Proving Non-Interference on Reachability Properties: a Refinement Approach

Marc Frappier[1] and Amel Mammar[2]

[1] GRIL, Université de Sherbrooke, Québec, Canada
[2] Institut Telecom SudParis, CNRS/SAMOVAR, Paris, France
marc.frappier@usherbrooke.ca, amel.mammar@it-sudparis.eu

Abstract. This paper proposes an approach to prove interference freedom for a reachability property of the form $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \phi)$ in a B specification. Such properties frequently occur in security policies and information systems. Reachability is proved by constructing using stepwise algorithmic refinement an abstract program that refines $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \phi)$. We propose proof obligations to show *non-interference*, *i.e.*, to prove that other operations can be executed in interleaving with this program while preserving the reachability property, to cater for the multi-user aspect of information systems. Proof obligations are discharged using conventional B provers (*eg*, Atelier B). Since refinement preserves these reachability properties and non-interference, proofs can be conducted on abstract machines rather than implementation code.

Keywords: B Notation, reachability, non-interference, proof, refinement calculus

1 Introduction

Reachability properties frequently occur 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 books and then borrow the desired 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.

Such reachability properties are 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 "$[p]\phi$" is the traditional substitution semantic operator of the B theory [1], 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 operators to construct p, then the B theory can be used to prove this statement.

Existing B tools (*eg*, 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 B tools to prove it. However, the proof obligations generated from $\psi \Rightarrow [p]\phi$ can be huge, thus hard to prove. In [4], we have described how to prove these statements using the refinement calculus of Carroll Morgan [7] in a B context, in order to obtain smaller proof obligations. The main results are summarized up in Section 2.

Proving reachability is usually insufficient in the context of information systems (IS), our target application domain. When specifying an IS using B machines, operations denote actions that a user can execute. Thus, p shows the existence of a path, *i.e.*, of a scenario that allows to reach ϕ from ψ. However, the system cannot guarantee that p is executed atomically, because users execute actions independently from one another. Thus, one must also ensure that actions executed by some users cannot interfere with p. Proving non-interference consists in showing that, while other operations are executed in interleaving with p, p still terminates in ϕ. The main contribution of this paper is to propose necessary and sufficient conditions to prove non-interference in a stepwise manner using the refinement tree of p, thereby generating simple proof obligations.

This paper is structured as follows. Section 2 briefly describes how reachability proofs are conducted using refinement, illustrating it on a small library system. Section 3 describes how to prove non-interference. Section 4 discusses the non-interference proofs for the library system. We conclude in Section 5 by a discussion of these results and a comparison with related work.

We assume some familiarity with the B notation. In the B terminology, a *substitution* is an arbitrary combination of programming and specification operators applied to specifications or program statements, in the spirit of the refinement calculus. A B abstract machine contains state variables which may be modified by executing machine operations.

2 Proving Reachability using Refinement

In [7], 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 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. Its wp-semantics is defined as follows:

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

Morgan's refinement is the traditional algorithmic refinement, which consists in weakening the precondition and strengthening the postcondition. One can prove the refinement of a specification statement by a substitution S as follows:

$$\textsc{spec}(pre\ ,\ post) \sqsubseteq S \quad\Leftrightarrow\quad (pre \Rightarrow [S]post) \tag{1}$$

Thus, the problem of proving a CTL reachability formula $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \ \phi)$ can be formulated as finding a program p such that $\textsc{spec}(\psi\ ,\ \phi) \sqsubseteq p$. In [4], we have shown how it is possible to conduct these proofs using refinement laws proposed by Morgan. The basic idea is to decompose the initial specification $\textsc{spec}(\psi\ ,\ \phi)$ into a set of sub-specifications using the usual control structures (sequencing, loop, condition, etc). The aim of this decomposition is to exhibit the path the user has in mind to reach ϕ from ψ and to prove that it really permits to fulfill the goal. The decomposition process is repeated until we obtain terminal specifications that are directly refined with operation calls.

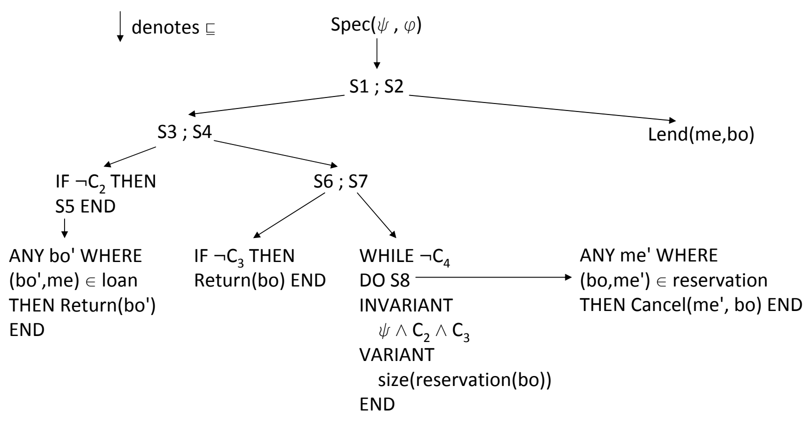
We have applied this process on a B machine which describes the behavior of a library system, to show that a member can always borrow a book (see Appendix A for an excerpt). This property is described by the following CTL formula:

$$\mathsf{AG}\ (me \in member \wedge bo \in book \Rightarrow \mathsf{EF}\ \ bo \mapsto me \in loan) \tag{2}$$

where the B machine variables *member*, *book* and *loan* respectively denote the set of members, books and loans of the library. A path that fulfills this property is described in Figure 1. It is decomposed as follows:

1. A state where the book is available and the member can borrow it (denoted by C_1) should be reached first using S_1; the next step is to reach the goal using S_2, which is refined by operation **Lend**.
2. To reach the goal of S_1, we have to reach a state using S_3 where the member has not met his loan limit (C_2); then we must achieve a state where the book is available (C_3) and no reservation exists on it (C_4) using S_4.
3. To reach the goal of S_3, the member has to return a book if he has reached his loan limit (S_5). Similarly, to reach the goal of S_4, the book should be returned if it is lent (S_6) and all the possible reservations on the book should be cancelled (S_7).

Variable n is a fresh integer variable not used in the B machine, *i.e.*, what Morgan calls in [7] a *logical constant*, which holds the value of the loop variant before the execution of a loop iteration. The proof of the refinement of $\textsc{spec}(\psi\ ,\ \phi)$, S_1 and S_4 are trivial using Morgan's laws [7]. The proofs of refinement for S_2, S_3, S_6, and S_7 are discharged by adding assertions in the B machine. These assertions generate 14 proof obligations (POs) in Atelier B; 10 are automatically proved by Atelier B's prover; 4 are easily proved with the


denotes ⊑
Spec(ψ , φ)
S1 ; S2
S3 ; S4
Lend(me,bo)
IF ¬C2 THEN S5 END
S6 ; S7
ANY bo' WHERE (bo',me) ∈ loan THEN Return(bo') END
IF ¬C3 THEN Return(bo) END
WHILE ¬C4 DO S8 INVARIANT ψ ∧ C2 ∧ C3 VARIANT size(reservation(bo)) END
ANY me' WHERE (bo,me') ∈ reservation THEN Cancel(me', bo) END

$S_1 \triangleq \text{SPEC}(\psi\ ,\ \psi \wedge C_1)$ $\quad S_2 \triangleq \text{SPEC}(\psi \wedge C_1\ ,\ \phi)$

$C_1 \triangleq C_2 \wedge C_3 \wedge C_4$ $\quad C_2 \triangleq card(loan \rhd \{me\}) < MaxNbLoans$

$C_3 \triangleq bo \notin \mathsf{dom}(loan)$ $\quad C_4 \triangleq reservation(bo) = [\,]$

$S_3 \triangleq \text{SPEC}(\psi\ ,\ \psi \wedge C_2)$ $\quad S_4 \triangleq \text{SPEC}(\psi \wedge C_2\ ,\ \psi \wedge C_1)$

$S_6 \triangleq \text{SPEC}(\psi \wedge C_2\ ,\ C_5)$ $\quad S_7 \triangleq \text{SPEC}(C_5\ ,\ \psi \wedge C_1)$

$C_5 \triangleq \psi \wedge C_2 \wedge C_3$ $\quad V \triangleq \mathsf{size}(reservation(bo))$

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

$S_8 \triangleq \text{SPEC}(C_5 \wedge \neg C_4 \wedge n = V\ ,\ C_5 \wedge V \geq 0 \wedge V < n)$

```
p ≜ 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. Refinement tree for proving property (2)

interactive prover. Program p is obtained by piecing together the leaves of the refinement equations. Program p executes operation calls, thus it provides an execution path from ψ to ϕ. Technically, we have shown that it is possible for a member to borrow a book. Thanks to refinement, any implementation of the Library machine will also satisfy this property. However, this program (path) is feasible when executed alone. In a more realistic context, other users are also executing operations in interleaving with this program. Thus, one must wonder if these other operation calls can prevent a user from reaching the desired state by following the path of p. This problem, related to the interference of an operation on a program, is tackled in the next section.

3 Proving Non-Interference

This section defines sufficient and necessary conditions that permit to detect possible interferences on a path. It is important to note that some operations, by user requirements, can interfere with a given reachability property, while others should not. In the library system for example, if the librarian deletes the book, then obviously a member can't borrow it. However, the librarian can acquire or delete other books, other members should be able to borrow other books, or even try to borrow the same target book, in concurrency with the target member *me* trying to borrow *bo*. These are desirable properties of the library specification. Thus, some operations naturally interfere while others should not.

3.1 Defining Non-Interference with Reachability Properties

Let $\mathbf{X}$ denote an operation call that should not interfere with a reachability property. If we want to prove non-interference of $\mathbf{X}$ on p, then we must show that the execution of $\mathbf{X}$ in between two operation calls during the execution of p does not prevent p from terminating in ϕ. For example, suppose that $\textsc{spec}(\psi\ ,\ \phi) \sqsubseteq T_1\,;T_2$, then, intuitively, we must show that:

$$\textsc{spec}(\psi\ ,\ \phi) \quad \sqsubseteq \quad T_1\,; \textsc{if}\ \mathit{pre}(\mathbf{X})\ \textsc{then}\ \mathbf{X}\ \textsc{end}\,; T_2$$

Expression $\mathit{pre}(\mathbf{X})$ denotes the precondition of $\mathbf{X}$. The IF statement surrounding $\mathbf{X}$ ensures that if the precondition of $\mathbf{X}$ is not satisfied, then there is no abortion. Indeed, a user cannot call an operation when its precondition is not satisfied. Thus, the state of the machine is preserved and it may resume its execution with the next operation.

There may be several operations $\mathbf{X}_j$, which we denote by the set $\mathbf{X}_J$, that should not interfere, and they may be called arbitrary many times between T_1 and T_2, so we need a more general proof obligation. Let us first introduce some conventions where $\mathbf{X}_J$ denotes the set of operations that may interfere with a program.

$$\begin{aligned}
\mathbf{X}^\sharp &\triangleq \textsc{if}\ \mathit{pre}(\mathbf{X})\ \textsc{then}\ \mathbf{X}\ \textsc{end}\\
\mathbb{X}_J &\triangleq [\!]_{j\in J}\mathbf{X}_j^\sharp\\
S^* &\triangleq [\!]_{i\geq 0}S^i\\
S^i &\triangleq \text{if } i=0 \text{ then } \textsc{skip} \text{ else } S\,;S^{i-1}
\end{aligned}$$

Operator "$[\!]$" is the choice operator of B, which is sometimes called *demonic choice* in the refinement calculus literature, and is the greatest lower bound wrt "$\sqsubseteq$". Operator "*" is the traditional closure, denoting a finite but arbitrary number of iterations on a statement. Thus, expression $\mathbb{X}_J^*$ denotes an arbitrary execution of operation calls from $\mathbf{X}_J$.

Next, we define an operator $\mathbf{Z}$ which takes as input a program p and returns a program that includes arbitrary operation calls from $\mathbf{X}_J$ at each intermediate execution step in p. One can then show non-interference by proving SPEC(ψ , ϕ) $\sqsubseteq \mathbf{Z}(p)$.

```
$\mathbf{Z}(\text{SKIP}) \triangleq \text{SKIP}$
$\mathbf{Z}(call) \triangleq call$
$\mathbf{Z}(S_1 ; S_2) \triangleq \mathbf{Z}(S_1) ; \mathbb{X}_J^* ; \mathbf{Z}(S_2)$
$\mathbf{Z}(\text{IF } C \text{ THEN } S_1 \text{ ELSE } S_2 \text{ END}) \triangleq$
    $\text{IF } C \text{ THEN } \mathbf{Z}(S_1) \text{ ELSE } \mathbf{Z}(S_2) \text{ END}$
$\mathbf{Z}(\text{WHILE } C \text{ DO } S_1 \text{ INVARIANT } I \text{ VARIANT } V \text{ END}) \triangleq$
    $\text{WHILE } C \text{ DO}$
        $\mathbf{Z}(S_1) ; \text{IF } C \text{ THEN } \mathbb{X}_J^* \text{ END}$
    $\text{INVARIANT } I \text{ VARIANT } V \text{ END}$
$\mathbf{Z}(\text{ANY } u \text{ WHERE } P \text{ THEN } S_1 \text{ END }) \triangleq$
    $\text{ANY } u \text{ WHERE } P \text{ THEN } \mathbf{Z}(S_1)\text{END}$
```

Operator $\mathbf{Z}$ inserts calls to $\mathbb{X}_J^*$ in each sequential composition. For a loop, the call to $\mathbb{X}_J^*$ must be inserted between two successive iterations of the loop body, but it must not be executed after the last iteration of a loop, when C is false, thus the call to $\mathbb{X}_J^*$ is surrounded by a conditional IF C THEN $\mathbb{X}_J^*$ END. This is especially important when the last statement of a program p is a WHILE. When exiting p, the condition ϕ has been reached, and there is no need to prove that $\mathbb{X}_J^*$ does not interfere at that point. Also, operator $\mathbf{Z}$ does not insert calls to $\mathbb{X}_J^*$ neither at the beginning of a path nor at the end. Indeed, an operation $\mathbb{X}_J^*$ interferes on a path p, defined from ψ to ϕ, if $\mathbb{X}_J^*$ prevents the path p, starting from ψ, from reaching ϕ. So before considering any possible interference, we have to executes the first operation call of p in order to ensure that the path actually starts from ψ. Similarly, we don't need to insert calls to $\mathbb{X}_J^*$ after executing the last operation since we are supposed to have already reached the goal ϕ.

Definition 1. Let S = SPEC(ψ , ϕ) (*i.e.*, a reachability property) and p a program such that $S \sqsubseteq p$. We say that $\mathbf{X}_J$ does not interfere with p under the context of specification S iff $S \sqsubseteq \mathbf{Z}(p)$. □

Definition 1 essentially states that p permits to reach ϕ from ψ even if it is executed in interleaving with arbitrary calls from $\mathbf{X}_J$. We use this definition is the next subsections to define the sufficient and necessary conditions such that $\mathbf{X}_J$ does not interfere with p under the context of specification S.

3.2 Necessary and Sufficient Conditions for Non-Interference

Figure 2 illustrates the relationships involved in a non-interference proof for inequations SPEC(ψ , ϕ) $\sqsubseteq S \triangleq (S_1 ; S_2)$, $S_1 \sqsubseteq p_1$, and $S_2 \sqsubseteq p_2$. By monotonicity

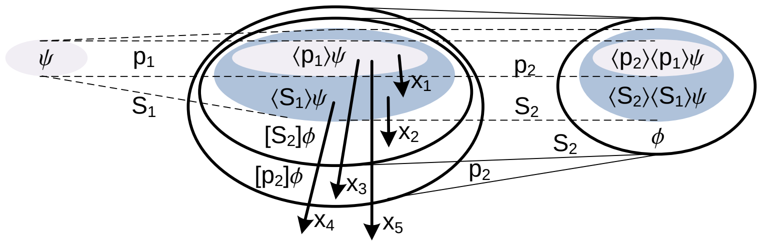


Fig. 2. Interference in sequential composition

of B operators wrt $\sqsubseteq$, these inequations entail $\text{SPEC}(\psi\ ,\ \phi) \sqsubseteq p \triangleq (p_1\,;p_2)$. Let $\langle S\rangle\psi$ denote the states reached after executing S from ψ, which is commonly called the *strongest postcondition*. These refinement inequations induce the following subsumptions among predicates for strongest postconditions and weakest preconditions denoting intermediate states reached after p_1 and before executing p_2:

$$\langle p_1\rangle\psi \;\Rightarrow\; \langle S_1\rangle\psi \;\Rightarrow\; [S_2]\phi \;\Rightarrow\; [p_2]\phi \qquad (3)$$

Operations $\mathbf{X}_1$ and $\mathbf{X}_3$ do not interfere with p, since it is still possible to reach ϕ after their executions, whereas operation $\mathbf{X}_5$ interferes with p since the state reached is beyond the set from which p_2 establishes ϕ. Operations $\mathbf{X}_2$ and $\mathbf{X}_4$ do not interfere either with p, because they are defined for states not reachable from ψ after executing p_1. Operations $\mathbf{X}_1$ and $\mathbf{X}_2$ do not interfere with S, while $\mathbf{X}_3$, $\mathbf{X}_4$, and $\mathbf{X}_5$ do interfere with S, since it is impossible to reach ϕ from ψ by executing $(S_1\,;\mathbf{X}_i\,;S_2)$, for $i \in 3..5$. This illustrates that the more refined a program is, the less it is subject to interference. Moreover, one can see that if $\mathbf{X}$ does not interfere with S, then it does not interfere with S' when $S \sqsubseteq S'$. This leads us to define the following conditions.

Necessary and Sufficient Non-Interference Condition NSNIC-1:
Given a refinement $\text{SPEC}(\psi\ ,\ \phi) \sqsubseteq p_1\,;p_2$, for each $j \in J$ prove

$$\langle p_1\rangle\psi \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j][p_2]\phi \qquad \square$$

Intuitively, condition NSNIC-1 states that operation $\mathbf{X}_j$ must establish $[p_2]\phi$ from $\langle p_1\rangle\psi$. Formula $[p_2]\phi$ denotes the states from which p_2 is guaranteed to establish ϕ; it represents the *freedom* that $\mathbf{X}_j$ has for not interfering with $(p_1\,;p_2)$ in the refinement of $\text{SPEC}(\psi\ ,\ \phi)$. For example, the proof obligation for refinement step $\text{SPEC}(\psi\ ,\ \phi) \sqsubseteq (S_1\,;S_2)$ of Figure 1 is the following:

$$\begin{array}{ll} & \langle p_1\rangle\psi \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j][p_2]\phi \\ \Leftrightarrow & \qquad\qquad \langle\ \langle p_1\rangle\psi \equiv [p_2]\phi \equiv (\psi \wedge C_1)\ \rangle \\ & \psi \wedge C_1 \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j](\psi \wedge C_1) \end{array}$$

If we consider $\mathbf{X}_j \triangleq \text{PRE } me \neq me' \wedge bo \neq bo' \text{ THEN } \mathbf{Lend}(me', bo') \text{ END}$, then condition NSNIC-1 means that another member me' borrowing another book

bo' should preserves $\psi \wedge C_1$. In this particular case it holds since $\mathbf{Lend}(me', bo')$ does not modify the reservations and loans of bo and me and the existence of bo and me in the library.

The next two conditions cater for WHILE statements. Given a refinement

$$\textsc{spec}(I\ ,\ I \wedge \neg C) \sqsubseteq \textsc{while}\ C\ \textsc{do}\ p\ \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}$$

for each $j \in J$ prove:
Non-Interference Condition NSNIC-2:

$$\langle p \rangle (C \wedge I) \wedge C \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j] I \qquad \Box$$

Non-Interference Condition NSNIC-3:

$$\forall n \cdot \langle p \rangle (C \wedge I) \wedge C \wedge V = n \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j] V \leq n \qquad \Box$$

Condition NSNIC-2 (resp. NSNIC-3) means that operation $\mathbf{X}_j$ must establish invariant I (resp. must not increase variant V) from states reached after p (*i.e.*, $\langle p \rangle (C \wedge I)$ holds) where it is possible to loop again (*i.e.*, C holds). We can now introduce the main theorem of the paper.

Theorem 1. Let $S = \textsc{spec}(\psi\ ,\ \phi)$ and p a program obtained from a refinement tree of S, *i.e.*, $S \sqsubseteq p$. We say that $\mathbf{X}_J$ does not interfere with p under the context of specification S iff conditions NSNIC-1, NSNIC-2 and NSNIC-3 hold.

Proof. The proof is based on the refinement of the reachability property, as illustrated for instance in Figure 1. Each inequation is of the form $\textsc{spec}(\psi\ ,\ \phi) \sqsubseteq \Theta(\ldots, S_i, \ldots)$, where Θ is a B operator. Each S_i of the form $\textsc{spec}(\ldots\ ,\ \ldots)$ must be represented by a child of this node with a refinement inequation $S_i \sqsubseteq \ldots$. We also assume that operator ";" can only occur as a refinement of a $\textsc{spec}(\ ,\ )$ statement. A WHILE of the form WHILE C DO S INVARIANT I VARIANT V END can only occur as a refinement of a specification of the form $\textsc{spec}(I\ ,\ I \wedge \neg C)$. This way, we ensure that NSNIC-1, NSNIC-2 and NSNIC-3 can be used. The proof is conducted by induction on the structure of the refinement tree of S. Program p is obtained from these refinement inequations, *i.e.*, it is made of the leaves of the tree composed with internal node operators, as shown in Figure 1. Each case relies on the monotonicity of B operators wrt "$\sqsubseteq$", as proved in [1]. For the sake of concision, we only illustrate some key elements of the proof. A detailed calculational proof is provided in the long version of this paper [4]. The proof that NSNIC-1, 2 and 3 are sufficient goes as follows for the case $p = p_1 \,; p_2$:

$$\begin{array}{ll}
 & S \\
\sqsubseteq & \qquad \langle \text{ Lemma 1 with NSNIC-1 } \rangle \\
 & p_1 \,; \mathbb{X}_J^* \,; p_2 \\
\sqsubseteq & \qquad \langle \text{ ind. hyp. } p_1 \sqsubseteq \mathbf{Z}(p_1),\ p_2 \sqsubseteq \mathbf{Z}(p_2), \text{ mon. of ";" } \rangle \\
 & \mathbf{Z}(p_1) \,; \mathbb{X}_J^* \,; \mathbf{Z}(p_2) \\
= & \qquad \langle \text{ def. of } \mathbf{Z} \ \rangle \\
 & \mathbf{Z}(p_1 \,; p_2)
\end{array}$$

Lemma 1. If $\textsc{spec}(\psi\ ,\ \phi) \sqsubseteq p_1\,;p_2$ and NSNIC-1 hold, then

$$\textsc{spec}(\psi\ ,\ \phi) \sqsubseteq p_1\,;\mathbb{X}_J^*\,;p_2.$$

Intuitively, one can see that Lemma 1 holds by the diagram of Figure 2. A computation starting from ψ reaches $\langle\psi\rangle p_1$ after p_1. An operation $\mathbf{X}$ that satisfies NSNIC-1 can only make a transition within $[p_2]\phi$. From these states, it is guaranteed that p_2 establishes ϕ. A similar reasoning goes for a WHILE statement.

The proof that NSNIC-1, 2 and 3 are necessary requires some additional notation. Let $\widehat{S}$ denote the before-after predicate of substitution S, with free variables x and x' respectively denoting the value before and after. By convention, x denotes the free variables of a state predicate. For the sake of concision we note $A(y) \equiv A[x := y]$ and $\widehat{S}(y, y') \equiv \widehat{S}[x, x' := y, y']$. The next laws relate the before-after predicate of a substitution to weakest precondition and strongest postcondition.

$$[S]\phi \;\Leftrightarrow\; \forall\, x' \cdot \widehat{S}(x, x') \Rightarrow \phi(x') \tag{4}$$

$$\langle S\rangle\phi \;\Leftrightarrow\; \exists\, x_0 \cdot \widehat{S}(x_0, x) \wedge \phi(x_0) \tag{5}$$

The proof goal that NSNIC-1 is a necessary condition is rewritten as follows:

$$\begin{array}{ll}
 & \textsc{spec}(\psi\ ,\ \phi) \sqsubseteq \mathbf{Z}(p_1\,;p_2) \;\Rightarrow\; \text{NSNIC-1} \\
\Leftrightarrow & \qquad\qquad \langle\ \text{contraposite}\ \rangle \\
 & \neg\text{NSNIC-1} \;\Rightarrow\; \textsc{spec}(\psi\ ,\ \phi) \not\sqsubseteq \mathbf{Z}(p_1\,;p_2) \\
\Leftrightarrow & \qquad\qquad \langle\ \text{negation of def. of refinement (1)}\ \rangle \\
 & \neg\text{NSNIC-1} \;\Rightarrow\; \exists\, x \cdot \psi \wedge \neg[\mathbf{Z}(p_1\,;p_2)]\phi
\end{array}$$

Since by definition $\mathbf{Z}(p_1\,;p_2) = \mathbf{Z}(p_1)\,;\mathbb{X}_J^*\,;\mathbf{Z}(p_2)$, we can simplify that goal using the following deduction:

$$\begin{array}{ll}
 & \mathbb{X}_J^* \sqsubseteq \mathbf{X}^\sharp \\
\Rightarrow & \\
 & \mathbf{Z}(p_1\,;p_2) \sqsubseteq \mathbf{Z}(p_1)\,;\mathbf{X}^\sharp\,;\mathbf{Z}(p_2) \\
\Rightarrow & \qquad\qquad \langle\ \mathbf{Z}(S) \sqsubseteq S \text{ since } \mathbb{X}_J^* \sqsubseteq \textsc{skip}\ \rangle \\
 & \mathbf{Z}(p_1\,;p_2) \sqsubseteq p_1\,;\mathbf{X}^\sharp\,;p_2 \\
\Rightarrow & \qquad\qquad \langle\ \text{refinement properties}\ \rangle \\
 & [\mathbf{Z}(p_1\,;p_2)]\phi \Rightarrow [p_1\,;\mathbf{X}^\sharp\,;p_2]\phi \\
\Leftrightarrow & \qquad\qquad \langle\ \text{contraposite}\ \rangle \\
 & \neg[p_1\,;\mathbf{X}^\sharp\,;p_2]\phi \Rightarrow \neg[\mathbf{Z}(p_1\,;p_2)]\phi
\end{array}$$

Thus, given an operation $\mathbf{X}$ verifying $\neg$NSNIC-1:

$$\exists\, x \cdot (\langle p_1\rangle\psi \wedge Pre(\mathbf{X}) \wedge \neg[\mathbf{X}][p_2]\phi) \tag{H$_1$}$$

it suffices to prove that:

$$\exists\, x.(\psi \wedge \neg[p_1\,;\mathbf{X}^\sharp\,;p_2]\phi) \tag{G$_1$}$$

Let x_1 be a value of x that verifies H_1:

$$(\langle p_1\rangle\psi)(x_1) \tag{H$_2$}$$
$$Pre(X)(x_1) \tag{H$_3$}$$
$$(\neg[\mathbf{X}][p_2]\phi)(x_1) \tag{H$_4$}$$

H_2 gives:

$$\exists\, x_0 \cdot (\widehat{p_1}(x_0, x_1) \wedge \psi(x_0)) \tag{H$_5$}$$

Let ${x_0}^1$ be a value of x_0 that verifies H_5

$$\psi({x_0}^1) \tag{H$_6$}$$
$$\widehat{p_1}({x_0}^1, x_1) \tag{H$_7$}$$

Let us prove that ${x_0}^1$ establishes G_1:

$$\psi({x_0}^1) \tag{${G_1}^1$}$$
$$(\neg[p_1\,;\mathbf{X}^\sharp\,;p_2]\phi)({x_0}^1) \tag{${G_1}^2$}$$

Subgoal ${G_1}^1$ is discharged by H_6. Subgoal ${G_1}^2$ is discharged as follows:

$$\begin{array}{ll}
 & (\neg[p_1\,;\mathbf{X}^\sharp\,;p_2]\phi)({x_0}^1) \\
\Leftrightarrow & \qquad \langle \text{ apply } [] \rangle \\
 & (\neg([p_1]([\mathbf{X}^\sharp\,;p_2]\phi)))({x_0}^1) \\
\Leftrightarrow & \qquad \langle \text{ apply } (4) \rangle \\
 & (\neg(\forall\, x'.(\widehat{p_1}(x,x') \Rightarrow ([\mathbf{X}^\sharp\,;p_2]\phi)(x'))))({x_0}^1) \\
\Leftrightarrow & \qquad \langle \text{ apply Negation operator } \rangle \\
 & (\exists\, x'.(\widehat{p_1}(x,x') \wedge \neg(([\mathbf{X}^\sharp\,;p_2]\phi)(x'))))({x_0}^1) \\
\Leftarrow & \qquad \langle\ \exists \text{ introduction } \rangle \\
 & (\widehat{p_1}(x,x_1) \wedge \neg(([\mathbf{X}^\sharp\,;p_2]\phi)(x_1)))({x_0}^1) \\
\Leftrightarrow & \qquad \langle\ H_3\ \rangle \\
 & (\widehat{p_1}(x,x_1) \wedge \neg(([\mathbf{X}\,;p_2]\phi)(x_1)))({x_0}^1) \\
\Leftrightarrow & \qquad \langle\ H_4\ \rangle \\
 & (\widehat{p_1}(x,x_1))({x_0}^1) \\
\Leftrightarrow & \\
 & H_7
\end{array}$$

The proof of necessary conditions for WHILE is very similar and provided in [4].

3.3 Sufficient Conditions for Non-Interference

Sufficient and necessary conditions are better since they permit to produce a precise verdict about the interference or not of an operation **X** on a program *p*. Nevertheless, they require some calculations that may be tedious even if they can be automated. The next laws determine how to compute the strongest post-condition from the weakest precondition and the before-after predicate.

$$\langle S\rangle\phi \;\Leftrightarrow\; \exists\, x_0 \cdot \widehat{S}(x_0, x) \wedge \phi(x_0) \tag{16}$$
$$\widehat{S} \;\Leftrightarrow\; \neg[S](x' = x) \wedge [S](x = x) \tag{17}$$

For a program p, $\langle p \rangle \phi$ may be cumbersome to compute and use in proof. For specification statements, it is easier.

$$\langle \text{SPEC}(\psi\ ,\ \phi)\rangle\psi \ \Leftrightarrow\ \phi \tag{18}$$

$$[\text{SPEC}(\psi\ ,\ \phi)]\phi \ \Leftrightarrow\ \psi \tag{19}$$

Moreover, non-interference is inherited by all refinements of a program, as expressed by the following proposition, which stems easily from the monotonicity of B operators.

Proposition 1. If $\mathbf{X}$ does not interfere with p for $\text{SPEC}(\psi\ ,\ \phi)$, then p' such that $p \sqsubseteq p'$ also does not interfere with $\text{SPEC}(\psi\ ,\ \phi)$. □

Thus, a non-interference proof obligation can be easier to discharge with the abstract specification derived during refinement (*eg*, $S = S_1\,;S_2$), instead of the final refined program (*eg*, $p = p_1\,;p_2$) when $S_1 \sqsubseteq p_1$ and $S_2 \sqsubseteq p_2$.

This leads us to propose sufficient conditions for proving non-interference based on the abstract specifications. Their proof follows easily from the subsumptions of (3) and theorem 1.

Sufficient Non-Interference Condition NIC-1:
Given a refinement $\text{SPEC}(\psi\ ,\ \phi) \sqsubseteq S_1\,;S_2$, for each $j \in J$ prove

$$[S_2]\phi \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j][S_2]\phi$$ □

Intuitively, condition NIC-1 states that operation $\mathbf{X}_j$ must preserve $[S_2]\phi$. If we consider again the non-interference proof for $\text{SPEC}(\psi\ ,\ \phi) \sqsubseteq (S_1\,;S_2)$ of Figure 1 with NIC-1, we obtain the same proof obligation as with NSNIC-1.

$$\begin{array}{ll} & [S_2]\phi \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j][S_2]\phi \\ \Leftrightarrow & \qquad \langle\ [S_2]\phi = \psi \wedge C_1, \text{ using } (1)\ \rangle \\ & \psi \wedge C_1 \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j](\psi \wedge C_1) \end{array}$$

The next two sufficient conditions cater for WHILE statements. Given a refinement

$$\text{SPEC}(I\ ,\ I \wedge \neg C) \sqsubseteq \text{WHILE } C \text{ DO } S \text{ INVARIANT } I \text{ VARIANT } V \text{ END}$$

Sufficient Non-Interference Condition NIC-2: for each $j \in J$ prove

$$C \wedge I \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j]I$$ □

Sufficient Non-Interference Condition NIC-3: for each $j \in J$ prove

$$\forall n \cdot C \wedge V = n \wedge pre(\mathbf{X}_j) \Rightarrow [\mathbf{X}_j]V \leq n$$ □

Condition NIC-2 and NIC-3 ensure that an operation $\mathbf{X}_j$ preserves the loop invariant and does not increase the loop variant. Thus, the loop can still terminate on $I \wedge \neg C$.

The next example illustrates the differences between NSNIC and NIC conditions.

$$\begin{array}{l}
\textsc{spec}(x = 0 \wedge y = 0\ ,\ x = 13 \wedge y = 5)\ \sqsubseteq\ S_1\,;S_2 \\
S_1\ \triangleq \textsc{spec}(x = 0 \wedge y = 0\ ,\ x = 6 \wedge y = 5)\ \sqsubseteq\ op_1 \\
op_1 \triangleq \textsc{begin}\ x := x + 6 \parallel y := 5\ \textsc{end} \\
S_2\ \triangleq \textsc{spec}(x = 6 \wedge y = 5\ ,\ x = 13 \wedge y = 5)\ \sqsubseteq\ op_2 \\
op_2 \triangleq \textsc{begin}\ x := 2 \times x + 1 \parallel y := 5\ \textsc{end} \\
\mathbf{X}\ \triangleq \textsc{begin}\ y := y + 1\ \textsc{end} \\
\langle S_1\rangle x = 0 \wedge y = 0 \quad\ \Leftrightarrow x = 6 \wedge y = 5 \\
[S_2](x = 13 \wedge y = 5) \Leftrightarrow x = 6 \wedge y = 5 \\
[p_2](x = 13 \wedge y = 5) \Leftrightarrow x = 6 \\
[X](x = 6 \wedge y = 5) \quad \Leftrightarrow x = 6 \wedge y = 4 \\
[X]x = 6 \qquad\qquad\quad \Leftrightarrow x = 6
\end{array}$$

Operation $\mathbf{X}$ doesn't interfere with path $(op_1\,;op_2)$ since $\mathbf{X}$ doesn't modify variable x and whatever the value of y after execution of $\mathbf{X}$, operation op_2 permits to reach $(x = 13 \wedge y = 5)$. However, condition NIC-1 is not satisfied by $\mathbf{X}$. Indeed after executing $\mathbf{X}$ on state $(x = 6 \wedge y = 5)$, the value of y will not be equal to 5. Therefore, the approach we advise consists in verifying the sufficient conditions NIC-1,2,3 with $[S_2]\phi$ at first if $\langle p_1\rangle\psi$ and $[p_2]\phi$ return complex expressions. If they fail, then conditions NSNIC-1,2,3 can be checked to determine if there is interference or not.

4 Application to the Case Study

In this section, we report and discuss the results obtained on non-interference analysis of the program $(Path_1)$ depicted in Figure 1. Applying the non-interference rules provided in the previous section gives the following POs. In this particular case, NIC-i and NSNIC-i give the same POs:

1. $\forall \vec{v}\ .([S_2]\phi \wedge pre(\mathbf{X}) \Rightarrow [\mathbf{X}][S_2]\phi)$
2. $\forall \vec{v}\ .([S_4](\psi \wedge C_1) \wedge pre(\mathbf{X}) \Rightarrow [\mathbf{X}][S_4](\psi \wedge C_1))$
3. $\forall \vec{v}\ .([S_7](\psi \wedge C_1) \wedge pre(\mathbf{X}) \Rightarrow [\mathbf{X}][S_7](\psi \wedge C_1))$
4. $\forall \vec{v}\ .(pre(\mathbf{X}) \wedge C_5 \wedge \neg C_4 \Rightarrow [\mathbf{X}]C_5)$
5. $\forall \vec{v}\ .(pre(\mathbf{X}) \wedge n = V \wedge \neg C_4 \Rightarrow [\mathbf{X}]V \leq n)$

where $\vec{v}$ includes the free variables $\vec{v_1}$ of formula $(\mathsf{AG}\ \psi \Rightarrow \mathsf{EF}\ \phi)$, the formal input parameters $\vec{v_2}$ of operation $\mathbf{X}$ and n in the last PO. To discharge NIC-1 (PO 1, 2 and 3) under Atelier B, a new machine M' that includes the Library machine is created. Such a machine defines variables $\vec{v_1}$ as concrete variables and an invariant to describe the property we want to maintain (*eg*, $[S_2]\phi$). For each operation $\mathbf{X}$, we define a new operation $\mathbf{X}'$ that calls $\mathbf{X}$ with exactly the same precondition as $\mathbf{X}$; other constraints on input parameters can be added in the precondition of $\mathbf{X}'$ to precise the context where there should not be interference. Proof obligations related to loop invariants (NIC-2, PO 4) are dealt similarly by using a new machine M'' which includes the loop condition in the precondition of $\mathbf{X}'$. Finally, proof obligations related to loop variants (NIC-3, PO 5) are added as assertions into the included machine since they make reference to before-after variable values. For the above formulas applied to operations **Lend**, **Reserve**, **Take** and **Cancel**, the prover of Atelier B generates 13 POs: 7 POs have been discharged (automatically or interactively), the six remaining ones are not valid and correspond to interference issues. According to our analysis of these six POs, we can conclude that some operations of the system interfere in specific points of the path, which illustrates the adequacy of our proof obligations.

1. Operation **Lend** interferes with the defined path in the following situations:
   – if it is executed on book *bo* by another member $me' \neq me$, when *bo* is available and its reservation queue is empty, just at the end of the loop.
   – if it is executed by member *me* to borrow another book $bo' \neq bo$, when *me* has $(MaxNbLoans - 1)$ loans.
2. Operation **Take** interferes with the defined path in the following situations:
   – if it is executed on book *bo* as soon as *bo* becomes available and its reservation queue is not empty.
   – it is executed on member *me*, when *me* has $(MaxNbLoans - 1)$ loans.
3. Operation **Reserve** interferes with the defined path if it is executed on book *bo*: either it increases the loop variant during the loop, thus potentially preventing the loop from terminating, or it makes the reservation queue not empty after the end of the loop.

The analysis of the different interference causes permits to propose a new path ($Path_2$) that is free from any interference:(1) make a reservation on *bo* for member *me*, (2) cancel all the reservations made by the other members (or less drastically, let them use their reservations and return the books) such that *me* becomes at the head of the queue. In that case, the variant of the related loop would be the position of *me* in the queue, (3) If the book is not available then return the book, (4) return a book *bo'* if $MaxNbLoans$ is reached by *me*, (5) finally, lend the book *bo* to *me*. This path generates 19 POs for NIC that have been successfully discharged. Table 1 gives the results we have obtained during the proof phase. NIC for $Path_2$ generates slightly more proof obligations than for $Path_1$ since its loop variant is a little bit more complicated than that of $Path_1$. It is easier to prove that the length of the queue is not increasing than to prove that the position of an element in a queue doesn't increase.

	Automatic POs	Interactive POs	Erroneous POs	Total POs
Reachability Proof $Path_1$	8	6	0	14
NIC $Path_1$	2	5	6	13
NIC $Path_2$	2	17	0	19

Table 1. Proof results

5 Discussion and Conclusion

We have shown how to prove a CTL reachability property using substitution refinement and how to prove interference freedom for such a property. We construct by stepwise refinement, reusing Morgan's specification statement and laws, a program whose elementary statements are operations calls. Interference freedom is proved using three necessary and sufficient conditions on the refinement tree of the program. Thanks to the use of refinement in our proofs, reachability and non-interference properties are preserved by any implementation of the B specification.

Program p proving $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \phi)$ is not hard to invent. It is a path the specifier typically has in mind to reach ϕ from ψ. It comes from the use cases identified during requirements analysis. Intermediate predicates are not hard either to invent: they come from the precondition of the last operation of p; they are also easy to extract from use cases. The number of assertions to prove refinement is typically equal to the number of leaves in the refinement tree, because internal nodes are proved with Morgan's laws in most cases. The number of assertions to prove non-interference is $(m + n * 2) * k$, where m and n are the number of ";" and "WHILE", and k the number of non-interfering operations. This is certainly manageable, and of the same order of magnitude as proving a safety property like an invariant, something which is routinely done in numerous industrial B projects.

To the extent of our knowledge, this is the first paper that provides a practical and simple method, based on well established techniques, supported by industrial-strength case tools, to prove reachability and non-interference, two crucial properties for information systems.

Abrial and Mussat [2] introduced the *leadsto* modality of UNITY for an ancestor of Event B. UNITY's leadsto, denoted by $\psi \leadsto \phi$, is defined in LTL as $\Box(\psi \Rightarrow \diamond\phi)$. Being an LTL property, it is too strong for our reachability properties (not all path needs to reach ψ from ϕ). CTL reachability allows us to use refinement to decompose the proof into simple steps. The work of [2] is extended in [3] 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 execute an action. Fairness is typically handled by the implementation container (*eg*, a servlet engine) which ensures that all user requests are served. However, we must take into account progress in the context of while loops. Additional proof rules for *ensures* and *leadsto* are proposed in [10].

The work of Pnueli *et al* [6] is probably the closest to ours for reachability proofs. 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. Rule E-UNTIL of [6] is the main one to prove $\mathsf{AG}\ (\psi \Rightarrow \mathsf{EF}\ \phi)$. It requires to find an invariant and a variant, even for a simple sequential decomposition It does not support hierarchical decomposition of the proof into simple POs. Moreover, rule E-UNTIL uses an existential quantification, which is essentially an *angelic* sequential composition, which is not monotonic with respect to refinement. Thus, properties proved using E-UNTIL are not preserved by refinement; in our approach, reachability properties are indeed preserved by refinement.

None of the work above addresses non-interference. Non-interference has been studied mainly in the context of two programs p and q executing in interleaving (*eg*, [8,5] and several others). The idea is to show that p and q satisfy their respective specification even when executed in interleave. In our context, q is $\mathbb{X}_J^*$, an arbitrary choice over machine operations and it has no specification to satisfy. Thus we only need to ensure that p satisfies its own specification. This leads us to identify necessary and sufficient conditions using the refinement tree of p.

Model checking is typically used to check CTL reachability properties. On top of the state explosion problem and bounded state space limitations, CTL model checking does not guarantee that an implementation of the abstract machine still satisfies the reachability property or that it is free from interference. A model checker simply finds a path, which may disappear during refinement, since the computation of the transition system by a model checker does not take into account demonic sequential composition as done in the refinement calculus; a model checker uses *angelic* sequential composition, which is not monotonic wrt to refinement.

References

1. Abrial., J.-R., *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.
2. 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).
3. 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).
4. Frappier, M., Mammar, A.: Proving Reachability and Non-Interference in B, Tech. Rep. 34, Dept. of Comp. Sc., U of Sherbrooke (QC), Canada, 2010. http://www.dmi.usherb.ca/˜frappier/Papers/TR-DI-34.pdf
5. Jones, C.B., Wanted: a compositional approach to concurrency, in *Programming Methodology*, (eds.) A. McIver and C. Morgan, Springer-Verlag, 2000, pp. 1–15.
6. Kesten, Y., Amir Pnueli, A Compositional Approach to CTL* Verification, *Theoretical Computer Science*, 331(2-3), pp 397–428, February 2005.
7. Morgan, C.C., *Programming from Specifications*, Prentice Hall, 1998.
8. Owicki, S.S., Gries,D., An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976.
9. Pnueli, A., Y. Sa'ar, All You Need Is Compassion, in *VMCAI 2008*, LNCS 4905, pp 233–247, Springer-Verlag (2008)
10. J. Misra, *A Discipline of Multiprogramming*, Springer-Verlag, 2001.

A The B Specification of the Library System

```
MACHINE $Library$
SETS $MEMBERID$; $BOOKID$
ABSTRACT_CONSTANTS $MaxNbLoans$
PROPERTIES $MaxNbLoans \in \mathcal{N}$
ABSTRACT_VARIABLES $loan$ , $member$ , $book$ , $reservation$
INVARIANT
  $member \subseteq MEMBERID \wedge book \subseteq BOOKID \wedge loan \in book \nrightarrow member \wedge$
  $reservation \in book \rightarrow \mathbf{iseq}(member) \wedge$
  $\forall\ mm\ .\ (\ mm \in member \Rightarrow \mathbf{card}\ (\ loan \rhd \{\ mm\ \}\ ) \leq MaxNbLoans\ )$
INITIALISATION
  $loan := \emptyset \mid\mid book := \emptyset \mid\mid member := \emptyset \mid\mid reservation := \emptyset$
OPERATIONS
Lend ( $member\_$ , $book\_$ ) = PRE
    $member\_ \in MEMBERID \wedge member\_ \in member \wedge book\_ \in BOOKID \wedge$
    $book\_ \in book \wedge book\_ \notin \mathbf{dom}\ (\ loan\ ) \wedge reservation(book\_)=[] \wedge$
    $\mathbf{card}\ (\ loan \rhd \{\ member\_\ \}\ ) < MaxNbLoans$
  THEN    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
    $reservation := reservation \mathbin{<\!\!+}$
                        $\{\ book\_ \mapsto ((reservation(book\_) \leftarrow member\_))\ \}$
  END ;
Return ( $book\_$ ) = PRE
    $book\_ \in BOOKID \wedge book\_ \in book \wedge$     $book\_ \in \mathbf{dom}\ (\ loan\ )$
  THEN $loan := \{\ book\_\ \} \mathbin{\lhd\!\!-} loan$ 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
    loan ( $book\_$ ) := $member\_ \mid\mid$
    $reservation := reservation \mathbin{<\!\!+} \{book\_ \mapsto \mathbf{tail}(reservation(book\_))\}$
  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
    reservation($book\_$) :=
      $(reservation(book\_) \uparrow (Index(book\_, member\_)$ - $1)) \frown$
      $(reservation\ (book\_) \downarrow Index(book\_, member\_))$
  END
END
```

B Detailed Proof of Theorem 1

B.1 Sufficient Condition

Proof.
Case $S \sqsubseteq call$: $S \sqsubseteq call = Z(call)$.
Case $S \sqsubseteq \textsc{skip}$: $S \sqsubseteq \textsc{skip} = Z(\textsc{skip})$.
Case $S \sqsubseteq p_1\,;p_2$:

$$\begin{array}{ll}
 & S \\
\sqsubseteq & \qquad \langle \text{ Lemma 1 with NSNIC-1 } \rangle \\
 & p_1\,;\mathbb{X}_J^*\,;p_2 \\
\sqsubseteq & \qquad \langle \text{ ind. hyp. } p_1 \sqsubseteq \mathbf{Z}(p_1),\ p_2 \sqsubseteq \mathbf{Z}(p_2), \text{ mon. of ``;'' } \rangle \\
 & \mathbf{Z}(p_1)\,;\mathbb{X}_J^*\,;\mathbf{Z}(p_2) \\
= & \qquad \langle \text{ def. of } \mathbf{Z} \ \rangle \\
 & \mathbf{Z}(p_1\,;p_2)
\end{array}$$

Case $S \sqsubseteq \textsc{if}\ C\ \textsc{then}\ p_1\ \textsc{else}\ p_2\ \textsc{end}$:

$$\begin{array}{ll}
 & S \\
\sqsubseteq & \\
 & \textsc{if}\ C\ \textsc{then}\ p_1\ \textsc{else}\ p_2\ \textsc{end} \\
\sqsubseteq & \qquad \langle \text{ ind. hyp. } p_1 \sqsubseteq \mathbf{Z}(p_1),\ p_2 \sqsubseteq \mathbf{Z}(p_2), \text{ mon. of } \textsc{if} \ \rangle \\
 & \textsc{if}\ C\ \textsc{then}\ \mathbf{Z}(p_1)\ \textsc{else}\ \mathbf{Z}(p_2)\ \textsc{end} \\
= & \qquad \langle \text{ def. of } \mathbf{Z} \ \rangle \\
 & \mathbf{Z}(\textsc{if}\ C\ \textsc{then}\ p_1\ \textsc{else}\ p_2\ \textsc{end})
\end{array}$$

Case $S \sqsubseteq \textsc{while}\ C\ \textsc{do}\ p_1\ \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}$:

$$\begin{array}{ll}
 & S \\
\sqsubseteq & \qquad \langle \text{ Lemma 6 with NSNIC-2 and NSNIC-3 } \rangle \\
 & \textsc{while}\ C\ \textsc{do}\ p_1\,;\textsc{if}\ C\ \textsc{then}\ \mathbb{X}_J^*\ \textsc{end} \\
 & \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end} \\
\sqsubseteq & \qquad \langle \text{ ind. hyp. } p_1 \sqsubseteq Z(p(p_1)), \text{ mon. of } \textsc{while} \ \rangle \\
 & \textsc{while}\ C\ \textsc{do}\ \mathbf{Z}(p_1)\,;\textsc{if}\ C\ \textsc{then}\ \mathbb{X}_J^*\ \textsc{end} \\
 & \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end} \\
= & \qquad \langle \text{ def. of } \mathbf{Z} \ \rangle \\
 & \mathbf{Z}(\textsc{while}\ C\ \textsc{do}\ p_1\ \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end})
\end{array}$$

The proof for ANY is similar to the proof of IF. □

Proof of Lemmas used in the Sequence Case

Lemma 1. If $\textsc{spec}(\psi\ ,\ \phi) \sqsubseteq S_1\,;S_2$ and NSNIC-1 hold, then

$$\textsc{spec}(\psi\ ,\ \phi) \sqsubseteq S_1\,;\mathbb{X}_J^*\,;S_2.$$

Proof. By (1), we have to show $\psi \Rightarrow [S_1\,;\mathbb{X}_J^*\,;S_2]\phi$.

$[S_1; \mathbb{X}^*_J; S_2]\phi$
$\Leftrightarrow$ $\langle$ def of "[ ]" for ";" $\rangle$
$[S_1][\mathbb{X}^*_J][S_2]\phi$
$\Leftarrow$ $\langle$ Lemma 2 with NSNIC-1 $\rangle$
$[S_1]\langle S_1\rangle\psi$
$\Leftrightarrow$ $\langle$ $[S1]\langle S_1\rangle\psi \Leftrightarrow \psi$ $\rangle$
ψ

□

Lemma 2. If NSNIC-1, then $\langle S_1\rangle\psi \Rightarrow [\mathbb{X}^*_J][S_2]\phi$.

Proof. $[\mathbb{X}^*_J][S_2]\phi$
$\Leftrightarrow$ $\langle$ def. of "[ ]" for "*" $\rangle$
$\bigwedge_{i>=0}[\mathbb{X}^i_J][S_2]\phi$
$\Leftarrow$ $\langle$ Lemma 3, NSNIC-1 and monotonicity of $\wedge$ for $\Rightarrow$ $\rangle$
$\bigwedge_{i>=0}\langle S_1\rangle\psi$
$\Leftrightarrow$ $\langle$ idempotence $\rangle$
$\langle S_1\rangle\psi$ □

Lemma 3. If NSNIC-1 holds, then $\forall\, i \geq 0 \cdot \langle S_1\rangle\psi \Rightarrow [\mathbb{X}^i_J][S_2]\phi$

Proof. Case $i = 0$, we have

$[\textsc{skip}][S_2]\phi$
$=$
$[S_2]\phi$
$\Leftarrow$
$\langle S_1\rangle\psi$.

For $i > 0$: by induction.
Induction basis: $i = 1$: by Lemma 4.
Induction step:

$[\mathbb{X}^n_J][S_2]\phi$
$\Leftrightarrow$ $\langle$ def. $\mathbb{X}^n_J$ for $n > 0$ $\rangle$
$[\mathbb{X}_J\,; \mathbb{X}^{n-1}_J][S_2]\phi$
$\Leftrightarrow$ $\langle$ def. "[ ]" for ";" $\rangle$
$[\mathbb{X}_J][\mathbb{X}^{n-1}_J][S_2]\phi$
$\Leftarrow$ $\langle$ ind. hyp. $\rangle$
$[\mathbb{X}_J]\langle S_1\rangle\psi$
$\Leftarrow$ $\langle$ NSNIC-1 and Lemma 4 $\rangle$
$\langle S_1\rangle\psi$ □

Lemma 4. If NSNIC-1 holds, then $[S_2]\phi \Rightarrow [\mathbb{X}_J][S_2]\phi$

Proof. $[\mathbb{X}_J][S_2]\phi$
$\Leftrightarrow$ $\langle$ def. $\mathbb{X}_J$ $\rangle$
$[\mathbin{[\!]}_{j\in J}\mathbf{X}^\sharp_j][S_2]\phi$
$\Leftrightarrow$ $\langle$ def. "[ ]" for $[\!]$ $\rangle$

$$\begin{array}{ll}
 & \bigwedge_{j\in J}[\mathbf{X}_j^\sharp][S_2]\phi \\
\Leftarrow & \qquad\qquad \langle\ \text{Lemma 5}\ \rangle \\
 & \bigwedge_{j\in J}\langle S_1\rangle\psi \\
\Leftrightarrow & \qquad\qquad \langle\ \text{idempotence} \wedge\ \rangle \\
 & \langle S_1\rangle\psi
\end{array}$$

Lemma 5. If NSNIC-1, then $\langle S_1\rangle\psi \Rightarrow [\mathbf{X}^\sharp][S_2]\phi$.

Proof. $[\mathbf{X}^\sharp][S_2]\phi$

$$\begin{array}{ll}
\Leftrightarrow & \qquad\qquad \langle\ \text{apply } [\mathbf{X}^\sharp]\ \rangle \\
 & (pre(\mathbf{X}) \wedge [\mathbf{X}][S_2]\phi) \vee (\neg pre(\mathbf{X}) \wedge [\textsc{skip}][S_2]\phi) \\
\Leftarrow & \qquad\qquad \langle\ \text{NSNIC-1 and } [\textsc{skip}]\phi = \phi\ \rangle \\
 & (pre(\mathbf{X}) \wedge \langle S_1\rangle\psi) \vee (\neg pre(\mathbf{X}) \wedge [S_2]\phi) \\
\Leftarrow & \qquad\qquad \langle\ \langle S_1\rangle\psi \Rightarrow [S_2]\phi\ \rangle \\
 & (pre(\mathbf{X}) \wedge \langle S_1\rangle\psi) \vee (\neg pre(\mathbf{X}) \wedge \langle S_1\rangle\psi) \\
\Leftrightarrow & \\
 & \langle S_1\rangle\psi
\end{array}$$

□

Proof of Lemmas used in the While Case

Lemma 6. If NSNIC-2 and NSNIC-3 and

$$\textsc{spec}(I\ ,\ I \wedge \neg C) \sqsubseteq \textsc{while}\ C\ \textsc{do}\ S\ \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end} \tag{20}$$

hold, then $\textsc{spec}(I\ ,\ I \wedge \neg C) \sqsubseteq \textsc{while}\ C\ \textsc{do}\ S\,;\textsc{if}\ C\ \textsc{then}\ \mathbb{X}_J^*\ \textsc{end}$ $\textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}.$

Proof. From

$$\textsc{spec}(I\ ,\ I \wedge \neg C) \sqsubseteq \textsc{while}\ C\ \textsc{do}\ S\ \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}$$

we can deduce using (1) and the semantics of WHILE

$$\begin{array}{ll}
 & I \Rightarrow \\
\text{(H1)} & I \wedge \\
\text{(H2)} & \forall w \cdot (C \wedge I \Rightarrow [S]I) \wedge \\
\text{(H3)} & \forall w \cdot (I \Rightarrow V \in \mathbb{N}) \wedge \\
\text{(H4)} & \forall w, n \cdot (C \wedge I \wedge n = V \Rightarrow [S]V < n) \wedge \\
\text{(H5)} & \forall w \cdot (\neg C \wedge I \Rightarrow I \wedge \neg C)
\end{array}$$

We now rewrite the goal to obtain simpler ones.

$$\begin{array}{ll}
 & \quad \textsc{spec}(I\ ,\ I \wedge \neg C) \\
\sqsubseteq & \qquad \textsc{while}\ C\ \textsc{do}\ S\,;\textsc{if}\ C\ \textsc{then}\ \mathbb{X}_J^*\ \textsc{end} \\
 & \qquad \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}. \\
\Leftrightarrow & \qquad\qquad \langle\ (1)\ \rangle \\
 & I \Rightarrow \\
 & \quad [\textsc{while}\ C\ \textsc{do}\ S\,;\textsc{if}\ C\ \textsc{then}\ \mathbb{X}_J^*\ \textsc{end}
\end{array}$$

$$
\begin{array}{ll}
 & \quad\quad\quad\quad \text{INVARIANT } I \text{ VARIANT } V \text{ END}](I \wedge \neg C) \\
\Leftrightarrow & \quad\quad\quad \langle \text{ def. of ``[WHILE]''} \ \rangle \\
 & I \Rightarrow \\
 & \text{(G1)} \quad I \wedge \\
 & \text{(G2)} \quad \forall w \cdot (C \wedge I \Rightarrow [S\,; \text{IF } C \text{ THEN } \mathbb{X}_J^* \text{ END}]I) \wedge \\
 & \text{(G3)} \quad \forall w \cdot (I \Rightarrow V \in \mathbb{N}) \wedge \\
 & \text{(G4)} \quad \forall w, n \cdot (C \wedge I \wedge n = V \Rightarrow [S\,; \text{IF } C \text{ THEN } \mathbb{X}_J^* \text{ END}]V < n) \wedge \\
 & \text{(G5)} \quad \forall w \cdot (\neg C \wedge I \Rightarrow I \wedge \neg C)
\end{array}
$$

(G1) is immediate from hypothesis I.
(G3) follows from H3.
(G5) trivially holds.
It remains to prove (G2) and (G4). We prove (G2) first.

$$
\begin{array}{ll}
 & [S\,; \text{IF } C \text{ THEN } \mathbb{X}_J^* \text{ END}]I \\
\Leftrightarrow & \quad\quad \langle \text{ def. ``[;]'' } \rangle \\
 & [S][\text{IF } C \text{ THEN } \mathbb{X}_J^* \text{ END}]I \\
\Leftrightarrow & \quad\quad \langle \text{ def. [IF] } \rangle \\
 & [S]((C \Rightarrow [\mathbb{X}_J^*]I) \wedge (\neg C \Rightarrow [\text{SKIP}]I)) \\
\Leftrightarrow & \quad\quad \langle \text{ def. [SKIP] } \rangle \\
 & [S]((C \Rightarrow [\mathbb{X}_J^*]I) \wedge (\neg C \Rightarrow I))) \\
\Leftarrow & \quad\quad \langle \text{ NSNIC-2 with a lemma similar to Lemma 2, but for } I \text{ instead} \\
 & \quad\quad \text{of } [S_2]\phi \ \rangle \\
 & [S](C \Rightarrow (\langle S\rangle(I \wedge C) \wedge C) \wedge (\neg C \Rightarrow I)) \\
\Leftrightarrow & \\
 & [S](C \Rightarrow (\langle S\rangle(I \wedge C)) \wedge (\neg C \Rightarrow I)) \\
\Leftarrow & \\
 & [S]\langle S\rangle(I \wedge C) \wedge [S]I \\
\Leftrightarrow & \quad\quad \langle \text{ by (20), } S \text{ terminates for } I \wedge C \text{, thus } [S]\langle S\rangle(I \wedge C) \Leftrightarrow I \wedge C \ \rangle \\
 & (I \wedge C) \wedge [S]I \\
\Leftarrow & \quad\quad \langle \text{ H2 } \rangle \\
 & C \wedge I
\end{array}
$$

The proof above uses a lemma whose proof is similar to Lemma 2, the only difference being that it is done for I instead of $[S_2]\phi$ using hypothesis NSNIC-2. Thus, we only need to provide the proof of $\langle S\rangle(I \wedge C) \wedge C \Rightarrow [\mathbf{X}^\sharp]I$ under NSNIC-2.

$$
\begin{array}{ll}
 & [\mathbf{X}^\sharp]I \\
\Leftrightarrow & \\
 & (pre(\mathbf{X}) \wedge [\mathbf{X}]I) \vee (\neg pre(\mathbf{X}) \wedge I) \\
\Leftarrow & \quad\quad \langle \text{ NSNIC-2 } \rangle \\
 & (pre(\mathbf{X}) \wedge (\langle S\rangle(C \wedge I) \wedge C) \vee (\neg pre(\mathbf{X}) \wedge I) \\
\Leftarrow & \quad\quad \langle\ \langle S\rangle(I \wedge C) \Rightarrow I\ \rangle \\
 & \langle S\rangle(I \wedge C) \wedge C
\end{array}
$$

We now prove (G4); it is essentially the same proof as (G2), but using $V < n$ and NSNIC-3 instead.

$$
\begin{array}{ll}
 & [S\,;\textsc{if}\ C\ \textsc{then}\ \mathbb{X}_J^*\ \textsc{end}]V < n \\
\Leftrightarrow & \qquad \langle \text{ def. "[;]" } \rangle \\
 & [S][\textsc{if}\ C\ \textsc{then}\ \mathbb{X}_J^*\ \textsc{end}]V < n \\
\Leftrightarrow & \qquad \langle \text{ def. } [\textsc{if}] \rangle \\
 & [S]((C \Rightarrow [\mathbb{X}_J^*]V < n) \wedge (\neg C \Rightarrow [\textsc{skip}]V < n)) \\
\Leftrightarrow & \qquad \langle \text{ def. } [\textsc{skip}] \rangle \\
 & [S]((C \Rightarrow [\mathbb{X}_J^*]V < n) \wedge (\neg C \Rightarrow V < n))) \\
\Leftarrow & \qquad \langle \text{ NSNIC-2 with a lemma similar to Lemma 2, but for } V < n \\
 & \qquad \text{instead of } [S_2]\phi \rangle \\
 & [S](C \Rightarrow (\langle S\rangle(I \wedge C) \wedge C \wedge V < n) \wedge (\neg C \Rightarrow V < n)) \\
\Leftrightarrow & \\
 & [S](C \Rightarrow (\langle S\rangle(I \wedge C) \wedge V < n) \wedge (\neg C \Rightarrow V < n)) \\
\Leftarrow & \\
 & [S]\langle S\rangle(I \wedge C) \wedge [S]V < n \\
\Leftrightarrow & \qquad \langle \text{ by (20), } S \text{ terminates for } I \wedge C\text{, thus } [S]\langle S\rangle(I \wedge C) \Leftrightarrow I \wedge C \rangle \\
 & (I \wedge C) \wedge [S]V < n \\
\Leftarrow & \qquad \langle \text{ H4 } \rangle \\
 & C \wedge I \wedge V = n
\end{array}
$$

As in (G2), the proof above uses a lemma whose proof is similar to Lemma 2, the only difference being that it is done for $V < n$ instead of $[S_2]\phi$ using hypothesis NSNIC-3. Thus, we only need to provide the proof of $\langle S\rangle(I \wedge C) \wedge C \wedge V < n \Rightarrow [\mathbf{X}^\sharp]V < n$ under NSNIC-3, which stems easily since **X** does not increase V by NSNIC-3.

B.2 Proof of Necessary Condition for the While

Proof of the necessity of NSNIC-2

To prove that NSNIC-2 is a necessary condition, it is sufficient to prove that the execution any operation **X** of **J** that doesn't verify NSNIC-2 really interferes with a program such that:

$$\textsc{spec}(I\ ,\ I \wedge \neg C) \sqsubseteq \textsc{while}\ C\ \textsc{do}\ p\ \textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}$$

Thus, operation **X** verifies:

$$\exists\, x.(\langle p\rangle(C \wedge I) \wedge C \wedge pre(X) \wedge \neg[X]I) \tag{H$_8$}$$

We have to demonstrate that

$$
\begin{array}{r}
\exists\, x.(I \wedge \\
(\neg[\textsc{while}\ C\ \textsc{do}\ \mathbf{Z}(p)\,;\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end} \\
\textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}](I \wedge \neg C)))
\end{array} \tag{G$_2$}
$$

Let x_1 be a value of x verifying H_8:

$$\langle p\rangle(C \wedge I)(x_1) \tag{H$_9$}$$
$$C(x_1) \tag{H$_{10}$}$$
$$pre(X)(x_1) \tag{H$_{11}$}$$
$$(\neg[X]I)(x_1) \tag{H$_{12}$}$$

Rewriting H_9 with (5) gives:

$$\exists\, x_0 \cdot (\widehat{p}(x_0, x_1) \wedge (C \wedge I)(x_0)) \tag{H$_{13}$}$$

Let ${x_0}^1$ be a value of x_0 that verifies H_{13}

$$(C \wedge I)({x_0}^1) \tag{H$_{14}$}$$
$$\widehat{p}({x_0}^1, x_1) \tag{H$_{15}$}$$

Let us prove that ${x_0}^1$ establishes G_2:

$$I({x_0}^1) \tag{${G_1}^3$}$$
$$\begin{array}{r}(\neg[\text{WHILE } C \text{ DO } \mathbf{Z}(p)\,;\text{IF } C \text{ THEN } \mathbf{X}^\sharp \text{ END} \\ \text{INVARIANT } I \text{ VARIANT } V \text{ END}](I \wedge \neg C))({x_0}^1)\end{array} \tag{${G_1}^4$}$$

Sub-goal ${G_1}^3$ is discharged by H_{13}. To establish ${G_1}^4$, it is sufficient to prove it for p, since $\mathbf{Z}(p) \sqsubseteq p$, because $\mathbb{X}_J^* \sqsubseteq$ SKIP; thus, if a path leads outside ϕ from ψ starting from p, this path also exists starting from $\mathbf{Z}(p)$:

$$\begin{array}{r}(\neg[\text{WHILE } C \text{ DO } p\,;\text{IF } C \text{ THEN } \mathbf{X}^\sharp \text{ END} \\ \text{INVARIANT } I \text{ VARIANT } V \text{ END}](I \wedge \neg C))({x_0}^1)\end{array} \tag{${G_1}^5$}$$

To discharge ${G_1}^5$, it suffices to prove that the invariant I is not preserved, that is:

$$I({x_0}^1) \tag{H$_{16}$}$$
$$C({x_0}^1) \tag{H$_{17}$}$$

and

$$(\neg[p\,;\text{IF } C \text{ THEN } \mathbf{X}^\sharp \text{ END}]I)({x_0}^1) \tag{${G_1}^6$}$$

Goal ${G_1}^6$ is discharged as follows:

$$\begin{array}{ll} & (\neg[p\,;\text{IF } C \text{ THEN } \mathbf{X}^\sharp \text{ END}]I)({x_0}^1) \\ \Leftrightarrow & \qquad\qquad \langle \text{ apply } [] \rangle \\ & (\neg([p]([\text{IF } C \text{ THEN } \mathbf{X}^\sharp \text{ END}]I)))({x_0}^1)\end{array}$$

$$
\begin{array}{ll}
\Leftrightarrow & \qquad \langle \text{ apply (4) } \rangle \\
& (\neg(\forall x'.(\widehat{p}(x,x') \Rightarrow ([\textsc{if } C \textsc{ then } \mathbf{X}^\sharp \textsc{ end}]I)(x'))(x_0{}^1) \\
\Leftrightarrow & \qquad \langle \text{ apply Negation operator } \rangle \\
& (\exists x'.(\widehat{p}(x,x') \wedge \neg(([\textsc{if } C \textsc{ then } \mathbf{X}^\sharp \textsc{ end}]I)(x'))))(x_0{}^1) \\
\Leftarrow & \qquad \langle \ \exists \text{ introduction } \rangle \\
& (\widehat{p}(x,x_1) \wedge \neg(([\textsc{if } C \textsc{ then } \mathbf{X}^\sharp \textsc{ end}]I)(x_1)))(x_0{}^1) \\
\Leftrightarrow & \qquad \langle \ \mathrm{H}_{11} \ \rangle \\
& (\widehat{p}(x,x_1) \wedge \neg(([\textsc{if } C \textsc{ then } \mathbf{X} \textsc{ end}]I)(x_1)))(x_0{}^1) \\
\Leftrightarrow & \qquad \langle \ \mathrm{H}_{10} \ \rangle \\
& (\widehat{p}(x,x_1) \wedge \neg(([\mathbf{X}]I)(x_1)))(x_0{}^1) \\
\Leftrightarrow & \qquad \langle \ \mathrm{H}_{12} \ \rangle \\
& (\widehat{p}(x,x_1))(x_0{}^1) \\
\Leftrightarrow & \\
& \mathrm{H}_{15}
\end{array}
$$

□

Proof of the necessity of NSNIC-3

To prove that NSNIC-3 is a necessary condition, it is sufficient to prove that the execution any operation **X** of **J** that doesn't verify NSNIC-3 really interferes with a program of the form WHILE C DO p INVARIANT I VARIANT V END. Thus, operation **X** verifies:

$$\exists(x,n).(\langle p\rangle(C \wedge I) \wedge C \wedge V = n \wedge pre(X) \wedge \neg[X](V < n)) \qquad (\mathrm{H}_{18})$$

We have to demonstrate that

$$
\begin{array}{r}
\exists x.(I \wedge \\
(\neg[\textsc{while } C \textsc{ do } \mathbf{Z}(p)\,;\textsc{if } C \textsc{ then } \mathbf{X}^\sharp \textsc{ end} \\
\textsc{invariant } I \textsc{ variant } V \textsc{ end}](I \wedge \neg C)))
\end{array}
\qquad (\mathrm{G}_3)
$$

Let x_1 and n_1 be the values of x and n verifying H_{18}:

$$\langle p\rangle(C \wedge I)(x_1) \qquad (\mathrm{H}_{19})$$

$$C(x_1) \qquad (\mathrm{H}_{20})$$

$$V = n_1 \qquad (\mathrm{H}_{21})$$

$$pre(X)(x_1) \qquad (\mathrm{H}_{22})$$

$$\neg[X](V < n_1))(x_1) \qquad (\mathrm{H}_{23})$$

H_{19} gives:

$$\exists\, x_0 \cdot (\widehat{p}(x_0,x_1) \wedge (C \wedge I)(x_0)) \qquad (\mathrm{H}_{24})$$

Let $x_0{}^1$ be a value of x_0 that verifies H_{24}

$$(C \wedge I)(x_0{}^1) \tag{H$_{25}$}$$
$$\widehat{p}(x_0{}^1, x_1) \tag{H$_{26}$}$$

Let us prove that $x_0{}^1$ establishes G$_3$:

$$I(x_0{}^1) \tag{$G_2{}^1$}$$
$$\begin{aligned}&(\neg[\textsc{while}\ C\ \textsc{do}\ \mathbf{Z}(p)\,;\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}\\ &\textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}](I \wedge \neg C))(x_0{}^1)\end{aligned} \tag{$G_2{}^2$}$$

Sub-goal $G_2{}^1$ is discharged by H$_{25}$. To establish $G_2{}^2$, it is sufficient to prove it for $\mathbf{Z}(p) = p$:

$$\begin{aligned}&(\neg[\textsc{while}\ C\ \textsc{do}\ \mathbf{Z}(p)\,;\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}\\ &\textsc{invariant}\ I\ \textsc{variant}\ V\ \textsc{end}](I \wedge \neg C))(x_0{}^1)\end{aligned} \tag{$G_2{}^3$}$$

To discharge $G_2{}^3$, it suffices to prove that the variant V does not decrease, that is:

$$I(x_0{}^1) \tag{H$_{27}$}$$
$$C(x_0{}^1) \tag{H$_{28}$}$$

and

$$(\neg[p\,;\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}](V < n_1))(x_0{}^1) \tag{$G_2{}^4$}$$

Goal $G_2{}^4$ is discharged as follows:

$$\begin{aligned}
& (\neg[p\,;\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}](V < n_1))(x_0{}^1)\\
\Leftrightarrow & \qquad \langle \text{ apply } [] \ \rangle\\
& (\neg([p]([\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}](V < n_1))))(x_0{}^1)\\
\Leftrightarrow & \qquad \langle \text{ apply } (4) \ \rangle\\
& (\neg(\forall x'.(\widehat{p}(x,x') \Rightarrow ([\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}](V < n_1))(x'))(x_0{}^1)\\
\Leftrightarrow & \qquad \langle \text{ apply Negation operator } \rangle\\
& (\exists x'.(\widehat{p}(x,x') \wedge \neg(([\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}](V < n_1))(x'))))(x_0{}^1)\\
\Leftarrow & \qquad \langle \ \exists \text{ introduction } \rangle\\
& (\widehat{p}(x,x_1) \wedge \neg(([\textsc{if}\ C\ \textsc{then}\ \mathbf{X}^\sharp\ \textsc{end}](V < n_1))(x_1)))(x_0{}^1)\\
\Leftrightarrow & \qquad \langle \text{ H}_{22} \ \rangle\\
& (\widehat{p}(x,x_1) \wedge \neg(([\textsc{if}\ C\ \textsc{then}\ \mathbf{X}\ \textsc{end}](V < n_1))(x_1)))(x_0{}^1)\\
\Leftrightarrow & \qquad \langle \text{ H}_{20} \ \rangle\\
& (\widehat{p}(x,x_1) \wedge \neg(([\mathbf{X}](V < n_1))(x_1)))(x_0{}^1)\\
\Leftrightarrow & \qquad \langle \text{ H}_{23} \ \rangle\\
& (\widehat{p}(x,x_1))(x_0{}^1)\\
\Leftrightarrow & \\
& \text{H}_{26}
\end{aligned}$$

□