

System Modelling and Design

Structuring Specifications A Library Case Study

Revision: 1.6, April 28, 2004

Ken Robinson

School of Computer Science & Engineering
The University of New South Wales, Sydney Australia

4th April 2005

©Ken Robinson 2005

`mailto::k.robinson@unsw.edu.au`

Outline I

- 1 Objectives of this lecture
- 2 A Small library development
- 3 A Book TYPE machine
- 4 Book Services
- 5 User Registration
 - The UserRegistration machine
- 6 The Borrowing Machine
 - Proof obligation problem
 - Explanation of the problem
- 7 Book reservation
 - Modelling the reservation queue
 - Modelling books for collection
 - The Reservation machine
- 8 Browsing Books in the Library

Outline II

- 9 Insecurity caused by USES
 - Putting it all together
 - The Bool_TYPE machine
 - The Library machine

- 10 A Robust Library machine

- 11 Implementing LibraryAPI

- 12 Implementing Library
 - Library Base Specification
 - The LibraryI implementation

Objectives of this lecture

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

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 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 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 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 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 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 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 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 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 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 TYPE machine

We need to have a representation of a book object. Books will be simply modelled using a set *BOOK*, and since this should be a universal, global set we will use a machine *Book_TYPE*, that contains a deferred set *BOOK*.

The rest of the development will access this machine using *SEES*.

```

MACHINE Book_TYPE ( maxbook )
CONSTRAINTS maxbook  $\in \mathbb{N}_1$ 
SETS BOOK
PROPERTIES card ( BOOK ) = maxbook
END
  
```

Note carefully This machine has a parameter that is used to configure the size of the set *BOOK*, but this parameter is *not* instantiated in the *SEES* clause, and nor is this parameter visible in the seeing machine.

A Book TYPE machine

We need to have a representation of a book object. Books will be simply modelled using a set *BOOK*, and since this should be a universal, global set we will use a machine *Book_TYPE*, that contains a deferred set *BOOK*.

The rest of the development will access this machine using *SEES*.

```

MACHINE Book_TYPE ( maxbook )
CONSTRAINTS maxbook  $\in \mathbb{N}_1$ 
SETS BOOK
PROPERTIES card ( BOOK ) = maxbook
END

```

Note carefully This machine has a parameter that is used to configure the size of the set *BOOK*, but this parameter is *not* instantiated in the *SEES* clause, and nor is this parameter visible in the seeing machine.

A Book TYPE machine

We need to have a representation of a book object. Books will be simply modelled using a set *BOOK*, and since this should be a universal, global set we will use a machine *Book_TYPE*, that contains a deferred set *BOOK*.

The rest of the development will access this machine using *SEES*.

```

MACHINE Book_TYPE ( maxbook )
CONSTRAINTS maxbook  $\in \mathbb{N}_1$ 
SETS BOOK
PROPERTIES card ( BOOK ) = maxbook
END
  
```

Note carefully This machine has a parameter that is used to configure the size of the set *BOOK*, but this parameter is *not* instantiated in the *SEES* clause, and nor is this parameter visible in the seeing machine.

A Book TYPE machine

We need to have a representation of a book object. Books will be simply modelled using a set *BOOK*, and since this should be a universal, global set we will use a machine *Book_TYPE*, that contains a deferred set *BOOK*.

The rest of the development will access this machine using *SEES*.

MACHINE *Book_TYPE* (*maxbook*)

CONSTRAINTS *maxbook* $\in \mathbb{N}_1$

SETS *BOOK*

PROPERTIES $\text{card} (\textit{BOOK}) = \textit{maxbook}$

END

Note carefully This machine has a parameter that is used to configure the size of the set *BOOK*, but this parameter is *not* instantiated in the *SEES* clause, and nor is this parameter visible in the seeing machine.

A Book TYPE machine

We need to have a representation of a book object. Books will be simply modelled using a set *BOOK*, and since this should be a universal, global set we will use a machine *Book_TYPE*, that contains a deferred set *BOOK*.

The rest of the development will access this machine using *SEES*.

MACHINE *Book_TYPE* (*maxbook*)

CONSTRAINTS *maxbook* $\in \mathbb{N}_1$

SETS *BOOK*

PROPERTIES $\text{card} (\textit{BOOK}) = \textit{maxbook}$

END

Note carefully This machine has a parameter that is used to configure the size of the set *BOOK*, but this parameter is *not* instantiated in the *SEES* clause, and nor is this parameter visible in the seeing machine.

Book Services I

The *BookServices* machine models the acquisition of books by the library.

Requirements:

- to acquire new books for the library;
- to provide operations for adding books to and removing books from the shelves of the library.

MACHINE *BookServices* (*maxlibrary*)

CONSTRAINTS *maxlibrary* $\in \mathbb{N}_1$

SEES *Book_TYPE*

VARIABLES

books_in_library ,

books_on_shelf

Book Services II

INVARIANT

books_in_library models all books acquired by the library, and *books_on_shelf* models those books currently on the shelves of the library.

$$\begin{aligned} & \text{books_in_library} \subseteq \text{BOOK} \wedge \\ & \text{card}(\text{books_in_library}) \leq \text{maxlibrary} \wedge \\ & \text{books_on_shelf} \subseteq \text{books_in_library} \end{aligned}$$

INITIALISATION

$$\text{books_in_library}, \text{books_on_shelf} := \{\}, \{\}$$

OPERATIONS

Book Services III

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*) $\hat{=}$

PRE $book \in BOOK \wedge book \notin books_in_library \wedge$
 $card (books_in_library) \neq maxlibrary$

THEN $books_in_library := books_in_library \cup \{ book \} \parallel$
 $books_on_shelf := books_on_shelf \cup \{ book \}$

END ;

Book Services IV

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*) $\hat{=}$

PRE *book* \in *books_in_library*

THEN *books_on_shelf* := *books_on_shelf* \cup { *book* }

END ;

RemoveBookFromShelf (*book*) $\hat{=}$

PRE *book* \in *books_in_library*

THEN *books_on_shelf* := *books_on_shelf* - { *book* }

END

END

User Registration

The *UserRegistration* machine will model the operation of registering a user of the library.

To model the identifiers issued to registered users we will use a deferred set *USER*. 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 *USER*.

User Registration

The *UserRegistration* machine will model the operation of registering a user of the library.

To model the identifiers issued to registered users we will use a deferred set *USER*. 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 *USER*.

User Registration

The *UserRegistration* machine will model the operation of registering a user of the library.

To model the identifiers issued to registered users we will use a deferred set *USER*. 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 *USER*.

User Context machine

MACHINE *User_CTX* (*maxuser*)
CONSTRAINTS *maxuser* $\in \mathbb{N}_1$
SETS *USER*
PROPERTIES $\text{card} (\textit{USER}) = \textit{maxuser}$

OPERATIONS

user \leftarrow **anyuser** $\hat{=}$

user $:\in \textit{USER}$

END

The UserRegistration machine I

The purpose of the *UserRegistration* machine is:

- to register users of the library; *only registered users of the library may borrow books.*

The machine is specified as an independent machine that sees *User_CTX*, its own *context* machine, and *Bool_TYPE*.

The UserRegistration machine II

MACHINE *UserRegistration* (*maxuser*)

CONSTRAINTS *maxuser* $\in \mathbb{N}_1$

SEES *User_CTX*

VARIABLES *users*

INVARIANT *users* $\subseteq USER$

INITIALISATION *users* := {}

OPERATIONS

The UserRegistration machine III

Requirements:

- to register users of the library;

newuser \leftarrow **NewUser** $\hat{=}$

PRE *users* \neq *USER*

THEN

ANY *user*

WHERE *user* \in *USER* – *users*

THEN *users* := *users* \cup { *user* } ||

newuser := *user*

END

END

END

The Borrowing Machine I

The purpose of the *Borrowing* machine is to:

- control the *borrowing capability* of the library;
Only registered users of the library may borrow books.
A registered use may borrow any number of books.
- maintain information on books on loan.

The machine is specified as an *extension* of the *BookServices* machine, and *uses* *UserRegistration* because the invariant is dependent on the variable *users*.

The Borrowing Machine II

MACHINE *Borrowing* (*maxlibrary*)

CONSTRAINTS *maxlibrary* $\in \mathbb{N}_1$

SEES *Book_TYPE*

The Borrowing Machine III

USES *UserRegistration*

EXTENDS *BookServices* (*maxlibrary*)

VARIABLES *books_on_loan*

INVARIANT

$books_on_loan \in books_in_library \leftrightarrow users \wedge$

$dom (books_on_loan) \cap books_on_shelf = \{ \}$

INITIALISATION $books_on_loan := \{ \}$

OPERATIONS

The Borrowing Machine IV

Allow a registered *user* to borrow a *book*.

Borrow (*user* , *book*) $\hat{=}$

PRE $user \in users \wedge$

$book \in books_in_library \wedge book \notin dom (books_on_loan)$

THEN *RemoveBookFromShelf* (*book*) \parallel

$books_on_loan := books_on_loan \cup \{ book \mapsto user \}$

END ;

Return a *book*.

Return (*book*) $\hat{=}$

PRE $book \in dom (books_on_loan)$

THEN *AddBookToShelf* (*book*) \parallel

$books_on_loan := \{ book \} \triangleleft books_on_loan$

END ;

The Borrowing Machine V

Enquire who has borrowed *book*.

```
user  $\leftarrow$  Borrowed ( book )  $\hat{=}$   
  PRE book  $\in$  BOOK  $\wedge$  book  $\in$  dom ( books_on_loan )  
  THEN user := books_on_loan ( book )  
  END ;
```

Cancel the loan of a *book*. This is a utility operation that will not finally be exported to the interface.

```
CancelLoan ( book )  $\hat{=}$   
  PRE book  $\in$  BOOK  
  THEN books_on_loan := { book }  $\triangleleft$  books_on_loan  
  END
```

END

Proof obligation problem

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

Why?

Proof obligation problem

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

Why?

Proof obligation problem

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 *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 *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 *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 *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 *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 *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.

Book reservation

Requirements:

- We would like to allow a registered user to reserve a book that is currently on loan.
- A user may reserve any number of books, but any particular book may be reserved at most once.
- A user may not reserve a book 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.

Book reservation

Requirements:

- We would like to allow a registered user to reserve a book that is currently on loan.
- A user may reserve any number of books, but any particular book may be reserved at most once.
- A user may not reserve a book 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.

Book reservation

Requirements:

- We would like to allow a registered user to reserve a book that is currently on loan.
- A user may reserve any number of books, but any particular book may be reserved at most once.
- A user may not reserve a book 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.

Book reservation

Requirements:

- We would like to allow a registered user to reserve a book that is currently on loan.
- A user may reserve any number of books, but any particular book may be reserved at most once.
- A user may not reserve a book 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.

Book reservation

Requirements:

- We would like to allow a registered user to reserve a book that is currently on loan.
- A user may reserve any number of books, but any particular book may be reserved at most once.
- A user may not reserve a book 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.

Book reservation

Requirements:

- We would like to allow a registered user to reserve a book that is currently on loan.
- A user may reserve any number of books, but any particular book may be reserved at most once.
- A user may not reserve a book 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.

Book reservation

Requirements:

- We would like to allow a registered user to reserve a book that is currently on loan.
- A user may reserve any number of books, but any particular book may be reserved at most once.
- A user may not reserve a book 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.

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned}reserved \in books_in_library &\leftrightarrow iseq (users) \wedge \\reserved \in books_in_library &\leftrightarrow seq_1 (users)\end{aligned}$$

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

$$\forall book . (book \in dom (reserved) \Rightarrow size (reserved (book)) \leq maxreserve)$$

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned} reserved \in books_in_library &\rightarrow iseq (users) \wedge \\ reserved \in books_in_library &\rightarrow seq_1 (users) \end{aligned}$$

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

$$\forall book . (book \in dom (reserved) \Rightarrow size (reserved (book)) \leq maxreserve)$$

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned}reserved &\in \text{books_in_library} \mapsto \text{iseq} (\text{users}) \wedge \\reserved &\in \text{books_in_library} \mapsto \text{seq}_1 (\text{users})\end{aligned}$$

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

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

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned} reserved \in books_in_library &\leftrightarrow iseq (users) \wedge \\ reserved \in books_in_library &\leftrightarrow seq_1 (users) \end{aligned}$$

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

$$\forall book . (book \in dom (reserved) \Rightarrow size (reserved (book)) \leq maxreserve)$$

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned} reserved \in books_in_library &\mapsto iseq (users) \wedge \\ reserved \in books_in_library &\mapsto seq_1 (users) \end{aligned}$$

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

$$\forall book . (book \in dom (reserved) \Rightarrow size (reserved (book)) \leq maxreserve)$$

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned} reserved \in books_in_library &\leftrightarrow iseq (users) \wedge \\ reserved \in books_in_library &\leftrightarrow seq_1 (users) \end{aligned}$$

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

$$\forall book . (book \in dom (reserved) \Rightarrow size (reserved (book)) \leq maxreserve)$$

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned} reserved \in books_in_library &\leftrightarrow iseq (users) \wedge \\ reserved \in books_in_library &\leftrightarrow seq_1 (users) \end{aligned}$$

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

$$\forall book . (book \in dom (reserved) \Rightarrow size (reserved (book)) \leq maxreserve)$$

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_in_library* to a sequence of *users*.

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.

$$\begin{aligned} reserved \in books_in_library &\leftrightarrow iseq (users) \wedge \\ reserved \in books_in_library &\leftrightarrow seq_1 (users) \end{aligned}$$

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

$$\forall book . (book \in dom (reserved) \Rightarrow size (reserved (book)) \leq maxreserve)$$

Modelling books for collection

Books available for collection (and borrowing) can be modelled by a partial function from *books_in_library* to *users*

$$\text{collect} \in \text{books_in_library} \leftrightarrow \text{users}$$

Books available for collection cannot be on loan.

$$\text{dom}(\text{collect}) \cap \text{dom}(\text{books_on_loan}) = \{\}$$

Nor should they be on the shelf.

$$\text{dom}(\text{collect}) \cap \text{books_on_shelf} = \{\}$$

Modelling books for collection

Books available for collection (and borrowing) can be modelled by a partial function from *books_in_library* to *users*

$$\text{collect} \in \text{books_in_library} \rightarrow \text{users}$$

Books available for collection cannot be on loan.

$$\text{dom}(\text{collect}) \cap \text{dom}(\text{books_on_loan}) = \{\}$$

Nor should they be on the shelf.

$$\text{dom}(\text{collect}) \cap \text{books_on_shelf} = \{\}$$

Modelling books for collection

Books available for collection (and borrowing) can be modelled by a partial function from *books_in_library* to *users*

$$\textit{collect} \in \textit{books_in_library} \rightarrow \textit{users}$$

Books available for collection cannot be on loan.

$$\text{dom}(\textit{collect}) \cap \text{dom}(\textit{books_on_loan}) = \{\}$$

Nor should they be on the shelf.

$$\text{dom}(\textit{collect}) \cap \textit{books_on_shelf} = \{\}$$

Modelling books for collection

Books available for collection (and borrowing) can be modelled by a partial function from *books_in_library* to *users*

$$\textit{collect} \in \textit{books_in_library} \rightarrow \textit{users}$$

Books available for collection cannot be on loan.

$$\text{dom}(\textit{collect}) \cap \text{dom}(\textit{books_on_loan}) = \{\}$$

Nor should they be on the shelf.

$$\text{dom}(\textit{collect}) \cap \textit{books_on_shelf} = \{\}$$

Modelling books for collection

Books available for collection (and borrowing) can be modelled by a partial function from *books_in_library* to *users*

$$\textit{collect} \in \textit{books_in_library} \rightarrow \textit{users}$$

Books available for collection cannot be on loan.

$$\text{dom}(\textit{collect}) \cap \text{dom}(\textit{books_on_loan}) = \{\}$$

Nor should they be on the shelf.

$$\text{dom}(\textit{collect}) \cap \textit{books_on_shelf} = \{\}$$

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* (*maxlibrary* , *maxreserve*)
CONSTRAINTS $maxlibrary \in \mathbb{N}_1 \wedge maxreserve \in \mathbb{N}_1$
SEES *User_CTX* , *Book_TYPE*
USES *UserRegistration*
INCLUDES *Borrowing* (*maxlibrary*)
PROMOTES *AddNewBook* , *Borrowed*

The *Reservation* machine II

VARIABLES

reserved , *collect*

INVARIANT

$reserved \in books_in_library \leftrightarrow iseq (users) \wedge$

$reserved \in books_in_library \leftrightarrow seq_1 (users) \wedge$

$\forall book . (book \in dom (reserved) \Rightarrow$

$size (reserved (book)) \leq maxreserve) \wedge$

$dom (reserved) \subseteq dom (books_on_loan) \cup dom (collect) \wedge$

$collect \in books_in_library \leftrightarrow users \wedge$

$dom (collect) \cap dom (books_on_loan) = \{ \} \wedge$

$dom (collect) \cap books_on_shelf = \{ \}$

INITIALISATION

$reserved , collect := \{ \} , \{ \}$

OPERATIONS

The *Reservation* machine III

Reserve(*user*, *book*)

Precondition:

- Only registered users may reserve books
- To be reserved, 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 this book

The Reservation machine IV

Reserve (*user* , *book*) $\hat{=}$

PRE $user \in users \wedge book \in books_in_library \wedge$
 $book \in dom (books_on_loan) \cup dom (collect) \wedge$
 $(book \in dom (books_on_loan) \Rightarrow$
 $books_on_loan (book) \neq user) \wedge$
 $(book \in dom (collect) \Rightarrow collect (book) \neq user) \wedge$
 $(book \in dom (reserved) \Rightarrow$
 $size (reserved (book)) \neq maxreserve) \wedge$
 $(book \in dom (reserved) \Rightarrow$
 $user \notin ran (reserved (book)))$

THEN IF $book \notin dom (reserved)$

THEN $reserved (book) := [user]$

ELSE $reserved (book) := reserved (book) \leftarrow user$

END

END ;

The *Reservation* machine V

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 VI

```
Return1 ( book )  $\hat{=}$   
  PRE book  $\in$  dom ( books_on_loan )  
  THEN IF book  $\in$  dom ( reserved )  
    THEN CancelLoan ( book ) ||  
      collect ( book ) := first ( reserved ( book ) ) ||  
      IF size ( reserved ( book ) ) = 1  
        THEN reserved := { book }  $\triangleleft$  reserved  
        ELSE reserved ( book ) := tail ( reserved ( book ) )  
        END  
      ELSE Return ( book )  
      END  
  END ;
```

The *Reservation* machine VII

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 VIII

```
CancelReservation ( user , book )  $\hat{=}$   
  PRE user  $\in$  users  $\wedge$  book  $\in$  dom ( reserved )  $\wedge$   
    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 )  $\wedge$   
      list = reserved ( book )  
    THEN  
      reserved ( book ) :=  
        list  $\uparrow$  pos - 1  $\frown$  ( list  $\downarrow$  pos )  
    END  
  END  
END ;
```

The *Reservation* machine IX

Borrow1(*user*, *book*): An upgrade of *Borrow*, this operation extends the *Borrow* operation to provide for the collection and borrowing of a reserved book.

If the book being borrowed is in the collect set then the borrower must be the person who reserved the book. Having collected the book *Borrow1* joins *Borrow*.

If the book is not in the collect set then *Borrow1* simply reverts to *Borrow*.

The *Reservation* machine X

```
Borrow1 ( user , book )  $\hat{=}$   
  PRE user  $\in$  users  $\wedge$  book  $\in$  books_in_library  $\wedge$   
    book  $\notin$  dom ( books_on_loan )  $\wedge$   
    ( book  $\in$  dom ( collect )  $\Rightarrow$  collect ( book ) = user )  
  THEN IF book  $\in$  dom ( collect )  
    THEN collect := { book }  $\triangleleft$  collect  
    END ||  
    Borrow ( user , book )  
END ;
```

The *Reservation* machine XI

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 XII

UnCollect (*user* , *book*) $\hat{=}$

PRE $user \in users \wedge book \in \text{dom} (collect) \wedge$
 $collect (book) = user$

THEN IF $book \in \text{dom} (reserved)$

THEN $collect (book) := \text{first} (reserved (book)) \parallel$

IF $\text{size} (reserved (book)) = 1$

THEN $reserved := \{ book \} \triangleleft reserved$

ELSE $reserved (book) := \text{tail} (reserved (book))$

END

ELSE $collect := \{ book \} \triangleleft collect \parallel$

$AddBookToShelf (book)$

END

END

END

Browsing Books in the Library I

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* (*maxlibrary* , *maxreserve*)

CONSTRAINTS

maxlibrary $\in \mathbb{N}_1 \wedge$

maxreserve $\in \mathbb{N}_1$

SEES

User_CTX ,

Book_TYPE

USES *UserRegistration*

INCLUDES *Reservation* (*maxlibrary* , *maxreserve*)

PROMOTES

AddNewBook ,

Borrowed ,

Reserve ,

Return1 ,

CancelReservation ,

UnCollect

VARIABLES *browsing*

Browsing Books in the Library III

INVARIANT $browsing \subseteq books_on_shelf$

INITIALISATION $browsing := \{\}$

Browsing Books in the Library IV

OPERATIONS

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.

BeginBrowse (*book*) $\hat{=}$

PRE $book \in books_on_shelf \wedge book \notin browsing$ **THEN**

$browsing := browsing \cup \{ book \}$

END ;

Browsing Books in the Library V

EndBrowse(book): End the browsing of a book in the library.

EndBrowse (*book*) $\hat{=}$

PRE *book* \in *browsing* **THEN**

browsing := *browsing* – { *book* }

END ;

Browsing Books in the Library VI

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.

```

RemoveBook ( book )  $\hat{=}$ 
  PRE book  $\in$  BOOK  $\wedge$ 
    ( book  $\in$  books_in_library  $\Rightarrow$ 
      book  $\in$  dom ( books_on_loan ) )
  THEN skip
  END ;

```

Browsing Books in the Library VII

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

Borrow2 ($user$, $book$) $\hat{=}$

PRE $user \in users \wedge book \in books_in_library \wedge$
 $book \notin dom (books_on_loan) \wedge$
 $(book \in dom (collect) \Rightarrow collect (book) = user)$

THEN $Borrow1 (user , book) \parallel$

IF $book \in browsing$ **THEN**

$browsing := browsing - \{ book \}$

END

END

END

Insecurity caused by *USES*

The invariant of the *Borrowing* machine and all subsequent machines up to *Browsing* depend on the variable *users* in the used machine *UserRegistration*, but the *Browsing* machine has no control over the value of that variable. This coupling between the two machines is insecure and all proof obligations discharged for *Browsing* are subject to that reservation.

B Method (B) insists that all using and used machines are included—using *INCLUDES* or *EXTENDS*—in a single machine to resolve all insecurities, and we will now proceed to do that.

Insecurity caused by *USES*

The invariant of the *Borrowing* machine and all subsequent machines up to *Browsing* depend on the variable *users* in the used machine *UserRegistration*, but the *Browsing* machine has no control over the value of that variable. This coupling between the two machines is insecure and all proof obligations discharged for *Browsing* are subject to that reservation.

B insists that all using and used machines are included —using *INCLUDES* or *EXTENDS*— in a single machine to resolve all insecurities, and we will now proceed to do that.

Insecurity caused by *USES*

The invariant of the *Borrowing* machine and all subsequent machines up to *Browsing* depend on the variable *users* in the used machine *UserRegistration*, but the *Browsing* machine has no control over the value of that variable. This coupling between the two machines is insecure and all proof obligations discharged for *Browsing* are subject to that reservation.

B insists that all using and used machines are included —using *INCLUDES* or *EXTENDS*— in a single machine to resolve all insecurities, and we will now proceed to do that.

Putting it all together

Objectives:

- To include all the component machine into a top-level machine, in which all the operations are robust.
- To resolve the insecurity of the using machine.

Putting it all together

Objectives:

- To include all the component machine into a top-level machine, in which all the operations are robust.
- To resolve the insecurity of the using machine.

Putting it all together

Objectives:

- To include all the component machine into a top-level machine, in which all the operations are robust.
- To resolve the insecurity of the using machine.

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 *bool*:

```
var := bool(expr)  $\hat{=}$  IF expr THEN var := TRUE  
                     ELSE var := FALSE  
                     END
```

```
MACHINE Bool_TYPE  
SETS  $BOOL = \{ FALSE , TRUE \}$   
END
```

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 *bool*:

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

```
MACHINE Bool_TYPE
SETS BOOL = { FALSE , TRUE }
END
```

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 *bool*:

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

MACHINE *Bool_TYPE*
SETS $BOOL = \{ FALSE , TRUE \}$
END

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

$maxbook \in \mathbb{N}_1 \wedge$

$maxuser \in \mathbb{N}_1 \wedge$

$maxlibrary \in \mathbb{N}_1 \wedge$

$maxreserve \in \mathbb{N}_1$

SEES

User_CTX ,

Book_TYPE ,

Bool_TYPE

The *Library* machine II

EXTENDS

UserRegistration (*maxuser*) ,

Browsing (*maxlibrary* , *maxreserve*)

INVARIANT

We now strengthen the relationship between *books_in_library*, *dom(books_on_loan)*, *collect* and *books_on_shelf*.

$$\forall bb . (bb \in books_in_library \wedge (bb \notin dom (books_on_loan) \wedge bb \notin dom (collect)) \Rightarrow bb \in books_on_shelf)$$

OPERATIONS

The *Library* machine III

We add a set of operations that give access to information required for determination of preconditions

$ok \leftarrow \mathbf{users_not_full} \hat{=}$

BEGIN $ok := \mathbf{bool} (users \neq USER)$ **END ;**

$ok \leftarrow \mathbf{user_registered} (user) \hat{=}$

PRE $user \in USER$

THEN $ok := \mathbf{bool} (user \in users)$

END ;

$ok \leftarrow \mathbf{book_in_library} (book) \hat{=}$

PRE $book \in BOOK$

THEN $ok := \mathbf{bool} (book \in books_in_library)$

END ;

$ok \leftarrow \mathbf{library_full} \hat{=}$

BEGIN

$ok := \mathbf{bool} (\mathbf{card} (books_in_library) = maxlibrary)$

The *Library* machine IV

END ;

$ok \leftarrow \mathbf{book_on_loan} (book) \hat{=}$

PRE $book \in BOOK$

THEN $ok := \mathbf{bool} (book \in \mathbf{dom} (\mathbf{books_on_loan}))$

END ;

$ok \leftarrow \mathbf{book_on_loan_to_user} (book , user) \hat{=}$

PRE $book \in BOOK \wedge user \in USER$

THEN $ok := \mathbf{bool} (book \in \mathbf{dom} (\mathbf{books_on_loan}) \wedge$
 $\mathbf{books_on_loan} (book) = user)$

END ;

$ok \leftarrow \mathbf{book_on_loan_or_collect} (book) \hat{=}$

PRE $book \in BOOK$

THEN $ok := \mathbf{bool} (book \notin \mathbf{dom} (\mathbf{books_on_loan}) \Rightarrow$
 $book \in \mathbf{dom} (\mathbf{collect}))$

END ;

$ok \leftarrow \mathbf{reservation_full} (book) \hat{=}$

PRE $book \in BOOK$

The *Library* machine V

THEN $ok := \text{bool} (book \in \text{dom} (reserved) \wedge$
 $\text{size} (reserved (book)) = \text{maxreserve})$

END ;

$ok \leftarrow \text{book_reserved} (book) \hat{=}$

PRE $book \in BOOK$

THEN $ok := \text{bool} (book \in \text{dom} (reserved))$

END ;

$ok \leftarrow \text{book_reserved_by_user} (book , user) \hat{=}$

PRE $book \in BOOK \wedge user \in USER$

THEN $ok := \text{bool} (book \in \text{dom} (reserved) \wedge$
 $user \in \text{ran} (reserved (book)))$

END ;

$ok \leftarrow \text{book_for_collection} (book) \hat{=}$

PRE $book \in BOOK$

THEN $ok := \text{bool} (book \in \text{dom} (collect))$

END ;

$ok \leftarrow \text{book_for_collection_by_user} (book , user) \hat{=}$

The *Library* machine VI

PRE $book \in BOOK \wedge user \in USER$

THEN $ok := \text{bool} (book \in \text{dom} (collect) \Rightarrow$
 $collect (book) = user)$

END ;

$ok \leftarrow$ **book_available_to_browse** ($book$) $\hat{=}$

PRE $book \in BOOK$

THEN $ok := \text{bool} (book \in \text{books_on_shelf} \wedge book \notin \text{browsing})$

END ;

$ok \leftarrow$ **book_in_browsing** ($book$) $\hat{=}$

PRE $book \in BOOK$

THEN $ok := \text{bool} (book \in \text{browsing})$

END

END

A Robust Library machine I

We now specify a “top-level” *API* version of the *Library* machine with robust operations.

MACHINE *LibraryAPI* (*maxbook* , *maxuser* , *maxlibrary* , *maxreserve*)

CONSTRAINTS

$maxbook \in \mathbb{N}_1 \wedge$

$maxuser \in \mathbb{N}_1 \wedge$

$maxlibrary \in \mathbb{N}_1 \wedge$

$maxreserve \in \mathbb{N}_1$

SEES *User_CTX* , *Book_TYPE*

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

A Robust Library machine II

SETS

RESPONSE is an enumerated set, whose symbolic values represent the status responses returned with every operation.

```
RESPONSE = { OK ,  
              BookInLibrary , LibraryFull ,  
              BookNotForLoan , BookNotOnLoan ,  
              BookNotAvailable , NotBeingBrowsed ,  
              InvalidReservation ,  
              NoNewUsers , NotRegisteredUser ,  
              NotForCollection , NotReservedForUser ,  
              ReserveQueueFull ,  
              UnBorrowedBook ,  
              FAIL }
```

A Robust Library machine III

OPERATIONS

$response, newuser \leftarrow \mathbf{RegisterUser} \hat{=}$

IF $users \neq USER$

THEN $newuser \leftarrow NewUser \parallel response := OK$

ELSE $newuser : \in USER \parallel response := NoNewUsers$

END ;

A Robust Library machine IV

```

response ← AcquireBook ( book ) ≐
  PRE book ∈ BOOK THEN
    SELECT book ∈ books_in_library
    THEN response := BookInLibrary
    WHEN card ( books_in_library ) = maxlibrary
    THEN response := LibraryFull
    ELSE CHOICE AddNewBook ( book ) ||
      response := OK
    OR
      response := FAIL
    END
  END
END ;

```

A Robust Library machine V

```
response  $\leftarrow$  ExitLibrary ( book )  $\hat{=}$   
  PRE book  $\in$  BOOK THEN  
    IF book  $\in$  books_in_library  $\wedge$   
      book  $\notin$  dom ( books_on_loan )  
    THEN response := UnBorrowedBook  
    ELSE RemoveBook ( book ) || response := OK  
    END  
END ;
```

A Robust Library machine VI

```

response ← BeginBrowseBook ( book ) ≐
  PRE book ∈ BOOK THEN
    IF book ∈ books_on_shelf ∧
      book ∉ browsing
    THEN
      CHOICE
        BeginBrowse ( book ) || response := OK
      OR
        response := FAIL
      END
    ELSE response := BookNotAvailable
    END
  END ;

```

A Robust Library machine VII

```
response  $\leftarrow$  EndBrowseBook ( book )  $\hat{=}$   
  PRE book  $\in$  BOOK THEN  
    IF book  $\notin$  browsing  
    THEN response := NotBeingBrowsed  
    ELSE EndBrowse ( book ) || response := OK  
    END  
END ;
```

A Robust Library machine VIII

```

response ← BorrowBook ( user , book ) ≐
  PRE user ∈ USER ∧ book ∈ BOOK THEN
    SELECT user ∉ users
    THEN response := NotRegisteredUser
    WHEN book ∉ books_in_library
    THEN response := BookNotForLoan
    WHEN book ∈ dom ( books_on_loan )
    THEN response := BookNotForLoan
    WHEN book ∈ dom ( collect ) ∧
      collect ( book ) ≠ user
    THEN response := BookNotForLoan
    ELSE Borrow2 ( user , book ) ||
      response := OK
  END
END ;
response ← ReturnBook ( book ) ≐

```

A Robust Library machine IX

```
PRE book ∈ BOOK THEN  
  IF book ∈ dom ( books_on_loan )  
  THEN Return1 ( book ) || response := OK  
  ELSE response := BookNotOnLoan  
  END  
END ;
```

A Robust Library machine X

```
response , user  $\leftarrow$  WhoBorrowed ( book )  $\hat{=}$   
  PRE book  $\in$  BOOK THEN  
    IF book  $\in$  dom ( books_on_loan ) THEN  
      response := OK ||  
      user  $\leftarrow$  Borrowed ( book )  
    ELSE response := BookNotOnLoan ||  
      user : $\in$  USER  
    END  
END ;
```

A Robust Library machine XI

```

response ← ReserveBook ( user , book ) ≐
  PRE user ∈ USER ∧ book ∈ BOOK THEN
    CHOICE
      SELECT user ∉ users
      THEN response := NotRegisteredUser
      WHEN book ∉ dom ( books_on_loan ) ∧
           book ∉ dom ( collect )
      THEN response := InvalidReservation
      WHEN book ∈ dom ( books_on_loan ) ∧
           books_on_loan ( book ) = user
      THEN response := InvalidReservation
      WHEN book ∈ dom ( collect ) ∧
           collect ( book ) = user
      THEN response := InvalidReservation
      WHEN book ∈ dom ( reserved ) ∧
           user ∈ ran ( reserved ( book ) )

```

A Robust Library machine XII

```
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 ;
```

A Robust Library machine XIII

```

response ← CancelBookReserve ( user , book ) ≐
  PRE user ∈ USER ∧ book ∈ BOOK THEN
    SELECT user ∉ users
    THEN response := NotRegisteredUser
    WHEN book ∈ dom ( reserved ) ⇒
      user ∉ ran ( reserved ( book ) )
    THEN response := NotReservedForUser
    ELSE
      CancelReservation ( user , book ) ||
      response := OK
    END
  END ;

```

A Robust Library machine XIV

```

response ← UnCollectBook ( user , book ) ≐
  PRE user ∈ USER ∧ book ∈ BOOK THEN
    SELECT user ∉ users
    THEN response := NotRegisteredUser
    WHEN book ∈ dom ( collect ) ⇒ collect ( book ) ≠ user
    THEN response := NotForCollection
    ELSE UnCollect ( user , book ) || response := OK
    END
  END
END

```

Implementing LibraryAPI

The **LibraryAPI** machine will be implemented by importing the **Library** machine as shown in **LibraryAPII**.

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.

Implementing LibraryAPI

The **LibraryAPI** machine will be implemented by importing the **Library** machine as shown in **LibraryAPII**.

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.

Implementing LibraryAPI

The **LibraryAPI** machine will be implemented by importing the **Library** machine as shown in **LibraryAPII**.

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.

Implementing LibraryAPI

The **LibraryAPI** machine will be implemented by importing the **Library** machine as shown in **LibraryAPII**.

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.

Implementing LibraryAPI

The **LibraryAPI** machine will be implemented by importing the **Library** machine as shown in **LibraryAPII**.

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.

Implementing LibraryAPI

The **LibraryAPI** machine will be implemented by importing the **Library** machine as shown in **LibraryAPII**.

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.

IMPLEMENTATION *LibraryAPII*

REFINES *LibraryAPI*

SEES *Bool_TYPE*

IMPORTS

User_CTX (maxuser) ,

Book_TYPE (maxbook) ,

Library (maxbook , maxuser , maxlibrary , maxreserve) ,

LibraryDBCtx

OPERATIONS

```
response , newuser  $\leftarrow$  RegisterUser  $\hat{=}$   
VAR bb IN  
  bb  $\leftarrow$  users_not_full ;  
  IF bb = TRUE THEN  
    newuser  $\leftarrow$  NewUser ; response := OK  
  ELSE newuser  $\leftarrow$  anyuser ; response := NoNewUsers  
  END  
END ;
```

```
response ← AcquireBook ( book ) ≐
  VAR bb IN
    bb ← book_in_library ( book ) ;
    IF bb = TRUE THEN
      response := BookInLibrary
    ELSE bb ← library_full ;
      IF bb = TRUE THEN
        response := LibraryFull
      ELSE AddNewBook ( book ) ; response := OK
    END
  END
END ;
```

```

response ← ExitLibrary ( book ) ≐
  VAR binl , bonl IN
    response := UnBorrowedBook ;
    binl ← book_in_library ( book ) ;
    IF binl = TRUE THEN
      bonl ← book_on_loan ( book )
    END ;
    IF binl = FALSE ∨ bonl = TRUE THEN
      RemoveBook ( book ) ; response := OK
    END
  END ;

```

```
response ← BeginBrowseBook ( book ) ≐  
VAR bb IN  
  bb ← book_available_to_browse ( book ) ;  
  IF bb = FALSE THEN  
    response := BookNotAvailable  
  ELSE BeginBrowse ( book ) ; response := OK  
  END  
END ;
```

```
response ← EndBrowseBook ( book ) ≐  
VAR bb IN  
  bb ← book_in_browsing ( book ) ;  
  IF bb = FALSE THEN  
    response := NotBeingBrowsed  
  ELSE EndBrowse ( book ) ; response := OK  
  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*) ;

IF *bb* = *TRUE* **THEN**

bb ← *book_on_loan* (*book*) ;

IF *bb* = *FALSE* **THEN**

bb ← *book_for_collection_by_user* (*book* , *user*) ;

IF *bb* = *TRUE* **THEN**

Borrow2 (*user* , *book*) ; *response* := *OK*

END

END

END

END

END ;

```
response ← ReturnBook ( book ) ≐
VAR bb IN
    response := BookNotOnLoan ;
    bb ← book_on_loan ( book ) ;
    IF bb = TRUE THEN
        Return1 ( book ) ; response := OK
    END
END ;
```

```
response , user  $\leftarrow$  WhoBorrowed ( book )  $\hat{=}$   
  VAR bb IN  
    bb  $\leftarrow$  book_on_loan ( book ) ;  
    IF bb = TRUE THEN  
      user  $\leftarrow$  Borrowed ( book ) ; response := OK  
    ELSE  
      user  $\leftarrow$  anyuser ; response := BookNotOnLoan  
    END  
END ;
```

```

response ← ReserveBook ( user , book ) ≐
VAR bb IN
    response := NotRegisteredUser ;
    bb ← user_registered ( user ) ;
    IF bb = TRUE THEN
        response := InvalidReservation ;
        bb ← book_on_loan_or_collect ( book ) ;
        IF bb = TRUE THEN
            bb ← book_on_loan_to_user ( book , user ) ;
            IF bb = FALSE THEN
                bb ← book_for_collection ( book ) ;
                IF bb = TRUE THEN
                    bb ← book_for_collection_by_user ( book , user )
                END ;
            IF bb = FALSE THEN
                bb ← book_reserved_by_user ( book , user ) ;
                IF bb = FALSE THEN

```

```
bb ← book_reserved ( book ) ;  
IF bb = TRUE THEN  
    response := ReserveQueueFull ;  
    bb ← reservation_full ( book )  
END ;  
IF bb = FALSE THEN  
    Reserve ( user , book ) ; response := OK  
END  
END  
END  
END  
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 ) ;
      IF bb = TRUE THEN
        bb ← book_reserved_by_user ( book , user ) ;
        IF bb = TRUE THEN
          CancelReservation ( user , book ) ;
          response := OK
        END
      END
    END
  END ;

```

response \leftarrow **UnCollectBook** (*user* , *book*) $\hat{=}$

VAR *bb* **IN**

response := *NotRegisteredUser* ;

bb \leftarrow *user_registered* (*user*) ;

IF *bb* = *TRUE* **THEN**

response := *NotForCollection* ;

bb \leftarrow *book_for_collection* (*book*) ;

IF *bb* = *TRUE* **THEN**

bb \leftarrow *book_for_collection_by_user* (*book* , *user*) ;

IF *bb* = *TRUE* **THEN**

UnCollect (*user* , *book*) ;

response := *OK*

END

END

END

END

END

Implementing Library

To complete the implementation the **Library** machine must be implemented. This will be done by using the B-Toolkit *Base* generator to generate a database machine. The base is specified in **LibraryDB** specification. The automatically generated base machine is shown in the appendix.

Library Base Specification

```
SYSTEM LibraryDB
SUPPORTS Libraryl
IS
  GLOBAL
    browsed ∈ SET ( librarydb ) [ 100 ]
  END ;

BASE librarydb
MANDATORY
  bookid ∈ BOOK
OPTIONAL
  loan ∈ USER ;
  reserveq ∈ SEQ ( USER ) [ 10 ] ;
  tocollect ∈ USER
END
END
```

The LibraryI implementation I

IMPLEMENTATION *LibraryI*

REFINES *Library*

SEES

User_CTX ,

Book_TYPE ,

LibraryDBCtx ,

Bool_TYPE

IMPORTS

user_Nvar (*maxuser*) ,

LibraryDB (*maxlibrary* , *BOOK* , *USER*)

PROPERTIES

USER = 1 .. *maxuser*

The Libraryl implementation II

INVARIANT

$$\begin{aligned} users &= 1 .. user_Nvar \wedge \\ books_in_library &= \text{ran} (bookid) \wedge \\ books_on_loan &= (bookid^{-1} ; loan) \wedge \\ reserved &= (bookid^{-1} ; reserveq) \wedge \\ collect &= (bookid^{-1} ; tocollect) \wedge \\ browsing &= bookid [browsed] \wedge \\ bookid &\in librarydb \mapsto BOOK \wedge \\ browsed &\subseteq librarydb \end{aligned}$$

The Libraryl implementation III

ASSERTIONS

$$\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge bookid (ldb) = bb \Rightarrow bb \in books_in_library) \wedge$$

$$\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge bookid (ldb) = bb \wedge ldb \in dom (loan) \Rightarrow bb \in dom (books_on_loan)) \wedge$$

$$\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge bookid (ldb) = bb \wedge ldb \in dom (reserveq) \Rightarrow bb \in dom (reserved)) \wedge$$

$$\forall (bb , uu , ldb) . (bb \in BOOK \wedge uu \in USER \wedge ldb \in librarydb \wedge bookid (ldb) = bb \wedge ldb \in dom (reserveq) \wedge uu \in ran (reserveq (ldb)) \Rightarrow uu \in ran (reserved (bb))) \wedge$$

$$\forall (bb , uu , ldb) . (bb \in BOOK \wedge uu \in USER \wedge ldb \in librarydb \wedge bookid (ldb) = bb \wedge ldb \in dom (reserveq) \wedge$$

The Libraryl implementation IV

$\text{size} (\text{reserveq} (ldb)) = \text{maxreserve} \Rightarrow$

$\text{size} (\text{reserved} (bb)) = \text{maxreserve}) \wedge$

$\forall (bb , ldb) . (bb \in \text{BOOK} \wedge ldb \in \text{librarydb} \wedge$

$\text{bookid} (ldb) = bb \wedge ldb \in \text{dom} (\text{tocollect}) \Rightarrow$

$bb \in \text{dom} (\text{collect})) \wedge$

$\forall (bb , ldb) . (bb \in \text{BOOK} \wedge ldb \in \text{librarydb} \wedge$

$\text{bookid} (ldb) = bb \wedge ldb \in \text{browsed} \Rightarrow$

$bb \in \text{browsing}) \wedge$

$\forall (bb , ldb) . (bb \in \text{BOOK} \wedge ldb \in \text{librarydb} \wedge$

$\text{bookid} (ldb) = bb \wedge \neg (ldb \in \text{dom} (\text{loan})) \Rightarrow$

$\neg (bb \in \text{dom} (\text{books_on_loan}))) \wedge$

$\forall bb . (bb \in \text{BOOK} \wedge \exists ldb . (ldb \in \text{librarydb} \wedge$

$\text{bookid} (ldb) = bb \wedge \neg (ldb \in \text{dom} (\text{reserveq}))) \Rightarrow$

$\neg (bb \in \text{dom} (\text{reserved}))) \wedge$

$\forall (bb , uu , ldb) . (bb \in \text{BOOK} \wedge uu \in \text{USER} \wedge ldb \in \text{librarydb} \wedge$

$\text{bookid} (ldb) = bb \wedge ldb \in \text{dom} (\text{reserveq}) \wedge$

The Libraryl implementation V

$$\neg (uu \in \text{ran} (\text{reserveq} (ldb))) \Rightarrow$$

$$\neg (uu \in \text{ran} (\text{reserved} (bb))) \wedge$$

$$\forall (bb , uu , ldb) . (bb \in \text{BOOK} \wedge uu \in \text{USER} \wedge ldb \in \text{librarydb} \wedge$$

$$\text{bookid} (ldb) = bb \wedge ldb \in \text{dom} (\text{reserveq}) \wedge$$

$$\neg (\text{size} (\text{reserveq} (ldb)) = \text{maxreserve}) \Rightarrow$$

$$\neg (\text{size} (\text{reserved} (bb)) = \text{maxreserve}) \wedge$$

$$\forall (bb , ldb) . (bb \in \text{BOOK} \wedge ldb \in \text{librarydb} \wedge$$

$$\text{bookid} (ldb) = bb \wedge \neg (ldb \in \text{dom} (\text{tocollect})) \Rightarrow$$

$$\neg (bb \in \text{dom} (\text{collect})) \wedge$$

$$\forall (bb , ldb) . (bb \in \text{BOOK} \wedge ldb \in \text{librarydb} \wedge$$

$$\text{bookid} (ldb) = bb \wedge \neg (ldb \in \text{browsed}) \Rightarrow$$

$$\neg (bb \in \text{browsing}))$$

OPERATIONS

The Libraryl implementation VI

```
newuser ← NewUser ≐  
BEGIN  
  user_INC_NVAR ;  
  newuser ← user_VAL_NVAR  
END ;
```

The Libraryl implementation VII

AddNewBook (*book*) $\hat{=}$

VAR *bb* , *db* **IN**

bb , *db* \leftarrow *make_librarydb* (*book*)

Ignore FALSE return

END ;

The Libraryl implementation VIII

```
RemoveBook ( book )  $\hat{=}$   
  skip ;
```

The Libraryl implementation IX

```
BeginBrowse ( book )  $\hat{=}$   
  VAR bb , db IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    bb  $\leftarrow$  add_browsed ( db )  
END ;
```

The Libraryl implementation X

```
EndBrowse ( book )  $\hat{=}$   
  VAR bb , db IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    del_browsed ( db )  
END ;
```

The Libraryl implementation XI

```
Borrow2 ( user , book )  $\hat{=}$   
  VAR bb , cc , db IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    cc  $\leftarrow$  def_tocollect ( db ) ;  
    IF cc = TRUE  
    THEN rem_tocollect ( db )  
    END ;  
    cre_loan ( db , user ) ;  
    mod_loan ( db , user )  
END ;
```

The Libraryl implementation XII

```
Return1 ( book )  $\hat{=}$   
  VAR bb , db , ii , uu IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    rem_loan ( db ) ;  
    bb  $\leftarrow$  def_reserveq ( db ) ;  
    uu  $\leftarrow$  vallth_reserveq ( db , 1 ) ;  
    cre_tocollect ( db , uu ) ;  
    ii  $\leftarrow$  length_reserveq ( db ) ;  
    IF ii = 1  
      THEN rem_reserveq ( db )  
      ELSE bb  $\leftarrow$  dellth_reserveq ( db , 1 )
```

Ignore FALSE return

The Libraryl implementation XIII

```
END  
END ;
```

The Libraryl implementation XIV

```
user ← Borrowed ( book ) ≐  
  VAR bb , db IN  
    bb , db ← key_search_bookid ( book ) ;  
    user ← val_loan ( db )  
END ;
```

The Libraryl implementation XV

```
Reserve ( user , book )  $\hat{=}$   
  VAR bb , db IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    bb  $\leftarrow$  cre_reserveq ( db ) ;  
    bb  $\leftarrow$  push_reserveq ( db , user )  
END ;
```

The Libraryl implementation XVI

```
CancelReservation ( user , book )  $\hat{=}$   
  VAR bb , db , ii , nn IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    nn  $\leftarrow$  length_reserveq ( db ) ;  
    IF nn = 1  
      THEN rem_reserveq ( db )  
      ELSE bb , ii  $\leftarrow$  within_reserveq ( db , user ) ;  
        bb  $\leftarrow$  dellth_reserveq ( db , ii )  
      END  
    END ;
```

The LibraryI implementation XVII

```
UnCollect ( user , book )  $\hat{=}$   
  VAR bb , db , ii , uu IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    bb  $\leftarrow$  def_reserveq ( db ) ;  
    IF bb = TRUE  
    THEN uu  $\leftarrow$  last_reserveq ( db ) ;  
        mod_tocollect ( db , uu ) ;  
        ii  $\leftarrow$  length_reserveq ( db ) ;  
        IF ii = 1  
        THEN rem_reserveq ( db )  
        ELSE bb  $\leftarrow$  dellth_reserveq ( db , ii )  
        END  
    ELSE rem_tocollect ( db )  
    END  
END ;
```

The Libraryl implementation XVIII

The Libraryl implementation XIX

```
ok ← users_not_full ≐  
  ok ← user_NEQ_NVAR ( maxuser ) ;  
ok ← user_registered ( user ) ≐  
  ok ← user_GEQ_NVAR ( user ) ;  
ok ← book_in_library ( book ) ≐  
  VAR db IN  
    ok , db ← key_search_bookid ( book )  
END ;
```

The Libraryl implementation XX

```
ok ← library_full  $\hat{=}$   
  VAR nn IN  
    nn ← nbr_librarydb ;  
    IF nn = maxlibrary  
    THEN ok := TRUE  
    ELSE ok := FALSE  
    END  
  END ;
```

The Libraryl implementation XXI

```
ok  $\leftarrow$  book_on_loan ( book )  $\hat{=}$   
  VAR bb , db IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    IF bb = TRUE  
      THEN bb  $\leftarrow$  def_loan ( db )  
    END ;  
    ok := bb  
END ;
```

The Libraryl implementation XXII

```
ok ← book_on_loan_to_user ( book , user ) ≐
  VAR bb , db IN
    bb , db ← key_search_bookid ( book ) ;
    IF bb = TRUE
      THEN bb ← def_loan ( db ) ;
        IF bb = TRUE THEN
          bb ← eql_loan ( db , user )
        END
      END ;
    ok := bb
  END ;
```

The LibraryI implementation XXIII

```
ok ← book_on_loan_or_collect ( book ) ≐
  VAR bb , db IN
    ok := TRUE ;
    bb , db ← key_search_bookid ( book ) ;
    IF bb = TRUE THEN
      bb ← def_loan ( db ) ;
      IF bb = TRUE THEN
        ok ← def_tocollect ( db )
      END
    END ;
    ok := bb
  END ;
```

The Libraryl implementation XXIV

```
ok ← reservation_full ( book ) ≐
  VAR bb , db , nn IN
    bb , db ← key_search_bookid ( book ) ;
    IF bb = TRUE THEN
      bb ← def_reserveq ( db ) ;
      IF bb = TRUE THEN
        nn ← length_reserveq ( db ) ;
        IF nn = maxreserve THEN
          bb := TRUE
        ELSE bb := FALSE
        END
      END
    END ;
  ok := bb
END ;
ok ← book_reserved ( book ) ≐
```

The Libraryl implementation XXV

```
VAR bb , db IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
IF bb = TRUE THEN  
    bb  $\leftarrow$  def_reserveq ( db )  
END ;  
ok := bb  
END ;
```

The Libraryl implementation XXVI

```
ok ← book_reserved_by_user ( book , user ) ≐
  VAR bb , db , nn IN
    bb , db ← key_search_bookid ( book ) ;
    IF bb = TRUE THEN
      bb ← def_reserveq ( db ) ;
      IF bb = TRUE THEN
        bb , nn ← within_reserveq ( db , user )
      END
    END ;
    ok := bb
  END ;
```

The LibraryI implementation XXVII

```
ok  $\leftarrow$  book_for_collection ( book )  $\hat{=}$   
  VAR bb , db IN  
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;  
    IF bb = TRUE THEN  
      bb  $\leftarrow$  def_tocollect ( db )  
    END ;  
    ok := bb  
END ;
```

The Libraryl implementation XXVIII

```
ok ← book_for_collection_by_user ( book , user ) ≐
  VAR bb , db IN
    ok := TRUE ;
    bb , db ← key_search_bookid ( book ) ;
    IF bb = TRUE THEN
      bb ← def_tocollect ( db ) ;
      IF bb = TRUE THEN
        ok ← eql_tocollect ( db , user )
      END
    END
  END ;
```

The Libraryl implementation XXIX

```
ok ← book_available_to_browse ( book ) ≐
  VAR bb , cc , dd , db IN
    bb , db ← key_search_bookid ( book ) ;
    IF bb = TRUE THEN
      bb ← def_loan ( db ) ;
      cc ← def_tocollect ( db ) ;
      dd ← member_browsed ( db ) ;
      IF bb = FALSE ∧ cc = FALSE ∧ dd = FALSE THEN
        bb := TRUE
      END
    END ;
    ok := bb
  END ;
```

The Libraryl implementation XXX

```
ok ← book_in_browsing ( book ) ≐
  VAR bb , db IN
    bb , db ← key_search_bookid ( book ) ;
    IF bb = TRUE
      THEN bb ← member_browsed ( db )
    END ;
    ok := bb
  END
END
```

Appendix: Base machine LibraryDB I

MACHINE *LibraryDB* (*max_librarydb* , *BOOK* , *USER*)

CONSTRAINTS

max_librarydb $\in 1 \dots 2147483646$

SEES

LibraryDBCtx , *Bool_TYPE* , *Scalar_TYPE*

VARIABLES

browsed ,

librarydb ,

locate_librarydb ,

bookid ,

loan ,

reserveq ,

tocollect

INVARIANT

browsed \subseteq *librarydb_ABSOBJ* \wedge

librarydb \subseteq *librarydb_ABSOBJ* \wedge

Appendix: Base machine LibraryDB II

$\text{card} (\text{librarydb}) \leq \text{max_librarydb} \wedge$
 $\text{locate_librarydb} \in 1 \dots \text{card} (\text{librarydb}) \mapsto \text{librarydb} \wedge$
 $\text{bookid} \in \text{librarydb} \rightarrow \text{BOOK} \wedge$
 $\text{loan} \in \text{librarydb} \leftrightarrow \text{USER} \wedge$
 $\text{reserveq} \in \text{librarydb} \leftrightarrow \text{seq} (\text{USER}) \wedge$
 $\text{tocollect} \in \text{librarydb} \leftrightarrow \text{USER}$

INITIALISATION

$\text{browsed} := \{ \} \parallel$
 $\text{librarydb} := \{ \} \parallel$
 $\text{locate_librarydb} := \{ \} \parallel$
 $\text{bookid} := \{ \} \parallel$
 $\text{loan} := \{ \} \parallel$
 $\text{reserveq} := \{ \} \parallel$
 $\text{tocollect} := \{ \}$

OPERATIONS

$\text{rep} \leftarrow \text{add_browsed} (\text{Elem_librarydb}) \hat{=}$
PRE $\text{Elem_librarydb} \in \text{librarydb}$ **THEN**

Appendix: Base machine LibraryDB III

CHOICE $browsed := browsed \cup \{ Elem_librarydb \} \parallel$ $rep := TRUE$ **OR** $rep := FALSE$ **END****END ;****del_browsed** ($Elem_librarydb$) $\hat{=}$ **PRE** $Elem_librarydb \in browsed$ **THEN** $browsed := browsed - \{ Elem_librarydb \}$ **END ;** $rep \leftarrow$ **member_browsed** ($Elem_librarydb$) $\hat{=}$ **PRE** $Elem_librarydb \in librarydb$ **THEN** $rep := bool (Elem_librarydb \in browsed)$ **END ;** $rep, Base_librarydb \leftarrow$ **make_librarydb** (Val_bookid) $\hat{=}$ **PRE** $Val_bookid \in BOOK \wedge$

Appendix: Base machine LibraryDB IV

$\text{card} (\text{librarydb}) < \text{max_librarydb}$

THEN

CHOICE

ANY *Base_librarydbx* , *loc* **WHERE**

$\text{Base_librarydbx} \in \text{librarydb_ABSOBJ} - \text{librarydb} \wedge$

$\text{loc} \in 1 .. \text{card} (\text{librarydb}) + 1 \mapsto \text{librarydb} \cup \{ \text{Base_librarydbx} \}$

THEN

$\text{librarydb} := \text{librarydb} \cup \{ \text{Base_librarydbx} \} \parallel$

$\text{bookid} (\text{Base_librarydbx}) := \text{Val_bookid} \parallel$

$\text{Base_librarydb} := \text{Base_librarydbx} \parallel$

$\text{locate_librarydb} := \text{loc} \parallel$

$\text{rep} := \text{TRUE}$

END

OR

ANY *Base_librarydbx* **WHERE**

$\text{Base_librarydbx} \in \text{librarydb_ABSOBJ}$

THEN

Appendix: Base machine LibraryDB V

```

    Base_librarydb := Base_librarydbx ||
    rep := FALSE
  END
  END
  END ;
  tot ← nbr_librarydb ≐ BEGIN tot := card ( librarydb ) END ;
  rep , Base_librarydb ← key_search_bookid ( Elem_BOOK ) ≐
  PRE Elem_BOOK ∈ BOOK THEN
    IF Elem_BOOK ∈ ran ( bookid ) THEN
      ANY Base_librarydbx WHERE
        Base_librarydbx ∈ librarydb ∧ bookid ( Base_librarydbx ) = Elem_BOOK
      THEN
        Base_librarydb := Base_librarydbx || rep := TRUE
      END
    ELSE
      Base_librarydb := librarydb_ABSOBJ || rep := FALSE
    END
  END ;

```

Appendix: Base machine LibraryDB VI

cre_loan (*Base_librarydb* , *Elem_USER*) $\hat{=}$

PRE *Base_librarydb* \in *librarydb* – *dom* (*loan*) \wedge *Elem_USER* \in *USER* **THEN**

loan (*Base_librarydb*) := *Elem_USER*

END ;

rem_loan (*Base_librarydb*) $\hat{=}$

PRE *Base_librarydb* \in *dom* (*loan*) **THEN**

loan := { *Base_librarydb* } \ll *loan*

END ;

rep \leftarrow **def_loan** (*Base_librarydb*) $\hat{=}$

PRE *Base_librarydb* \in *librarydb* **THEN**

rep := *bool* (*Base_librarydb* \in *dom* (*loan*))

END ;

Elem_USER \leftarrow **val_loan** (*Base_librarydb*) $\hat{=}$

PRE *Base_librarydb* \in *dom* (*loan*) **THEN**

Elem_USER := *loan* (*Base_librarydb*)

END ;

mod_loan (*Base_librarydb* , *Elem_USER*) $\hat{=}$

PRE *Base_librarydb* \in *dom* (*loan*) \wedge *Elem_USER* \in *USER* **THEN**

Appendix: Base machine LibraryDB VII

$$\text{loan} (\text{Base_librarydb}) := \text{Elem_USER}$$

END ;

$$\text{rep} \leftarrow \text{eql_loan} (\text{Base_librarydb} , \text{Elem_USER}) \hat{=}$$

PRE $\text{Base_librarydb} \in \text{dom} (\text{loan}) \wedge \text{Elem_USER} \in \text{USER}$ **THEN**

$$\text{rep} := \text{bool} (\text{loan} (\text{Base_librarydb}) = \text{Elem_USER})$$

END ;

$$\text{rep} \leftarrow \text{cre_reserveq} (\text{Base_librarydb}) \hat{=}$$

PRE $\text{Base_librarydb} \in \text{librarydb} - \text{dom} (\text{reserveq})$ **THEN**

CHOICE

$$\text{reserveq} (\text{Base_librarydb}) := [] \quad ||$$

$$\text{rep} := \text{TRUE}$$

OR

$$\text{rep} := \text{FALSE}$$

END

END ;

$$\text{rem_reserveq} (\text{Base_librarydb}) \hat{=}$$

PRE $\text{Base_librarydb} \in \text{dom} (\text{reserveq})$ **THEN**

$$\text{reserveq} := \{ \text{Base_librarydb} \} \triangleleft \text{reserveq}$$

Appendix: Base machine LibraryDB VIII

```

END ;
rep ← def_reserveq ( Base_librarydb ) ≐
PRE Base_librarydb ∈ librarydb THEN
    rep := bool ( Base_librarydb ∈ dom ( reserveq ) )
END ;
nbr ← length_reserveq ( Base_librarydb ) ≐
PRE Base_librarydb ∈ dom ( reserveq ) THEN
    nbr := size ( reserveq ( Base_librarydb ) )
END ;
Elem_USER ← last_reserveq ( Base_librarydb ) ≐
PRE Base_librarydb ∈ dom ( reserveq ) ∧ reserveq ( Base_librarydb ) ≠ [ ] TH
    Elem_USER := last ( reserveq ( Base_librarydb ) )
END ;
Elem_USER ← vallth_reserveq ( Base_librarydb , ii ) ≐
PRE Base_librarydb ∈ dom ( reserveq ) ∧ ii ∈ 1 .. size ( reserveq ( Base_librarydb ) )
    Elem_USER := reserveq ( Base_librarydb ) ( ii )
END ;
rep ← dellth_reserveq ( Base_librarydb , ii ) ≐

```

Appendix: Base machine LibraryDB IX

PRE $Base_librarydb \in \text{dom} (reserveq) \wedge ii \in 1 .. \text{size} (reserveq (Base_librarydb))$

CHOICE

$reserveq (Base_librarydb) :=$

$reserveq (Base_librarydb) \uparrow ii - 1 \wedge (reserveq (Base_librarydb) \downarrow ii) \parallel$

$rep := TRUE$

OR

$rep := FALSE$

END

END ;

$rep \leftarrow \text{push_reserveq} (Base_librarydb, Elem_USER) \hat{=}$

PRE $Base_librarydb \in \text{dom} (reserveq) \wedge Elem_USER \in USER$ **THEN**

CHOICE

$reserveq (Base_librarydb) := reserveq (Base_librarydb) \wedge [Elem_USER]$

$rep := TRUE$

OR

$rep := FALSE$

END

END ;

Appendix: Base machine LibraryDB X

```

rep , ii  $\leftarrow$  within_reserveq ( Base_librarydb , Elem_USER )  $\hat{=}$ 
  PRE Base_librarydb  $\in$  dom ( reserveq )  $\wedge$  Elem_USER  $\in$  USER THEN
    IF Elem_USER  $\in$  ran ( reserveq ( Base_librarydb ) ) THEN
      rep := TRUE || ii : $\in$  ( reserveq ( Base_librarydb ) )-1 [ { Elem_USER } ]
    ELSE
      rep := FALSE || ii : $\in$   $\mathbb{N}$ 
    END
  END ;
cre_tocollect ( Base_librarydb , Elem_USER )  $\hat{=}$ 
  PRE Base_librarydb  $\in$  librarydb – dom ( tocollect )  $\wedge$  Elem_USER  $\in$  USER THEN
    tocollect ( Base_librarydb ) := Elem_USER
  END ;
rem_tocollect ( Base_librarydb )  $\hat{=}$ 
  PRE Base_librarydb  $\in$  dom ( tocollect ) THEN
    tocollect := { Base_librarydb }  $\triangleleft$  tocollect
  END ;
rep  $\leftarrow$  def_tocollect ( Base_librarydb )  $\hat{=}$ 
  PRE Base_librarydb  $\in$  librarydb THEN

```

Appendix: Base machine LibraryDB XI

```

    rep := bool ( Base_librarydb ∈ dom ( tocollect ) )
  END ;
  mod_tocollect ( Base_librarydb , Elem_USER ) ≐
    PRE Base_librarydb ∈ dom ( tocollect ) ∧ Elem_USER ∈ USER THEN
      tocollect ( Base_librarydb ) := Elem_USER
    END ;
  rep ← eql_tocollect ( Base_librarydb , Elem_USER ) ≐
    PRE Base_librarydb ∈ dom ( tocollect ) ∧ Elem_USER ∈ USER THEN
      rep := bool ( tocollect ( Base_librarydb ) = Elem_USER )
    END
  END
END

```