

System Modelling and Design

Structuring Specifications

A Library Case Study

Revision: 1.6, April 28, 2004

Ken Robinson

4th April 2005

©Ken Robinson 2005

[mailto::k.robinson@unsw.edu.au](mailto:k.robinson@unsw.edu.au)

Contents

1 Objectives of this lecture	2
2 A Small library development	2
3 A Book TYPE machine	2
4 Book Services	3
5 User Registration	4
5.1 The UserRegistration machine	4
6 The Borrowing Machine	5
6.1 Proof obligation problem	6
6.2 Explanation of the problem	6
7 Book reservation	7
7.1 Modelling the reservation queue	7
7.2 Modelling books for collection	7
7.3 The Reservation machine	8
8 Browsing Books in the Library	11
9 Insecurity caused by USES	12
9.1 Putting it all together	12
9.2 The Bool_TYPE machine	13
9.3 The Library machine	13

10 A Robust Library machine	15
11 Implementing LibraryAPI	19
12 Implementing Library	24
12.1 Library Base Specification	24
12.2 The LibraryI implementation	24
A Base machine LibraryDB	33

1 Objectives of this lecture

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

2 A Small library development

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

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

3 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 ∈ ℕ1
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.

4 Book Services

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

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.

$books_in_library \subseteq BOOK \wedge$

$card (books_in_library) \leq maxlibrary \wedge$

$books_on_shelf \subseteq books_in_library$

INITIALISATION

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

OPERATIONS

AddNewBook(*book*) *Requirements:*

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

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

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 ;

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

```

5 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*.

```

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

```

OPERATIONS

```

  user  $\leftarrow$  anyuser  $\hat{=}$ 
    user  $\in$  USER
END

```

5.1 The UserRegistration machine

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*.

```

MACHINE UserRegistration (maxuser)
CONSTRAINTS maxuser  $\in$   $\mathbb{N}_1$ 

```

SEES *User_CTX*
VARIABLES *users*
INVARIANT $users \subseteq USER$
INITIALISATION $users := \{\}$

OPERATIONS

Requirements:

- to register users of the library;

$newuser \leftarrow \mathbf{NewUser} \hat{=} \\
\mathbf{PRE} \quad users \neq USER \\
\mathbf{THEN} \\
\quad \mathbf{ANY} \quad user \\
\quad \mathbf{WHERE} \quad user \in USER - users \\
\quad \mathbf{THEN} \quad users := users \cup \{ user \} \parallel \\
\quad \quad newuser := user \\
\quad \mathbf{END} \\
\mathbf{END} \\
\mathbf{END}$

6 The Borrowing Machine

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*.

MACHINE *Borrowing* (*maxlibrary*)
CONSTRAINTS $maxlibrary \in \mathbb{N}_1$
SEES *Book_TYPE*

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

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*) ||

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

END ;

Return a *book*.

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

PRE $book \in dom (books_on_loan)$

THEN *AddBookToShelf* (*book*) ||

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

END ;

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

6.1 Proof obligation problem

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

Why?

6.2 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.

7 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.

7.1 Modelling the reservation queue

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

We will model this using a partial function from *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.

$reserved \in books_in_library \mapsto i\text{seq}(users) \wedge [5ex]reserved \in books_in_library \mapsto seq_1(users)$

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

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

7.2 Modelling books for collection

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

$collect \in books_in_library \mapsto users$

Books available for collection cannot be on loan.

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

Nor should they be on the shelf.

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

7.3 The Reservation machine

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*

VARIABLES

reserved , *collect*

INVARIANT

reserved $\in \text{books_in_library} \leftrightarrow \text{iseq} (\text{users}) \wedge$

reserved $\in \text{books_in_library} \leftrightarrow \text{seq}_1 (\text{users}) \wedge$

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

$\text{size} (\text{reserved} (\text{book})) \leq \text{maxreserve}) \wedge$

$\text{dom} (\text{reserved}) \subseteq \text{dom} (\text{books_on_loan}) \cup \text{dom} (\text{collect}) \wedge$

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

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

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

INITIALISATION

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

OPERATIONS

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

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

PRE *user* $\in \text{users} \wedge$ *book* $\in \text{books_in_library} \wedge$

book $\in \text{dom} (\text{books_on_loan}) \cup \text{dom} (\text{collect}) \wedge$

```

( book ∈ dom ( books_on_loan ) ⇒
  books_on_loan ( book ) ≠ user ) ∧
( book ∈ dom ( collect ) ⇒ collect ( book ) ≠ user ) ∧
( book ∈ dom ( reserved ) ⇒
  size ( reserved ( book ) ) ≠ maxreserve ) ∧
( book ∈ dom ( reserved ) ⇒
  user ∉ ran ( reserved ( book ) ) )
THEN IF book ∉ dom ( reserved )
THEN reserved ( book ) := [ user ]
ELSE reserved ( book ) := reserved ( book ) ← user
END
END ;

```

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.

```

Return1 ( book ) ≐
PRE book ∈ dom ( books_on_loan )
THEN IF book ∈ dom ( reserved )
THEN CancelLoan ( book ) ||
  collect ( book ) := first ( reserved ( book ) ) ||
IF size ( reserved ( book ) ) = 1
THEN reserved := { book } ≪ reserved
ELSE reserved ( book ) := tail ( reserved ( book ) )
END
ELSE Return ( book )
END
END ;

```

CancelReservation(user, book): A user who has reserved a book may wish to cancel the reservation.

The reservation must be deleted from the reservation queue, or the queue should be deleted if the reservation is the only one on the queue.

```

CancelReservation ( user , book ) ≐
PRE user ∈ users ∧ book ∈ dom ( reserved ) ∧
  user ∈ ran ( reserved ( book ) )
THEN IF size ( reserved ( book ) ) = 1
THEN reserved := { book } ≪ reserved
ELSE ANY pos , list WHERE

```

```

    pos = ( reserved ( book ) ) -1 ( user ) ∧
    list = reserved ( book )
THEN
    reserved ( book ) :=
        list ↑ pos - 1 ∧ ( list ↓ pos )
END
END
END ;

```

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*.

```

Borrow1 ( user , book ) ≐
PRE user ∈ users ∧ book ∈ books_in_library ∧
    book ∉ dom ( books_on_loan ) ∧
    ( book ∈ dom ( collect ) ⇒ collect ( book ) = user )
THEN IF book ∈ dom ( collect )
    THEN collect := { book } ⋈ collect
    END ||
    Borrow ( user , book )
END ;

```

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.

```

UnCollect ( user , book ) ≐
PRE user ∈ users ∧ book ∈ dom ( collect ) ∧
    collect ( book ) = user
THEN IF book ∈ dom ( reserved )
    THEN collect ( book ) := first ( reserved ( book ) ) ||
        IF size ( reserved ( book ) ) = 1
        THEN reserved := { book } ⋈ reserved
        ELSE reserved ( book ) := tail ( reserved ( book ) )
        END
    ELSE collect := { book } ⋈ collect ||
        AddBookToShelf ( book )
    END
END
END
END

```

8 Browsing Books in the Library

Requirements: to model

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

This machine upgrades the current borrowing operation.

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

MACHINE *Browsing* (*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*

INVARIANT *browsing* \subseteq *books_on_shelf*

INITIALISATION *browsing* := {}

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 ;

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

```
EndBrowse ( book )  $\hat{=}$   
  PRE book  $\in$  browsing THEN  
    browsing := browsing - { book }  
  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.

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

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 ) ||  
    IF book  $\in$  browsing THEN  
      browsing := browsing - { book }  
    END  
  END  
END
```

9 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.

9.1 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.

9.2 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*:

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

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

9.3 The Library machine

This “top-level” machine is build on top of the *Browsing* machine.

The *UserRegistration* machine, which was previously *used*, is now *included*.

```
MACHINE Library ( maxbook , maxuser , maxlibrary , maxreserve )
```

```
CONSTRAINTS
```

```
maxbook ∈ ℕ1 ∧
```

```
maxuser ∈ ℕ1 ∧
```

```
maxlibrary ∈ ℕ1 ∧
```

```
maxreserve ∈ ℕ1
```

```
SEES
```

```
User_CTX ,
```

```
Book_TYPE ,
```

```
Bool_TYPE
```

```
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*.

$$\forall bb . (bb \in books_in_library \wedge ($$

$$bb \notin \text{dom} (books_on_loan) \wedge bb \notin \text{dom} (collect)) \Rightarrow$$

$$bb \in books_on_shelf)$$

OPERATIONS

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

```

ok ← users_not_full ≐
  BEGIN ok := bool ( users ≠ USER ) END ;
ok ← user_registered ( user ) ≐
  PRE user ∈ USER
  THEN ok := bool ( user ∈ users )
  END ;
ok ← book_in_library ( book ) ≐
  PRE book ∈ BOOK
  THEN ok := bool ( book ∈ books_in_library )
  END ;
ok ← library_full ≐
  BEGIN
    ok := bool ( card ( books_in_library ) = maxlibrary )
  END ;
ok ← book_on_loan ( book ) ≐
  PRE book ∈ BOOK
  THEN ok := bool ( book ∈ dom ( books_on_loan ) )
  END ;
ok ← book_on_loan_to_user ( book , user ) ≐
  PRE book ∈ BOOK ∧ user ∈ USER
  THEN ok := bool ( book ∈ dom ( books_on_loan ) ∧
    books_on_loan ( book ) = user )
  END ;
ok ← book_on_loan_or_collect ( book ) ≐
  PRE book ∈ BOOK
  THEN ok := bool ( book ∉ dom ( books_on_loan ) ⇒
    book ∈ dom ( collect ) )
  END ;
ok ← reservation_full ( book ) ≐
  PRE book ∈ BOOK
  THEN ok := bool ( book ∈ dom ( reserved ) ∧
    size ( reserved ( book ) ) = maxreserve )
  END ;
ok ← book_reserved ( book ) ≐
  PRE book ∈ BOOK
  THEN ok := bool ( book ∈ dom ( reserved ) )

```

```

END ;
ok ← book_reserved_by_user ( book , user ) ≐
PRE book ∈ BOOK ∧ user ∈ USER
THEN ok := bool ( book ∈ dom ( reserved ) ∧
    user ∈ ran ( reserved ( book ) ) )
END ;
ok ← book_for_collection ( book ) ≐
PRE book ∈ BOOK
THEN ok := bool ( book ∈ dom ( collect ) )
END ;
ok ← book_for_collection_by_user ( book , user ) ≐
PRE book ∈ BOOK ∧ user ∈ USER
THEN ok := bool ( book ∈ dom ( collect ) ⇒
    collect ( book ) = user )
END ;
ok ← book_available_to_browse ( book ) ≐
PRE book ∈ BOOK
THEN ok := bool ( book ∈ books_on_shelf ∧ book ∉ browsing )
END ;
ok ← book_in_browsing ( book ) ≐
PRE book ∈ BOOK
THEN ok := bool ( book ∈ browsing )
END
END

```

10 A Robust Library machine

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)

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 }

OPERATIONS

response , *newuser* \leftarrow **RegisterUser** $\hat{=}$
IF *users* \neq *USER*
THEN *newuser* \leftarrow *NewUser* || *response* := *OK*
ELSE *newuser* := *USER* || *response* := *NoNewUsers*
END ;

response \leftarrow **AcquireBook** (*book*) $\hat{=}$
PRE *book* \in *BOOK* **THEN**
SELECT *book* \in *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 ;

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 ;

```

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 ;

```

```

response ← EndBrowseBook ( book ) ≐
PRE book ∈ BOOK THEN
  IF book ∉ browsing
  THEN response := NotBeingBrowsed
  ELSE EndBrowse ( book ) || response := OK
  END
END ;

```

```

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 ) ≐
PRE book ∈ BOOK THEN
  IF book ∈ dom ( books_on_loan )

```

```

THEN Return1 ( book ) || response := OK
ELSE response := BookNotOnLoan
END
END ;

```

```

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 ;

```

```

response  $\leftarrow$  ReserveBook ( user , book )  $\hat{=}$ 
PRE user  $\in$  USER  $\wedge$  book  $\in$  BOOK THEN
  CHOICE
    SELECT user  $\notin$  users
      THEN response := NotRegisteredUser
    WHEN book  $\notin$  dom ( books_on_loan )  $\wedge$ 
      book  $\notin$  dom ( collect )
      THEN response := InvalidReservation
    WHEN book  $\in$  dom ( books_on_loan )  $\wedge$ 
      books_on_loan ( book ) = user
      THEN response := InvalidReservation
    WHEN book  $\in$  dom ( collect )  $\wedge$ 
      collect ( book ) = user
      THEN response := InvalidReservation
    WHEN book  $\in$  dom ( reserved )  $\wedge$ 
      user  $\in$  ran ( reserved ( book ) )
      THEN response := InvalidReservation
    WHEN book  $\in$  dom ( reserved )  $\wedge$ 
      size ( reserved ( book ) ) = maxreserve
      THEN response := ReserveQueueFull
    ELSE Reserve ( user , book ) || response := OK
  END
OR
  response := FAIL
END

```

END ;

$response \leftarrow \text{CancelBookReserve} (user , book) \hat{=}$

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

SELECT $user \notin users$

THEN $response := NotRegisteredUser$

WHEN $book \in \text{dom} (reserved) \Rightarrow$

$user \notin \text{ran} (reserved (book))$

THEN $response := NotReservedForUser$

ELSE

$CancelReservation (user , book) \parallel$

$response := OK$

END

END ;

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

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

SELECT $user \notin users$

THEN $response := NotRegisteredUser$

WHEN $book \in \text{dom} (collect) \Rightarrow collect (book) \neq user$

THEN $response := NotForCollection$

ELSE $UnCollect (user , book) \parallel response := OK$

END

END

END

11 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 \leftarrow **AcquireBook** (*book*) $\hat{=}$

VAR *bb* **IN**

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

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

response := *BookInLibrary*

ELSE *bb* \leftarrow *library_full* ;

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

response := *LibraryFull*

ELSE *AddNewBook* (*book*) ; *response* := *OK*

END

END

END ;

response \leftarrow **ExitLibrary** (*book*) $\hat{=}$

VAR *binl* , *bonl* **IN**

response := *UnBorrowedBook* ;

binl \leftarrow *book_in_library* (*book*) ;

IF *binl* = *TRUE* **THEN**

bonl \leftarrow *book_on_loan* (*book*)

END ;

IF *binl* = *FALSE* \vee *bonl* = *TRUE* **THEN**

RemoveBook (*book*) ; *response* := *OK*

END
END ;

response \leftarrow **BeginBrowseBook** (*book*) $\hat{=}$
VAR *bb* **IN**
 bb \leftarrow *book_available_to_browse* (*book*) ;
 IF *bb* = *FALSE* **THEN**
 response := *BookNotAvailable*
 ELSE *BeginBrowse* (*book*) ; *response* := *OK*
 END
END ;

response \leftarrow **EndBrowseBook** (*book*) $\hat{=}$
VAR *bb* **IN**
 bb \leftarrow *book_in_browsing* (*book*) ;
 IF *bb* = *FALSE* **THEN**
 response := *NotBeingBrowsed*
 ELSE *EndBrowse* (*book*) ; *response* := *OK*
 END
END ;

response \leftarrow **BorrowBook** (*user* , *book*) $\hat{=}$
VAR *bb* **IN**
 response := *NotRegisteredUser* ;
 bb \leftarrow *user_registered* (*user*) ;
 IF *bb* = *TRUE* **THEN**
 response := *BookNotForLoan* ;
 bb \leftarrow *book_in_library* (*book*) ;
 IF *bb* = *TRUE* **THEN**
 bb \leftarrow *book_on_loan* (*book*) ;
 IF *bb* = *FALSE* **THEN**
 bb \leftarrow *book_for_collection_by_user* (*book* , *user*) ;
 IF *bb* = *TRUE* **THEN**
 Borrow2 (*user* , *book*) ; *response* := *OK*
 END
 END
 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 ← WhoBorrowed ( book ) ≐
VAR bb IN
    bb ← book_on_loan ( book ) ;
    IF bb = TRUE THEN
        user ← Borrowed ( book ) ; response := OK
    ELSE
        user ← 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 ;
                END ;
            END ;
        END ;
    END ;

```

```

        END ;
        IF bb = FALSE THEN
            Reserve ( user , book ) ; response := OK
        END
    END
END
END
END
END ;

```

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

```

VAR bb IN
    response := NotRegisteredUser ;
    bb  $\leftarrow$  user_registered ( user ) ;
    IF bb = TRUE THEN
        response := NotReservedForUser ;
        bb  $\leftarrow$  book_reserved ( book ) ;
        IF bb = TRUE THEN
            bb  $\leftarrow$  book_reserved_by_user ( book , user ) ;
            IF bb = TRUE THEN
                CancelReservation ( user , book ) ;
                response := OK
            END
        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
END
END
END

12 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.

12.1 Library Base Specification

SYSTEM *LibraryDB*
SUPPORTS *LibraryI*
IS
GLOBAL
browsed \in SET (*librarydb*) [100]
END ;

BASE *librarydb*
MANDATORY
bookid \in *BOOK*
OPTIONAL
loan \in *USER* ;
reserveq \in SEQ (*USER*) [10] ;
tocollect \in *USER*
END
END

12.2 The LibraryI implementation

IMPLEMENTATION *LibraryI*
REFINES *Library*
SEES
User_CTX ,
Book_TYPE ,
LibraryDBCtx ,
Bool_TYPE
IMPORTS
user_Nvar (*maxuser*) ,
LibraryDB (*maxlibrary* , *BOOK* , *USER*)

PROPERTIES

$USER = 1 \dots maxuser$

INVARIANT

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

ASSERTIONS

$\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge bookid (ldb) = bb \Rightarrow$
 $bb \in books_in_library) \wedge$
 $\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge ldb \in \text{dom} (loan) \Rightarrow$
 $bb \in \text{dom} (books_on_loan)) \wedge$
 $\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge ldb \in \text{dom} (reserveq) \Rightarrow$
 $bb \in \text{dom} (reserved)) \wedge$
 $\forall (bb , uu , ldb) . (bb \in BOOK \wedge uu \in USER \wedge ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge ldb \in \text{dom} (reserveq) \wedge$
 $uu \in \text{ran} (reserveq (ldb)) \Rightarrow$
 $uu \in \text{ran} (reserved (bb))) \wedge$
 $\forall (bb , uu , ldb) . (bb \in BOOK \wedge uu \in USER \wedge ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge ldb \in \text{dom} (reserveq) \wedge$
 $\text{size} (reserveq (ldb)) = maxreserve \Rightarrow$
 $\text{size} (reserved (bb)) = maxreserve) \wedge$
 $\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge ldb \in \text{dom} (tocollect) \Rightarrow$
 $bb \in \text{dom} (collect)) \wedge$
 $\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge ldb \in browsed \Rightarrow$
 $bb \in browsing) \wedge$
 $\forall (bb , ldb) . (bb \in BOOK \wedge ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge \neg (ldb \in \text{dom} (loan)) \Rightarrow$
 $\neg (bb \in \text{dom} (books_on_loan))) \wedge$
 $\forall bb . (bb \in BOOK \wedge \exists ldb . (ldb \in librarydb \wedge$
 $bookid (ldb) = bb \wedge \neg (ldb \in \text{dom} (reserveq)))) \Rightarrow$

$$\begin{aligned}
& \neg (bb \in \text{dom} (reserved))) \wedge \\
& \forall (bb , uu , ldb) . (bb \in \text{BOOK} \wedge uu \in \text{USER} \wedge ldb \in \text{librarydb} \wedge \\
& \text{bookid} (ldb) = bb \wedge ldb \in \text{dom} (reserveq) \wedge \\
& \neg (uu \in \text{ran} (reserveq (ldb)))) \Rightarrow \\
& \quad \neg (uu \in \text{ran} (reserved (bb)))) \wedge \\
& \forall (bb , uu , ldb) . (bb \in \text{BOOK} \wedge uu \in \text{USER} \wedge ldb \in \text{librarydb} \wedge \\
& \text{bookid} (ldb) = bb \wedge ldb \in \text{dom} (reserveq) \wedge \\
& \neg (\text{size} (reserveq (ldb)) = \text{maxreserve}) \Rightarrow \\
& \quad \neg (\text{size} (reserved (bb)) = \text{maxreserve})) \wedge \\
& \forall (bb , ldb) . (bb \in \text{BOOK} \wedge ldb \in \text{librarydb} \wedge \\
& \text{bookid} (ldb) = bb \wedge \neg (ldb \in \text{dom} (tocollect))) \Rightarrow \\
& \quad \neg (bb \in \text{dom} (collect))) \wedge \\
& \forall (bb , ldb) . (bb \in \text{BOOK} \wedge ldb \in \text{librarydb} \wedge \\
& \text{bookid} (ldb) = bb \wedge \neg (ldb \in \text{browsed}) \Rightarrow \\
& \quad \neg (bb \in \text{browsing}))
\end{aligned}$$

OPERATIONS

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

BEGIN

user_INC_NVAR ;

newuser \leftarrow *user_VAL_NVAR*

END ;

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

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

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

Ignore FALSE return

END ;

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

skip ;

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

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

bb , *db* \leftarrow *key_search_bookid* (*book*) ;

bb \leftarrow *add_browsed* (*db*)

END ;

```

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

```

```

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 ;

```

```

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  $\leftarrow$  val_lth_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

```

  END
END ;

```

```

user  $\leftarrow$  Borrowed ( book )  $\hat{=}$ 
  VAR bb , db IN
    bb , db  $\leftarrow$  key_search_bookid ( book ) ;
    user  $\leftarrow$  val_loan ( db )
  END ;

```

```

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 ;

```

```

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 ;

```

```

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 ;

```

```

ok  $\leftarrow$  users_not_full  $\hat{=}$ 
  ok  $\leftarrow$  user_NEQ_NVAR ( maxuser ) ;
ok  $\leftarrow$  user_registered ( user )  $\hat{=}$ 
  ok  $\leftarrow$  user_GEQ_NVAR ( user ) ;
ok  $\leftarrow$  book_in_library ( book )  $\hat{=}$ 
  VAR db IN

```

```

    ok , db ← key_search_bookid ( book )
END ;

```

```

ok ← library_full ≐
VAR nn IN
    nn ← nbr_librarydb ;
    IF nn = maxlibrary
    THEN ok := TRUE
    ELSE ok := FALSE
    END
END ;

```

```

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

```

```

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 ;

```

```

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 ;

```

```

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
END ;
ok := bb
END ;

```

```

ok ← book_reserved ( book ) ≐
VAR bb , db IN
    bb , db ← key_search_bookid ( book );
    IF bb = TRUE THEN
        bb ← def_reserveq ( db )
    END ;
ok := bb
END ;

```

```

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

```

```
END ;  
  ok := bb  
END ;
```

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

```
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 ;
```

```
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 ;
```

```
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
```

A Base machine LibraryDB

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
card (*librarydb*) \leq *max_Librarydb* \wedge
locate_Librarydb $\in 1 \dots \text{card}$ (*librarydb*) \Rightarrow *librarydb* \wedge
bookid \in *librarydb* \rightarrow *BOOK* \wedge
loan \in *librarydb* \leftrightarrow *USER* \wedge
reserveq \in *librarydb* \leftrightarrow *seq* (*USER*) \wedge
tocollect \in *librarydb* \leftrightarrow *USER*

INITIALISATION

browsed := { } ||
librarydb := { } ||
locate_Librarydb := { } ||
bookid := { } ||
loan := { } ||
reserveq := { } ||
tocollect := { }

OPERATIONS

rep \leftarrow **add_browsed** (*Elem_Librarydb*) $\hat{=}$

PRE *Elem_Librarydb* \in *librarydb* **THEN**

CHOICE

browsed := *browsed* \cup { *Elem_Librarydb* } ||

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 ∈ librarydb THEN
  rep := bool ( Elem_Librarydb ∈ browsed )
END ;
rep , Base_Librarydb ← make_librarydb ( Val_bookid ) ≐
PRE
  Val_bookid ∈ BOOK ∧
  card ( librarydb ) < max_librarydb
THEN
  CHOICE
    ANY Base_Librarydbx , loc WHERE
      Base_Librarydbx ∈ librarydb_ABSOBJ − librarydb ∧
      loc ∈ 1 . . card ( librarydb ) + 1 ↦ librarydb ∪ { Base_Librarydbx }
    THEN
      librarydb := librarydb ∪ { Base_Librarydbx } ||
      bookid ( Base_Librarydbx ) := Val_bookid ||
      Base_Librarydb := Base_Librarydbx ||
      locate_librarydb := loc ||
      rep := TRUE
    END
    OR
    ANY Base_Librarydbx WHERE
      Base_Librarydbx ∈ librarydb_ABSOBJ
    THEN
      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 ;
cre_loan ( Base_Librarydb , Elem_USER ) ≐
  PRE Base_Librarydb ∈ librarydb − dom ( loan ) ∧ Elem_USER ∈ USER THEN
    loan ( Base_Librarydb ) := Elem_USER
  END ;
rem_loan ( Base_Librarydb ) ≐
  PRE Base_Librarydb ∈ dom ( loan ) THEN
    loan := { Base_Librarydb } ≪ loan

```

```

END ;
rep ← def_loan ( Base_Librarydb ) ≐
  PRE Base_Librarydb ∈ librarydb THEN
    rep := bool ( Base_Librarydb ∈ dom ( loan ) )
  END ;
Elem_USER ← val_loan ( Base_Librarydb ) ≐
  PRE Base_Librarydb ∈ dom ( loan ) THEN
    Elem_USER := loan ( Base_Librarydb )
  END ;
mod_loan ( Base_Librarydb , Elem_USER ) ≐
  PRE Base_Librarydb ∈ dom ( loan ) ∧ Elem_USER ∈ USER THEN
    loan ( Base_Librarydb ) := Elem_USER
  END ;
rep ← eq_loan ( Base_Librarydb , Elem_USER ) ≐
  PRE Base_Librarydb ∈ dom ( loan ) ∧ Elem_USER ∈ USER THEN
    rep := bool ( loan ( Base_Librarydb ) = Elem_USER )
  END ;
rep ← cre_reserveq ( Base_Librarydb ) ≐
  PRE Base_Librarydb ∈ librarydb – dom ( reserveq ) THEN
    CHOICE
      reserveq ( Base_Librarydb ) := [] ||
      rep := TRUE
    OR
      rep := FALSE
    END
  END ;
rem_reserveq ( Base_Librarydb ) ≐
  PRE Base_Librarydb ∈ dom ( reserveq ) THEN
    reserveq := { Base_Librarydb } ≪ reserveq
  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 ) ≠ [] THEN
    Elem_USER := last ( reserveq ( Base_Librarydb ) )
  END ;
Elem_USER ← valIth_reserveq ( Base_Librarydb , ii ) ≐
  PRE Base_Librarydb ∈ dom ( reserveq ) ∧ ii ∈ 1 .. size ( reserveq ( Base_Librarydb ) ) THEN
    Elem_USER := reserveq ( Base_Librarydb ) ( ii )
  END ;
rep ← dellth_reserveq ( Base_Librarydb , ii ) ≐
  PRE Base_Librarydb ∈ dom ( reserveq ) ∧ ii ∈ 1 .. size ( reserveq ( Base_Librarydb ) ) THEN
    CHOICE

```

```

    reserveq ( Base_librarydb ) :=
    reserveq ( Base_librarydb ) ↑ ii - 1 ∧ ( reserveq ( Base_librarydb ) ↓ ii ) ||
    rep := TRUE
OR
    rep := FALSE
END
END ;
rep ← push_reserveq ( Base_librarydb , Elem_USER ) ≐
PRE Base_librarydb ∈ dom ( reserveq ) ∧ Elem_USER ∈ USER THEN
CHOICE
    reserveq ( Base_librarydb ) := reserveq ( Base_librarydb ) ∧ [ Elem_USER ] ||
    rep := TRUE
OR
    rep := FALSE
END
END ;
rep , ii ← within_reserveq ( Base_librarydb , Elem_USER ) ≐
PRE Base_librarydb ∈ dom ( reserveq ) ∧ Elem_USER ∈ USER THEN
IF Elem_USER ∈ ran ( reserveq ( Base_librarydb ) ) THEN
    rep := TRUE || ii := ( reserveq ( Base_librarydb ) )-1 [ { Elem_USER } ]
ELSE
    rep := FALSE || ii := ℕ
END
END ;
cre_tocollect ( Base_librarydb , Elem_USER ) ≐
PRE Base_librarydb ∈ librarydb - dom ( tocollect ) ∧ Elem_USER ∈ USER THEN
    tocollect ( Base_librarydb ) := Elem_USER
END ;
rem_tocollect ( Base_librarydb ) ≐
PRE Base_librarydb ∈ dom ( tocollect ) THEN
    tocollect := { Base_librarydb } ≪ tocollect
END ;
rep ← def_tocollect ( Base_librarydb ) ≐
PRE Base_librarydb ∈ librarydb THEN
    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

```