# COMP2111 Assignment 3

Ken Robinson

16th April 2012

Name of assignment: **ass3**
Due date: **27th April 2012**
Assessment: 15 marks
Submission: give cs2111 ass3 Library.zip

## 1 Overview of assignment

This assignment extends the tutorial example of a simple library (see 2.4.2). The extensions are:

1. addition of a borrowing limit;
2. addition of a reservation capability

The assignment archive *Library.zip* contains the following components:

**Library_ctx:** context for *Library* machine;

**List_ctx:** list context for *LibraryR2*

Each extension should be modelled as a refinement.

**Note** To create a refinement of a machine it is best to use the Event-B Explorer in Rodin; *don't create the refinement from scratch* by hand.

In the Event-B Explorer right click on the machine you want to refine and then choose *Refine* from the options. Fill in the name of the refinement machine and Rodin will create a base for your refinement with all events being extended. In some cases you will not want an extension, for example when you want to modify the guards of an event, not simply add more guards. In such cases you will want to turn off extension for such events.



## 2 Refinements

### 2.1 LibraryR0: Borrowing limit

LibraryR0 is a very simple refinement that use the constant *maxloan* to set a borrowing limit on number of books that a member can borrow at any time. The constant does not have a value; it is simply of type $\mathbb{N}1$. For animation with AnimB a value would be required in the AnimB values for Library_ctx.

You should be try to discharge the proof obligations, but some are a little tricky. They probably will not be all auto-proved. You will have to use proof by cases (the *dc* button in the proof control) as most of the lemmas will have $m \in members$ and there will be two cases: $m = member$ and $m \neq member$.

## 2.2 Adding book reservation

Book reservation is concerned with reserving a book that is currently borrowed.

The following constraints apply to reservation of *book* by *member*:

**Person reserving must be a *member*** same as constraint for borrowing;

**Book being reserved must be currently *onloan***

**Book must not be currently reserved for *member*** a book may be reserved at most once for any particular member;

**Books may have multiple reservations** by different members.

**Reservations can be cancelled** by the member that requested the reservation.

Modelling of reservation should be done in two stages represented by *LibraryR1* and *LibraryR2*.

**LibraryR1** refines Library, and

**LibraryR2** refines LibraryR1



## 2.3 LibraryR1

*LibraryR1* should refine *LibraryR0* and model reservation with no priority. That is, when a book that has been reserved is returned it can be borrowed by any one of the members who reserved that book.



A book cannot be reserved by the current borrower of the book.

## 2.4 LibraryR2



*LibraryR2* should refine *LibraryR1* and reservations should now satisfy the following:

1. Reservations are queued. That is reservations for the same book are queued in the order in which the reservation requests were made.
2. When a reserved book is returned it is then available for the first member on the queue to borrow. The book is not available for general borrowing until all on the reservation queue have borrowed it.
3. A member who has requested reservation of a book can cancel their reservation, in which case the queue must "close up".

This refinement will be a *data refinement*.

### 2.4.1 The List context

*List_ctx* contains a list algebra that you should use for the book reservation queue. Lists are modelled as functions, so $l(i)$ is the i-th element of the list $l$. There are two types of list models provided: $LIST$ ordinary lists and $ILIST$ injective lists, which are lists in which there are no repeated values. The algebra provides you with the following operations on lists:

**APPEND** $APPEND(l \mapsto m)$ appends $m$ to the end of the list $l$, so you could write $l := APPEND(l \mapsto m)$;

**DEQUEUE** $DEQUEUE(l)$ removes the head (first) item on the list, for example
$l := DEQUEUE(l)$;

**JOIN** $JOIN(l1 \mapsto l2)$ joins the two lists $l1$ and $l2$, for example
$l := JOIN(l1 \mapsto l2)$;

**DELETE** $DELETE(l \mapsto i)$ deletes the i-th element of the list $l$, for example
$l := DELETE(l \mapsto i)$.

**IDELETE** a slightly simpler version of $DELETE$ that can be used on injective lists. IDELETE deletes the member $m$ from the injective list $l$
$l := IDELETE(l \mapsto m)$.

The context provides quantifiers for determining $dom$, $ran$ and indexing of list combinations.

### 2.4.2 What you should do

First, download the archive, *Library.zip*, containing the contexts *Library_ctx*, *List_ctx* and the machine *Library*.

*Partial versions of the refinements are not included as it is best if you use Rodin to generate the initial refinements*



**Create and complete LibraryR0** to introduce a borrowing limit.

**Create and modify LibraryR1** to add simple reservation and cancelling of a reservation.



**Create and modify LibraryR2** to add a priority queue for reservation

**Discharge or at least review the proof obligations** it can be expected that the POs will be generally difficult, but they should be reviewed to detect errors in your models.

**CONTEXT** Library_ctx

**SETS**

MEMBER

BOOK

**CONSTANTS**

maxloan

**AXIOMS**

axm1 : $finite(MEMBER)$

axm2 : $finite(BOOK)$

axm3 : $maxloan \in \mathbb{N}_1$

**END**

**MACHINE** Library

**SEES** Library_ctx

**VARIABLES**

`books` books contained in the library

`members` members of the library

`borrowed` books borrowed by each member

**INVARIANTS**

`inv1` : $books \subseteq BOOK$

`inv2` : $members \subseteq MEMBER$

`inv3` : $borrowed \in books \pfun members$

**EVENTS**

**Initialisation**

**begin**

`act1` : $books := \varnothing$

`act2` : $members := \varnothing$

`act3` : $borrowed := \varnothing$

**end**



**Event** *NewMember* $\widehat{=}$

Add a new member of the library

**any**

*member*

**where**

`grd1` : $member \in MEMBER \setminus members$

**then**

`act1` : $members := members \cup \{member\}$

**end**

**Event** *AddBook* $\widehat{=}$

Add a book to the library; this could be a new book for the library or an extra copy

**any**

*book*

**where**

`grd1` : $book \in BOOK \setminus books$

**then**

`act1` : $books := books \cup \{book\}$

**end**

**Event** *Borrrow* $\widehat{=}$

A member borrows a book

**any**

*book*

*member*

**where**

grd1 : $book \in books$

grd2 : $member \in members$

grd3 : $book \notin dom(borrowed)$

**then**

act1 : $borrowed(book) := member$

**end**

**Event** *Return* $\widehat{=}$

A member returns a book

**any**

*book*

**where**

grd1 : $book \in dom(borrowed)$

**then**

act1 : $borrowed := \{book\} \lhd borrowed$

**end**

**END**

**CONTEXT** List_ctx

This context presents a small theory of lists.
Lists might also be called sequences.
Both injective and non-injective lists will be modelled.
All elements of an injective list are distinct.

**EXTENDS** Library_ctx

**CONSTANTS**

`LENGTH` Finite limit on length of lists

`LIST` Set of lists

`ILIST` Set of injective lists

`JOIN` List concatenation operator

`APPEND` Append an item to tail of list

`IAPPEND` Append maintaining injectivity

`DEQUEUE` Delete head of list

`DELETE` Delete an element from any position of a list

`IDELETE` Delete an element from an injective list



**AXIOMS**

axm1 : $LENGTH \in \mathbb{N}$

axm2 : $LIST = \{l \mid l \in \mathbb{N}_1 \pfun MEMBER \land dom(l) = 1 \upto card(l)\}$

axm60 : $finite(LIST)$

axm3 : $\varnothing \in LIST$

axm4 : $ILIST = \{l \mid l \in LIST \land l \in \mathbb{N}_1 \pinj MEMBER\}$

axm5 : $ILIST \subset LIST$

axm6 : $\varnothing \in ILIST$

axm7 : $\forall l \cdot l \in ILIST \Leftrightarrow l \in LIST \land l \in 1 \upto LENGTH \tinj MEMBER$

axm8 : $\forall l \cdot l \in LIST \Rightarrow ran(l) = l[dom(l)]$

axm9 : $\forall l \cdot l \in LIST \Rightarrow ran(l) = l[1 \upto card(l)]$

axm10 : $\forall l \cdot l \in LIST \land card(ran(l)) = card(dom(l))$
$\Rightarrow l \in ILIST$

axm11 : $JOIN \in (LIST \times LIST) \to LIST$

axm12 : $dom(JOIN) = LIST \times LIST$

axm13 : $\forall l1, l2 \cdot l1 \in LIST \land l2 \in LIST$
$\Rightarrow$
$dom(JOIN(l1 \mapsto l2)) = 1 \upto card(l1) + card(l2)$

axm14 : $\forall l1, l2, i \cdot l1 \in LIST \land l2 \in LIST \land i \in dom(JOIN(l1 \mapsto l2))$
$\Rightarrow$
$(i \in 1 \upto card(l1) \Rightarrow JOIN(l1 \mapsto l2)(i) = l1(i))$
$\land$
$(i - card(l1) \in 1 \upto card(l2) \Rightarrow JOIN(l1 \mapsto l2)(i) = l2(i - card(l1)))$

axm15 : $\forall l \cdot l \in LIST$
$\Rightarrow$
$JOIN(l \mapsto \varnothing) = l$

axm16 : $\forall l \cdot l \in LIST$
$\Rightarrow$
$JOIN(\varnothing \mapsto l) = l$

axm17 : $\forall l1, l2 \cdot l1 \in ILIST \land l2 \in ILIST \land ran(l1) \cap ran(l2) = \varnothing$
$\Rightarrow$
$JOIN(l1 \mapsto l2) \in ILIST$

axm18 : $\forall l1, l2 \cdot l1 \in LIST \land l2 \in LIST$
$\Rightarrow$
$ran(JOIN(l1 \mapsto l2)) = ran(l1) \cup ran(l2)$

axm19 : $\forall l1, l2 \cdot l1 \in LIST \land l2 \in LIST$
$\Rightarrow$
$card(JOIN(l1 \mapsto l2)) = card(l1) + card(l2)$

axm20 : $APPEND \in (LIST \times MEMBER) \rightarrow LIST$

axm21 : $dom(APPEND) = LIST \times MEMBER$

axm22 : $\forall l, m \cdot l \in LIST$
$\Rightarrow$
$dom(APPEND(l \mapsto m)) = 1 \mathinner{..} card(l) + 1$

axm23 : $\forall l, m, i \cdot l \in LIST \land i \in dom(APPEND(l \mapsto m))$
$\Leftrightarrow$
$(i \in dom(l) \Rightarrow APPEND(l \mapsto m)(i) = l(i))$
$\land$
$(i = card(l) + 1 \Rightarrow APPEND(l \mapsto m)(i) = m)$

axm24 : $\forall l, m \cdot l \in LIST \land m \in MEMBER$
$\Rightarrow$
$ran(APPEND(l \mapsto m)) = ran(l) \cup \{m\}$

axm25 : $\forall l, m \cdot l \in LIST \land m \in MEMBER$
$\Rightarrow$
$card(APPEND(l \mapsto m)) = card(l) + 1$

axm26 : $\forall l, m \cdot l \in ILIST \land m \in MEMBER \land m \notin ran(l)$
$\Rightarrow$
$APPEND(l \mapsto m) \in ILIST$

axm27 : $IAPPEND \in (ILIST \times MEMBER) \rightarrow ILIST$

axm28 : $dom(IAPPEND) = ILIST \times MEMBER$

axm29 : $\forall l, m \cdot l \in LIST$
$\Rightarrow$
$dom(IAPPEND(l \mapsto m)) = dom(APPEND(l \mapsto m))$

axm30 : $\forall l, m \cdot l \in ILIST \land m \notin ran(l)$
$\Rightarrow$
$IAPPEND(l \mapsto m) = APPEND(l \mapsto m)$

axm31 : $\forall l, m \cdot l \in LIST \land m \in MEMBER$
$\Rightarrow$
$card(APPEND(l \mapsto m)) = card(l) + 1$

axm32 : $DEQUEUE \in LIST \rightarrow LIST$

axm33 : $dom(DEQUEUE) = LIST$

axm34 : $\forall l \cdot l \in LIST \land l \neq \varnothing$
$\Rightarrow$
$dom(DEQUEUE(l)) = 1 \mathinner{..} card(l) - 1$

axm35 : $\forall l, i \cdot l \in LIST \land l \neq \varnothing \land i \in 1 \mathinner{..} card(l) - 1$
$\Rightarrow$
$DEQUEUE(l)(i) = l(i + 1)$

axm36 : $\forall l \cdot l \in ILIST \land l \neq \varnothing$
$\Rightarrow$
$DEQUEUE(l) \in ILIST$

axm37 : $\forall l \cdot l \in LIST \land l \neq \varnothing$
$\Rightarrow$
$ran(DEQUEUE(l)) = ran(l) \setminus \{l(1)\}$

axm38 : $DELETE \in (LIST \times (1 \mathinner{..} LENGTH)) \pfun LIST$

axm39 : $dom(DELETE) \subseteq LIST \times (1 \mathinner{..} LENGTH)$

axm40 : $\forall l, i \cdot l \in LIST \land i \in dom(l)$
$\Rightarrow$
$l \mapsto i \in dom(DELETE)$

axm41 : $\forall l, i \cdot l \in LIST \land i \in dom(l)$
$\Rightarrow$
$dom(DELETE(l \mapsto i)) = 1 \mathinner{..} card(l) - 1$

axm42 : $\forall l, i, j \cdot l \in LIST \land i \in dom(l) \land j \in 1 \mathinner{..} card(l) - 1$
$\Rightarrow$
$j \in dom(DELETE(l \mapsto i))$

axm43 : $\forall l, i, j \cdot l \in LIST \land i \in dom(l) \land j \in 1 \mathinner{..} i - 1$
$\Rightarrow$
$DELETE(l \mapsto i)(j) = l(j)$

axm44 : $\forall l, i, j \cdot l \in LIST \land i \in dom(l) \land j \in i \mathinner{..} card(l) - 1$
$\Rightarrow$
$DELETE(l \mapsto i)(j) = l(j + 1)$

axm45 : $\forall l, i \cdot l \in LIST \land i \in dom(l)$
$\Rightarrow$
$card(DELETE(l \mapsto i)) = card(l) - 1$

axm46 : $\forall l, i \cdot l \in LIST \land i \in dom(l)$
$\Rightarrow$
$ran(DELETE(l \mapsto i)) = ran(l) \setminus \{l(i)\}$

axm47 : $\forall l, i \cdot l \in LIST \land i \in dom(l)$
$\Rightarrow$
$card(DELETE(l \mapsto i)) = card(l) - 1$

axm48 : $IDELETE \in (ILIST \times MEMBER) \pfun ILIST$

axm49 : $dom(IDELETE) = ILIST \times MEMBER$

axm50 : $\forall l, m \cdot l \in ILIST \land m \in MEMBER \land m \in ran(l)$
$\Rightarrow$
$l \mapsto m \in dom(IDELETE)$

axm51 : $\forall l, m \cdot l \in ILIST \land m \in ran(l)$
$\Rightarrow$
$dom(IDELETE(l \mapsto m)) = 1 \mathinner{..} card(l) - 1$

axm52 : $\forall l, m \cdot l \in ILIST \land m \in ran(l)$
$\Rightarrow$
$IDELETE(l \mapsto m) = DELETE(l \mapsto l^{-1}(m))$

axm53 : $\forall l, m \cdot l \in ILIST \land m \in ran(l)$
$\Rightarrow$
$card(IDELETE(l \mapsto m)) = card(l) - 1$

axm54 : $\forall l, m \cdot l \in ILIST \land m \in ran(l)$
$\Rightarrow$
$ran(IDELETE(l \mapsto m)) = ran(l) \setminus \{m\}$

axm55 : $\forall l \cdot l \in LIST \land l \neq \varnothing$
$\Rightarrow$
$DEQUEUE(l) = DELETE(l \mapsto 1)$

axm56 : $\forall l \cdot l \in LIST \land l \neq \varnothing$
$\Rightarrow$
$card(DEQUEUE(l)) = card(l) - 1$



axm57 : $\forall l, i \cdot l \in ILIST \land i \in dom(l)$
$\Rightarrow$
$DEQUEUE(l) \in ILIST$



axm58 : $\forall l, i \cdot l \in ILIST \land i \in dom(l)$
$\Rightarrow$
$DELETE(l \mapsto i) \in ILIST$



axm59 : $\forall l, m \cdot l \in LIST \land m \in MEMBER$
$\Rightarrow$
$DELETE(APPEND(l \mapsto m) \mapsto card(l) + 1) = l$

**END**