks\_onloan}) \Rightarrow \]
\[ \text{card} (\text{librarybooks\_onloan}^{-1}[\{\text{user}\}]) = \]
\[ \text{card} (\text{books\_onloan}^{-1}[\{\text{user}\}]) \]

As illustrated in the above, it is useful to have a variable books\_onloan, but this variable is depend-ent on and completely defined by other variables. See Definitions at the end of this machine.

ASSERTIONS

\[ \text{books\_onloan} \in \text{books} \leftrightarrow \text{libraryusers} \land \]
\[ \text{ran} (\text{books\_onloan}) = \text{ran} (\text{librarybooks\_onloan}) \land \]
\[ \text{dom} (\text{librarybooks\_onloan}) \cap \text{books\_onshelf} = \{\} \]

INITIALISATION

\[ \text{libraryusers}, \text{librarybooks\_onloan} := \{\}, \{\} \]

OPERATIONS

\[ \text{newuser} \leftarrow \text{NewUser} \]
Requirement: to register a new user by allocating a new user token. Pre-
condition: not all USER tokens have been allocated.
newuser ←\text{NewUser} ≜ 

\text{pre card (libraryusers) \neq maxuser then}

any user

where user ∈ LIBRARYUSER − libraryusers then

libraryusers := libraryusers ∪ \{ user \} ∥

card := user

newuser := user

end

end ;

\text{Borrow(user, book) Requirement: to record the borrowing of a book. Precondition: the borrower must be a register user; the book must belong to the library and must not be on loan.}

\text{Borrow (user, book) ≜}

\text{pre user ∈ libraryusers \land }

book ∈ librarybooks \land \text{book \notin dom (librarybooks_onloan) \land }

user \notin books_onloan \{ \{ book\text{\_copies (book) } \} \} \text{ then}

\text{RemoveBookFromShelf (book) \parallel}

librarybooks_onloan (book) := user

\text{end ;}


\text{Return (book) ≜}

\text{pre book ∈ dom (librarybooks_onloan) then}

\text{AddBookToShelf (book) \parallel}

librarybooks_onloan := \{ book \} \leftarrow \text{librarybooks_onloan}

\text{end ;}

\text{users ← Borrowed(book) Requirement: report the borrowers of all copies of a book.}

\text{users ← Borrowed (book) ≜}

\text{pre book ∈ BOOK then}

users := books_onloan \{ \{ book \} \}

\text{end ;}

\text{CancelLoan(book) Requirement: cancel a loan; the book is not placed on the library shelf. This is a utility operation that will be used in subsequent machines.}

\text{CancelLoan (book) ≜}

\text{pre book ∈ LIBRARYBOOK then}

\text{end ;}
librarybooks_onloan := \{ book \} \preceq librarybooks_onloan

DEFINITIONS
books_onloan \triangleq book_copies^{-1} ; librarybooks_onloan

END

4.2 Proof obligation problem

There is a proof obligation for the operation \texttt{AddBookToShelf} which is impossible to discharge.

\textit{Why?}

4.3 Explanation of the problem

It might even seem strange that we are presented with a proof obligation for an operation that belongs to another machine; a machine for which all proof obligations have already been discharged.

The answer is that we have chosen to promote \texttt{AddBookToShelf} and this operation can break the invariant of \texttt{Borrowing}.

We are promoting this operation because we will need it in subsequent machines that will include \texttt{Borrowing}.

If the \texttt{AddBookToShelf} operation is used in appropriate contexts then it will be possible that the invariant of \texttt{Borrowing} will not be broken.

This operation has been simply promoted to illustrate this type of problem; an alternative would be to repackage the \texttt{AddBookToShelf} operation with a strengthened precondition to ensure safety.

5 Book reservation

\textit{Requirements:}

- We would like to allow a registered user to reserve a book when no copies of that book are available for loan.
- A user may reserve any number of books, but only one copy of any particular book may be reserved.
- A user may not reserve a book if a copy of that book is currently borrowed by that user.
- Many users may reserve the same book, and reservations are queued in the order in which the reservation requests were received.
- There will be some limit on the size of each reservation queue.
- When a reserved book is returned the book is available for collection and borrowing by the user at the head of the queue.
- Books are reserved on global book identifiers.
5.1 Modelling the reservation queue

For each reserved book will have a queue of registered users.

We will model this using a partial function from \( \text{books} \) to a sequence of \( \text{libraryusers} \).

Since each user can reserve any book at most once we will use an injective sequence to model the queue.

Also, we do not want to keep empty reservation queues so we will use non-empty sequences.

\[
\text{reserved} \in \text{books} \rightarrow \text{iseq} \left( \text{libraryusers} \right) \land \left[ .5 \text{ex} \right] \text{reserved} \in \text{books} \rightarrow \text{seq}_{1} \left( \text{libraryusers} \right)
\]

The size of any queue may not exceed \( \text{maxreserve} \).

\[
\forall \text{book} . \left( \text{book} \in \text{dom} \left( \text{reserved} \right) \Rightarrow \text{size} \left( \text{reserved} \left( \text{book} \right) \right) \leq \text{maxreserve} \right)
\]

5.2 Modelling books for collection

For collection we have to model a number of library users who have books available for collection and the books being held for them to collect. We also have to model the local book identifiers of books to be collected. There are a number of possible models. We will use two relations:

1. a relation from books to the actual copies of those books being held for collection \( \text{books} \rightarrow \text{librarybooks} \)

2. a relation from library users to the books waiting collection \( \text{users} \rightarrow \text{books} \rightarrow \text{libraryusers} \)

\[
\text{dom} \left( \text{books} \rightarrow \text{collect} \right)
\]

Other constrains will be given in the invariant.

5.3 The Reservation machine

The \( \text{Reservation} \) machine partially extends the \( \text{Borrowing} \) machine. Reservation interacts with borrowing and returning of books, so the \( \text{Borrow} \) and \( \text{Return} \) operations will be redefined.

\[
\begin{align*}
\text{MACHINE} & \text{ Reservation} \left( \text{maxuser} , \text{maxlibrary} , \text{maxreserve} \right) \\
\text{CONSTRAINTS} & \text{maxuser} \in \mathbb{N}_1 \land \text{maxlibrary} \in \mathbb{N}_1 \land \text{maxreserve} \in \mathbb{N}_1 \\
\text{SEES} & \text{User} \_\text{ctx} , \text{Book} \_\text{ctx} , \text{LibraryBook} \_\text{ctx} \\
\text{INCLUDES} & \text{UserServices} \left( \text{maxuser} , \text{maxlibrary} \right) \\
\text{PROMOTES} & \text{AddNewBook} , \text{Borrowed}
\end{align*}
\]

\[
\text{VARIABLES}
\begin{align*}
\text{reserved} , \text{books} \rightarrow \text{collect}
\end{align*}
\]

\[
\text{INVARIANT}
\begin{align*}
\text{reserved} \in \text{books} \rightarrow \text{iseq} \left( \text{libraryusers} \right) \land \\
\text{reserved} \in \text{books} \rightarrow \text{seq}_{1} \left( \text{libraryusers} \right) \land \\
\forall \text{book} . \left( \text{book} \in \text{dom} \left( \text{reserved} \right) \Rightarrow \\
\text{size} \left( \text{reserved} \left( \text{book} \right) \right) \leq \text{maxreserve} \right) \land \\
\text{ Reserved books must be either on loan or awaiting collection}
\end{align*}
\]
\( \text{dom}(\text{reserved}) \subseteq \text{dom}(\text{books\_onloan}) \cup \text{ran}(\text{dom}(\text{books\_to\_collect})) \land \)
\(\text{books\_to\_collect} \in \text{libraryusers} \times \text{books} \rightarrow \text{librarybooks} \land \)

Any user has at most one book to collect
\(\text{dom}(\text{books\_to\_collect}) \in \text{libraryusers} \rightarrow \text{books} \land \)

for all book, user pairs, the books set aside for collection are library copies of that book
\(\forall (\text{user, book}) . (\text{user} \in \text{libraryusers} \land \text{book} \in \text{books} \land \)
\(\text{user} \mapsto \text{book} \in \text{dom}(\text{books\_to\_collect}) \Rightarrow \)
\(\text{books\_to\_collect}(\text{user} \mapsto \text{book}) \in \text{book\_copies}^{-1} [\{\text{book}\}] \land \)

library books awaiting collection cannot be on the shelf
\(\text{ran}(\text{books\_to\_collect}) \cap \text{books\_on\_shelf} = \{\} \land \)

nor can they be on loan
\(\text{ran}(\text{books\_to\_collect}) \cap \text{dom}(\text{librarybooks\_onloan}) = \{\} \land \)

books awaiting collection by a user cannot be on loan to the same user
\(\forall (\text{book, user}) . (\text{book} \in \text{books} \land \text{user} \in \text{libraryusers} \land \)
\(\text{user} \mapsto \text{book} \in \text{dom}(\text{books\_to\_collect}) \Rightarrow \)
\(\text{user} \notin \text{books\_onloan}[\{\text{book}\}] \land \)

books awaiting collection by a user cannot be reserved by the same user
\(\forall (\text{book, user}) . (\text{book} \in \text{books} \land \text{user} \in \text{libraryusers} \land \)
\(\text{user} \mapsto \text{book} \in \text{dom}(\text{books\_to\_collect}) \Rightarrow \)
\(\text{user} \notin \text{ran}(\text{reserved}(\text{book})) \)

**ASSERTIONS**

a consequence of \(\text{ran}(\text{books\_to\_collect}) \land \text{books\_on\_shelf} = \{\} \)
\(\forall \text{book} . (\text{book} \in \text{ran}(\text{books\_to\_collect}) \Rightarrow \text{book} \notin \text{books\_on\_shelf}) \land \)

a consequence of \(\text{ran}(\text{books\_to\_collect}) \land \text{dom}(\text{librarybooks\_onloan}) = \{\} \)
\(\forall \text{book} . (\text{book} \in \text{ran}(\text{books\_to\_collect}) \Rightarrow \)
\(\text{book} \notin \text{dom}(\text{librarybooks\_onloan}) \)

**INITIALISATION**
\(\text{reserved}, \text{books\_to\_collect} := \{\}, \{\} \)

**OPERATIONS**

\(\text{Reserve}(\text{user, book}) \) **Precondition:**

- Only registered users may reserve books
- To be reserved, all copies of a book must be on loan or awaiting collection
- The reserver cannot be the borrower or collector
- The reservation queue for this book must not be full
• The reserver may not have already reserved a copy of this book

\textbf{Reserve} \ ( \ user \ , \ book \ ) \ \overset{\triangleleft}{=} \\
\textbf{pre} \ \ user \in libraryusers \land book \in books \land \\
book \in \text{dom} \ ( \ \text{books} \text{-onloan} \ ) \cup \text{ran} \ ( \ \text{dom} \ ( \ \text{books} \text{-to_collect} \ ) \ ) \land \\
book\text{-copies}^{-1} \ [ \ \{ \ \text{book} \ \} \ ] \subseteq \text{dom} \ ( \ \text{librarybooks} \text{-onloan} \ ) \cup \\
\text{ran} \ ( \ \text{books} \text{-to_collect} \ ) \land \\
user \notin \text{books} \text{-onloan} \ [ \ \{ \ \text{book} \ \} \ ] \land \\
user \mapsto \text{book} \in \text{dom} \ ( \ \text{books} \text{-to_collect} \ ) \land \\
( \ \text{book} \in \text{dom} \ ( \ \text{reserved} \ ) \ \Rightarrow \\
\text{size} \ ( \ \text{reserved} \ ( \ \text{book} \ ) \ ) \neq \text{maxreserve} \ ) \land \\
( \ \text{book} \in \text{dom} \ ( \ \text{reserved} \ ) \ \Rightarrow \ user \notin \text{ran} \ ( \ \text{reserved} \ ( \ \text{book} \ ) \ ) \ )

\textbf{Borrow1} ( \ user \ , \ book \ ) : \ An \ upgrade \ of \ \textbf{Borrow} , \ this \ operation \ strengthens \ the \ precondition \ to \ take \ account \ of \ the \ new \ state.

\textbf{Borrow1} \ ( \ user \ , \ book \ ) \ \overset{\triangleleft}{=} \\
\textbf{pre} \ \ user \in libraryusers \land book \in librarybooks \land \\
book \notin \text{dom} \ ( \ librarybooks \text{-onloan} \ ) \land \\
user \notin \text{books} \text{-onloan} \ [ \ \{ \ \text{book} \text{-copies} \ ( \ \text{book} \ ) \ \} \ ] \land \\
book \notin \text{ran} \ ( \ \text{books} \text{-to_collect} \ ) \land \\
user \mapsto \text{book} \text{-copies} \ ( \ \text{book} \ ) \notin \text{dom} \ ( \ \text{books} \text{-to_collect} \ ) \ \textbf{then} \\
\textbf{Borrow} \ ( \ user \ , \ book \ )

\textbf{Return1} ( \ book \ ) : \ A \ replacement \ for \ the \ \textbf{Return} \ operation. \ When \ a \ book \ is \ returned, \ we \ need \ to \ check \ the \ reservation \ list.

If \ the \ book \ is \ reserved \ we \ cancel \ the \ loan \ without \ putting \ the \ book \ on \ the \ shelf, \ and \ we \ put \ the \ book \ on \ the \ collect \ list \ for \ the \ user \ who \ is \ at \ the \ head \ of \ the \ reservation \ queue.

If \ the \ book \ is \ not \ reserved, \ then \ we \ simply \ revert \ to \ the \ current \ \textbf{Return} \ operation.

\textbf{Return1} \ ( \ book \ ) \ \overset{\triangleleft}{=} \\
\textbf{pre} \ \ \text{book} \in \text{dom} \ ( \ librarybooks \text{-onloan} \ ) \ \textbf{then} \\
\textbf{any} \ \text{bbook} \ \textbf{where} \ \text{bbook} \in BOOK \land \\
\text{bbook} = \text{book} \text{-copies} \ ( \ \text{book} \ ) \ \textbf{then} \\
\textbf{if} \ \text{bbook} \in \text{dom} \ ( \ \text{reserved} \ ) \ \textbf{then}
CancelLoan (book) ||
books_to_collect (first (reserved (bbook)) ->
bbook := book ||

if size (reserved (bbook)) = 1 then
reserved := {bbook} $\triangleleft$ reserved
else reserved (bbook) := tail (reserved (bbook))
end
else Return (book)
end
end;

CancelReservation(user, book): A user who has reserved a book may wish to cancel the reservation.
The reservation must be deleted from the reservation queue, or the queue should be deleted if the reservation is the only one on the queue.

CancelReservation (user, book) ⇔
pre user $\in$ libraryusers $\land$ book $\in$ dom (reserved) $\land$
user $\in$ ran (reserved (book)) then
if size (reserved (book)) = 1 then
reserved := {book} $\triangleleft$ reserved
else any pos, list where
pos = (reserved (book)) $^{-1}$ (user) $\land$
list = reserved (book) then
reserved (book) :=
list $\uparrow$ pos $\rightarrow$ 1 $\leftarrow$ (list $\downarrow$ pos)
end
end
end;

CollectnBorrow(user, book): This operation allows a user to collect a book and immediately borrow a copy of that book. Collection is through a global book identifier, just as for the Reserve operation.

CollectnBorrow (user, book) ⇔
pre user $\in$ libraryusers $\land$ book $\in$ books $\land$
user $\mapsto$ book $\in$ dom (books_to_collect) $\land$
user $\notin$ books_onloan [ {book} ] then
any bookid where bookid $\in$ librarybooks $\land$
book_copies (bookid) = book $\land$
bookid = books_to_collect ( user ↦→ book ) then
Borrow ( user , bookid ) ||
books_to_collect := { user ↦→ book } ≲ books_to_collect
end
end ;

UnCollect (user , book): Allow a user tUnCollect (user , book): o surrender their right to collect a reserved book.

If the reservation queue for the book is not empty then the book should be made available for the user at the head of the queue, otherwise the book should be put on the library shelf.

UnCollect (user , book) ⇔
pre user ∈ libraryusers ∧ book ∈ books ∧
user ↦→ book ∈ dom ( books_to_collect ) then
any bookid where bookid ∈ librarybooks ∧
bookid = books_to_collect ( user ↦→ book ) then
if book ∈ dom ( reserved ) then
books_to_collect := { user ↦→ book } ≲ books_to_collect ∪
{ first ( reserved ( book ) ) ↦→ book ↦→ bookid } ||
if size ( reserved ( book ) ) = 1 then
reserved := { book } ≲ reserved
else reserved ( book ) := tail ( reserved ( book ) )
end
else books_to_collect := books_to_collect −
{ user ↦→ book ↦→ bookid } ||
AddBookToShelf ( bookid )
end
end
end
END

6 Browsing Books in the Library

Requirements: to model

• browsing books on the shelves of the library;
• putting books back on the shelves;
• a book exiting the library.

This machine upgrades the current borrowing operation.
Notice that we want to model browsers reading books that are on the library shelf, but this information is held in the variable books_on_shelf, which is “owned” by the BookServices machine. We will use USES to view the variables in that machine, and use them in the invariant.

MACHINE Browsing (maxuser, maxlibrary, maxreserve)

CONSTRAINTS
maxuser ∈ ℕ₁ ∧
maxlibrary ∈ ℕ₁ ∧
maxreserve ∈ ℕ₁

SEES
User_ctx,
Book_ctx,
LibraryBook_ctx

INCLUDES Reservation (maxuser, maxlibrary, maxreserve)

PROMOTES
AddNewBook,
Borrowed,
Reserve,
Return1,
CollectnBorrow,
CancelReservation,
UnCollect

VARIABLES browsing

INVARIANT browsing ⊆ books_on_shelf

ASSERTIONS
a consequence of ran(books_to_collect) ∧ books_on_shelf = {}

∀ book. (book ∈ ran(books_to_collect) ⇒ book ∉ books_on_shelf) ∧
a consequence of ran(books_to_collect) wedge dom(librarybooks_onloan) = {}

∀ book. (book ∈ ran(books_to_collect) ⇒
    book ∉ dom(librarybooks_onloan))

INITIALISATION browsing := {}

OPERATIONS

It must be “available”, which means it must be on the shelf and can’t be currently being browsed.

BeginBrowse(book) \iff
pre  book ∈ books_on_shelf ∧ book ∉ browsing then
    browsing := browsing ∪ {book}
end ;
**EndBrowse**(*book*): End the browsing of a book in the library.

\[
\text{EndBrowse (} book \text{)} \equiv \\
\text{pre} \quad book \in \text{browsing} \quad \text{then} \\
\quad \text{browsing} := \text{browsing} - \{ \text{book} \} \\
\text{end} ;
\]

**RemoveBook**(*book*): Check a book being removed from the library.
If it belongs to the library, then it must be borrowed.
Notice that this operation is only a precondition.

\[
\text{RemoveBook (} book \text{)} \equiv \\
\text{pre} \quad book \in \text{LIBRARYBOOK} \land \\
\quad ( \text{book} \in \text{librarybooks} \Rightarrow \text{book} \in \text{dom (librarybooks}_\text{onloan}) ) \\
\text{then} \quad \text{skip} \\
\text{end} ;
\]

**Borrow2**(*user*, *book*): The same as **Borrow1**(*user*, *book*) except it also removes *book* from *browsing*.

\[
\text{Borrow2 (} user, \text{book} \text{)} \equiv \\
\text{pre} \quad user \in \text{libraryusers} \land \text{book} \in \text{librarybooks} \land \\
\quad \text{book} \notin \text{dom (librarybooks}_\text{onloan}) \land \\
\quad user \notin \text{books}_\text{onloan} [ \{ \text{book}_\text{copies (} book \text{)} \} ] \land \\
\quad \text{book} \notin \text{ran (books}_\text{to_collect}) \land \\
\quad user \mapsto \text{book}_\text{copies (} book \text{)} \notin \text{dom (books}_\text{to_collect}) \\
\text{then} \\
\quad \text{Borrow1 (} user, \text{book} \text{)} \parallel \\
\text{browsing} := \text{browsing} - \{ \text{book} \} \\
\text{end}
\]

**END**

### 6.1 Putting it all together

**Objectives:**

- To include all the component machine into a top-level LIBRARY machine;
- To resolve any insecurity due to the use of USES.

### 6.2 The Library machine

This “top-level” machine is build on top of the Browsing machine.
The UserRegistration machine, which was previously used, is now included.
MACHINE Library (maxbook, maxuser, maxlibrary, maxreserve)
CONSTRAINTS
maxbook ∈ N₁ ∧
maxuser ∈ N₁ ∧
maxlibrary ∈ N₁ ∧
maxreserve ∈ N₁
SEES
User_ctx, Book_ctx, LibraryBook_ctx
INCLUDES Browsing (maxuser, maxlibrary, maxreserve)
PROMOTES
AddNewBook, Borrow2, Return1, Borrowed, Reserve, CollectnBorrow, CancelReservation, UnCollect, BeginBrowse, EndBrowse, RemoveBook

INVARIANT
We now assert a conservation relationship between the books owned by the library, the books on the shelf, the books on loan and the books awaiting collection.
librarybooks = books_on_shelf ∪ dom (librarybooks_onloan) ∪ ran (books_to_collect)

7 Preparing for a Robust machine

We are about to build an API version of the Library machine but, we also want to implement the API machine and we plan to do a modular implementation, importing some version of the Library machine. This produces a problem for the evaluation of guards in the API machine.

For that reason we will augment the Library machine with operations that can be used for the evaluation of guards.

Problems with exporting/importing sets
At the very topmost level, it is not possible to export or import sets or other structured objects. This is a general problem with interfaces and is not a problem peculiar to the B Method (B) implementations.

The library machine currently has an operation, Borrowed, that returns a set of library users. We will replace that single operation by two operations, BorrowedFirst and BorrowedNext, that achieve the same effect by iteration.

7.1 The Bool_TYPE machine

A library machine, Bool_TYPE, is also seen. This machine contains the enumerated set BOOL = {FALSE, TRUE}, and models the concrete Boolean as implemented by real machines, as distinct from the abstract Boolean of logic.
Inclusion of this machine also provides the function \texttt{bool}:

\[
\texttt{var} := \texttt{bool}(\texttt{expr}) \equiv \begin{cases} \text{IF} \ \texttt{expr} \ \text{THEN} \ \texttt{var} := \text{TRUE} \\ \text{ELSE} \ \texttt{var} := \text{FALSE} \end{cases}
\]

\[
\text{END}
\]

\[
\text{MACHINE} \ \texttt{Bool\_TYPE}
\]
\[
\text{SETS} \ \texttt{BOOL} = \{ \text{FALSE} , \text{TRUE} \}
\]
\[
\text{END}
\]

7.2 The LibraryPlus machine

This machine partially extends the Library machine adding supplementary operation that enable satisfaction of preconditions to be determined from outside the machine. The operation Borrowed is iterated to overcome the problem of not being able to pass sets through the topmost interface.

\[
\text{MACHINE} \ \texttt{LibraryPlus} (\ \texttt{maxbook} , \texttt{maxuser} , \texttt{maxlibrary} , \\
\ \texttt{maxreserve} , \texttt{maxiter} )
\]
\[
\text{CONSTRAINTS} \\
\ \texttt{maxbook} \in \mathbb{N}_1 \land \\
\ \texttt{maxuser} \in \mathbb{N}_1 \land \\
\ \texttt{maxlibrary} \in \mathbb{N}_1 \land \\
\ \texttt{maxreserve} \in \mathbb{N}_1 \land \\
\ \texttt{maxiter} \in \mathbb{N}_1
\]
\[
\text{SEES} \\
\ \texttt{User\_CTX} , \\
\ \texttt{Book\_TYPE} , \\
\ \texttt{LibraryBook} , \\
\ \texttt{IterKey} , \\
\ \texttt{Bool\_TYPE}
\]
\[
\text{INCLUDES} \ \texttt{Library} (\ \texttt{maxbook} , \texttt{maxuser} , \texttt{maxlibrary} , \texttt{maxreserve} )
\]
\[
\text{PROMOTES} \\
\ \texttt{AddNewBook} , \texttt{Borrow2} , \texttt{Return1} , \texttt{Reserve} , \texttt{CollectnBorrow} , \\
\ \texttt{CancelReservation} , \texttt{UnCollect} , \texttt{BeginBrowse} , \texttt{EndBrowse} , \\
\ \texttt{RemoveBook} , \texttt{NewUser}
\]
\[
\text{VARIABLES} \ \texttt{bkeys} , \texttt{borrowers}
\]
\[
\text{INVARIANT} \\
\ \texttt{bkeys} \subseteq \texttt{ITERKEY} \land \text{card} (\ \texttt{bkeys} ) \leq \texttt{maxiter} \land \\
\ \texttt{borrowers} \in \texttt{bkeys} \rightarrow \mathbb{P} (\ \texttt{libraryusers} )
\]
\[
\text{INITIALISATION} \ \texttt{bkeys} , \texttt{borrowers} := \{ \} , \{ \}
\]
\[
\text{OPERATIONS}
\]

Sets cannot be transmitted, in either direction, through the final API or UAI interface, so we will iterate the Borrowed operation.
\texttt{user, iterkey \leftarrow BorrowedFirst(book) \triangleq}

\texttt{pre book \in BOOK \land \text{card(bkeys)} \neq \text{maxiter} \land}
\texttt{books\_onloan[\{book\}] \neq \{} \text{then}
\texttt{any bkey, userid where}
\texttt{bkey \in ITERKEY \land bkey \in ITERKEY - bkeys \land}
\texttt{userid \in libraryusers \land}
\texttt{userid \in books\_onloan[\{book\}] \text{then}}

\texttt{if books\_onloan[\{book\}] - \{userid\} \neq \{} \text{then}
\texttt{bkeys := bkeys \cup \{bkey\} ||}
\texttt{borrowers(bkey) := books\_onloan[\{book\}] - \{userid\}}
\texttt{end ||}
\texttt{user, iterkey := userid, bkey}
\texttt{end}
\texttt{end ;}

\texttt{user \leftarrow BorrowedNext(bkey) \triangleq}

\texttt{pre bkey \in ITERKEY \land bkey \in bkeys then}
\texttt{any userid where userid \in libraryusers \land}
\texttt{userid \in borrowers(bkey) then}
\texttt{if borrowers(bkey) - \{userid\} \neq \{} \text{then}
\texttt{borrowers(bkey) := borrowers(bkey) - \{userid\}}
\texttt{else borrowers := \{bkey\} \triangleq borrowers}
\texttt{end ||}
\texttt{user := userid}
\texttt{end}
\texttt{end ;}

This machine will be used as imported machine in the implementation of LibraryAPI. The following set of operations provide access to information required for the guards in that machine, the preconditions of the Library machine.

\texttt{ok \leftarrow users\_not\_full \triangleq}
\texttt{begin ok := bool ( card(libraryusers) \neq \text{maxuser} ) end ;}
\texttt{ok \leftarrow user\_registered(user) \triangleq}
\texttt{pre user \in LIBRARYUSER}
\texttt{then ok := bool ( user \in libraryusers )}
\texttt{end ;}
\texttt{ok \leftarrow book\_in\_library(book) \triangleq}
\texttt{pre book \in LIBRARYBOOK}
\texttt{then ok := bool ( book \in librarybooks )}
\begin{verbatim}
end ;
ok ← book_in_books ( book ) ≜
  pre  book ∈ BOOK
  then  ok := bool ( book ∈ books )
end ;
ok ← library_not_full ≜
  begin  ok := bool ( card ( librarybooks ) ≠ maxlibrary ) end ;
ok ← librarybook_onloan ( book ) ≜
  pre  book ∈ LIBRARYBOOK
  then  ok := bool ( book ∈ dom ( librarybooks_onloan ) )
end ;
ok ← book_onloan ( book ) ≜
  pre  book ∈ BOOK
  then  ok := bool ( book ∈ dom ( books_onloan ) )
end ;
ok ← not_book_onloan_to_user ( book , user ) ≜
  pre  book ∈ LIBRARYBOOK ∧ user ∈ LIBRARYUSER
  then  ok := bool ( user ∉ books_onloan [ { book_catalogue ( book ) } ] )
end ;
ok ← not_book_copy_onloan_to_user ( book , user ) ≜
  pre  book ∈ BOOK ∧ user ∈ LIBRARYUSER
  then  ok := bool ( user ∉ books_onloan [ { book } ] )
end ;
ok ← book_onloan_or_notlibrary ( book ) ≜
  pre  book ∈ LIBRARYBOOK
  then
    ok := bool ( book ∈ librarybooks ⇒
      book ∈ dom ( librarybooks_onloan ) )
end ;
ok ← book_onloan_or_collect ( book ) ≜
  pre  book ∈ BOOK
  then
    ok := bool ( book ∉ dom ( books_onloan ) ⇒
      book ∈ ran ( dom ( books_to_collect ) ) )
end ;
ok ← not_librarybook_onloan_or_collect ( book ) ≜
  pre  book ∈ LIBRARYBOOK
  then
    ok := bool ( book ∉ dom ( librarybooks_onloan ) ∧
      book ∉ ran ( books_to_collect ) )
end ;
ok ← reservation_full ( book ) ≜
\end{verbatim}
pre  book ∈ BOOK
then
  ok := bool ( book ∈ dom ( reserved ) ∧
  size ( reserved ( book ) ) = maxreserve )
end ;
ok ← book_reserved ( book ) ≡
pre  book ∈ BOOK
then  ok := bool ( book ∈ dom ( reserved ) )
end ;
ok ← book_reserved_by_user ( book , user ) ≡
pre  book ∈ BOOK ∧ user ∈ LIBRARYUSER
then  ok := bool ( book ∈ dom ( reserved ) ∧
  user ∈ ran ( reserved ( book ) ) )
end ;
ok ← book_for_collection ( book ) ≡
pre  book ∈ BOOK
then  ok := bool ( book ∈ ran ( dom ( books_to_collect ) ) )
end ;
ok ← not_book_copy_for_collection_by_user ( book , user ) ≡
pre  book ∈ LIBRARYBOOK ∧ user ∈ LIBRARYUSER
then  ok := bool ( user ↦→ book_catalogue ( book ) \∉
  dom ( books_to_collect ) )
end ;
ok ← book_for_collection_by_user ( book , user ) ≡
pre  book ∈ BOOK ∧ user ∈ LIBRARYUSER
then  ok := bool ( user ↦→ book ∈ dom ( books_to_collect ) )
end ;
ok ← book_copy_not_on_shelf ( book ) ≡
pre  book ∈ BOOK
then  ok := bool ( book_catalogue ^−1 [ { book } ] \∩ books_on_shelf = {} )
end ;
ok ← book_available_to_browse ( book ) ≡
pre  book ∈ LIBRARYBOOK
then  ok := bool ( book ∈ books_on_shelf ∧ book \∉ browsing )
end ;
ok ← book_in_browsing ( book ) ≡
pre  book ∈ LIBRARYBOOK
then  ok := bool ( book ∈ browsing )
end ;
ok ← more_keys ≡
  ok := bool ( card ( bkeys ) \neq maxiter ) ;
key ← any_iterkey ≜
key :∈ ITERKEY ;
ok ← valid_bkey ( bkey ) ≜
pre bkey :∈ ITERKEY
then ok := bool ( bkey ∈ bkeys )
end
END

7.3 The LibraryAPI machine

MACHINE LibraryAPI ( maxbook , maxuser , maxlibrary ,
maxreserve , maxiter )
CONSTRAINTS
maxbook ∈ N₁ ∧
maxuser ∈ N₁ ∧
maxlibrary ∈ N₁ ∧
maxreserve ∈ N₁ ∧
maxiter ∈ N₁
SEES
User_CTX ,
Book_TYPE ,
LibraryBook ,
IterKey
INCLUDES LibraryPlus ( maxbook , maxuser , maxlibrary ,
maxreserve , maxiter )

SETS
RESPONSE is an enumerated set, whose symbolic values represent the status responses returned with every operation.
RESPONSE = { OK ,
BookNotInLibrary , LibraryFull , BookNotForLoan ,
BookNotOnLoan , BookCopyOnLoan , BookCopyOnCollection ,
BookNotAvailable , NotBeingBrowsed , InvalidReservation ,
NoNewUsers , NotRegisteredUser , NotForCollection ,
NotReservedForUser , ReserveQueueFull , UnBorrowedBook ,
NoBorrowers , NoMoreBorrowers ,
FAIL }

OPERATIONS

response , newuser ← RegisterUser ≜
if card ( libraryusers ) ≠ maxuser
then newuser ← NewUser || response := OK
else newuser ∈ LIBRARYUSER || response := NoNewUsers
end ;

response ← AcquireBook ( book ) ≜
pre book ∈ BOOK then
    select card ( librarybooks ) = maxlibrary
    then response := LibraryFull
    else choice AddNewBook ( book ) ||
        response := OK
        or
        response := FAIL
    end
end
end ;

response ← ExitLibrary ( book ) ≜
pre book ∈ LIBRARYBOOK then
    if book ∈ librarybooks \ book ∉ dom ( librarybooks_onloan )
        then response := UnBorrowedBook
    else RemoveBook ( book ) || response := OK
end
end ;

response ← BeginBrowseBook ( book ) ≜
pre book ∈ LIBRARYBOOK then
    if book ∈ books_on_shelf \ book ∉ browsing
        then
            choice
                BeginBrowse ( book ) || response := OK
            or
                response := FAIL
        end
    else response := BookNotAvailable
end
end ;
response ←  **EndBrowseBook** ( book ) ≜
pre  book ∈ LIBRARYBOOK then
  if  book /∈ browsing
  then  response := NotBeingBrowsed
  else  EndBrowse ( book ) || response := OK
end
end ;

response ←  **BorrowBook** ( user , book ) ≜
pre  user ∈ LIBRARYUSER ∧ book ∈ LIBRARYBOOK then
  select  user /∈ libraryusers
  then  response := NotRegisteredUser
  when  book /∈ librarybooks
  then  response := BookNotForLoan
  when  book ∈ dom ( librarybooks_onloan )
  then  response := BookNotForLoan
  when  user ∈ books_onloan [ { book_catalogue ( book ) } ]
  then  response := BookCopyOnLoan
  when  book ∈ ran ( books_to_collect )
  then  response := BookNotForLoan
  when  user ↦→ book_catalogue ( book ) ∈ dom ( books_to_collect )
  then  response := BookCopyOnCollection
  else  response := OK || Borrow2 ( user , book )
end
end ;

response ←  **ReturnBook** ( book ) ≜
pre  book ∈ LIBRARYBOOK then
  if  book ∈ dom ( librarybooks_onloan )
  then  Return1 ( book ) || response := OK
  else  response := BookNotOnLoan
end
end ;

response , user , iterkey ←  **WhoBorrowedFirst** ( book ) ≜
pre  book ∈ BOOK then
  select  card ( bkeys ) = maxiter
  then  response := FAIL ||
          user ∈ libraryusers ||
iterkey ∈ ITERKEY

when books_onloan [ { book } ] = {} then response := NoBorrowers ||
    user ∈ libraryusers ||
    iterkey ∈ ITERKEY
else response := OK ||
    user, iterkey ← BorrowedFirst ( book )
end
end;

response, user ← WhoBorrowedNext ( bkey ) ≜

pre bkey ∈ ITERKEY then
    select bkey /∈ bkeys
    then response := NoMoreBorrowers || user ∈ libraryusers
    else response := OK || user ← BorrowedNext ( bkey )
end
end;

response ← ReserveBook ( user, book ) ≜

pre user ∈ LIBRARYUSER ∧ book ∈ BOOK then
    choice
        select user /∈ libraryusers
        then response := NotRegisteredUser
        when book /∈ books
        then response := BookNotInLibrary
        when book /∈ dom ( books_onloan ) ∧
            book /∈ ran ( dom ( books_to_collect ) )
        then response := InvalidReservation
        when book_catalogue −1 [ { book } ] ∩ books_on_shelf ≠ {} 
        then response := InvalidReservation
        when user ∈ books_onloan [ { book } ]
        then response := BookCopyOnLoan
        when book = dom ( books_to_collect ) ( user )
        then response := BookCopyOnCollection
        when book ∈ dom ( reserved ) ∧
            user ∈ ran ( reserved ( book ) )
        then response := InvalidReservation
        when book ∈ dom ( reserved ) ∧ size ( reserved ( book ) ) = maxreserve
        then response := ReserveQueueFull

24
else Reserve (user, book) || response := OK
end
or
response := FAIL
end
end ;

response ← CollectAndBorrow (user, book) ⊑
pre user ∈ LIBRARYUSER ∧ book ∈ BOOK then
select user ∉ libraryusers
then response := NotRegisteredUser
when book ∉ books
then response := BookNotInLibrary
when book ≠ dom (books_to_collect) (user)
then response := NotForCollection
when user ∈ books_onloan [ {book} ]
then response := BookCopyOnLoan
else CollectnBorrow (user, book) ||
response := OK
end
end ;

response ← CancelBookReserve (user, book) ⊑
pre user ∈ LIBRARYUSER ∧ book ∈ BOOK then
select user ∉ libraryusers
then response := NotRegisteredUser
when book ∈ dom (reserved) ⇒
user ∉ ran (reserved (book))
then response := NotReservedForUser
else
CancelReservation (user, book) ||
response := OK
end
end ;

response ← UnCollectBook (user, book) ⊑
pre user ∈ LIBRARYUSER ∧ book ∈ BOOK then
select user ∉ libraryusers
then response := NotRegisteredUser
when \( \text{book} \notin \text{books} \) 
then \( \text{response} := \text{BookNotInLibrary} \)
when \( \text{book} \neq \text{dom}(\text{books to collect})(\text{user}) \)
then \( \text{response} := \text{NotForCollection} \)
else \( \text{UnCollect}(\text{user},\text{book}) \parallel \text{response} := \text{OK} \)
end
end

END

8 Implementing LibraryAPI

We will implement the LibraryAPI machine in two steps:

1. implement LibraryAPI by importing LibraryPlus, and then
2. implement LibraryPlus.

Some facts about implementation:

- The implementation has no state of its own.
- Machines are imported and these machines provide surrogate state variables. These variables are used to “implement” the variables of the machine being implemented.
- The invariant of the implementation machine expresses the relation between the variables of the imported machines and the variables of the machine being implemented.

Some notes on the implementation of LibraryAPI

- the implementation of the operations of LibraryAPI are systematic translations of the specifications to the implementation domain with guards replaced by BOOL result returning operations and sequential composition replaced by sequential composition;
- select constructs are replaced by IF-THEN constructs;
- the intrinsically nested IF-THEN-IF-THEN \ldots END END control structures are flattened to sequential IF B THEN END; IF B THEN END where there are runs of the same guard.

8.1 The LibraryAPIII machine

IMPLEMENTATION LibraryAPIII
REFINES LibraryAPI
SEES
\( \text{User}_{\text{CTX}} \),
\( \text{Book\_TYPE} \),
\( \text{LibraryBook} \),
IterKey,
Bool_TYPE
IMPORTS LibraryPlus (maxbook, maxuser, maxlibrary,
maxreserve, maxiter)

OPERATIONS

response, newuser ← RegisterUser ≜
var bb in
    bb ← users_not_full;
    if bb = TRUE then
        response := OK; newuser ← NewUser
    else newuser ← anyuser; response := NoNewUsers
end
end;

response ← AcquireBook (book) ≜
var bb in
    bb ← library_not_full;
    response := LibraryFull;
    if bb = TRUE then
        response := OK; AddNewBook (book)
end
end;

response ← ExitLibrary (book) ≜
var bb in
    response := UnBorrowedBook;
    bb ← book_onloan_or_notlibrary (book);
    if bb = TRUE then
        response := OK; RemoveBook (book)
end
end;

response ← BeginBrowseBook (book) ≜
var bb in
    response := BookNotAvailable;
    bb ← book_available_to_browse (book);
    if bb = TRUE then
response := OK ; BeginBrowse ( book )

end
end ;

response ←→ EndBrowseBook ( book ) ≜

var bb in

response := NotBeingBrowsed ;
bb ←→ book_in_browsing ( book ) ;
if bb = TRUE then
    response := OK ; EndBrowse ( book )
end
end ;

response ←→ BorrowBook ( user , book ) ≜

var bb in

response := NotRegisteredUser ;
bb ←→ user_registered ( user ) ;
if bb = TRUE then
    response := BookNotForLoan ;
    bb ←→ book_in_library ( book )
end ;
if bb = TRUE then
    response := BookCopyOnLoan ;
    bb ←→ not_book_onloan_to_user ( book , user )
end ;
if bb = TRUE then
    response := BookNotForLoan ;
    bb ←→ not_librarybook_onloan_or_collect ( book )
end ;
if bb = TRUE then
    response := BookCopyOnCollection ;
    bb ←→ not_book_copy_for_collection_by_user ( book , user )
end ;
if bb = TRUE then
    Borrow2 ( user , book ) ; response := OK
end
end ;

response ←→ ReturnBook ( book ) ≜
var \( bb \) in 
\( \text{response} := \text{BookNotOnLoan} \); 
\( bb \leftarrow \text{librarybook\_onloan (book)} \); 
if \( bb = \text{TRUE} \) then 
\( \text{Return1 (book)} ; \text{response} := \text{OK} \)
end 
end ;

\( \text{response, user, iterkey} \leftarrow \text{WhoBorrowedFirst (book)} \) \( \triangleq \)
var \( bb \) in 
\( \text{user} \leftarrow \text{anyuser} \); 
\( \text{iterkey} \leftarrow \text{any\_iterkey} \); 
\( bb \leftarrow \text{more\_keys} \); 
\( \text{response} := \text{FAIL} \); 
if \( bb = \text{TRUE} \) then 
\( bb \leftarrow \text{book\_onloan (book)} \); 
\( \text{response} := \text{NoBorrowers} \)
end ;
if \( bb = \text{TRUE} \) then 
\( \text{response} := \text{OK} ; \) 
\( \text{user, iterkey} \leftarrow \text{BorrowedFirst (book)} \)
end 
end ;

\( \text{response, user} \leftarrow \text{WhoBorrowedNext (bkey)} \) \( \triangleq \)
var \( bb \) in 
\( \text{user} \leftarrow \text{anyuser} \); 
\( bb \leftarrow \text{valid\_bkey (bkey)} \); 
\( \text{response} := \text{NoMoreBorrowers} \); 
if \( bb = \text{TRUE} \) then 
\( \text{response} := \text{OK} ; \) 
\( \text{user} \leftarrow \text{BorrowedNext (bkey)} \)
end 
end ;

\( \text{response} \leftarrow \text{ReserveBook (user, book)} \) \( \triangleq \)
var \( bb \) in 
\( \text{response} := \text{NotRegisteredUser} \); 
\( bb \leftarrow \text{user\_registered (user)} \); 
if \( bb = \text{TRUE} \) then 
\( \text{response} := \text{BookNotInLibrary} \);
\[ bb \leftarrow \text{book\_in\_books}\ (\text{book}) \]

\begin{verbatim}
end ;
if \( bb = \text{TRUE} \) then
  response := \text{InvalidReservation} ;
  bb \leftarrow \text{book\_onloan\_or\_collect}\ (\text{book})
end ;
if \( bb = \text{TRUE} \) then
  response := \text{InvalidReservation} ;
  bb \leftarrow \text{book\_copy\_not\_on\_shelf}\ (\text{book})
end ;
if \( bb = \text{TRUE} \) then
  response := \text{BookCopyOnLoan} ;
  bb \leftarrow \text{not\_book\_copy\_onloan\_to\_user}\ (\text{book}, \text{user})
end ;
if \( bb = \text{TRUE} \) then
  response := \text{BookCopyOnCollection} ;
  bb \leftarrow \text{book\_for\_collection}\ (\text{book}) ;
  if \( bb = \text{FALSE} \) then
    response := \text{InvalidReservation} ;
    bb \leftarrow \text{book\_reserved\_by\_user}\ (\text{book}, \text{user})
  end ;
  if \( bb = \text{FALSE} \) then
    response := \text{ReserveQueueFull} ;
    bb \leftarrow \text{reservation\_full}\ (\text{book})
  end ;
  if \( bb = \text{FALSE} \) then
    response := \text{OK} ; \text{Reserve}\ (\text{user}, \text{book})
  end
end
end

response \leftarrow \text{CollectAndBorrow}\ (\text{user}, \text{book}) \Rightarrow
\begin{verbatim}
var \ bb \ in
  response := \text{NotRegisteredUser} ;
  bb \leftarrow \text{user\_registered}\ (\text{user}) ;
  if \( bb = \text{TRUE} \) then
    response := \text{BookNotInLibrary} ;
    bb \leftarrow \text{book\_in\_books}\ (\text{book})
  end ;
  if \( bb = \text{TRUE} \) then
    response := \text{NotForCollection} ;
\end{verbatim}

30
bb ← book_for_collection_by_user ( book , user )
end ;
if bb = TRUE then
response := BookCopyOnLoan ;
bb ←¬ not_book_copy_onloan_to_user ( book , user )
end ;
if bb = TRUE then
CollectnBorrow ( user , book ) ; response := OK
end
end

response ←→ CancelBookReserve ( user , book ) ≡

var bb in
response := NotRegisteredUser ;
bb ←¬ user_registered ( user ) ;
if bb = TRUE then
response := NotReservedForUser ;
bb ←¬ book_reserved ( book )
end ;
if bb = TRUE then
response := NotReservedForUser ;
bb ←¬ book_reserved_by_user ( book , user )
end ;
if bb = TRUE then
response := OK ; CancelReservation ( user , book )
end
end
end

response ←→ UnCollectBook ( user , book ) ≡

var bb in
response := NotRegisteredUser ;
bb ←¬ user_registered ( user ) ;
if bb = TRUE then
response := NotForCollection ;
bb ←¬ book_for_collection_by_user ( book , user )
end ;
if bb = TRUE then
response := OK ;
UnCollect ( user , book )
end
end
END
9 Implementing LibraryPlus

To complete the implementation we now have to implement LibraryPlus. This is not currently presented here.