Outline I

1. Objectives of this lecture
2. A Small library development
3. What is a book?
   - A Book Context machine
   - LibraryBook context machine
   - Book Services
4. User Services
   - The UserServices machine
   - Proof obligation problem
   - Explanation of the problem
5. Book reservation
   - Modelling the reservation queue
   - Modelling books for collection
   - The Reservation machine
Outline II

6 Browsing Books in the Library
   - Putting it all together
   - The Library machine

7 Preparing for a Robust machine
   - The Bool_TYPE machine
   - The LibraryPlus machine
   - The LibraryAPI machine

8 Implementing LibraryAPI
   - The LibraryAPIII machine

9 Implementing LibraryPlus
Objectives of this lecture

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


```
MACHINE Book_ctx
SETS BOOK
END
```
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.


MACHINE $Book_ctx$
SETS $BOOK$
END
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.


```plaintext
MACHINE Book_ctx
SETS BOOK
END
```
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
```
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.

```plaintext
MACHINE LibraryBook_ctx
SETS LIBRARYBOOK
END
```
We will use a set \textit{LIBRARYBOOK} to model the identity of copies of books acquired by the library.

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

\textit{LIBRARYBOOK} models unique book identifiers that are assigned each book in the library’s collection.
We will use a set \( \text{LIBRARYBOOK} \) to model the identity of copies of books acquired by the library.

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

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

\[
\begin{align*}
\text{MACHINE} & \quad \text{LibraryBook\_ctx} \\
\text{SETS} & \quad \text{LIBRARYBOOK} \\
\text{END}
\end{align*}
\]
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
```
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)
CONSTRAINTS maxbooks ∈ N₁
SEES Book_ctx, LibraryBook_ctx
VARIABLES
  librarybooks,
  books,
  book_copies,
  books_on_shelf
Book Services III

IN Variant

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.

librarybooks ⊆ LIBRARYBOOK ∧
card (librarybooks) ≤ maxbooks ∧
books ⊆ BOOK ∧
book_copies ∈ librarybooks → books ∧
books_on_shelf ⊆ librarybooks

Initialisation

librarybooks , books := {}, {}  ||
books_on_shelf , book_copies := {}, {}
Book Services IV

OPERATIONS
Book Services V

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 ) ≜

pre book ∈ BOOK ∧ card ( librarybooks ) ≠ maxbooks then

any bookid

where bookid ∈ LIBRARYBOOK − librarybooks then

librarybooks := librarybooks ∪ { bookid } ∥
books := books ∪ { book } ∥
books_on_shelf := books_on_shelf ∪ { bookid } ∥
book_copies ( bookid ) := book

end

end ;
Book Services VI

books ← LocateBook(book)
returns the set of library book identifiers for book ∈ BOOK

bookids ← LocateBook(book) ≜
pre book ∈ BOOK then
    bookids := book_copies⁻¹ [ { book } ]
end ;
**AddBookToShelf** *(book)*, **RemoveBookFromShelf** *(book)*

Utility Operations to add a book to and remove a book from the library shelf.

These operations enable other machines to modify *books_on_shelf*. They will not finally be exported to the interface.

**AddBookToShelf** *(book)* ≜

- pre *book* ∈ LIBRARYBOOK ∧ *book* ∈ librarybooks then
  - *books_on_shelf* := *books_on_shelf* ∪ { *book* }

end ;

**RemoveBookFromShelf** *(book)* ≜

- pre *book* ∈ LIBRARYBOOK ∧ *book* ∈ librarybooks then
  - *books_on_shelf* := *books_on_shelf* − { *book* }

end

END
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_ctx*, to contain the set and a constant *anyuser*, an arbitrary member of *LIBRARYUSER*. 
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_ctx`, to contain the set and a constant `anyuser`, an arbitrary member of `LIBRARYUSER`. 
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_ctx*, to contain the set and a constant *anyuser*, an arbitrary member of *LIBRARYUSER*. 
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_ctx*, to contain the set and a constant *anyuser*, an arbitrary member of *LIBRARYUSER*. 
The *UserService* 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_ctx*, to contain the set and a constant *anyuser*, an arbitrary member of *LIBRARYUSER*. 
MACHINE  User_ctx
SETS  LIBRARYUSER

OPERATIONS

\[user \leftarrow \text{anyuser} \equiv \]
\[user \in LIBRARYUSER\]

END
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  \textit{UserServices} ( \textit{maxuser} , \textit{maxlibrary} )

CONSTRAINTS
\begin{align*}
\text{maxuser} & \in \mathbb{N} \setminus 1 \\
\text{maxlibrary} & \in \mathbb{N} \setminus 1
\end{align*}

SEES
\begin{align*}
\text{User}_\text{ctx} \\
\text{Book}_\text{ctx} , \text{LibraryBook}_\text{ctx}
\end{align*}

EXTENDS \textit{BookServices} ( \textit{maxlibrary} )

VARIABLES
\begin{align*}
\text{libraryusers} \\
\text{librarybooks_onloan}
\end{align*}
The UserRegistration machine III

INvariant

We will model the set of registered users by the set \textit{libraryusers}:

\begin{align*}
\text{libraryusers} & \subseteq \text{LIBRARYUSER} \land \\
\text{card (libraryusers)} & \leq \text{maxuser} \land \\
\text{each library book is on loan to at most one library user} & \\
\text{librarybooks\_onloan} & \in \text{librarybooks} \rightarrow \text{libraryusers} \land \\
\text{a book cannot be on loan and on the shelf at the same time} & \\
\text{dom (librarybooks\_onloan)} & \cap \text{books\_on\_shelf} = \emptyset \land \\
\text{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} &
\end{align*}
The UserRegistration machine IV

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

As illustrated in the above, it is useful to have a variable \textit{books\_on\_loan}, but this variable is dependent on and completely defined by other variables. See Definitions at the end of this machine.

**ASSERTIONS**

\[ \text{books\_on\_loan} \in \text{books} \leftrightarrow \text{libraryusers} \land \]

\[ \text{ran}(\text{books\_on\_loan}) = \text{ran}(\text{librarybooks\_onloan}) \land \]

\[ \text{dom}(\text{librarybooks\_onloan}) \cap \text{books\_on\_shelf} = \{\} \]

**INITIALISATION** \text{libraryusers} , \text{librarybooks\_onloan} := \{\} , \{\}

OPERATIONS

newuser ← NewUser

Requirement: to register a new user by allocating a new user token.
Precondition: not all USER tokens have been allocated.

newuser ← NewUser \equiv
pre \text{ card}(\text{libraryusers}) \neq \text{maxuser}
then
  any user
where user \in LIBRARYUSER \setminus \text{libraryusers} then
  libraryusers := libraryusers \cup \{user\} \parallel
  newuser := user
end
The UserRegistration machine VI

\[
\text{Borrow}(\text{user}, \text{book})
\]

**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.

\[
\text{Borrow} \ ( user \ , \ book \ ) \ \equiv \\
\text{pre} \ \begin{array}{c}
\text{user} \in \text{libraryusers} \\
\text{book} \in \text{librarybooks} \land \text{book} \notin \text{dom} ( \text{librarybooks\_onloan} ) \land \\
\text{user} \notin \text{books\_onloan} [ \{ \text{book\_copies} ( \text{book} ) \} ] \text{ then} \\
\text{RemoveBookFromShelf} ( \text{book} ) \parallel \\
\text{librarybooks\_onloan} ( \text{book} ) : = \text{user}
\end{array}
\]

end ;
Return\( (book) \)

*Requirement:* to record the return of a book. The book is placed on the library shelf.

*Precondition:* the book must be on loan.

\[
\text{Return}\ (\ book \ ) \equiv \\
\text{pre } \ book \in \text{dom}\ (\ librarybooks\_onloan \ ) \text{ then } \\
\text{AddBookToShelf}\ (\ book \ ) \parallel \\
\text{librarybooks\_onloan} := \{ \ book \} \triangleleft \text{librarybooks\_onloan} \\
\text{end } ;
\]
The UserRegistration machine VIII

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

users ← Borrowed(book) ≜
  pre book ∈ BOOK then
    users := books_onloan [ { book } ]
  end ;

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.
The UserRegistration machine IX

\[
\text{CancelLoan} \ ( book ) \ \triangleq \\
\text{pre } \ book \in \text{LIBRARYBOOK } \text{then } \\
\quad \text{librarybooks\_onloan} := \{ \ book \} \leftarrow \text{librarybooks\_onloan} \\
\text{end} \\
\text{DEFINITIONS} \\
\quad \text{books\_onloan} \ \triangleq \ book\_copies^{-1} ; \text{librarybooks\_onloan} \\
\text{END}
\]
There is a proof obligation for the operation AddBookToShelf which is impossible to discharge.

Why?
There is a proof obligation for the operation AddBookToShelf which is impossible to discharge.

Why?
There is a proof obligation for the operation AddBookToShelf which is impossible to discharge.

Why?
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 \textit{AddBookToShelf} and this operation can break the invariant of \textit{Borrowing}.

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

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

This operation has been simply promoted to illustrate this type of problem; an alternative would be to repackaged the \textit{AddBookToShelf} operation with a strengthened precondition to ensure safety.
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 \textit{AddBookToShelf} and this operation can break the invariant of \textit{Borrowing}.

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

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

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

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

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

This operation has been simply promoted to illustrate this type of problem; an alternative would be to repackage the AddBookToShelf operation with a strengthened precondition to ensure safety.
Explaination 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 AddBookToShelf and this operation can break the invariant of Borrowing.

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

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

This operation has been simply promoted to illustrate this type of problem; an alternative would be to repackage the AddBookToShelf operation with a strengthened precondition to ensure safety.
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 `AddBookToShelf` and this operation can break the invariant of `Borrowing`.

We are promoting this operation because we will need it in subsequent machines that will include `Borrowing`.

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

This operation has been simply promoted to illustrate this type of problem; an alternative would be to repackage the `AddBookToShelf` operation with a strengthened precondition to ensure safety.
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 \textit{AddBookToShelf} and this operation can break the invariant of \textit{Borrowing}.

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

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

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

**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.
Book reservation

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.
**Book reservation**

**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.
Book reservation

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.
Book reservation

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.
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.
Book reservation

**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.
Book reservation

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

We will model this using a partial function from books to a sequence of 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.

\[ \text{reserved} \in \text{books} \mapsto \text{iseq} ( \text{libraryusers} ) \land \]  
\[ \text{reserved} \in \text{books} \mapsto \text{seq}_1 ( \text{libraryusers} ) \]

The size of any queue may not exceed maxreserve.

\[ \forall \text{book} . \ ( \text{book} \in \text{dom} ( \text{reserved} ) \Rightarrow \]  
\[ \text{size ( reserved ( book ) ) } \leq \text{maxreserve} ) \]
Modelling the reservation queue

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

We will model this using a partial function from *books* to a sequence of *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.

\[
\text{reserved} \in \text{books} \rightarrow \text{iseq (libraryusers)} \land \\
\text{reserved} \in \text{books} \rightarrow \text{seq}_1 (\text{libraryusers})
\]

The size of any queue may not exceed *maxreserve*.

\[
\forall \text{book} . \ ( \text{book} \in \text{dom (reserved)} \Rightarrow \\
\text{size (reserved (book))} \leq \text{maxreserve})
\]
Modelling the reservation queue

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

We will model this using a partial function from *books* to a sequence of *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.

\[
\text{reserved} \in \text{books} \rightarrow \text{iseq} ( \text{libraryusers} ) \land \\
\text{reserved} \in \text{books} \rightarrow \text{seq}_1 ( \text{libraryusers} )
\]

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

\[
\forall \text{book} . \ ( \text{book} \in \text{dom} ( \text{reserved} ) \Rightarrow \size ( \text{reserved} ( \text{book} ) ) \leq \text{maxreserve} )
\]
Modelling the reservation queue

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

We will model this using a partial function from books to a sequence of 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.

\[ \text{reserved} \in \text{books} \mapsto \text{iseq ( libraryusers )} \land \]
\[ \text{reserved} \in \text{books} \mapsto \text{seq}_1 ( \text{libraryusers} ) \]

The size of any queue may not exceed maxreserve.

\[ \forall \text{book} . ( \text{book} \in \text{dom ( reserved )} \Rightarrow \text{size ( reserved ( book ) )} \leq \text{maxreserve} ) \]
Modelling the reservation queue

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

We will model this using a partial function from *books* to a sequence of *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.

\[
\text{reserved} \in \text{books} \mapsto \text{iseq} \left( \text{libraryusers} \right) \land \\
\text{reserved} \in \text{books} \mapsto \text{seq}_1 \left( \text{libraryusers} \right)
\]

The size of any queue may not exceed *maxreserve*.

\[
\forall \text{book} . \left( \text{book} \in \text{dom} \left( \text{reserved} \right) \Rightarrow \right) \\
\text{size} \left( \text{reserved} \left( \text{book} \right) \right) \leq \text{maxreserve}
\]
For each reserved book will have a queue of registered users.

We will model this using a partial function from *books* to a sequence of *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.

\[
\text{reserved} \in \text{books} \rightarrow \text{iseq (libraryusers)} \land \\
\text{reserved} \in \text{books} \rightarrow \text{seq}_1 (\text{libraryusers})
\]

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

\[
\forall \text{book} . (\text{book} \in \text{dom (reserved)} \Rightarrow \text{size (reserved (book))} \leq \text{maxreserve})
\]
Modelling the reservation queue

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

We will model this using a partial function from \textit{books} to a sequence of \textit{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.

\[\textit{reserved} \in \textit{books} \leftrightarrow \text{iseq ( libraryusers )} \land\]
\[\textit{reserved} \in \textit{books} \leftrightarrow \text{seq}_1 ( \text{libraryusers} )\]

The size of any queue may not exceed \textit{maxreserve}.

\[\forall \textit{book} . ( \textit{book} \in \text{dom ( reserved )} \Rightarrow \text{size ( reserved ( book ) )} \leq \text{maxreserve} )\]
Modelling the reservation queue

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

We will model this using a partial function from $books$ to a sequence of $libraryusers$.

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.

\[
\text{reserved} \in books \leftrightarrow \text{iseq} ( libraryusers ) \land \\
\text{reserved} \in books \leftrightarrow \text{seq}_1 ( libraryusers )
\]

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

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

- A relation from books to the actual copies of those books being held for collection.
  \[ books \to collect \subseteq books \leftrightarrow librarybooks \]

- A relation from library users to the books waiting collection.
  \[ users \to collect \subseteq library\_users \leftrightarrow \text{dom}(books \to collect) \]

Other constrains will be given in the invariant.
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
   \[ books_{to\_collect} \in books \leftrightarrow librarybooks \]

2. a relation from library users to the books waiting collection
   \[ users_{to\_collect} \in library\_users \leftrightarrow \text{dom} ( books_{to\_collect} ) \]

Other constrains will be given in the invariant.
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
   \[ \text{books	extunderscore to	extunderscore collect} \in \text{books} \leftrightarrow \text{librarybooks} \]

2. a relation from library users to the books waiting collection
   \[ \text{users	extunderscore to	extunderscore collect} \in \text{library	extunderscore users} \leftrightarrow \text{dom ( books	extunderscore to	extunderscore collect )} \]

Other constrains will be given in the invariant.
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
   
   \[ \text{books\_to\_collect} \in \text{books} \leftrightarrow \text{librarybooks} \]

2. a relation from library users to the books waiting collection
   
   \[ \text{users\_to\_collect} \in \text{library\_users} \leftrightarrow \text{dom (books\_to\_collect)} \]

Other constrains will be given in the invariant.
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
   \[ \text{books}\_\text{to}\_\text{collect} \in \text{books} \leftrightarrow \text{librarybooks} \]

2. a relation from library users to the books waiting collection
   \[ \text{users}\_\text{to}\_\text{collect} \in \text{library}\_\text{users} \leftrightarrow \text{dom} ( \text{books}\_\text{to}\_\text{collect} ) \]

Other constrains will be given in the invariant.
The Reservation machine I

The *Reservation* machine partially extends the *Borrowing* machine. Reservation interacts with borrowing and returning of books, so the *Borrow* and *Return* operations will be redefined.

**MACHINE** *Reservation* (maxuser, maxlibrary, maxreserve)

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

**SEES** User_ctx, Book_ctx, LibraryBook_ctx

**INCLUDES** UserServices (maxuser, maxlibrary)

**PROMOTES** AddNewBook, Borrowed
The Reservation machine II

VARIABLES
reserved, books_to_collect

INVARIANT
reserved ∈ books ↔ iseq ( libraryusers ) ∧
reserved ∈ books ↔ seq₁ ( libraryusers ) ∧
∀ book . ( book ∈ dom ( reserved ) ⇒
size ( reserved ( book ) ) ≤ maxreserve ) ∧

Reserved books must be either on loan or awaiting collection

dom ( reserved ) ⊆ dom ( books_onloan ) ∪
ran ( dom ( books_to_collect ) ) ∧
books_to_collect ∈ libraryusers × books ↦ librarybooks ∧

Any user has at most one book to collect

dom ( books_to_collect ) ∈ libraryusers ↦ books ∧
for all book, user pairs, the books set aside for collection are library copies of that book

$$\forall (\text{user}, \text{book}) . (\text{user} \in \text{libraryusers} \land \text{book} \in \text{books} \land$$

$$\text{user} \mapsto \text{book} \in \text{dom (books_to_collect)} \Rightarrow$$

$$\text{books_to_collect (user} \mapsto \text{book}) \in \text{book_copies}^{-1} [\{\text{book}\}] ) \land$$

library books awaiting collection cannot be on the shelf

$$\text{ran (books_to_collect)} \cap \text{books_on_shelf} = \{\} \land$$

nor can they be on loan

$$\text{ran (books_to_collect)} \cap \text{dom (librarybooks_onloan)} = \{\} \land$$

books awaiting collection by a user cannot be on loan to the same user
The Reservation machine IV

∀ (book, user). (book ∈ books ∧ user ∈ libraryusers ∧ 
user ↦→ book ∈ dom (books_to_collect) ⇒
user ∉ books_onloan [ {book}] ) ∧

books awaiting collection by a user cannot be reserved by the same user

∀ (book, user). (book ∈ books ∧ user ∈ libraryusers ∧ 
user ↦→ book ∈ dom (books_to_collect) ⇒
user ∉ ran (reserved (book)) )

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) ∧
dom(librarybooks_onloan) = { }
∀ book. ( book ∈ ran (books_to_collect) ⇒ book ∉ dom (librarybooks_onloan) )

INITIALISATION
reserved, books_to_collect := {}, {}

OPERATIONS
The *Reservation* machine VI

Reserve(*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
The Reservation machine VII

\[ \text{Reserve} \ ( user \ , \ book ) \ \overset{\text{def}}{=} \]

\[ \text{pre} \quad user \in libraryusers \ \land \ book \in books \ \land \]

\[ book \in \text{dom} \ ( books_{onloan} ) \cup \text{ran} \ ( \text{dom} \ ( books_{to\_collect} ) ) \land \]

\[ book_{\text{copies}}^{-1} \ [ \{ \text{book} \} ] \subseteq \text{dom} \ ( librarybooks_{onloan} ) \cup \]

\[ \text{ran} \ ( books_{to\_collect} ) \land \]

\[ user \notin books_{onloan} [ \{ \text{book} \} ] \land \]

\[ user \mapsto book \in \text{dom} \ ( books_{to\_collect} ) \land \]

\[ ( book \in \text{dom} \ ( reserved ) \Rightarrow \]

\[ \text{size} \ ( reserved \ ( book ) ) \neq \text{maxreserve} ) \land \]

\[ ( book \in \text{dom} \ ( reserved ) \Rightarrow user \notin \text{ran} \ ( reserved \ ( book ) ) ) \]
The Reservation machine VIII

\[
    \text{then if } \quad book \notin \text{dom (reserved)} \\
    \text{then } \quad \text{reserved} (book) := [\text{user}] \\
    \text{else } \quad \text{reserved} (book) := \text{reserved} (book) \leftarrow \text{user} \\
    \text{end} \\
    \text{end ;}
\]
Borrow\(_1\)(user, book): An upgrade of Borrow, this operation strengthens the precondition to take account of the new state.

\[
\text{Borrow}_1 (\text{user}, \text{book}) \triangleq \\
\text{pre } \text{user} \in \text{libraryusers} \land \text{book} \in \text{librarybooks} \land \\
\text{book} \notin \text{dom (librarybooks\_onloan)} \land \\
\text{user} \notin \text{books\_onloan} [\{ \text{book\_copies (book) } \}] \land \\
\text{book} \notin \text{ran (books\_to\_collect)} \land \\
\text{user} \mapsto \text{book\_copies (book)} \notin \text{dom (books\_to\_collect)} \text{ then }
\text{Borrow (user, book)}
\text{ end ;}
\]
The *Reservation* machine X

*Return1*(*book*): A replacement for the *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 *Return* operation.
The *Reservation* machine XI

\[
\text{Return1 ( } book \text{ ) } \triangleq \\
\text{pre } \quad book \in \text{dom ( librarybooks\_onloan ) then any bbook where bbook } \in \text{BOOK } \land \\
\quad bbook = \text{book\_copies ( } book \text{ ) then if bbook } \in \text{dom ( reserved ) then } \\
\quad \text{CancelLoan ( } book \text{ ) } \parallel \\
\text{books\_to\_collect ( first ( reserved ( bbook ) ) } \mapsto \\
\quad bbook ) := book \quad \parallel 
\]
if size (reserved (bbook)) = 1 then
    reserved := \{ bbook \} \leftarrow reserved
else
    reserved (bbook) := tail (reserved (bbook))
end
else
    Return (book)
end
end ;
The Reservation machine XIII

\( \text{CancelReservation(user, book)} \): 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.
The **Reservation** machine XIV

\[
\text{CancelReservation} \ ( \ user \ , \ book \ ) \triangleq \\
\text{pre} \ \ user \in libraryusers \land book \in \text{dom} \ ( \ reserved \ ) \land \\
user \in \text{ran} \ ( \ reserved \ ( \ book \ ) ) \text{ then} \\
\text{if} \ \ \text{size} \ ( \ reserved \ ( \ book \ ) ) = 1 \ \text{then} \\
\text{reserved} := \{ \ book \} \triangleleft \text{reserved} \\
\text{else any} \ \ pos \ , \ list \ \text{where} \\
\ \ pos = ( \ reserved \ ( \ book \ ) )^{-1} ( \ user ) \land \\
list = reserved \ ( \ book \ ) \text{ then} \\
\text{reserved} \ ( \ book \ ) := \\
\quad list \uparrow pos - 1 \bigcirc ( \ list \downarrow pos ) \\
\text{end} \\
\text{end} \\
\text{end} ;
\]
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.

\[
\text{CollectnBorrow}\ (user, book) \equiv \\
\text{pre} \quad user \in \text{libraryusers} \land book \in \text{books} \land \\
user \mapsto book \in \text{dom (books_to_collect)} \land \\
user \not\in \text{books_onloan [ \{ book \} ] then} \\
\text{any bookid where} \quad bookid \in \text{librarybooks} \land \\
book\_copies (bookid) = book \land \\
bookid = \text{books_to_collect (user \mapsto book)} \text{ then} \\
\text{Borrow (user, bookid) ||} \\
\text{books_to_collect := \{ user \mapsto book \} \triangleleft books_to_collect} \\
\text{end} \\
\text{end ;}
\]
The Reservation machine XVI
**UnCollect**(user, book): Allow a user to 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.
The **Reservation** machine XVIII

\[ \text{UnCollect} (user, book) \equiv \]

\[
\text{pre} \quad user \in libraryusers \land book \in books \land
\]

\[
user \mapsto book \in \text{dom} (books_to_collect) \land
\]

\[
\text{any} \quad bookid \text{ where } bookid \in librarybooks \land
\]

\[
bookid = books_to_collect (user \mapsto book) \land
\]

\[
\text{if } book \in \text{dom} (reserved) \text{ then}
\]

\[
books_to_collect := \{ user \mapsto book \} \leftarrow books_to_collect \cup
\]

\[
\{ \text{first} (reserved (book)) \mapsto book \mapsto bookid \} \lor
\]

\[
\text{if } \text{size} (reserved (book)) = 1 \text{ then}
\]

\[
reserved := \{ book \} \leftarrow reserved
\]

\[
\text{else } reserved (book) := \text{tail} (reserved (book))
\]

end
else  books_to_collect := books_to_collect –
    { user ↦ book ↦ bookid } || AddBookToShelf ( bookid )
end
end
end
END
**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.
Browsing Books in the Library II

**MACHINE** `Browsing (maxuser, maxlibrary, maxreserve)`

**CONSTRAINTS**

- `maxuser ∈ ℕ_1 ∧`
- `maxlibrary ∈ ℕ_1 ∧`
- `maxreserve ∈ ℕ_1`

**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 \( \text{ran}(\text{books_to_collect}) \land \text{books_on_shelf} = \{\} \)

\( \forall \text{book} . (\text{book} \in \text{ran}(\text{books_to_collect}) \Rightarrow \text{book} \notin \text{books_on_shelf}) \land \)

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** browsing := \{\}
Browsing Books in the Library IV

OPERATIONS

\( \text{BeginBrowse}(book) \): Begin browsing a book in the library.

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

\[
\text{BeginBrowse} \ (book) \ \equiv
\]

\[
\text{pre} \quad book \in books\_on\_shelf \ \land \ book \notin browsing \ \text{then}
\]

\[
browsing := browsing \cup \{ \text{book} \}
\]

\text{end} ;
EndBrowse\(^{(book)}\): End the browsing of a book in the library.

\[
\text{EndBrowse} (book) \triangleq \\
\text{pre } book \in \text{browsing } \text{then} \\
\text{browsing} := \text{browsing} \smallsetminus \{ \text{book} \} \\
\text{end} ;
\]
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.

\[
\text{RemoveBook} (\ book\ ) \triangleq \\
\begin{align*}
\text{pre} & \quad book \in LIBRARYBOOK \land \\
& \quad ( book \in librarybooks \Rightarrow book \in \text{dom} ( librarybooks\_onloan ) ) \\
\text{then} & \quad \text{skip} \\
\text{end} ;
\end{align*}
\]
Browsing Books in the Library VII

\[ \text{Borrow2}(user, book): \text{The same as Borrow1}(user, book) \text{ except it also removes book from browsing.} \]

\[
\text{Borrow2}(user, book) \triangleq \\
\quad \text{pre} \quad user \in libraryusers \land book \in librarybooks \land \\
\quad \quad book \notin \text{dom}(librarybooks_{onloan}) \land \\
\quad \quad user \notin books_{onloan}[\{book\_copies(book)\}] \land \\
\quad \quad book \notin \text{ran}(books\_to\_collect) \land \\
\quad \quad user \mapsto book\_copies(book) \notin \text{dom}(books\_to\_collect) \\
\quad \text{then} \\
\quad \quad \text{Borrow1}(user, book) \parallel \\
\quad \quad browsing := browsing - \{book\} \\
\quad \text{end}
\]

END
Objectives:

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

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

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)
The Library machine II

PROMOTES

AddNewBook, Borrow2, Return1, Borrowed, Reserve, CollectnBorrow, CancelReservation, UnCollect, BeginBrowse, EndBrowse, RemoveBook
The Library machine III

**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.

\[librarybooks = books\_on\_shelf \cup \text{dom}(librarybooks\_onloan) \cup \text{ran}(book\_loan)\]
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.
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.
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.
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.
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 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.
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 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 BorrowedNextNext, that achieve the same effect by iteration.
A library machine, *Bool_TYPE*, is also seen. This machine contains the enumerated set $BOOL = \{ \text{FALSE}, \text{TRUE} \}$, 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 $bool$:

$$\text{var} := bool(\text{expr}) \triangleq \begin{cases} \text{IF expr THEN var} := \text{TRUE} \\ \text{ELSE var} := \text{FALSE} \end{cases} \text{END}$$

```
MACHINE Bool_TYPE
SETS BOOL = \{ \text{FALSE} , \text{TRUE} \}
END
```
The \texttt{Bool\_TYPE} machine

A library machine, \texttt{Bool\_TYPE}, is also seen. This machine contains the enumerated set \texttt{BOOL} = \{\texttt{FALSE}, \texttt{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}:

\begin{verbatim}
var := \texttt{bool}(expr) \equiv IF \ expr \ THEN \ var := \texttt{TRUE} \\
ELSE \ var := \texttt{FALSE} \\
END
\end{verbatim}

\texttt{MACHINE \texttt{Bool\_TYPE}}
\texttt{SETS \texttt{BOOL} = \{ \texttt{FALSE} , \texttt{TRUE} \} }
\texttt{END}
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 `bool`:

$$\begin{align*}
\text{var} & := \text{bool}(\text{expr}) \triangleq \\
& \quad \text{IF } \text{expr} \text{ THEN } \text{var} := \text{TRUE} \\
& \quad \text{ELSE } \text{var} := \text{FALSE} \\
& \quad \text{END}
\end{align*}$$

$$\text{MACHINE } \text{Bool\_TYPE} \\
\text{SETS } BOOL = \{ \text{FALSE} , \text{TRUE} \} \\
\text{END}$$
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.

**MACHINE**  
*LibraryPlus* ( *maxbook*, *maxuser*, *maxlibrary*,  
  *maxreserve*, *maxiter*)  

**CONSTRAINTS**  
  
  *maxbook* ∈ $\mathbb{N}_1$ ∧  
  *maxuser* ∈ $\mathbb{N}_1$ ∧  
  *maxlibrary* ∈ $\mathbb{N}_1$ ∧  
  *maxreserve* ∈ $\mathbb{N}_1$ ∧  
  *maxiter* ∈ $\mathbb{N}_1$,  

**SEES**
The **LibraryPlus** machine II

*User_CTX*,

*Book_TYPE*,

*LibraryBook*,

*IterKey*,

*Bool_TYPE*

**INCLUDES** Library (maxbook, maxuser, maxlibrary, maxreserve)

**PROMOTES**

AddNewBook, Borrow2, Return1, Reserve, CollectnBorrow, CancelReservation, UnCollect, BeginBrowse, EndBrowse, RemoveBook, NewUser

**VARIABLES** bkeys, borrowers

**INVARIANT**

\[ bkeys \subseteq \text{ITERKEY} \land \text{card}(bkeys) \leq \text{maxiter} \land \]

\[ \text{borrowers} \in bkeys \rightarrow \mathbb{P}(\text{libraryusers}) \]

**INITIALISATION** bkeys, borrowers := \{\}, {}
The LibraryPlus machine III
The LibraryPlus machine IV

Sets cannot be transmitted, in either direction, through the final API or UAI interface, so we will iterate the Borrowed operation.

\[
\text{user}, \text{iterkey} \leftarrow \text{BorrowedFirst}(\text{book}) \equiv \\
\text{pre } \text{book} \in \text{BOOK} \land \text{card}(\text{bkeys}) \neq \text{maxiter} \land \\
\text{books\_onloan}[\{\text{book}\}] \neq \{\} \text{ then } \\
\text{any } \text{bkey}, \text{userid } \text{where } \\
\text{bkey} \in \text{ITERKEY} \land \text{bkey} \in \text{ITERKEY} - \text{bkeys} \land \\
\text{userid} \in \text{libraryusers} \land \\
\text{userid} \in \text{books\_onloan}[\{\text{book}\}] \text{ then }
\]
The LibraryPlus machine V

```
if \( \text{books\_onloan}[\{\text{book}\}] - \{\text{userid}\} \neq \{\}\) then
  \(bkeys := bkeys \cup \{bkey\}\) \parallel
  \(\text{borrowers}(bkey) := \text{books\_onloan}[\{\text{book}\}] - \{\text{userid}\}\)
end \parallel
user, iterkey := userid, bkey
end
end ;
```
The LibraryPlus machine VI

\[ \text{user} \leftarrow \text{BorrowedNext} \ (bkey) \triangleq \]
\[
\text{pre} \quad bkey \in \text{ITERKEY} \land bkey \in \text{bkeys} \quad \text{then}
\]
\[
\text{any} \quad \text{userid} \quad \text{where} \quad \text{userid} \in \text{libraryusers} \land
\]
\[
\text{userid} \in \text{borrowers} \ (bkey) \text{ then}
\]
\[
\text{if} \quad \text{borrowers} \ (bkey) - \{ \text{userid} \} \neq \{ \} \text{ then}
\]
\[
\text{borrowers} \ (bkey) := \text{borrowers} \ (bkey) - \{ \text{userid} \}
\]
\[
\text{else} \quad \text{borrowers} := \{ bkey \} \leftarrow \text{borrowers}
\]
\[
\text{end} \quad \|
\]
\[
\text{user} := \text{userid}
\]
\[
\text{end}
\]
\[
\text{end} ;
\]
The \textit{LibraryPlus} machine VII

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

\[\text{ok} \leftarrow \text{users\_not\_full} \triangleq\]
\[\text{begin} \quad \text{ok} := \text{bool} \left( \text{card} \left( \text{libraryusers} \right) \neq \text{maxuser} \right) \text{ end} ;\]
\[\text{ok} \leftarrow \text{user\_registered} \left( \text{user} \right) \triangleq\]
\[\text{pre} \quad \text{user} \in \text{LIBRARYUSER}\]
\[\text{then} \quad \text{ok} := \text{bool} \left( \text{user} \in \text{libraryusers} \right) \]
\[\text{end} ;\]
\[\text{ok} \leftarrow \text{book\_in\_library} \left( \text{book} \right) \triangleq\]
\[\text{pre} \quad \text{book} \in \text{LIBRARYBOOK}\]
\[\text{then} \quad \text{ok} := \text{bool} \left( \text{book} \in \text{librarybooks} \right) \]
\[\text{end} ;\]
\[\text{ok} \leftarrow \text{book\_in\_books} \left( \text{book} \right) \triangleq\]
\[\text{pre} \quad \text{book} \in \text{BOOK}\]
The *LibraryPlus* machine VIII

\[\text{then } \text{ok} := \text{bool } ( \text{book} \in \text{books} ) \]

\[\text{end ;} \]

\[\text{ok } \leftarrow \text{library_not_full } \triangleq \]

\[\text{begin } \text{ok} := \text{bool } ( \text{card } ( \text{librarybooks} ) \neq \text{maxlibrary} ) \text{ end ;} \]

\[\text{ok } \leftarrow \text{librarybook_onloan } ( \text{book} ) \triangleq \]

\[\text{pre } \text{book} \in \text{LIBRARYBOOK} \]

\[\text{then } \text{ok} := \text{bool } ( \text{book} \in \text{dom } ( \text{librarybooks_onloan} ) ) \]

\[\text{end ;} \]

\[\text{ok } \leftarrow \text{book_onloan } ( \text{book} ) \triangleq \]

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

\[\text{then } \text{ok} := \text{bool } ( \text{book} \in \text{dom } ( \text{books_onloan} ) ) \]

\[\text{end ;} \]

\[\text{ok } \leftarrow \text{not_book_onloan_to_user } ( \text{book} , \text{user} ) \triangleq \]

\[\text{pre } \text{book} \in \text{LIBRARYBOOK} \land \text{user} \in \text{LIBRARYUSER} \]

\[\text{then } \]

\[\text{ok} := \text{bool } ( \text{user} \notin \text{books_onloan} [ \{ \text{book_catalogue } ( \text{book} ) \} ] ) \]

\[\text{end ;} \]
The LibraryPlus machine IX

\[ \text{ok} \leftarrow \text{not_book_copy_onloan_to_user} \left( \text{book}, \text{user} \right) \triangleq \]
\[ \begin{align*}
\text{pre} & \quad \text{book} \in \text{BOOK} \land \text{user} \in \text{LIBRARYUSER} \\
\text{then} & \quad \text{ok} := \text{bool} \left( \text{user} \notin \text{books_onloan} \left\{ \text{book} \right\} \right) \\
\text{end} ;
\end{align*} \]

\[ \text{ok} \leftarrow \text{book_onloan_or_notlibrary} \left( \text{book} \right) \triangleq \]
\[ \begin{align*}
\text{pre} & \quad \text{book} \in \text{LIBRARYBOOK} \\
\text{then} & \quad \\
\text{end} ;
\end{align*} \]

\[ \text{ok} \leftarrow \text{book_onloan_or_collect} \left( \text{book} \right) \triangleq \]
\[ \begin{align*}
\text{pre} & \quad \text{book} \in \text{BOOK} \\
\text{then} & \quad \\
\text{end} ;
\end{align*} \]
The *LibraryPlus* machine $X$

\[
\text{ok } \leftarrow \ \text{not_librarybook_onloan_or_collect} \ (book) \ \triangleq \\
\text{pre } \ \text{book} \in \text{LIBRARYBOOK} \\
\text{then} \\
\text{ok} := \text{bool} \ (\ \text{book} \notin \text{dom} \ (\text{librarybooks_onloan}) \land \\
\text{book} \notin \text{ran} \ (\text{books_to_collect}) ) \\
\text{end} ; \\
\text{ok } \leftarrow \ \text{reservation_full} \ (book) \ \triangleq \\
\text{pre } \ \text{book} \in \text{BOOK} \\
\text{then} \\
\text{ok} := \text{bool} \ (\ \text{book} \in \text{dom} \ (\text{reserved}) \land \\
\text{size} \ (\text{reserved} \ (\text{book})) = \text{maxreserve}) \\
\text{end} ; \\
\text{ok } \leftarrow \ \text{book_reserved} \ (book) \ \triangleq \\
\text{pre } \ \text{book} \in \text{BOOK} \\
\text{then} \ \text{ok} := \text{bool} \ (\ \text{book} \in \text{dom} \ (\text{reserved}) ) \\
\text{end} ;
\]
The **LibraryPlus** machine XI

\[
\text{ok} \leftarrow \text{book\_reserved\_by\_user} \ ( \text{book}, \text{user} ) \ \cong \\
\text{pre} \quad \text{book} \in \text{BOOK} \land \text{user} \in \text{LIBRARYUSER} \\
\text{then} \\
\quad \text{ok} := \text{bool} \ ( \text{book} \in \text{dom} \ ( \text{reserved} ) \land \\
\quad \quad \text{user} \in \text{ran} \ ( \text{reserved} \ ( \text{book} ) ) ) \\
\text{end} ; \\
\text{ok} \leftarrow \text{book\_for\_collection} \ ( \text{book} ) \ \cong \\
\text{pre} \quad \text{book} \in \text{BOOK} \\
\text{then} \\
\quad \text{ok} := \text{bool} \ ( \text{book} \in \text{ran} \ ( \text{dom} \ ( \text{books\_to\_collect} ) ) ) \\
\text{end} ; \\
\text{ok} \leftarrow \text{not\_book\_copy\_for\_collection\_by\_user} \ ( \text{book}, \text{user} ) \ \cong \\
\text{pre} \quad \text{book} \in \text{LIBRARYBOOK} \land \text{user} \in \text{LIBRARYUSER} \\
\text{then} \\
\quad \text{ok} := \text{bool} \ ( \text{user} \mapsto \text{book\_catalogue} \ ( \text{book} ) \notin \\
\quad \quad \text{dom} \ ( \text{books\_to\_collect} ) ) \\
\text{end} ;
\]
The \textit{LibraryPlus} machine XII

\texttt{ok \leftarrow \text{book} \text{\_for\_collection\_by\_user} ( \text{book} , \text{user} ) \triangleq}

\texttt{pre \hspace{1em} book \in BOOK \land user \in LIBRARYUSER}

\texttt{then \hspace{1em} ok := bool ( user \leftrightarrow book \in \text{dom} ( \text{books\_to\_collect} ) )}

\texttt{end ;}

\texttt{ok \leftarrow \text{book} \text{\_copy\_not\_on\_shelf} ( \text{book} ) \triangleq}

\texttt{pre \hspace{1em} book \in BOOK}

\texttt{then \hspace{1em} ok := bool ( book\_catalogue^{-1} [ \{ book \} ] \cap books\_on\_shelf = \{ \} )}

\texttt{end ;}

\texttt{ok \leftarrow \text{book} \text{\_available\_to\_browse} ( \text{book} ) \triangleq}

\texttt{pre \hspace{1em} book \in LIBRARYBOOK}

\texttt{then \hspace{1em} ok := bool ( book \in books\_on\_shelf \land book \notin browsing )}

\texttt{end ;}

\texttt{ok \leftarrow \text{book} \text{\_in\_browsing} ( \text{book} ) \triangleq}

\texttt{pre \hspace{1em} book \in LIBRARYBOOK}

\texttt{then \hspace{1em} ok := bool ( book \in browsing )}

\texttt{end ;}

\texttt{ok \leftarrow more\_keys \triangleq}
The LibraryPlus machine XIII

\[
\begin{align*}
ok & := \text{bool} \left( \text{card} \left( bkeys \right) \neq \text{maxiter} \right) ; \\
key & \leftarrow \text{any_iterkey} \triangleq \\
key & :\in \text{ITERKEY} ; \\
ok & \leftarrow \text{valid_bkey} \left( bkey \right) \triangleq \\
\text{pre} & \hspace{1em} bkey \in \text{ITERKEY} \\
\text{then} & \hspace{1em} ok := \text{bool} \left( bkey \in bkeys \right) \\
\text{end} \\
\text{END}
\end{align*}
\]
MACHINE LibraryAPI (maxbook, maxuser, maxlibrary, maxreserve, maxiter)

CONSTRAINTS
  maxbook ∈ \mathbb{N}_1 ∧
  maxuser ∈ \mathbb{N}_1 ∧
  maxlibrary ∈ \mathbb{N}_1 ∧
  maxreserve ∈ \mathbb{N}_1 ∧
  maxiter ∈ \mathbb{N}_1

SEES
  User_CTX,
  Book_TYPE,
  LibraryBook,
  IterKey

INCLUDES LibraryPlus (maxbook, maxuser, maxlibrary, maxreserve, maxiter)
The LibraryAPI machine II

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 \}
The \textit{LibraryAPI} machine \textbf{III}

\begin{eqnarray*}
\text{response}, \ \text{newuser} & \leftarrow & \text{RegisterUser} \ \triangleq \\
\text{if} & \text{card}(\text{libraryusers}) & \neq \text{maxuser} \\
\text{then} & \text{newuser} & \leftarrow \text{NewUser} \ | \ | \ \text{response} := \text{OK} \\
\text{else} & \text{newuser} & \in \text{LIBRARYUSER} \ | \ | \ \text{response} := \text{NoNewUsers} \\
\text{end} ;
\end{eqnarray*}
response ← AcquireBook (book) ⇐
pre book ∈ BOOK then
select \( card(\text{librarybooks}) = \text{maxlibrary} \)
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 ;
The *LibraryAPI* machine VII

```plaintext
response ←→ EndBrowseBook (book) ≜
  pre book ∈ LIBRARYBOOK then
    if book ∉ browsing
      then response := NotBeingBrowsed
    else EndBrowse (book) || response := OK
  end
end;
```
The **LibraryAPI** machine

\[
\text{response} \leftarrow \text{BorrowBook} \ ( \text{user}, \text{book} ) \ \triangleq \\
\text{pre} \quad \text{user} \in \text{LIBRARYUSER} \land \text{book} \in \text{LIBRARYBOOK} \quad \text{then} \\
\quad \text{select} \quad \text{user} \notin \text{libraryusers} \\
\quad \text{then} \quad \text{response} := \text{NotRegisteredUser} \\
\quad \text{when} \quad \text{book} \notin \text{librarybooks} \\
\quad \text{then} \quad \text{response} := \text{BookNotForLoan} \\
\quad \text{when} \quad \text{book} \in \text{dom} ( \text{librarybooks}_\text{onloan} ) \\
\quad \text{then} \quad \text{response} := \text{BookNotForLoan} \\
\quad \text{when} \quad \text{user} \in \text{books}_\text{onloan} [ \{ \text{book}_\text{catalogue} ( \text{book} ) \} ] \\
\quad \text{then} \quad \text{response} := \text{BookCopyOnLoan} \\
\quad \text{when} \quad \text{book} \in \text{ran} ( \text{books}_\text{to}_\text{collect} ) \\
\quad \text{then} \quad \text{response} := \text{BookNotForLoan} \\
\quad \text{when} \quad \text{user} \mapsto \text{book}_\text{catalogue} ( \text{book} ) \in \text{dom} ( \text{books}_\text{to}_\text{collect} ) \\
\quad \text{then} \quad \text{response} := \text{BookCopyOnCollection} \\
\quad \text{else} \quad \text{response} := \text{OK} \parallel \text{Borrow2} ( \text{user}, \text{book} ) \\
\text{end}
\]
end ;
The `LibraryAPI` machine X

```
response ← ReturnBook ( book ) ≜
  pre book ∈ LIBRARYBOOK then
    if book ∈ dom ( librarybooks_onloan )
      then Return1 ( book ) ∥ response := OK
    else response := BookNotOnLoan
  end
end ;
```
The *LibraryAPI* machine XI

\[
\text{response} \, , \, \text{user} \, , \, \text{iterkey} \leftarrow \text{WhoBorrowedFirst} \,(\, \text{book} \, ) \equiv \\
\text{pre} \quad \text{book} \in \text{BOOK} \quad \text{then} \\
\text{select} \quad \text{card} \,(\, \text{bkeys} \, ) = \text{maxiter} \\
\text{then} \quad \text{response} := \text{FAIL} \quad \parallel \\
\quad \text{user} : \in \text{libraryusers} \quad \parallel \\
\quad \text{iterkey} : \in \text{ITERKEY} \\
\text{when} \quad \text{books\_onloan} \,(\, \{ \, \text{book} \, \} \, ) = \{\} \\
\text{then} \quad \text{response} := \text{NoBorrowers} \quad \parallel \\
\quad \text{user} : \in \text{libraryusers} \quad \parallel \\
\quad \text{iterkey} : \in \text{ITERKEY} \\
\text{else} \quad \text{response} := \text{OK} \quad \parallel \\
\quad \text{user} \, , \, \text{iterkey} \leftarrow \text{BorrowedFirst} \,(\, \text{book} \, ) \\
\text{end} \\
\text{end} ;
\]
response, user ← WhoBorrowedNext ( bkey )

pre bkey ∈ ITERKEY then
select bkey ∉ bkeys
then response := NoMoreBorrowers || user ∈ libraryusers
else response := OK || user ← BorrowedNext ( bkey )
end
end ;
The LibraryAPI machine XIII

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
The *LibraryAPI* machine XIV

```plaintext

when book ∈ dom ( reserved ) ∧
    user ∈ ran ( reserved ( book ) )
then response := InvalidReservation

when book ∈ dom ( reserved ) ∧
    size ( reserved ( book ) ) = maxreserve
then response := ReserveQueueFull

else Reserve ( user , book ) || response := OK
end

or

response := FAIL

end
end ;
```
The \textit{LibraryAPI} machine XV

\[\text{response} \leftarrow \text{CollectAndBorrow} \ (user, \ book) \equiv\]

\text{pre} \quad user \in \text{LIBRARYUSER} \land book \in \text{BOOK} \quad \text{then}

\text{select} \quad user \notin \text{libraryusers}

\text{then} \quad \text{response} := \text{NotRegisteredUser}

\text{when} \quad book \notin \text{books}

\text{then} \quad \text{response} := \text{BookNotInLibrary}

\text{when} \quad book \neq \text{dom (books\_to\_collect)} (user)

\text{then} \quad \text{response} := \text{NotForCollection}

\text{when} \quad user \in \text{books\_onloan} [\{book\}]

\text{then} \quad \text{response} := \text{BookCopyOnLoan}

\text{else} \quad \text{CollectnBorrow} (user, \ book) \parallel

\quad \text{response} := \text{OK}

\text{end}

\text{end} ;
The \textit{LibraryAPI} machine XVI

\begin{verbatim}
response ← CancelBookReserve ( user , book ) ≜ 
  pre  user ∈ LIBRARYUSER ∧ book ∈ BOOK  then 
    select  user \notin libraryusers 
    then  response := NotRegisteredUser 
    when  book ∈ dom ( reserved ) ⇒ 
      user \notin ran ( reserved ( book ) ) 
    then  response := NotReservedForUser 
  else 
    CancelReservation ( user , book ) \parallel 
    response := OK 
  end 
end ;
\end{verbatim}
The LibraryAPI machine XVII

response ← UnCollectBook(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
else UnCollect(user, book) || response := OK
end
end

END
Implementing LibraryAPI

We will implement the LibraryAPI machine in two steps:

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

We will implement the LibraryAPI machine in two steps:

1. implement LibraryAPI by importing LibraryPlus, and then
2. implement LibraryPlus.
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 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 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 ... END END control structures are flattened to sequential IF B THEN END; IF B THEN END where there are runs of the same guard.
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 . . . END END control structures are flattened to sequential IF B THEN END; IF B THEN END where there are runs of the same guard.
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 ... END END control structures are flattened to sequential IF B THEN END; IF B THEN END where there are runs of the same guard.
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 . . . END END control structures are flattened to sequential IF B THEN END; IF B THEN END where there are runs of the same guard.
The LibraryAPII machine

IMPLEMENTATION  LibraryAPII
REFINES  LibraryAPI
SEES
   User_CTX,
   Book_TYPE,
   LibraryBook,
   IterKey,
   Bool_TYPE
IMPORTS  LibraryPlus ( maxbook, maxuser, maxlibrary, maxreserve, maxiter )

OPERATIONS
The LibraryAPI machine II

\[\text{response} , \text{newuser} \leftarrow \text{RegisterUser} \triangleq\]

\[\text{var } bb \text{ in}\]

\[bb \leftarrow \text{users\_not\_full} ;\]

\[\text{if } bb = \text{TRUE } \text{then}\]

\[\text{response} := \text{OK} ; \text{newuser} \leftarrow \text{NewUser}\]

\[\text{else } \text{newuser} \leftarrow \text{anyuser} ; \text{response} := \text{NoNewUsers}\]

\[\text{end}\]

\[\text{end} ;\]
The LibraryAPIII machine III

\[ \text{response} \leftarrow \text{AcquireBook} \left( \text{book} \right) \]

\[
\begin{align*}
\text{var} & \quad \text{bb in} \\
& \quad \text{bb} \leftarrow \text{library\_not\_full} \\
& \quad \text{response} := \text{LibraryFull} \\
& \quad \text{if } \text{bb} = \text{TRUE} \text{ then} \\
& \quad \quad \text{response} := \text{OK} ; \text{AddNewBook} \left( \text{book} \right) \\
& \quad \text{end} \\
& \quad \text{end} \\
\end{align*}
\]
The LibraryAPII machine IV

\[
\text{response} \leftarrow \text{ExitLibrary} \ (book) \ \triangleright \quad \\
\text{var} \quad bb \ \text{in} \\
\text{response} := \text{UnBorrowedBook} \ ; \\
bb \leftarrow \text{book\_onloan\_or\_notlibrary} \ (book) \ ; \\
\text{if} \quad bb = \text{TRUE} \quad \text{then} \\
\text{response} := \text{OK} \ ; \text{RemoveBook} \ (book) \\
\text{end} \\
\text{end} \\
\]
The LibraryAPII machine V

```
response ← BeginBrowseBook (book) ≜ 
  var bb in 
    response ::= BookNotAvailable ;
    bb ← book_available_to_browse (book) ;
    if bb = TRUE then 
      response ::= OK ; BeginBrowse (book)
    end 
  end ;
```
The LibraryAPIII machine VI

\[ response \leftarrow \text{EndBrowseBook} \left( book \right) \]

\[ \text{var } bb \text{ in} \]

\[ response := \text{NotBeingBrowsed} ; \]
\[ bb \leftarrow \text{book\_in\_browsing} \left( book \right) ; \]
\[ \text{if } bb = \text{TRUE} \text{ then} \]
\[ response := \text{OK} ; \text{EndBrowse} \left( book \right) \]
\[ \text{end} \]
\[ \text{end} ; \]
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 ;
The LibraryAPII machine VIII

\[
\begin{align*}
\text{if} & \quad bb = \text{TRUE} \quad \text{then} \\
& \quad \text{response} := \text{BookCopyOnCollection} ; \\
& \quad bb \leftarrow \text{not_book_copy_for_collection_by_user} ( \text{book} , \text{user} ) \\
\text{end} ; \\
\text{if} & \quad bb = \text{TRUE} \quad \text{then} \\
& \quad \text{Borrow2} ( \text{user} , \text{book} ) ; \text{response} := \text{OK} \\
\text{end} \\
\text{end} ;
\end{align*}
\]
The LibraryAPI machine IX

\[ \text{response} \leftarrow \text{ReturnBook} ( \text{book} ) \overset{\Delta}{=} \]
\[ \text{var } bb \text{ in} \]
\[ \text{response} := \text{BookNotOnLoan} ; \]
\[ bb \leftarrow \text{librarybook\_onloan} ( \text{book} ) ; \]
\[ \text{if } bb = TRUE \text{ then} \]
\[ \text{Return1} ( \text{book} ) ; \text{response} := \text{OK} \]
\[ \text{end} \]
\[ \text{end} ; \]
The LibraryAPIII machine X

response, user, iterkey ← WhoBorrowedFirst (book) ≜

var bb in

user ← anyuser;
iterkey ← any_iterkey;
bb ← more_keys;
response := FAIL;
if bb = TRUE then
  bb ← book_onloan (book);
  response := NoBorrowers
end;
if bb = TRUE then
  response := OK;
  user, iterkey ← BorrowedFirst (book)
end
end;
response, user ← WhoBorrowedNext (bkey) ≜
The *LibraryAPIII* machine XI

\[
\begin{align*}
\textbf{var} & \quad bb \ \textbf{in} \\
& user \leftarrow \text{anyuser} \\
& bb \leftarrow \text{valid}\_\text{bkey} ( bkey ) \\
& \text{response} := \text{NoMoreBorrowers} \\
& \textbf{if} \quad bb = \text{TRUE} \quad \textbf{then} \\
& & \text{response} := \text{OK} \\
& & user \leftarrow \text{BorrowedNext} ( bkey ) \\
& \textbf{end} \\
& \textbf{end} ;
\end{align*}
\]
The *LibraryAPI* machine XII

\[ \text{response} \leftarrow \text{ReserveBook} \left( \text{user}, \text{book} \right) \uplus \]

\[ \text{var } bb \text{ in} \]

\[ \text{response} := \text{NotRegisteredUser} ; \]
\[ bb \leftarrow \text{user_registered} \left( \text{user} \right) ; \]
\[ \text{if } bb = \text{TRUE} \text{ then} \]
\[ \text{response} := \text{BookNotInLibrary} ; \]
\[ bb \leftarrow \text{book_in_books} \left( \text{book} \right) \]
\[ \text{end} ; \]
\[ \text{if } bb = \text{TRUE} \text{ then} \]
\[ \text{response} := \text{InvalidReservation} ; \]
\[ bb \leftarrow \text{book_onloan_or_collect} \left( \text{book} \right) \]
\[ \text{end} ; \]
\[ \text{if } bb = \text{TRUE} \text{ then} \]
\[ \text{response} := \text{InvalidReservation} ; \]
\[ bb \leftarrow \text{book_copy_not_on_shelf} \left( \text{book} \right) \]
\[ \text{end} ; \]
The LibraryAPIII machine XIII

\[
\begin{align*}
    \text{if } \; & \; bb = \text{TRUE} \; \text{then} \\
    \quad & \; \text{response} := \text{BookCopyOnLoan} \; ; \\
    \quad & \; bb \leftarrow \text{not\_book\_copy\_onloan\_to\_user} ( \text{book}, \text{user} ) \\
    \text{end} ; \\
    \text{if } \; & \; bb = \text{TRUE} \; \text{then} \\
    \quad & \; \text{response} := \text{BookCopyOnCollection} \; ; \\
    \quad & \; bb \leftarrow \text{book\_for\_collection} ( \text{book} ) \; ; \\
    \text{if } \; & \; bb = \text{FALSE} \; \text{then} \\
    \quad & \; \text{response} := \text{InvalidReservation} \; ; \\
    \quad & \; bb \leftarrow \text{book\_reserved\_by\_user} ( \text{book}, \text{user} ) \\
    \text{end} ; \\
    \text{if } \; & \; bb = \text{FALSE} \; \text{then} \\
    \quad & \; \text{response} := \text{ReserveQueueFull} \; ; \\
    \quad & \; bb \leftarrow \text{reservation\_full} ( \text{book} ) \\
    \text{end} ; \\
    \text{if } \; & \; bb = \text{FALSE} \; \text{then}
\end{align*}
\]
The LibraryAPI machine XIV

```plaintext
response := OK ; Reserve ( user , book )
end
end
end ;
```
The *LibraryAPII* machine XV

\[ \text{response} \leftarrow \text{CollectAndBorrow}(\text{user}, \text{book}) \triangleq \]

\[
\text{var} \quad bb \quad \text{in} \\
\quad \text{response} := \text{NotRegisteredUser} ; \\
\quad bb \leftarrow \text{user\_registered}(\text{user}) ; \\
\quad \text{if} \quad \text{bb} = \text{TRUE} \quad \text{then} \\
\qquad \text{response} := \text{BookNotInLibrary} ; \\
\qquad bb \leftarrow \text{book\_in\_books}(\text{book}) \\
\qquad \text{end} ; \\
\quad \text{if} \quad \text{bb} = \text{TRUE} \quad \text{then} \\
\qquad \text{response} := \text{NotForCollection} ; \\
\qquad bb \leftarrow \text{book\_for\_collection\_by\_user}(\text{book}, \text{user}) \\
\qquad \text{end} ; \\
\quad \text{if} \quad \text{bb} = \text{TRUE} \quad \text{then} \\
\qquad \text{response} := \text{BookCopyOnLoan} ; \\
\qquad bb \leftarrow \text{not\_book\_copy\_onloan\_to\_user}(\text{book}, \text{user}) \\
\quad \text{end} ; \\
\]
The LibraryAPII machine XVI

if \( bb = TRUE \) then

\[
\text{CollectnBorrow ( user , book )} ; \text{response} := OK
\]

end

end ;
The *LibraryAPII* machine XVII

\[
\text{response} \leftarrow \text{CancelBookReserve} \ (user, \ book) \triangleq \\
\text{var} \ bb \ \text{in} \\
\quad \text{response} := \text{NotRegisteredUser} \\
\quad bb \leftarrow user\_registered \ (user) \\
\quad \text{if} \ bb = \text{TRUE} \ \text{then} \\
\quad \quad \text{response} := \text{NotReservedForUser} \\
\quad \quad bb \leftarrow book\_reserved \ (book) \\
\quad \text{end} ; \\
\quad \text{if} \ bb = \text{TRUE} \ \text{then} \\
\quad \quad \text{response} := \text{NotReservedForUser} \\
\quad \quad bb \leftarrow book\_reserved\_by\_user \ (book, \ user) \\
\quad \text{end} ; \\
\quad \text{if} \ bb = \text{TRUE} \ \text{then} \\
\quad \quad \text{response} := \text{OK} ; \text{CancelReservation} \ (user, \ book) \\
\quad \text{end} \\
\text{end} ;
\]
The **LibraryAPI** machine XVIII

\[
\begin{align*}
\text{response} & \leftarrow \text{UnCollectBook}(\text{user}, \text{book}) \equiv \\
\text{var} & \quad bb \text{ in} \\
& \quad \text{response} := \text{NotRegisteredUser} ; \\
& \quad bb \leftarrow \text{user\_registered}(\text{user}) ; \\
& \quad \text{if } bb = \text{TRUE} \text{ then} \\
& \quad \quad \text{response} := \text{NotForCollection} ; \\
& \quad \quad bb \leftarrow \text{book\_for\_collection\_by\_user}(\text{book}, \text{user}) \\
& \quad \text{end} ; \\
& \quad \text{if } bb = \text{TRUE} \text{ then} \\
& \quad \quad \text{response} := \text{OK} ; \\
& \quad \quad \text{UnCollect}(\text{user}, \text{book}) \\
& \quad \text{end} \\
& \text{end}
\end{align*}
\]

END
Implementing LibraryPlus

To complete the implementation we now have to implement LibraryPlus.

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

This is not currently presented here.
Implementing LibraryPlus

To complete the implementation we now have to implement LibraryPlus.

This is not currently presented here.