Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 280 (2011) 47–56

www.elsevier.com/locate/entcs

Proving Reachability in B using Substitution Refinement

Marc Frappier[a,1,2] Fama Diagne[a,b,3] Amel Mammar[b,4]

[a] *GRIL, Dép. d'informatique, Université de Sherbrooke, Sherbrooke (Québec), Canada*

[b] *Institut Telecom SudParis, CNRS/SAMOVAR, Paris, France*

Abstract

This paper proposes an approach to prove reachability properties of the form AG ($\psi \Rightarrow$ EF ϕ) using substitution refinement in classical B. Such properties denote that there exists an execution path for each state satisfying ψ to a state satisfying ϕ. These properties frequently occur in security policies and information systems. We show how to use Morgan's specification statement to represent a property and refinement laws to prove it. The idea is to construct by stepwise refinement a program whose elementary statements are operation calls. Thus, the execution of such a program provides an execution satisfying AG ($\psi \Rightarrow$ EF ϕ). Proof obligations are represented using assertions (ASSERTIONS clause of B) and can be discharged using Atelier B.

Keywords: B Notation, proof, reachability, CTL, refinement calculus

1 Introduction

Reachability properties frequently occurs in information systems and security policies. For example, in a library system, a typical property is that a member should always be able to borrow a book. If the book is available and the member hasn't reached his loan limit, he can proceed immediately and borrow the book; if he has reached his loan limit, he can return one of his borrowed book and then borrow the book. If the book is already borrowed by another member, then he can make a reservation and wait for his turn to borrow the book. In this description, we see that the specifier must take into account several cases when proving such a property. They are documented in use cases and scenarios during requirements analysis. In a

[1] This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC)

[2] Email: Marc.Frappier@USherbrooke.ca

[3] Email: Fama.Diagne@USherbrooke.ca

[4] Email: Amel.Mammar@it-sudparis.eu

doi:10.1016/j.entcs.2011.11.017

clinical information system, similar properties arise, especially when access control rules and patient consent rules are used. One wants to ensure that in case of emergencies, doctors can still reach the desired information, or appropriately process patient transfers between departments, etc.

Such reachability properties can be expressed in CTL [8], because one only needs to show the existence of a path. LTL [18] is inappropriate, because LTL properties must be satisfied by all execution paths. Our reachability properties need not be satisfied by all execution path: for instance, a member is never forced to borrow a book. In fact, such properties triggered the definition of CTL in the early 80's.

This form of reachability is expressed in CTL as $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \ \phi)$. This formula denotes that there exists an execution path from each state satisfying ψ to a state satisfying ϕ. In the context of proving CTL properties for classical B abstract machines, an execution path is a sequence of operation calls. One way to prove a reachability property is to provide a program p, whose elementary statements are operation calls, which are combined with some operators, and to show that $\psi \Rightarrow [p]\phi$. Operator "[ ]" is the traditional substitution semantic operator of the B theory, which is the same as Dijkstra's weakest precondition operator, denoted by $wp(p, \phi)$. This statement essentially states that p, when started in ψ, is guaranteed to terminate in a state satisfying ϕ. By proving this statement, one proves the existence of a path from ψ to ϕ. If one uses B substitutions to construct p, then the B theory can be used to prove this statement.

Existing tools like Atelier B cannot directly handle an expression like $\psi \Rightarrow [p]\phi$, but such an expression can easily be translated into an assertion, using the laws of "[ ]". One can then use traditional tools like Atelier B to prove it. However, the proof obligations generated from $\psi \Rightarrow [p]\phi$ can be large and complex, so hard to prove. This is why the idea of refinement calculus was introduced [2,3,10,13,15,17]. In this paper, we show how to prove these statements using the well-documented refinement calculus of Carroll Morgan [14] in a B context.

2 Proving Reachability using Substitution Refinement

Morgan has proposed a number of refinement rules to develop sequential programs in a stepwise fashion. He introduced the notion of *specification statement*, denoted by $w : [pre, post]$, that specifies a computation which, when started in a state satisfying pre, must terminate in a state satisfying $post$, by modifying variables w. To avoid any confusion with "[...]" of the B notation, let us write Morgan's specification statement as $\mathsf{Spec}(pre\ ,\ post)$ and consider it as a new B substitution. We eliminate w from the notation, because it suffices for our purpose to implicitly let w denote all variables of a B machine. Note that this statement can be written in B as:

$$w : [pre, post] = \text{PRE } pre \text{ THEN ANY } w' \text{ WHERE } post' \text{ THEN } w := w' \qquad (1)$$

The wp-semantics of $\mathsf{Spec}(pre\ ,\ post)$ is defined as follows:

$$[\mathsf{Spec}(pre\ ,\ post)]Q \quad \Leftrightarrow \quad pre \wedge (\forall\, w \cdot post \Rightarrow Q)$$

One can prove the refinement of a specification statement by a substitution S as follows:

$$\mathsf{Spec}(pre\ ,\ post) \sqsubseteq S \quad \Leftrightarrow \quad (pre \Rightarrow [S]post) \tag{2}$$

Thus, the problem of proving a CTL reachability formula $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \ \phi)$ can be formulated as finding a program S such that $\mathsf{Spec}(\psi\ ,\ \phi) \sqsubseteq S$. We will show how to conduct these proofs using refinement laws proposed by Morgan in [14].

As a first example, let us consider a B machine which describes the behavior of a library system, and show that a member can always borrow a book (see Appendix A). Thus we want to prove the following:

$$\mathsf{AG}\ (me \in member \wedge bo \in book \Rightarrow \mathsf{EF}\ \ bo \mapsto me \in loan)$$

The B machine variables *member*, *book* and *loan* respectively denote the set of members, books and loans of the library. All the variables of this CTL formula are implicitly universally quantified. In the sequel, we will use the following abbreviations:

$$\psi \triangleq me \in member \wedge bo \in book$$
$$\phi \triangleq bo \mapsto me \in loan$$

Thus, we must find a program S such that $\mathsf{Spec}(\psi\ ,\ \phi) \sqsubseteq S$. We will construct S by stepwise, algorithmic refinement (*ie*, B substitution refinement, not B machine refinement) using Morgan's laws. The execution path that S will follow can be summarized as follows: i) if the member has reached his loan limit, return one book borrowed by the member; ii) if the book is borrowed, return it; iii) if the book has reservations, cancel them. This is a brute force strategy in terms of path. Other paths corresponding to more probable scenarios could also be proved. We shall come back to this discussion at the end of the section. For now, this brute force strategy is sufficient to prove our CTL formula.

The first step is to decompose the specification in two parts: the first part will establish a condition C_1 sufficient to satisfy the precondition of operation **Lend**; the second part will start from C_1 and establish ϕ, and we will then show its refinement by a call to **Lend**. So our first refinement step is the following:

$$\mathsf{Spec}(\psi\ ,\ \phi) \sqsubseteq S_1\,;S_2 \tag{3}$$

where

$$S_1 \triangleq \mathsf{Spec}(\psi\ ,\ \psi \wedge C_1)$$
$$S_2 \triangleq \mathsf{Spec}(\psi \wedge C_1\ ,\ \phi)$$
$$C_1 \triangleq C_2 \wedge C_3 \wedge C_4$$
$$C_2 \triangleq card(loan \rhd \{me\}) < MaxNbLoans$$
$$C_3 \triangleq bo \notin \mathsf{dom}(loan)$$
$$C_4 \triangleq reservation(bo) = [\,]$$

Refinement (3) is immediately proved using law 3.3 of [14]:

$$\mathsf{Spec}(pre\ ,\ post) \sqsubseteq \mathsf{Spec}(pre\ ,\ mid)\,;\mathsf{Spec}(mid\ ,\ post) \tag{4}$$

We can refine S_2 by an operation call to **Lend**.

$$S_2 \sqsubseteq \mathbf{Lend}(me, bo) \tag{5}$$

This refinement can be proved using (2).

$$S_2 \sqsubseteq \mathbf{Lend}(me, bo) \quad \Leftrightarrow \quad (\psi \wedge C_1 \Rightarrow [\mathbf{Lend}(me, bo)]\phi)$$

This proof can be discharged by rewriting $[\mathbf{Lend}(me, bo)]\phi$ using standard B axioms, which results in the following formula that can be proved by adding it as an assertion in the Library machine.

$$\begin{aligned} \psi \wedge C_1 \Rightarrow\ & me \in MEMBERID \wedge \\ & me \in member \wedge \\ & bo \in BOOKID \wedge \\ & bo \in book \wedge \\ & bo \notin \mathsf{dom}(loan) \wedge \\ & reservation(bo) = [\,] \wedge \\ & \mathsf{card}(loan \rhd \{me\}) < MaxNbLoans \wedge \\ & bo \mapsto me \in loan \leftarrow\!\!+ \{bo \mapsto me\} \end{aligned}$$

We have now solved S_2 by refining it into a substitution whose elementary statements are operation calls. We will solve S_1 in three steps.

$$\begin{aligned} & S_1 \sqsubseteq S_3\,;S_4 \\ & S_3 \triangleq \mathsf{Spec}(\psi\ ,\ \psi \wedge C_2) \\ & S_4 \triangleq \mathsf{Spec}(\psi \wedge C_2\ ,\ \psi \wedge C_1) \end{aligned} \tag{6}$$

We can again prove (6) using (4).

Specification S_3 states that the member should not have reached it loan limit. To solve it, we can test C_2, which ensures that the loan limit is not reached; if C_2 is false, then we nondeterministically chose a borrowed book of the member and return it, to establish C_2. Thus S_3 is solved as follows:

$$S_3 \sqsubseteq \text{IF } \neg C_2 \text{ THEN } S_5 \text{ END} \tag{7}$$

where

$$S_5 \triangleq \text{ANY } bo' \text{ WHERE } bo' \in loan^{-1}[\{me\}] \text{ THEN } \mathbf{Return}(bo') \text{ END} \tag{8}$$

We can prove (7) using (2) and B axioms for "[IF]", generating the appropriate proof obligation. We can solve S_4 is a similar fashion using an IF statement and a loop.

$S_4 \sqsubseteq S_6 \,;\, S_7$ (9)

$S_6 \triangleq \mathsf{Spec}(\psi \wedge C_2 \ , \ \psi \wedge C_2 \wedge C_3)$

$S_7 \triangleq \mathsf{Spec}(\psi \wedge C_2 \wedge C_3 \ , \ \psi \wedge C_1)$

$S_6 \sqsubseteq$ IF $\neg C_3$ THEN **Return**(bo) END (10)

$S_7 \sqsubseteq$ WHILE $\neg C_4$ (11)
DO S_8
INVARIANT $\psi \wedge C_2 \wedge C_3$
VARIANT $\mathsf{size}(reservation(bo))$ END

$S_8 \triangleq$ ANY me' WHERE $me' \in \mathsf{ran}(reservation(bo))$
THEN **Cancel**(me', bo) END

We can again immediately prove (9) using (4). We can prove (10) using (2) and the B axiom for "[IF]" to generate a proof obligation.

The WHILE statement of (11) is not a syntactically accepted WHILE statement in the B notation, because its loop body does not use implementation substitutions, but the semantics and rules of the WHILE statement are valid for any substitution, so we can use them to generate proof obligations. We can prove (11) using law 5.5 of [14], provided that we discharge the following proof obligations.

$\psi \wedge C_2 \wedge C_3 \wedge \neg C_4 \Rightarrow [S_8]\psi$ (* loop body preserves the invariant *)
$\psi \wedge C_2 \wedge C_3 \wedge \neg C_4 \Rightarrow [n := V][S_8](V < n)$ (* loop body decreases the variant $V \triangleq \mathsf{size}(reservation(bo))$ *)
$\psi \wedge C_2 \wedge C_3 \wedge \neg C_4 \Rightarrow V \in \mathbb{N}$ (* the variant is a natural number *)

Note that there is no proof obligation for the initialisation of the loop, since the loop refines S_7, whose precondition is the loop invariant.

We have added as assertions the proof obligations that we manually generated from (5), (7), (10), and (11). They are provided in the Library specification of Appendix A in the ASSERTIONS clause. These assertions generate 14 PO; 10 are automatically proved and 4 are easily proved with the interactive prover.

Figure 1 provides the program obtained by piecing together the leafs of this refinement tree. By transitivity of refinement, this program refines $\mathsf{Spec}(\psi \ , \ \phi)$, itself representing the CTL formula $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \phi)$. This program executes operation calls, thus it provides an execution path from ψ to ϕ. Other solutions could explore more probable scenarios. For instance, the user could make a reservation when the book is borrowed; a loop over operations **Take** and **Return** or **Cancel** would empty the list of reservations and allow the member to ultimately borrow the book. These alternative solutions should also be devisable by stepwise refinement.

3 Related Work

There were other attempts at implementing Morgan's refinement calculus. The Refinement Calculator [6] uses HOL to formalize the refinement calculus and conduct proofs; PRT [7] uses Ergo. By reusing a B tool, we avoid to formalise the theory of refinement; we instead reuse the theory of B.

IF $\neg C_2$ THEN
  ANY bo' WHERE $bo' \in loan^{-1}[\{me\}]$ THEN **Return**(bo') END
END ;
IF $\neg C_3$ THEN **Return**(bo) END ;
WHILE $\neg C_4$ DO
  ANY me' WHERE $me' \in \mathsf{ran}(reservation(bo))$
  THEN **Cancel**(me', bo) END
 INVARIANT $\psi \wedge C_2 \wedge C_3$
 VARIANT $\mathsf{size}(reservation(bo))$ END ;
Lend(me, bo)

Fig. 1. The program which refines the reachability property "a member can always borrow a book"

In a companion paper [12], we outline an alternative approach to prove reachability properties. We propose an algorithm that, given a *path expression*, generates proof obligations for $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \phi)$. A path expression is an angelic choice between basic paths. A basic path starts with a precondition and includes operation calls or loops on operation calls. The generated proof obligations are slightly more complex than those obtained by the approach proposed here, but the specifier does not have to go through a stepwise refinement process. On the other hand, stepwise refinement provides greater freedom in constructing a program showing the existence of a path.

Brown and Méry [5] have shown how to encode UNITY's *ensures* and *leadsto* modalities in Atelier B using the wp-calculus. Abrial and Mussat [1] introduced the *leadsto* modality of UNITY for an ancestor of Event B. UNITY's *leadsto*, denoted by $\psi \rightsquigarrow \phi$, is defined in LTL as $\Box(\psi \Rightarrow \Diamond\phi)$. This modality is proved by showing that a set of events decrease a variant V, pretty much like the while loop rule 5.5 of [14] which we have used. In [1], events have a guard, which is refined by strengthening it, thus potentially eliminating some execution paths. To ensure that execution paths are preserved by guard refinement, one must also prove that the disjunction of the guards is not strengthened. Because we use classical B with preconditions for operations, substitution refinement directly preserves the existence of a path from ψ to ϕ, so we do not have this additional proof obligation. This also means that any implementation of a B abstract machine will also satisfy our reachability properties.

The work of [1] is extended in [4] by adding UNITY's *ensures* modality as well as weak fairness and minimal progress. We do not take into account fairness in our work, because it is not relevant in information systems (IS), since an IS user is never forced to invoke an action. However, we must take into account progress in the context of while loops. Additional proof rules for *ensures* and *leadsto* are proposed in [20].

The work of Pnueli *et al* [11,19] is probably the closest to ours. It includes a number of rules to prove CTL* properties, also taking into account fairness and justice. Their approach is to reduce CTL* formula into basic formula without temporal connectors which can then be proved using elementary rules. We plan to investigate how to adapt these rules to a refinement context, in order to deal

with more complex patterns of CTL formula. Rule E-UNTIL of [19] is the main one involved for a proof of $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \phi)$. It requires to find an invariant φ and a variant V. We provide it below, instantiating it for EF .

$$\frac{\begin{array}{l}\psi \to \varphi \\ \varphi \to \phi \vee \exists\, w' \cdot \rho(w, w') \wedge \varphi' \wedge V' < V\end{array}}{\psi \to\ \mathsf{EF}\ \phi}\ \text{E-UNTIL}$$

Symbol ρ denotes the transition relation of the system. The notation $\psi_1 \to \psi_2$ denotes $\mathsf{AG}\ \psi_1 \Rightarrow \psi_2$, which means that $\psi_1 \Rightarrow \psi_2$ holds in all *reachable* states. To prove it, it is sufficient to prove $\psi_1 \Rightarrow \psi_2$ for any state (not only states reachable from the initial state). In a B context, one can use machine invariants to prove this. Essentially, rule E-UNTIL is similar to a WHILE proof rule. It is not so easy to use in practice. Our approach allows one to restrict this invariant and variant identification to the iterative part of the program p that we have to construct. The rest is done with other B operators, where termination is implicit. Moreover, E-UNTIL uses an existential quantification, which is an *angelic* sequential composition, which is not monotonic with respect to refinement, because the transition relation is coded like a precondition. During refinement, if the path chosen does not match the value selected by this existential quantification, then reachability is lost during sequential composition of actions. Thus, properties proved using E-UNTIL are not preserved by refinement; in our approach, properties are indeed preserved by refinement. Finally, using refinement allows one to decompose a proof into several small proof obligations, instead of one monolithic proof as required by E-UNTIL.

4 Discussion and Conclusion

We have shown how to prove a reachability property using substitution refinement. We construct by stepwise refinement, reusing Morgan's specification statement and laws, a program whose elementary statements are operations calls. Thus, the execution of this program provides a path to prove the reachability property. Such a program is not written in B0; we take the liberty of freely combining all substitutions offered by the B notation, since our purpose is not to built an executable program. Our only constraint is that elementary substitutions are operation calls.

We could have used machine refinement, instead of substitution refinement, to conduct this reachability proof. The idea is to write an abstract machine that includes the library specification, and that contains an operation specifying the reachability property, *ie*, an operation that describes the same behavior as $\mathsf{Spec}(\psi\ ,\ \phi)$ using (1). One then implements this machine into an executable program similar to the one of Figure 1, except that it must satisfy the syntactic restrictions of the B notation for implementations and use only concrete expressions and substitutions. Getter operations have to be added to code conditions of IF and WHILE, since these conditions cannot use the variables of the imported Library machine. Similarly, ANY substitutions have to be encapsulated as B operations, since ANY is not allowed in implementations. A machine refinement approach has the advantage of letting Atelier B generate the proof obligations. On the other hand, it generates

significantly more POs: for our case study, it generates 28 POs, with 14 remaining to prove interactively. These turned out to be hard to prove. We added extra variables to help in the proof of the WHILE loop, which brought the number of POs to 88, out of which 30 had to be proved interactively. This is significantly higher than our approach based on assertions and substitution refinement, which generates 14 POs, of which only 4 had to be proved interactively. Stepwise substitution refinement allows the specifier to work directly with abstract variables and manage the size of proof obligations by properly decomposing the proof into several small steps. Moreover, by applying refinement laws, we avoid some POs that the machine refinement approach has to generate. This is somehow similar to the BART tool [9], which proposes automatic refinement laws.

The automation of our approach could be quite simple. The specifier would provide the refinement steps under the form of a list of refinement inequations and definitions, as illustrated in this paper, and provide a reference to a database of refinement laws (*eg*, Morgan's laws in [14]) to justify each refinement step. The tool could syntactically check the call to refinement laws, generate the proof obligations and insert them as assertions in the B abstract specification. These assertions could then be proved using a tool like Atelier B.

References

[1] Abrial, J.-R., L. Mussat, Introducing Dynamic Constraints in B. In *B'98: Recent Advances in the Development and Use of the B Method*, LNCS 1393, pp 83–128, Springer-Verlag (1998).

[2] Back, R.J.R.: On the Correctness of Refinement in Program Development. Ph. D. Thesis Report A-1978-4, Dept. of Computer Science, University of Helsinki, 1978.

[3] Back, R.J.R.: Correctness Preserving Program Refinements: Proof Theory and Applications. Mathematical Center Tracts 131, Mathematical Centre, Amsterdam, 1980.

[4] Barradas, H.R., D. Bert, "Specification and Proof of Liveness Properties under Fairness Assumptions in B Event Systems", in *Integrated Formal Methids 2002*, LNCS 2335, pp 360–379 (2002).

[5] Brown, N., Méry, D.:"A Proof Environment for Concurrent Programs", in FME '93: Industrial-Strength Formal Methods, LNCS 670, Springer-Verlag, pp 196–215 (1993).

[6] Butler, M., J. Grundy , T. Långbacka, R. Rukšėnas, J. von Wright, "The Refinement Calculator: Proof Support for Program Refinement", in *Formal Methods Pacific '97*, pp 40–61, Springer-Verlag (1997).

[7] Carrington, D., I. Hayes, R. Nickson, G. Watson, J. Welsh, "A Tool for Developing Correct Programs By Refinement", Technical report 95-49, The University of Queensland (1996).

[8] Clarke, E.M., E.A. Emerson, "Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic", in *Logic of Programs Workshop*. LNCS 131, pp 52–71, Springer-Verlag (1982).

[9] Clearsy: Bart : B Automatic Refinement Tool, Clearsy System Engineering, Aix-en-provence, France, `http://www.tools.clearsy.com/index.php5?title=BART_Project`.

[10] Hehner, E.C.R.: *A Practical Theory of Programming*. Springer Verlag, 1993.

[11] Kesten, Y., Amir Pnueli, "A Compositional Approach to CTL* Verification", *Theoretical Computer Science*, 331(2-3), pp 397–428, February 2005.

[12] Mammar, A., Frappier, M., Diagne, F.: A Proof-Based Approach to Verifying Reachability Properties, 26^{th} ACM Symposium on Applied Computing,Tunghai University, TaiChung, Taiwan, March 21-24, 2011, *to appear*.

[13] Morgan, C.: The Specification Statement. *ACM Transactions on Programming Languages and Systems* **11**(4) (1988) 517–561.

[14] Morgan, C.C., *Programming from Specifications*, Second edition, Prentice Hall, 1998. `http://web2.comlab.ox.ac.uk/oucl/publications/books/PfS/PfS.ps.gz`.

[15] Morris, J.M.: A Theoretical Basis for Stepwise Refinmement and the Programming Calculus. *Science of Computer Programming* **9** (1987) 287–306.

[16] Morris, J.M.: Laws of Data Refinement. *Acta Informatica* **26** (1989) 287–308.

[17] Nelson, G.: A Generalization of Disjktra's Calculus. *ACM Transactions on Programming Languages and Systems* **11**(4) (1989) 517–560.

[18] Pnueli, A.: "The Temporal Logic of Programs", in *18th IEEE Annual Symposium on Foundations of Computer Science*, pp 46–57, IEEE Computer Society (1977).

[19] Pnueli, A., Y. Kesten, A Deductive Proof System for CTL*, in *13th International Conference on Concurrency Theory 2002*, LNCS 2421, pp 24–40, Springer-Verlag (2002).

[20] J. Misra, *A Discipline of Multiprogramming*, Springer-Verlag, 2001.

Appendix A The B Specification of the Library System

MACHINE *Library*

SETS *MEMBERID*; *BOOKID*

ABSTRACT_CONSTANTS *MaxNbLoans*

PROPERTIES $MaxNbLoans \in \mathcal{N}$

DEFINITIONS
$Index(bb, mm) == ((reservation(bb))^{-1}(mm))$;
$phi == ((bo \mapsto me) \in loan)$;
$psi == (me \in member \wedge bo \in book)$;
$C1 == (C2 \wedge C3 \wedge C4)$;
$C2 == (\mathbf{card}(loan \rhd \{ me \}) < MaxNbLoans)$;
$C3 == (bo \notin \mathbf{dom}(loan))$;
$C4 == (reservation(bo) = [])$

ABSTRACT_VARIABLES
loan , *member* , *book* , *reservation*

INVARIANT
$member \subseteq MEMBERID \wedge book \subseteq BOOKID \wedge loan \in book \nrightarrow member \wedge$
$reservation \in book \rightarrow \mathbf{iseq}(member) \wedge$
$\forall mm . (mm \in member \Rightarrow \mathbf{card} (loan \rhd \{ mm \}) \leq MaxNbLoans)$

ASSERTIONS

/* PO (5) : Lend refines S2 */
$\forall (bo,me) . (psi \wedge C1 \Rightarrow ($
$me \in member \wedge bo \in BOOKID \wedge bo \in book \wedge bo \notin \mathbf{dom}(loan) \wedge$
$reservation(bo) = [] \wedge \mathbf{card}(loan \rhd \{ me \}) < MaxNbLoans \wedge$
$bo \mapsto me \in loan \leftarrow\!\!+ \{ bo \mapsto me \}))$;

/* PO (7) Refinement of S3 */

$\forall (bo,me) .(psi \Rightarrow ($
$\neg (C2) \Rightarrow ($
$\forall bop . (bop \in loan^{-1}[\{me\}] \Rightarrow ($
$bop \in BOOKID \wedge bop \in book \wedge bop \in \mathbf{dom} (loan) \wedge psi \wedge$
$(\mathbf{card}((\{ bop \} \lhd loan) \rhd \{me\}) < MaxNbLoans))))$
$\wedge$
$(C2 \Rightarrow (psi \wedge C2))))$;

/* PO (10) Refinement of S6 */

$\forall (bo,me) . ((psi \wedge C2) \Rightarrow ($
$\neg (C3) \Rightarrow ($
$bo \in BOOKID \wedge bo \in book \wedge bo \in \mathbf{dom} (loan) \wedge psi \wedge$
$(\mathbf{card}((\{ bo \} \lhd loan) \rhd \{me\}) < MaxNbLoans) \wedge$
$(bo \notin \mathbf{dom}(\{ bo \} \lhd loan)))$
$\wedge$
$(C3 \Rightarrow (psi \wedge C2 \wedge C3))))$;

/* PO (11) Refinement of S7 */

$\forall (bo,me) . ($

$(psi \wedge C2 \wedge C3 \wedge \neg (C4)) \Rightarrow$
$\forall\ mep\ .\ (mep \in \mathbf{ran}(reservation(bo)) \Rightarrow ($
$mep \in MEMBERID \wedge mep \in member \wedge bo \in BOOKID \wedge$
$bo \in book \wedge bo \in \mathbf{dom}(reservation) \wedge$
$mep \in \mathbf{ran}(reservation(bo)) \wedge$
$\mathbf{size}((reservation(bo) \uparrow (Index(bo, mep) - 1)) \frown$
$(reservation\ (bo) \downarrow Index(bo, mep)))$
$< \mathbf{size}(reservation(bo)))))$

INITIALISATION
$loan := \emptyset \ ||\ book := \emptyset \ ||\ member := \emptyset \ ||\ reservation := \emptyset$

OPERATIONS

Lend ($member\_$, $book\_$) =
PRE
$member\_ \in MEMBERID \wedge member\_ \in member \wedge$
$book\_ \in BOOKID \wedge book\_ \in book \wedge$
$book\_ \notin \mathbf{dom}\ (\ loan\ ) \wedge reservation(book\_)=[] \quad \wedge$
$\mathbf{card}\ (\ loan \rhd \{\ member\_\ \}\ ) < MaxNbLoans$
THEN
$\mathbf{loan}\ (\ book\_\ ) := member\_$
END ;

Reserve ($member\_$, $book\_$) =
PRE
$member\_ \in MEMBERID \wedge member\_ \in member \wedge$
$book\_ \in BOOKID \wedge book\_ \in book \wedge$
$member\_ \notin \mathbf{ran}(reservation(book\_)) \wedge$
$book\_ \mapsto member\_ \notin loan \wedge$
$(\ book\_ \in \mathbf{dom}\ (\ loan\ ) \vee reservation(book\_) \neq []\ )$
THEN
$reservation := reservation <+$
$\{\ book\_ \mapsto ((reservation(book\_) \leftarrow member\_))\ \}$
END ;
Return ($book\_$) =
PRE
$book\_ \in BOOKID \wedge book\_ \in book \wedge$
$book\_ \in \mathbf{dom}\ (\ loan\ )$
THEN
$loan := \{\ book\_\ \} \lhd loan$
END ;

Take ($member\_$, $book\_$) =
PRE
$member\_ \in MEMBERID \wedge member\_ \in member \wedge$
$book\_ \in BOOKID \wedge book\_ \in book \wedge$
$book\_ \notin \mathbf{dom}\ (\ loan\ ) \wedge$
$\mathbf{card}\ (\ loan \rhd \{\ member\_\ \}\ ) < MaxNbLoans \wedge$
$\mathbf{size}(reservation(book\_)) \neq 0 \wedge$
$\mathbf{first}(reservation(book\_)) = member\_$
THEN
$\mathbf{loan}\ (\ book\_\ ) := member\_ \ ||$
$reservation := reservation <+ \{book\_ \mapsto \mathbf{tail}(reservation(book\_))\}$
END ;

Cancel ($member\_$, $book\_$) =
PRE
$member\_ \in MEMBERID \wedge member\_ \in member \wedge$
$book\_ \in BOOKID \wedge book\_ \in book \wedge$
$member\_ \in \mathbf{ran}(reservation(book\_))$
THEN
$\mathbf{reservation}(book\_) :=$
$(reservation(book\_) \uparrow (Index(book\_, member\_) - 1))$
$\frown$
$(reservation\ (book\_) \downarrow Index(book\_, member\_))$
END

END