PROVING INVARIANCE PROPERTIES OF PARALLEL PROGRAMS BY BACKWARD INDUCTION

Radhia COUSOT

LRIM-82-02 Mars 1981

UNIVERSITÉ DE METZ

LABORATOIRE DE RECHERCHE EN INFORMATIQUE
DE METZ

Faculté des Sciences
Ile du Saulcy
57045 METZ CEDEX
FRANCE

PROVING INVARIANCE PROPERTIES OF PARALLEL PROGRAMS BY BACKWARD INDUCTION

Radhia Cousot

ABSTRACT

We discuss induction principles for proving invariance (safety) properties of programs. We next show that a sound and complete invariance proof method can be mathematically constructed out of an operational semantics, an induction principle, an assertion language and its semantics.

This approach is applied to the construction of invariance proof methods by backward induction for parallel processes with shared variables and synchronized by conditional critical sections. For partial correctness, we obtain a generalization of Morris & Wegbreit's subgoal induction. For freedom from deadlock, mutual exclusion, non-termination, ... this introduces backward proof methods by reductio ad absurdum. Backward invariance proof methods are finally compared with the classical forward ones à la Lamport-Owicki & Gries.

RESUME

Nous présentons divers principes d'induction permettant de démontrer des propriétés d'invariance des programmes. Nous montrons ensuite comment construire formellement une méthode de preuve d'invariance étant donnés une sémantique opérationnelle, un principe d'induction, un langage d'assertions et sa sémantique.

Nous appliquons ensuite cette approche à la construction de méthodes de preuve d'invariance par induction en arrière pour des programmes composés de processus parallèles partageant des données communes et synchronisés au moyen de sections critiques conditionnelles. Dans le cas de la correction partielle ceci nous conduit à généraliser la méthode de Morris & Wegbreit ('subgoal induction'). Pour l'absence d'interblocages globaux permanents, l'exclusion mutuelle, la non-terminaison, ... cette approche nous permet d'introduire des méthodes de preuve par l'absurde et en arrière. Nous comparons finalement les méthodes de preuve d'invariance en arrière aux méthodes à la Lamport-Owicki & Gries qui procèdent en avant.

PROVING INVARIANCE PROPERTIES OF PARALLEL PROGRAMS BY BACKWARD INDUCTION

Radhia Cousot*

1. INTRODUCTION

Invariance properties of parallel programs include *partial correctness*, *non-termination*, *clean-behavior* (absence of run-time errors), *freedom from deadlock*, *mutual exclusion*. Floyd[67]-Naur[66]-Hoare[69]'s method for proving invariance properties of sequential programs has been generalized to the case of parallel programs by Ashcroft[75], Keller[76], Newton[75], Owicki & Gries[76a], Lamport[77]; Hoare[75], Owicki & Gries[76b]; Apt,Francez & de Roever[80], Cousot & Cousot[80], Levin[79], ... All these proof methods use a *forward induction step* (in the direction of the program's control flow).

We introduce *invariance proof methods* using a *backward induction step* (in the opposite direction of the program's control flow). Therefore we generalize subgoal induction (Manna[70], Morris & Wegbreit[77], King[79]) and contrapositive induction (King[79]) to the case of parallel processes sharing global variables and synchronized by conditional critical sections.

Most proof methods have been empirically designed and some of them have been a posteriori shown to be sound and complete. In contrast we have been able to *mathematically construct the proof methods* which will be introduced. This systematic construction a priori ensures their *soundness* and *completeness*.

This approach is explained in the first part of the paper. We show that an invariance proof method for a programming language is determined by the choice of

- an operational semantics,
- an induction principle,
- an assertion language and its semantics.

Then the design of the corresponding verification conditions amounts to formal

---

* Attaché de Recherche au CNRS. CRIN-LA °262.

** This work was supported by ATP-CNRS "Intelligence Artificielle"

calculus and mainly involves algebraic simplifications.

This approach is applied in the second part to the systematic design of backward invariance proof methods. Examples illustrate these proof methods. Finally forward and backward inductions are compared from a methodological point of view.

2. SYSTEMATIC DESIGN OF INVARIANCE PROOF METHODS

2.1 OPERATIONAL SEMANTICS

A programming language can be formally defined by an operational semantics which associates to each syntactically correct program P a set of states $S[\![P]\!]$ and a (non-deterministic) transition relation $t[\![P]\!]$ between a state and its possible successors. The style of such a formal definition is not relevant to our discussion. Therefore the purpose of this paragraph is only to give one possible operational semantics of the programming language which is used throughout the paper.

2.1.1 Sequential Programs

2.1.1.1 Syntax

A sequential program consists of a list of sequentially executed commands. Commands can be null, assignment, conditional or iteration commands. Each command is preceded and followed by unique labels the only purpose of which is to designate program points.

Let L, V, E and B be respectively given sets of labels, variables, expressions and boolean expressions. The following context-free grammar then defines the sets CL of command lists and C of commands :

$CL ::= L_1:C_1;\ldots;L_m:C_m;L_{m+1}:$ $\quad (m\geq 1)$

$C ::= \underline{\text{skip}} \mid V:=E \mid \underline{\text{if}}\ B\ \underline{\text{then}}\ CL_1\ \underline{\text{else}}\ CL_2\ \underline{\text{fi}} \mid \underline{\text{while}}\ B\ \underline{\text{do}}\ CL\ \underline{\text{od}}$

Given a terminal string N deriving from the non-terminal N we write $N\equiv\alpha_1 N_1 \ldots \alpha_n N_n \alpha_{n+1}$ to state that there exist maybe empty terminal strings α_i and non-empty terminal strings N_i such that N is of the form $\alpha_1 N_1 \ldots \alpha_n N_n \alpha_{n+1}$ and each N_i derives from the non-terminal N_i (i.e. $N \overset{*}{\rightarrow} \alpha_1 N_1 \ldots \alpha_n N_n \alpha_{n+1} \overset{*}{\rightarrow} \alpha_1 N_1 \ldots \alpha_n N_n \alpha_{n+1} = N$).

This notation will be used to express context-sensitive properties of programs such as, for example, that labels can only appear once in a program :

$\forall\ CL \in \mathcal{CL},\ (CL \equiv \alpha L_1 : \beta L_2 : \gamma) \Rightarrow (L_1 \neq L_2)$

2.1.1.2 Semantics

- A state of a sequential program consists in a pair <control state,memory state>. The control state is similar to a program location counter. It is a label which designates a program point. The memory state is a function which associates a value to variables. The domain M of values of the variables is left unspecified.

Formally, the set S[[CL]] of states of program CL is :

$S[\![CL]\!] = \{L \in \mathcal{L} \mid CL \equiv \alpha L : \beta\} \times (V \to M)$

- The transition relation ts[[CL]] between a state <L,M> of program CL and its only (if any) successor <L',M'> will be considered as a binary function on states with boolean result :

$ts[\![CL]\!] \in (S[\![CL]\!] \times S[\![CL]\!]) \to \{tt, ff\}$
$ts[\![CL]\!](\langle L,M\rangle,\langle L',M'\rangle) = [\underline{cond}[\![CL]\!](L,L')(M) \wedge M' = \underline{succ}[\![CL]\!](L)(M)]$

<L',M'> is a successor state of <L,M> if and only if when executing the command designated by label L in memory state M, control goes to the program point designated by label L' (i.e. $\underline{cond}[\![CL]\!](L,L')(M)$ holds) and memory state is changed to $M' = \underline{succ}[\![CL]\!](L)(M)$.

Memory states can only be changed by assignment commands :

$\underline{succ}[\![CL]\!](L)(M) = \underline{if}\ (CL \equiv \alpha L : V := E ; \beta)\ \underline{then}\ \underline{assign}[\![V := E]\!](M)\ \underline{else}\ M$

Let $\mathbb{E} \in (E \to ((V \to M) \to M))$ be a given semantic function, such that $\mathbb{E}[\![E]\!](M)$ is the value of expression E in memory state M. The effect of an assignment V:=E is to evaluate expression E and assign its value to variable V. No other variable is modified since side-effects are disallowed :

$\underline{assign}[\![V := E]\!](M) = M'$

where $M'(V) = \mathbb{E}(E)(M)$ and $M'(W) = M(W)$ when $W \in (\mathcal{V} - \{V\})$.

In order to define the control flow we assume we are given a semantic function $\mathbb{B} \in (B \to ((V \to M) \to \{tt, ff\}))$ such that $\mathbb{B}[\![B]\!](M)$ is the value of boolean expression B in memory state M. Both $\mathbb{E}[\![E]\!]$ and $\mathbb{B}[\![B]\!]$ are partial functions because of possible run-time errors. We denote by $dom(\mathbb{E}[\![E]\!])$ and $dom(\mathbb{B}[\![B]\!])$ their respective domains.

The transition relation cond[[CL]](L,L')(M) between a control state L and its successor L' in memory state M is defined by cases as follows :

$$
\begin{array}{l}
\underline{\mathrm{cond}}[\![\mathrm{CL}]\!](\mathrm{L},\mathrm{L}')(\mathrm{M}) = \\
\quad [\\
\quad\quad [\bigvee \begin{array}{l} \mathrm{CL}\equiv\alpha\mathrm{L}:\underline{\mathrm{skip}};\mathrm{L}':\beta \\ \mathrm{CL}\equiv\alpha\mathrm{L}:\underline{\mathrm{else}}\ \mathrm{CL}'\ \underline{\mathrm{fi}};\mathrm{L}':\beta \\ \mathrm{CL}\equiv\alpha\mathrm{L}:\underline{\mathrm{fi}};\mathrm{L}':\beta \\ \mathrm{CL}\equiv\alpha\mathrm{L}:\mathrm{V}:=\mathrm{E};\mathrm{L}':\beta \wedge \mathrm{M}\in \mathit{dom}(\mathbb{E}[\![\mathrm{E}]\!]) \end{array}] \\
\quad\quad \vee \\
\quad\quad [(\bigvee \begin{array}{l} \mathrm{CL}\equiv\alpha\mathrm{L}:\underline{\mathrm{if}}\ \mathrm{B}\ \underline{\mathrm{then}}\ \mathrm{L}':\beta \\ \mathrm{CL}\equiv\alpha\mathrm{L}:\underline{\mathrm{while}}\ \mathrm{B}\ \underline{\mathrm{do}}\ \mathrm{L}':\beta \\ \mathrm{CL}\equiv\alpha\underline{\mathrm{while}}\ \mathrm{B}\ \underline{\mathrm{do}}\ \mathrm{L}':\mathrm{C}_1;\ldots;\mathrm{L}_m:\mathrm{C}_m;\mathrm{L}:\underline{\mathrm{od}};\beta \end{array}) \\
\quad\quad\quad \wedge\ \mathrm{M}\in \mathit{dom}(\mathbb{B}[\![\mathrm{B}]\!]) \wedge \mathbb{B}[\![\mathrm{B}]\!](\mathrm{M})=\mathrm{tt} \\
\quad\quad] \\
\quad\quad \vee \\
\quad\quad [(\bigvee \begin{array}{l} \mathrm{CL}\equiv\alpha\mathrm{L}:\underline{\mathrm{if}}\ \mathrm{B}\ \underline{\mathrm{then}}\ \mathrm{CL}'\ \underline{\mathrm{else}}\ \mathrm{L}':\beta \\ \mathrm{CL}\equiv\alpha\mathrm{L}:\underline{\mathrm{while}}\ \mathrm{B}\ \underline{\mathrm{do}}\ \mathrm{CL}'\ \underline{\mathrm{od}};\mathrm{L}':\beta \\ \mathrm{CL}\equiv\alpha\underline{\mathrm{while}}\ \mathrm{B}\ \underline{\mathrm{do}}\ \mathrm{L}_1:\mathrm{C}_1;\ldots;\mathrm{L}_m:\mathrm{C}_m;\mathrm{L}:\underline{\mathrm{od}};\mathrm{L}':\beta \end{array}) \\
\quad\quad\quad \wedge\ \mathrm{M}\in \mathit{dom}(\mathbb{B}[\![\mathrm{B}]\!]) \wedge \mathbb{B}[\![\mathrm{B}]\!](\mathrm{M})=\mathrm{ff} \\
\quad\quad] \\
\quad]
\end{array}
$$

For example, if control is at point L before executing a while loop or after executing its body then control goes to point L' designating the first command of its body if the test is well-defined and true, control exits the loop if the test is well-defined and false and the program stops (at L which has no possible successor) if the test ill-defined (so that its evaluation causes a run-time error).

2.1.2 Parallel Programs

A parallel program consists of sequential processes executed concurrently. The processes share global variables. They can be synchronized using conditional critical sections.

2.1.2.1 Syntax

P ::= L_1:cobegin $CL_1 \| \ldots \| CL_n$ coend;L_2:

C ::= await B then CL end

Conditional critical sections cannot be nested (i.e. whenever P≡ αawait B then CL end β then not CL≡α'await B' then CL' end β').

2.1.2.2 Semantics

- A state is a pair <control state,memory state> :

$S[\![P]\!] = C[\![P]\!] \times (V \to M)$

If $P \equiv \underline{L}:\underline{\text{cobegin}}\ CL_1 \| \ldots \| CL_i \| \ldots \| CL_n\ \underline{\text{coend}};\overline{L}:$ then a control state is the entry label $\underline{L}$ (the program is not yet started), the exit label $\overline{L}$ (the program has finished) or a tuple $(L_1,\ldots,L_n)$ of labels (the program is executing and each label L_i designates the current position of process CL_i) :

$$C[\![P]\!] = \{\underline{L}\} \cup \prod_{i=1}^{n} (La[\![CL_i]\!]) \cup \{\overline{L}\}$$

where

$$La[\![CL]\!] = \{L \in L \mid (CL \equiv \alpha L:\beta) \wedge \neg(CL \equiv \alpha\underline{\text{await}}\ B\ \underline{\text{then}}\ \beta L:\gamma\ \underline{\text{end}}\ \delta)\}$$

- The transition relation

$t[\![P]\!] \in ((S[\![P]\!] \times S[\![P]\!]) \to \{tt,ff\})$

is defined by cases :

. All processes of a parallel program start simultaneously :

(2.1.2.2-1)

$$t[\![P]\!](<\underline{L},\underline{M}>,<(L_1,\ldots,L_n),M'>) =$$
$$[P \equiv \underline{L}:\underline{\text{cobegin}}\ L_1:\alpha_1 \| \ldots \| L_n:\alpha_n\ \underline{\text{coend}}\ \beta \wedge M'=\underline{M}]$$

. Parallel execution of processes is modelized by non-deterministic interleaving of atomic actions. A program step consists in the indivisible execution of an atomic step of some process CL_i while the other processes CL_j, $j \neq i$ do not evolve. The indivisible step either corresponds to the evaluation of an assignment or a test (as previously defined for sequential programs by ts) or to a conditional critical section. It has already been observed that this atomicity condition on tests and assignments can be lifted when memory reference is indivisible and tests and assignments at most refer once to at most one variable which can be changed by another process (Owicki & Gries[76a]). The evaluation of a conditional critical section is delayed until the condition is true in which case the body is executed in mutual exclusion. Fairness conditions on the scheduler are irrelevant to invariance properties.

(2.1.2.2-2)

$$t[\![P]\!](<(L_1,\ldots,L_n),M>,<(L'_1,\ldots,L'_n),M'>) =$$
$$[\exists i \in [1,n] \mid P \equiv \underline{L}:\underline{\text{cobegin}}\ CL_1 \| \ldots \| CL_i \| \ldots \| CL_n\ \underline{\text{coend}};\overline{L}:$$
$$\wedge\ (\forall j \in [1,n]-\{i\},\ L'_j=L_j)$$
$$\wedge\ ((ts[\![CL_i]\!](<L_i,M>,<L'_i,M'>)$$
$$\vee (CL_i \equiv \alpha L_i:\underline{\text{await}}\ B\ \underline{\text{then}}\ CL\ \underline{\text{end}};L'_i:\beta \wedge M \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M)=tt$$
$$\wedge\ CL \equiv \underline{L}_1:\gamma;\overline{L}_2: \wedge\ ts[\![CL]\!]^*(<\underline{L}_1,M>,<\overline{L}_2,M'>))$$
$$)$$
$$]$$

(The reflexive transitive closure of a relation θ is denoted θ^*).

. The parallel program ends only when its constituent processes have all finished :

(2.1.2.2-3)

$$t[\![P]\!](<(L_1,\ldots,L_n),M>,<\overline{L},\overline{M}>) =$$
$$[P \equiv \alpha \underline{\text{cobegin}}\ \alpha_1;L_1:\|\ldots\|\alpha_n;L_n:\underline{\text{coend}};\overline{L}: \wedge\ \overline{M}=M]$$

2.2 INDUCTION PRINCIPLES

2.2.1 Invariance Properties

A property Ψ is invariant for a program P if and only if it holds for any pair $(\underline{s},\overline{s})$ of states such that the final state $\overline{s}$ is a possible descendant of the initial state $\underline{s}$ when executing P.

More formally, let $\varepsilon \in (S[\![P]\!] \to \{tt,ff\})$ and $\sigma \in (S[\![P]\!] \to \{tt,ff\})$ be characteristic predicates of respectively initial and final states.
$\Psi \in ((S[\![P]\!] \times S[\![P]\!]) \to \{tt,ff\})$ is an invariant property of program P with respect to conditions ε and σ if and only if :

$$\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2,\quad [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s},\overline{s})$$

Example : Non-termination is an invariance property : a program does not terminate if and only if no possible descendant of the entry states is an exit state. Formally one can define :

$$\varepsilon(\underline{s}) = [\exists \underline{L} \in \mathcal{L}, \underline{M} \in (V \to M) \mid \underline{s}=<\underline{L},\underline{M}> \wedge P \equiv \underline{L}:\alpha]$$
$$\sigma(\overline{s}) = tt$$
$$\Psi(\underline{s},\overline{s}) = [(P \equiv \alpha;\overline{L}:) \Rightarrow (\forall \overline{M} \in (V \to M), \overline{s} \neq <\overline{L},\overline{M}>)] \quad \square$$

2.2.2 Forward Positive Induction *I*

In order to prove that Ψ is invariant for a program P, the fundamental forward proof method consists in inventing an induction hypothesis $I(\underline{s},s)$ which relates the current state s to the initial state $\underline{s}$ and such that :

. I holds when the current state is an initial state,

. assuming that I is true for some current state s, then I remains true for all possible successors s' of s,

. when the current state is a final state then I implies Ψ.

THEOREM 2.2.2.1 : *Forward Positive Induction I*

$$[\exists I \in (S[\![P]\!]^2 \to \{tt,ff\}) \mid \forall (\underline{s},s,s',\overline{s}) \in S[\![P]\!]^4,$$
$$(a) \quad \varepsilon(\underline{s}) \Rightarrow I(\underline{s},\underline{s})$$
$$(b) \quad \wedge\ [\varepsilon(\underline{s}) \wedge I(\underline{s},s) \wedge t[\![P]\!](s,s')] \Rightarrow I(\underline{s},s')$$
$$(c) \quad \wedge\ [\varepsilon(\underline{s}) \wedge I(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s},\overline{s}) \quad]$$
$$\Leftrightarrow$$
$$[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2,\ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s},\overline{s})]$$

The soundness proof consists in showing that $\forall n \geq 0,\ \varepsilon(\underline{s}) \wedge t[\![P]\!]^n(\underline{s},s) \Rightarrow I(\underline{s},s)$. This is done by induction on n, using (a) for the basis n=0 and (b) for the induction step. Then (c) implies that Ψ is invariant. The completeness proof consists in remarking that $I(\underline{s},s)$ can be chosen as $t[\![P]\!]^*(\underline{s},s)$.

This induction principle was used by Manna[70], Floyd[67], Naur[66], Hoare[69] and their followers do not relate the current and initial states but only use a predicate on the current state :

COROLLARY 2.2.2-2 : *Forward Positive Induction i*

$$[\exists i \in (S[\![P]\!] \to \{tt,ff\}) \mid \forall (\underline{s},s,s',\overline{s}) \in S[\![P]\!]^4,$$
$$(a) \quad \varepsilon(\underline{s}) \Rightarrow i(\underline{s})$$
$$(b) \quad \wedge\ [i(s) \wedge t[\![P]\!](s,s')] \Rightarrow i(s')$$
$$(c) \quad \wedge\ [i(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s}) \quad]$$
$$\Leftrightarrow$$
$$[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2,\ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})]$$

The proof consists in defining $\Psi(\underline{s},\overline{s})=\psi(\overline{s})$, $I(\underline{s},s)=\varepsilon(\underline{s})\wedge i(s)$ and applying theorem 2.2.2-1. In fact, when Ψ does not depend upon the initial state, these two proof methods are equivalent in the sense that a proof by either of these two methods can be used to produce a proof by the other :

THEOREM 2.2.2-3 : $I \cong i$

If $\Psi(\underline{s},s)=\psi(s)$ then conditions 2.2.2-1(a)-(c) and 2.2.2-2(a)-(c) are equivalent by defining :

$$I(\underline{s},s) = \varepsilon(\underline{s}) \wedge i(s)$$
$$i(\overline{s}) = (\exists \underline{s} \in \overline{S[\![P]\!]} \mid \varepsilon(\underline{s}) \wedge I(\underline{s},s))$$

2.2.3 Backward Positive Induction I^{-1}

In order to prove that Ψ is invariant for a program P, the backward proof method consists in inventing an induction hypothesis $J(s,\overline{s})$ which relates the current state s to the final state $\overline{s}$ and such that :

. J holds when the current state is a final state,

. assuming that J is true for some current state s' then J remains true for all possible predecessors s of s',

. when the current state is an initial state, J implies Ψ.

THEOREM 2.2.3-1 : *Backward Positive Induction* I^{-1}

$$[\exists J \in (S[\![P]\!]^2 \to \{tt,ff\}) \mid \forall (\underline{s},s,s',\overline{s}) \in S[\![P]\!]^4,$$
$$(a) \quad \sigma(\overline{s}) \Rightarrow J(\overline{s},\overline{s})$$
$$(b) \quad \wedge \ [t[\![P]\!](s,s') \wedge J(s',\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow J(s,\overline{s})$$
$$(c) \quad \wedge \ [\varepsilon(\underline{s}) \wedge J(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s},\overline{s}) \quad]$$
$$\Leftrightarrow$$
$$[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2, \ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s},\overline{s})]$$

The proof consists in applying forward induction I to the inverses Ψ^{-1} and $t[\![P]\!]^{-1}$ of Ψ and $t[\![P]\!]$ respectively, using σ as initial and ε as final conditions. One then applies the definition $\theta^{-1}(s_1,s_2)=\theta(s_2,s_1)$, remarks that $\theta^{-1*}=\theta^{*-1}$, renames the dummy variables $\underline{s}$, s, s', $\overline{s}$ respectively as $\overline{s}$, s', s, $\underline{s}$ and finally defines $J(s_1,s_2)$ as $I(s_2,s_1)$.

Theorem 2.2.3-1 can also be proved with respect to 2.2.2-1 using the following theorem which also shows that forward and backward inductions are formally equivalent :

THEOREM 2.2.3-2 : $I \equiv I^{-1}$

The conditions 2.2.2-1(a)-(c) and 2.2.3-1(a)-(c) are equivalent by defining :

$$I(\underline{s},s) = [\forall \overline{s} \in S[\![P]\!], \ (J(s,\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \Psi(\underline{s},\overline{s})]$$
$$J(\overline{s},\overline{s}) = [\forall \underline{s} \in S[\![P]\!], \ (\varepsilon(\underline{s}) \wedge I(\underline{s},s)) \Rightarrow \Psi(\underline{s},\overline{s})]$$

In the proof 2.2.3-1(a) follows from 2.2.2-1(c), 2.2.3-1(b) follows from 2.2.2-1(b), 2.2.3-1(c) follows from 2.2.2-1(a) and reciprocally by simple formal calculus.

The backward induction principle I^{-1} is involved in subgoal induction. Morris & Wegbreit[77] have also shown that subgoal induction is equivalent to Floyd's inductive assertion method for the partial correctness proof of terminating while loops. Theorem 2.2.3-2 generalizes this result independently of a particular invariance property and a particular class of programs.

When Ψ does not depend upon the final states, one can use a particular case i^{-1} of induction principle I^{-1} which is equivalent to I^{-1}.

THEOREM 2.2.3-3 : *Backward Positive Induction* i^{-1}

$$[\exists j \in (S[\![P]\!] \to \{tt,ff\}) \mid \forall (\underline{s},s,s',\overline{s}) \in S[\![P]\!]^4,$$
$$(a) \quad \sigma(\overline{s}) \Rightarrow j(\overline{s})$$
$$(b) \quad \wedge \ [t[\![P]\!](s,s') \wedge j(s')] \Rightarrow j(s)$$
$$(c) \quad \wedge \ [\varepsilon(\underline{s}) \wedge j(\underline{s})] \Rightarrow \Psi(\underline{s}) \qquad]$$
$$\Leftrightarrow$$
$$[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2, [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s})]$$

THEOREM 2.2.3-4 : $I^{-1} \simeq i^{-1}$

If $\Psi(\underline{s},\overline{s})=\Psi(\underline{s})$ then conditions 2.2.3-1(a)-(c) and 2.2.3-3(a)-(c) are equivalent by defining :

$$j(s) = [\exists \overline{s} \in S[\![P]\!] \mid J(s,\overline{s}) \wedge \sigma(\overline{s})]$$
$$J(s,\overline{s}) = [j(s) \wedge \sigma(\overline{s})]$$

2.2.4 Backward Contrapositive Induction $\overline{I}$

Invariance properties can be proved by reductio ad absurdum : In order to prove that final values satisfy some assertion Ψ, the contrary is assumed. Examining the necessary behavior of the program leading to such final states not satisfying Ψ, one invents an induction hypothesis $\overline{I}$ which is first shown to be satisfied by the final states not satisfying Ψ, and using an induction step, is next shown to be satisfied by all possible ascendants of the final states not satisfying Ψ. It is finally shown that the initial states do not satisfy $\overline{I}$. Therefore assuming that the initial states are possible ascendants of the final states, we get a contradiction and Ψ must be achieved by final states. Formally speaking, we have :

THEOREM 2.2.4-1 : *Backward Contrapositive Induction* $\overline{I}$

$$[\exists \overline{I} \in (S[\![P]\!]^2 \to \{tt,ff\}) \mid \forall (\underline{s},s,s',\overline{s}) \in S[\![P]\!]^4,$$
$$(a) \quad [\varepsilon(\underline{s}) \wedge \sigma(\overline{s}) \wedge \neg\Psi(\underline{s},\overline{s})] \Rightarrow \overline{I}(\underline{s},\overline{s})$$
$$(b) \quad \wedge \ [\varepsilon(\underline{s}) \wedge t[\![P]\!](s,s') \wedge \overline{I}(\underline{s},s')] \Rightarrow \overline{I}(\underline{s},s)$$
$$(c) \quad \wedge \ \varepsilon(\underline{s}) \Rightarrow \neg\overline{I}(\underline{s},\underline{s}) \qquad]$$
$$\Leftrightarrow$$
$$[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2, [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s},\overline{s})]$$

The proof consists in using theorem 2.2.2-1 where $I=\neg\overline{I}$ and the tautology $P \Rightarrow Q$ if and only if $\neg Q \Rightarrow \neg P$. As a consequence $\overline{I}$ is the contrapositive version of I :

THEOREM 2.2.4-2 : $I \equiv \overline{\overline{I}}$

Conditions 2.2.1-1(a)-(c) and 2.2.4-1(c)-(a) are equivalent by defining $I=\neg\overline{I}$ and $\overline{I}=\neg I$.

THEOREM 2.2.4-3 : *Backward Contrapositive Induction* $\overline{i}$

$$
\begin{array}{l}
[\exists \overline{i} \in (S[\![P]\!] \to \{tt,ff\}) \mid \forall (\underline{s},s,s',\overline{s}) \in S[\![P]\!]^4, \\
\quad (a) \quad [\sigma(\overline{s}) \wedge \neg\psi(\overline{s})] \Rightarrow \overline{i}(\overline{s}) \\
\quad (b) \quad \wedge\ [t[\![P]\!](s,s') \wedge \overline{i}(s')] \Rightarrow \overline{i}(s) \\
\quad (c) \quad \wedge\ \varepsilon(\underline{s}) \Rightarrow \neg\overline{i}(\underline{s}) \qquad] \\
\Leftrightarrow \\
[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2,\ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})]
\end{array}
$$

This theorem either results from theorem 2.2.4-1 or theorem 2.2.2-2 and as a corollary we get :

THEOREM 2.2.4-4 : $\overline{I} \cong \overline{\overline{i}} \equiv \overline{i}$

If $\Psi(\underline{s},\overline{s})=\psi(\underline{s})$ then conditions 2.2.4-3(a)-(c) are equivalent to conditions 2.2.4-1(a)-(c) by defining :

$\overline{I}(\underline{s},s) = [\varepsilon(\underline{s}) \wedge \overline{i}(s)]$

$\overline{i}(s) = [\exists \underline{s} \in S[\![P]\!] \mid \varepsilon(\underline{s}) \wedge \overline{I}(\underline{s},s)]$

and to conditions 2.2.2(c)-(a) by defining $i=\neg\overline{i}$ and $\overline{i}=\neg i$.

2.2.5 Forward Contrapositive Induction $\overline{I^{-1}}$

A last series of theorems is obtained by constructing $\overline{I^{-1}}$ which is shown to be equivalent to I^{-1} therefore to I. $(\overline{I})^{-1}$ turns out to be $\overline{I^{-1}}$. The variants $\overline{i^{-1}} = (\overline{i})^{-1}$ are also easily constructed. We will only give one

THEOREM 2.2.5-1 : *Forward Contrapositive Induction* $\overline{I^{-1}}$

$$
\begin{array}{l}
[\exists \overline{J} \in (S[\![P]\!]^2 \to \{tt,ff\}) \mid \forall (\underline{s},s,s',\overline{s}) \in S[\![P]\!]^4, \\
\quad (a) \quad [\varepsilon(\underline{s}) \wedge \sigma(\overline{s}) \wedge \neg\Psi(\underline{s},\overline{s})] \Rightarrow \overline{J}(\underline{s},\overline{s}) \\
\quad (b) \quad \wedge\ [\overline{J}(\underline{s},\overline{s}) \wedge t[\![P]\!](s,s') \wedge \sigma(\overline{s})] \Rightarrow \overline{J}(s',\overline{s}) \\
\quad (c) \quad \wedge\ \sigma(\overline{s}) \Rightarrow \neg\overline{J}(\underline{s},\overline{s}) \qquad] \\
\Leftrightarrow \\
[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2,\ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \Psi(\underline{s},\overline{s})]
\end{array}
$$

2.2.6 Invariance Proof Methods

To sum up, we can distinguish between the cases when the invariant property is a relation or an assertion :

2.2.6.1 Ψ is a relation between initial and final states

We can choose an induction hypothesis which relates the current state to either the initial or the final states. Independently we can choose either a forward or a backward induction. Then the corresponding induction principle is given by the following table :

	The induction hypothesis relates the current state to the initial states	The induction hypothesis relates the current state to the final states
Forward Induction	I	$\overline{I^{-1}}$
Backward Induction	$\overline{I}$	I^{-1}

These invariance proof methods are equivalent $I \equiv I^{-1} \equiv \overline{I} \equiv \overline{I^{-1}} = \overline{I}^{-1}$ in the sense that a proof by either of these methods can immediately be rephrased as an equivalent proof by any of the other methods. In practice however, choosing the right method can sometimes make both the discovery of an induction hypothesis and the subsequent proof simpler.

2.2.6.2 Ψ is an assertion upon either initial or final states

The choice of one of the above proof methods is still possible but for simplicity the induction hypothesis can be taken to be an assertion on the current state, the choice beeing between forward and backward induction :

	The invariant property Ψ is an assertion upon initial states	The invariant property Ψ is an assertion upon final states
Forward Induction	$\overline{i^{-1}}$	i
Backward Induction	i^{-1}	$\overline{i}$

Again these proof methods are formally equivalent $i \equiv \overline{i}$ and $i^{-1} \equiv \overline{i^{-1}}$.

2.3 ASSERTION LANGUAGE

Assume Ψ has to be proved invariant for a program P with respect to initial and final conditions ε and σ. This proof can be done using one of the invariance proof methods discussed at paragraph 2.2. These proof methods all have the same form. First, an induction hypothesis I has to be invented. This induction hypothesis I belongs to an assertion language denoted $A[\![P]\!]$ (i.e. I is either a relation and $A[\![P]\!]=(S[\![P]\!]^2 \to \{tt,ff\})$ or I is an assertion and $A[\![P]\!]=(S[\![P]\!] \to \{tt,ff\})$). Next, I must be shown to satisfy some verification condition C depending upon P, ε, σ and Ψ. Formally speaking all invariance proof methods have the form :

(2.3-1)

$$[\exists I \in A[\![P]\!] \mid C[\![P]\!](\varepsilon,\sigma)(\Psi)(I)]$$

Moreover these proof methods are sound and complete so that :

(2.3-2)

$$[\exists I \in A[\![P]\!] \mid C[\![P]\!](\varepsilon,\sigma)(\Psi)(I)]$$
$$\iff$$
$$[\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2,\ (\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \Psi(\underline{s},\overline{s})]$$

In practice using the assertion language $A[\![P]\!]$ may not be convenient because a single assertion or relation has to be used in order to describe the possible behavior of program P. For example, when proving the partial correctness of a sequential program, local assertions on memory states attached to each program point are often much more simple to use than a single global assertion on a program location counter and memory states. Therefore, practical invariance proof methods have the form :

(2.3-3)

$$[\exists \tilde{I} \in \tilde{A}[\![P]\!] \mid \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{I})]$$

where $\tilde{A}[\![P]\!]$ is the assertion language and $\tilde{C}[\![P]\!]$ the verification condition. For example, in the case of sequential programs, the assertions of $\tilde{A}[\![P]\!]$ can be chosen as a vector of predicates on memory states, each element of this vector corresponding to a program point (or to a loop cut-point).

2.4 CONSTRUCTION OF AN INVARIANCE PROOF METHOD

We show in this paragraph that once an assertion language $\tilde{A}[\![P]\!]$ has been chosen to express induction hypothesis $\tilde{I}$, the verification condition $\tilde{C}[\![P]\!]$

can be mathematically constructed, this contruction involving mainly algebraic simplifications.

First of all, the invariance proof method which is designed must be correct, i.e.

$$[\exists \tilde{I}\in\tilde{A}[\![P]\!] \mid \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{I})]$$
$$\Rightarrow$$
$$[\forall(\underline{s},\overline{s})\in S[\![P]\!]^2,\ (\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \Psi(\underline{s},\overline{s})]$$

By 2.3-2 this is equivalent to :

$$[\exists \tilde{I}\in\tilde{A}[\![P]\!] \mid \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{I})] \Rightarrow [\exists I\in A[\![P]\!] \mid C[\![P]\!](\varepsilon,\sigma)(\Psi)(I)]$$

or

$$[\forall \tilde{I}\in\tilde{A}[\![P]\!],\ \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{I}) \Rightarrow [\exists I\in A[\![P]\!] \mid C[\![P]\!](\varepsilon,\sigma)(\Psi)(I)]]$$

or using a Skolem function $\rho[\![P]\!]$

(2.4-1)

$$[\exists \tilde{\rho}[\![P]\!]\in(\tilde{A}[\![P]\!]\to A[\![P]\!]) \mid$$
$$\forall \tilde{I}\in\tilde{A}[\![P]\!],\ \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{I}) \Rightarrow C[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{\rho}[\![P]\!](\tilde{I}))]$$

Therefore in order to design a variant 2.3-3 of 2.3-1 one must first choose an assertion language $\tilde{A}[\![P]\!]$ and define its semantics $\tilde{\rho}[\![P]\!]$ in term of $A[\![P]\!]$. This variant is sound if and only if the verification condition $\tilde{C}[\![P]\!]$ applied to an invariant $\tilde{I}$ implies the verification condition $C[\![P]\!]$ applied to the meaning $\tilde{\rho}[\![P]\!](\tilde{I})$ of $\tilde{I}$. This leads to a mathematical construction of $\tilde{C}[\![P]\!]$ by formal calculus of $C[\![P]\!](\varepsilon,\sigma)(\Psi)\circ\tilde{\rho}[\![P]\!]$. Moreover algebraic simplifications are possible because implication by and not equality with $\tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)$ is required.

Once a sound invariance proof method has been designed it is desirable to check its completeness that is :

$$[\forall(\underline{s},\overline{s})\in S[\![P]\!]^2,\ (\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \Psi(\underline{s},\overline{s})]$$
$$\Rightarrow$$
$$[\exists \tilde{I}\in\tilde{A}[\![P]\!] \mid \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{I})]$$

by 2.3-2 this is equivalent to :

$$[\exists I\in A[\![P]\!] \mid C[\![P]\!](\varepsilon,\sigma)(\Psi)(I)] \Rightarrow [\exists \tilde{I}\in\tilde{A}[\![P]\!] \mid \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{I})]$$

or as above :

(2.4-2)

$$[\exists \rho[\![P]\!]\in(A[\![P]\!]\to\tilde{A}[\![P]\!]) \mid$$
$$\forall I\in A[\![P]\!],\ C[\![P]\!](\varepsilon,\sigma)(\Psi)(I) \Rightarrow \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\rho[\![P]\!](I))]$$

Hence the completeness check consists in first defining $\rho[\![P]\!]$ (i.e. how an assertion I of $A[\![P]\!]$ can be coded by an assertion $\rho[\![P]\!](I)$ of $\tilde{A}[\![P]\!]$) and

next proving that the verification condition $C[\![P]\!]$ applied to an invariant I implies the verification condition $\tilde{C}[\![P]\!]$ applied to the coding $\rho[\![P]\!](I)$ of I.

3. CONSTRUCTION OF A PARTIAL CORRECTNESS PROOF METHOD FOR PARALLEL PROGRAMS BY BACKWARD INDUCTION I^{-1}

Let $\phi \in ((V \to M) \to \{tt,ff\})$ and $\psi \in ((V \to M)^2 \to \{tt,ff\})$ be input and output specifications of a program P. The sets of entry or exit states are respectively characterized by :

$$\varepsilon(\underline{s}) = [\exists \underline{L} \in L, \underline{M} \in (V \to M) \mid \underline{s} = <\underline{L},\underline{M}> \wedge P \equiv \underline{L}:\alpha \wedge \phi(\underline{M})]$$

$$\sigma(\overline{s}) = [\exists \overline{L} \in L, \overline{M} \in (V \to M) \mid \overline{s} = <\overline{L},\overline{M}> \wedge P \equiv \alpha;\overline{L}:]$$

Define

$$\Psi(\underline{s},\overline{s}) = [\exists (\underline{L},\overline{L}) \in L^2,\ (\underline{M},\overline{M}) \in (V \to M)^2 \mid \underline{s} = <\underline{L},\underline{M}> \wedge \overline{s} = <\overline{L},\overline{M}> \wedge \psi(\underline{M},\overline{M})]$$

Then P is partially correct for ϕ and ψ if and only if Ψ is invariant for P with respect to ε and σ. Informally when P is executed starting with initial values $\underline{M}$ of the variables satisfying ϕ and this execution terminates with final values $\overline{M}$ of the variables then $\psi(\underline{M},\overline{M})$ must hold.

We will now design a partial correctness proof method for parallel programs as defined at paragraph 2.1. We will use the induction principle I^{-1} so that :

$$A[\![P]\!] = (S[\![P]\!]^2 \to \{tt,ff\})$$

$$C[\![P]\!](\varepsilon,\sigma)(\Psi)(I) = [C_1[\![P]\!](\sigma)(I) \wedge C_2[\![P]\!](\sigma)(I) \wedge C_3[\![P]\!](\varepsilon,\sigma)(\Psi)(I)]$$

where

$$C_1[\![P]\!](\sigma)(I) = [\forall \overline{s} \in S[\![P]\!], \sigma(\overline{s}) \Rightarrow I(\overline{s},\overline{s})]$$

$$C_2[\![P]\!](\sigma)(I) = [\forall (s,s',\overline{s}) \in S[\![P]\!]^3, (t[\![P]\!](s,s') \wedge I(s',\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow I(s,\overline{s})]$$

$$C_3[\![P]\!](\varepsilon,\sigma)(\Psi)(I) = [\forall (\underline{s},\overline{s}) \in S[\![P]\!]^2, (\varepsilon(\underline{s}) \wedge I(\underline{s},\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \Psi(\underline{s},\overline{s})]$$

3.1 SEQUENTIAL PROGRAMS

We first consider sequential programs and subgoal induction. This example is simple enough to illustrate our approach for designing invariance proof methods and has almost no other interest.

3.1.1 Choice of an Assertion Language and its Semantics

We choose :

$$\tilde{A}[\![CL]\!] = \prod_{L \in La[\![CL]\!]} ((V \to M)^2 \to \{tt, ff\})$$

in order to be able to associate a relation on memory states to each program point. The meaning of such a vector of relations is formally defined by the semantic function :

$$\tilde{\rho}[\![CL]\!] \in (\tilde{A}[\![CL]\!] \to A[\![CL]\!])$$

$$\tilde{\rho}[\![CL]\!](\tilde{I}) = \lambda(<L,M>,<\bar{L},\bar{M}>).[\tilde{I}(L)(M,\bar{M})]$$

Intuitively when control is at L, the relation $\tilde{I}(L)$ is true between the current memory state M and the (ultimate) memory state $\bar{M}$ (when the program halts at $\bar{L}$).

3.1.2 Construction of Sound Verification Conditions

We have to construct the verification condition $\tilde{C}[\![CL]\!]$ such that :

$$\tilde{C}[\![CL]\!](\varepsilon,\sigma)(\Psi)(\tilde{I}) \Rightarrow C[\![CL]\!](\varepsilon,\sigma)(\Psi)(\tilde{\rho}[\![CL]\!](\tilde{I}))$$

Since $C[\![CL]\!]$ is a conjunction of three conditions, we choose $\tilde{C}[\![CL]\!]$ to be also a conjunction of three conditions, each one satisfying the above soundness criterion.

- (3.1.2-1) Finalization :

$$\begin{aligned}
&C_1[\![CL]\!](\sigma)(\tilde{\rho}[\![CL]\!](\tilde{I})) \\
&= [\forall \bar{s} \in S[\![CL]\!],\ \sigma(\bar{s}) \Rightarrow \tilde{\rho}[\![CL]\!](\tilde{I})(\bar{s},\bar{s})] \\
&= [\forall \bar{L} \in La[\![CL]\!],\ \forall \bar{M} \in (V \to M),\ \sigma(<\bar{L},\bar{M}>) \Rightarrow \tilde{\rho}[\![CL]\!](\tilde{I})(<\bar{L},\bar{M}>,<\bar{L},\bar{M}>)] \quad \text{(Def. of } S[\![CL]\!]) \\
&= [\forall \bar{L} \in L,\forall \bar{M} \in (V \to M),\ (CL \equiv \alpha;\bar{L}:) \Rightarrow \tilde{I}(\bar{L})(\bar{M},\bar{M})] \quad \text{(Def. of } \sigma \text{ and } \tilde{\rho}[\![CL]\!]) \\
&= [\forall \bar{M} \in (V \to M),\ \tilde{I}(\bar{L})(\bar{M},\bar{M}) \quad \text{where } CL \equiv \alpha;\bar{L}] \\
&= \tilde{C}_1[\![CL]\!](\sigma)(\tilde{I})
\end{aligned}$$

Intuitively this verification condition states that the last assertion $\tilde{I}(\bar{L})$ encountered at program exit in any terminating execution must be true.

- (3.1.2-2) Induction :

$$\begin{aligned}
&C_2[\![CL]\!](\sigma)(\tilde{\rho}[\![CL]\!](\tilde{I})) \\
&= [\forall (s,s',\bar{s}) \in S[\![CL]\!]^3,\ (ts[\![CL]\!](s,s') \wedge \tilde{\rho}[\![CL]\!](\tilde{I})(s',\bar{s}) \wedge \sigma(\bar{s})) \Rightarrow \tilde{\rho}[\![CL]\!](\tilde{I})(s,\bar{s})]
\end{aligned}$$

$= [\forall L \in \mathcal{L} \mid CL \equiv \alpha L:\beta,\ \forall (M,\overline{M}) \in (V \to M)^2,$

$[\exists L' \in \mathcal{L} \mid \underline{cond}[\![CL]\!](L,L')(M) \wedge \tilde{I}(L')(\underline{succ}[\![CL]\!](L)(M),\overline{M})] \Rightarrow \tilde{I}(L)(M,\overline{M})]]$

(Def. of $S[\![CL]\!]$, σ, $\tilde{\rho}[\![CL]\!]$ and $ts[\![CL]\!]$)

$= \tilde{C}_2[\![CL]\!](\sigma)(\tilde{I})$

Intuitively this verification condition states that if an atomic program step starting from program point L with memory state M can go to an immediately following program point L' with memory state M' such that $M' = \underline{succ}[\![CL]\!](L)(M)$ then the subgoal assertion $\tilde{I}(L')(M',\overline{M})$ after this step must imply the subgoal assertion $\tilde{I}(L)(M,\overline{M})$ before this step. Notice that $\tilde{C}_2[\![CL]\!](\sigma)(\tilde{I})$ is a conjunction of verification conditions, one for each program label L. Depending upon the nature of the command which is designated by L and using the definitions of $\underline{cond}[\![CL]\!]$ and $\underline{succ}[\![CL]\!]$ we can detail the verification condition :

$[\exists L' \in \mathcal{L} \mid \underline{cond}[\![CL]\!](L,L')(M) \wedge \tilde{I}(L')(\underline{succ}[\![CL]\!](L)(M),\overline{M})] \Rightarrow \tilde{I}(L)(M,\overline{M})$

into subcases. For example, when L designates a while loop we get :

$[CL \equiv \alpha L:\underline{while}\ B\ \underline{do}\ L':\beta \wedge M \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = tt \wedge \tilde{I}(L')(M,\overline{M})]$

$\Rightarrow \tilde{I}(L)(M,\overline{M})$

and

$[CL \equiv \alpha L:\underline{while}\ B\ \underline{do}\ CL'\ \underline{od};L':\beta \wedge M \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = ff \wedge \tilde{I}(L')(M,\overline{M})]$

$\Rightarrow \tilde{I}(L)(M,\overline{M})$

- (3.1.2-3) Initialization :

$C_3[\![CL]\!](\varepsilon,\sigma)(\Psi)(\tilde{\rho}[\![CL]\!](\tilde{I}))$

$= [\forall (\underline{s},\overline{s}) \in S[\![CL]\!]^2,\ (\varepsilon(\underline{s}) \wedge \tilde{\rho}[\![CL]\!](\tilde{I})(\underline{s},\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \Psi(\underline{s},\overline{s})]$

$= [\forall (\underline{L},\overline{L}) \in \mathcal{L}^2, \forall (\underline{M},\overline{M}) \in (V \to M)^2,$

$(CL \equiv \underline{L}:\alpha \wedge \tilde{I}(\underline{L})(\underline{M},\overline{M}) \wedge CL \equiv \beta;\overline{L}:) \Rightarrow (\phi(\underline{M}) \Rightarrow \psi(\underline{M},\overline{M}))]$

(Def. of $S[\![CL]\!]$, ε, $\tilde{\rho}[\![CL]\!]$, σ and Ψ)

$= [(\phi(\underline{M}) \wedge \tilde{I}(\underline{L})(\underline{M},\overline{M})) \Rightarrow \psi(\underline{M},\overline{M})$ where $CL \equiv \underline{L}:\alpha]$

$= \tilde{C}_3[\![CL]\!](\varepsilon,\sigma)(\Psi)(\tilde{I})$

Intuitively, the input specification and the subgoal assertion on program entry must imply the output specification.

3.1.3 Completeness Check

Let us define $\rho[\![CL]\!] \in (A[\![CL]\!] \to \tilde{A}[\![CL]\!])$ which specifies how any induction hypothesis $I \in A[\![CL]\!]$ can be coded by a vector of invariants $\tilde{I}(L)$ associated with each point L of program CL :

$\rho[\![CL]\!](I) = \lambda L.[\lambda(M,\overline{M}).[\forall \overline{L}\in\mathcal{L},\ (CL\equiv\alpha;\overline{L}:) \Rightarrow I(<L,M>,<\overline{L},\overline{M}>)]$

Since we have chosen $\tilde{C}[\![CL]\!](\varepsilon,\sigma)(\Psi) = C[\![CL]\!](\varepsilon,\sigma)(\Psi)\circ\rho[\![CL]\!]$ the completeness check 2.4-2 is :

$\forall I\in A[\![P]\!], \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\rho[\![P]\!](I)) = C[\![P]\!](\varepsilon,\sigma)(\Psi)(\tilde{\rho}[\![P]\!](\rho[\![P]\!](I))) \Leftarrow C[\![P]\!](\varepsilon,\sigma)(\Psi)(I)$

since $C[\![P]\!](\varepsilon,\sigma)(\Psi)$ is isotone and $I \Rightarrow \tilde{\rho}[\![P]\!](\rho[\![P]\!](I))$.

This result disproves Misra[78]'s statement that "subgoal induction is not guaranteed to prove a correct program correct".

3.1.4 Summary of the Verification Conditions for the Partial Correctness Proof of a Sequential Program by Backward Induction

We use informal mnemonic notations. $\underline{x}$ ($\overline{x}$) is the vector of initial (final) program variables and P_i the assertion associated with program point l_i.

- Finalization :

  $\alpha;l_1$: $\quad \forall \overline{x},\ P_1(\underline{x},\overline{x})$

- Induction :

  . Null command :

  αl_1 : skip; l_2 : β $\quad P_2 \Rightarrow P_1$

  . Assignment command :

  αl_1 : V := E; l_2 : β $\quad P_2[E/V] \Rightarrow P_1$

  . Conditional command :

  αl_1 : if B then
  l_2 :
  β
  l_3 :
  else
  l_4 :
  γ
  l_5 :
  fi;
  l_6 : δ

  $((P_2 \wedge B)\vee(P_4 \wedge \neg B)) \Rightarrow P_1$

  $P_6 \Rightarrow (P_3 \wedge P_5)$

  . Iteration command :

  αl_1 : while B do
  l_2 :
  β
  l_3 :
  od;
  l_4 : γ

  $((P_2 \wedge B)\vee(P_4 \wedge \neg B)) \Rightarrow (P_1 \wedge P_3)$

- Initialization :

  l_1 : α $\quad \forall \underline{x},\overline{x},\ (\phi(\underline{x}) \wedge P_1(\underline{x},\overline{x})) \Rightarrow \psi(\underline{x},\overline{x})$

3.1.5 Proving Other Invariance Properties of Sequential Programs by Backward Induction

Morris & Wegbreit[77] assert that "a drawback of subgoal induction is that it cannot be used to prove invariants about non-terminating programs". The problem is to prove that when execution of a program P is started with an initial memory state $\underline{M}$ satisfying some input specification ϕ and reaches any program point L with memory state M then the invariant $\psi(L)(M)$ must hold. Formally it must be proved that :

$$[\forall(\underline{s},\overline{s})\in S[\![CL]\!]^2,\ (\varepsilon(\underline{s}) \wedge ts[\![CL]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \Psi(\underline{s},\overline{s})]$$

where :

$$\varepsilon(\underline{s}) = [\exists \underline{L}\in\mathcal{L},\ \underline{M}\in(V\to M) \mid \underline{s}=<\underline{L},\underline{M}> \wedge CL\equiv\underline{L}:\alpha \wedge \phi(\underline{M})]$$

$$\sigma(\overline{s}) = tt$$

$$\Psi(\underline{s},\overline{s}) = [\exists \overline{L}\in\mathcal{L},\ \overline{M}\in(V\to M) \mid \overline{s}=<\overline{L},\overline{M}> \wedge \psi(\overline{L})(\overline{M})]$$

The above definitions of σ and Ψ differ from the case of partial correctness hence of subgoal induction, so that Morris & Wegbreit 's remark is justified for subgoal induction but not for all backward proof methods.

In order to be complete, let us design a backward method for proving invariants about programs. Ψ is now an assertion upon final states ("final" meaning "any" in that case). According to paragraph 2.2.6-2 we see that the contrapositive backward induction principle $\overline{\iota}$ should be used. The construction of the proof method follows the previous lines except that one now associates a (contrapositive) assertion on the current memory state with each program point (and no longer a relation between the current and exit memory states). The verification conditions remain similar to the ones of paragraph 3.1.4 for the induction step 2.2.4-3(b) whereas 2.2.4-3(a) and 2.2.4-3(c) lead to :

- Finalization :

  $\alpha l_i:\beta$ $\qquad\qquad \neg\psi_i \Rightarrow P_i$

  (l_i is any label of the program)

- Initialization :

  $l_1:\alpha$ $\qquad\qquad \phi \Rightarrow \neg P_1$

Example : Let us illustrate this method using Morris & Wegbreit[77] simple (counter) example, which consists in proving that x is positive in the following program :

l_1: x:=1;
l_2: while true do
l_3: x:=x+1;
l_4: od;
l_5:

We choose $\phi(x)=tt$, $\psi_1(x)=tt$, $\psi_i(x)=[x>0]$, $i=2,\ldots,5$. The verification conditions are :

- Finalization :

$\neg\psi_i \Rightarrow P_i \quad i=1,\ldots,5$

- Induction :

$P_2[1/x] \Rightarrow P_1$
$((P_3 \wedge \underline{true}) \vee (P_5 \wedge \neg\underline{true})) \Rightarrow (P_2 \wedge P_4)$
$P_4[x+1/x] \Rightarrow P_3$

- Initialization :

$\phi \Rightarrow \neg P_1$

These verification conditions are obviously verified by $P_i=\neg\psi_i$, $i=1,\ldots,5$. □

3.2 PARALLEL PROGRAMS

3.2.1 Choice of an Assertion Language and its Semantics

In order to express a relationship between current and final states, we associate a relation with each program point not within a conditional critical section. At entry and exit program points it is a relation between the current and final memory states. At each program point of each process it is a relationship between the current values of the program location counters of the other processes, the current memory state and the final memory state. Therefore for a program :

$P \equiv \underline{L}$:cobegin $CL_1 \| \ldots \| CL_i \| \ldots \| CL_n$ coend;$\overline{L}$:

we choose :

$$\tilde{A}[\![P]\!] = \{\tilde{I} \mid [\tilde{I}(\underline{L}) \in ((V \to M)^2 \to \{tt,ff\})]$$
$$\wedge [\forall i \in [1,n],\ \forall L \in La[\![CL_i]\!],$$
$$\tilde{I}(i)(L) \in (((\prod_{\substack{j=1 \\ j\neq i}}^{n} La[\![CL_j]\!]) \times (V \to M) \times (V \to M)) \to \{tt,ff\})]$$
$$\wedge [\tilde{I}(\overline{L}) \in ((V \to M)^2 \to \{tt,ff\})]\}$$

The meaning of such a vector $\tilde{I}$ of relationships is formally defined by the semantic function $\tilde{\rho}[\![P]\!]$. We have $\tilde{\rho}[\![P]\!](\tilde{I})=I$ where by cases :

$I(\langle \underline{L},\underline{M}\rangle,\langle \overline{L},\overline{M}\rangle) = \tilde{I}(\underline{L})(\underline{M},\overline{M})$

$$I(\langle L_1,\ldots,L_r,M\rangle,\langle\overline{L},\overline{M}\rangle) = \bigwedge_{i=1}^{n} \tilde{I}(L_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M,\overline{M})$$

$$I(\langle\overline{L},M\rangle,\langle\overline{L},\overline{M}\rangle) = \tilde{I}(\overline{L})(M,\overline{M})$$

($\tilde{\rho}[\![P]\!](\tilde{I})(s,\overline{s})$ need not be defined when $\overline{s}$ is not a final state).

3.2.2 Construction of Sound Verification Conditions

The finalization and initialization conditions are similar to 3.1.2-1 and 3.1.2-2 and present no difficulties. The main point consists in finding $\tilde{C}_2[\![P]\!]$ such that :

$$\tilde{C}_2[\![P]\!](\sigma)(\tilde{I}) \Rightarrow C_2[\![P]\!](\sigma)(\tilde{\rho}[\![P]\!](\tilde{I}))$$

where :

$$C_2[\![P]\!](\sigma)(\tilde{\rho}[\![P]\!](\tilde{I}))$$

$$= [\forall(s,s',\overline{s})\in S[\![P]\!]^3 ; (t[\![P]\!](s,s') \wedge \tilde{\rho}[\![P]\!](\tilde{I})(s',\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \tilde{\rho}[\![P]\!](\tilde{I})(s,\overline{s})]$$

(3.2.2-1)

$$= [\forall(s,s')\in S[\![P]\!]^2,\ \overline{M}\in(V\to M),$$

$$(t[\![P]\!](s,s') \wedge \tilde{\rho}[\![P]\!](\tilde{I})(s',\langle\overline{L},\overline{M}\rangle) \wedge P\equiv\alpha;\overline{L}:) \Rightarrow \tilde{\rho}[\![P]\!](\tilde{I})(s,\langle\overline{L},\overline{M}\rangle)]$$

(Def. of $S[\![P]\!]$ and σ)

We split this verification condition into a conjunction of cases according to the possible forms of s which are defined by $S[\![P]\!]$:

Case 1 : $s=\langle\underline{L},\underline{M}\rangle$ where $P\equiv\underline{L}:\alpha$ and $\underline{M}\in(V\to M)$

Then according to the definition (2.1.2.2-1) of $t[\![P]\!]$, s' is necessarily of the form $\langle(L_1,\ldots,L_n),\underline{M}\rangle$ where $P\equiv\underline{L}:\underline{\text{cobegin}}\ L_1:\alpha_1\|\ldots\|L_n:\alpha_n\ \underline{\text{coend}};\overline{L}:$ so that by definition of $\tilde{\rho}[\![P]\!]$ the verification condition 3.2.2-1 is in that case equal to :

$$\bigwedge_{i=1}^{n} \tilde{I}(L_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,\underline{M},\overline{M}) \Rightarrow \tilde{I}(\underline{L})(\underline{M},\overline{M})$$

Intuitively, the conjunction of entry assertions of each process must imply the entry assertion of the program.

Case 2 : $s=\langle\overline{L},\overline{M}\rangle$ where $P\equiv\alpha;\overline{L}:$ and $\overline{M}\in(V\to M)$

Since an exit state has no possible successor, $t[\![P]\!](\langle\overline{L},\overline{M}\rangle,s')$ is false for any $s'\in S[\![P]\!]$ and condition 3.2.2-1 is trivially verified in that case.

Case 3 : $s=\langle(L_1,\ldots,L_n),M\rangle$

where $P\equiv\underline{L}:\underline{\text{cobegin}}\ CL_1\|\ldots\|CL_n\ \underline{\text{coend}};\overline{L}:$, $L_i\in La[\![CL_i]\!]$ and $M\in(V\to M)$

We distinguish two subcases according as the possible form of s' (by definition of $t[\![P]\!]$, s' cannot be an entry state).

Case 3.1 : $s'=<\overline{L},\overline{M}>$

According to the definition (2.1.2.2-3) of $t[\![P]\!]$ we necessarily have $M=\overline{M}$ and $P \equiv \underline{L}:\underline{cobegin}\ \alpha_1;L_1:\| \ldots \|\alpha_n;L_n:\underline{coend};\overline{L}$: so that by definition of $\tilde{\rho}[\![P]\!](\tilde{I})$ the verification condition 3.2.2-1 is in that case equal to :

$$\tilde{I}(\overline{L})(\overline{M},\overline{M}) \Rightarrow \bigwedge_{i=1}^{n} \tilde{I}(L_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,\overline{M},\overline{M})$$

Intuitively the exit assertion of the program must imply the exit assertion of each constituent process.

Case 3.2 : $s'=<(L'_1,\ldots,L'_n),M'>$ where $L'_i \in La[\![CL_i]\!]$ and $M' \in (V \to M)$.

According to the definition (2.1.2.2-2) of $t[\![P]\!]$ we split this case into a conjunction of two subcases according as the transition from s to s' corresponds to the execution of an *await* command or not.

Case 3.2.1 : Transition not corresponding to an *await* command.

By definition of $t[\![P]\!]$ and $\tilde{\rho}[\![P]\!]$, formula 3.2.2-1 equals :

$$\begin{aligned}
&[[\exists i \in [1,n] \mid P \equiv \underline{L}:\underline{cobegin}\ CL_1 \| \ldots \| CL_i \| \ldots \| CL_n\ \underline{coend};\overline{L}: \\
&\quad \wedge\ (\forall j \in [1,n]-\{i\},\ L'j=Lj) \wedge ts[\![CL_i]\!](<L_i,M>,<L'_i,M'>) \\
&\quad \wedge \bigwedge_{j=1}^{n} \tilde{I}(L'_j)(L'_1,\ldots,L'_{j-1},L'_{j+1},\ldots,L'_n,M',\overline{M})] \\
&\Rightarrow \bigwedge_{k=1}^{n} \tilde{I}(L_k)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M,\overline{M})\ ]
\end{aligned}$$

This condition can be split into a conjunction of verification conditions corresponding to each process CL_i, $i=1,\ldots,n$:

$$\begin{aligned}
&[[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M) \\
&\quad \wedge \tilde{I}(L'_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,\underline{succ}[\![CL_i]\!](L_i)(M),\overline{M}) \\
&\quad \wedge \bigwedge_{\substack{j=1 \\ j\neq i}}^{n} \tilde{I}(L_j)(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_{j-1},L_{j+1},\ldots,L_n,\underline{succ}[\![CL_i]\!](L_i)(M),\overline{M})] \\
&\Rightarrow \bigwedge_{k=1}^{n} \tilde{I}(L_k)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M,\overline{M})]
\end{aligned}$$

Again we split this condition into subcases depending on k :

Case 3.2.1.1 : Sequential proof (k=i)

$$\begin{aligned}
&[[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M) \wedge \tilde{I}(L'_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,\underline{succ}[\![CL_i]\!](L_i)(M),\overline{M}) \\
&\quad \wedge\ \underline{context}(i,i)] \Rightarrow \tilde{I}(L_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M,\overline{M})
\end{aligned}$$

where

$$\underline{context}(i,k) = \bigwedge_{\substack{j=1 \\ j\neq i\ j\neq k}}^{n} \tilde{I}(L_j)(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_{j-1},L_{j+1},\ldots,L_n,\underline{succ}[\![CL_i]\!](L_i)(M),\overline{M})$$

This verification condition corresponds to the case of sequential proofs (see 3.1.2-2) except for the term $\underline{context}(i,i)$. It means that if the assertion

$\tilde{I}(L'_i)$ is true after going from L_i to L'_i, the assertion $\tilde{I}(L_i)$ had to be true before going from L_i to L'_i. Moreover, the term context(i,i) states that in the proof we can use all information available about the other processes CL_j, $j \neq i$ before the transition. In order to have sequential proofs which are similar in the case of sequential or parallel programs we can neglect the term context(i,i) and choose the stronger verification condition :

$$[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M) \wedge \tilde{I}(L'_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,\underline{succ}[\![CL_i]\!](L_i)(M),\overline{M})]$$
$$\Rightarrow \tilde{I}(L_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M,\overline{M})$$

Notice that this simplification is sound since the above verification condition implies the original one. The completeness proof will show that this is also complete, since intuitively the information context(i,i) can be, if necessary, incorporated in each $\tilde{I}(L'_i)$ at each point L'_i of each process CL_i.

Case 3.2.1.2 : Interference freeness check ($k \neq i$)

For $k \in [1,n]-\{i\}$, we must prove :

$$[[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M)$$
$$\wedge\ \tilde{I}(L'_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,\underline{succ}[\![CL]\!](L_i)(M),\overline{M})$$
$$\wedge\ \tilde{I}(L_k)(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_{k-1},L_{k+1},\ldots,L_n,\underline{succ}[\![CL]\!](L_i)(M),\overline{M})$$
$$\wedge\ \underline{context}(i,k)] \Rightarrow \tilde{I}(L_k)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M,\overline{M})\ ]$$

Intuitively the assertions $\tilde{I}(L_k)$ in a process CL_k must not be invalidated by execution of commands in other processes CL_i, $i \neq k$. As above, it is sound (and complete) to neglect the term context(i,k).

We could further detail the above verification conditions depending upon the nature of the command designated by L_i. Since this is simple, we only give the result at paragraph 3.2.4.

Case 3.2.2 : Transition corresponding to an await command.

Formula 3.2.2-1 states that whenever :

$P \equiv \underline{L}:\underline{cobegin}\ CL_1 \| \ldots \| CL_i \| \ldots \| CL_n\ \underline{coend};\overline{L}:$

$CL_i \equiv \alpha L_i:\underline{await}\ B\ \underline{then}\ CL\ \underline{end};L'_i:\beta$

$CL \equiv \underline{L}_1:\gamma;\overline{L}_2:$

we must prove :

$$\underline{await\text{-}proof}(ts[\![CL]\!]^*(<\underline{L}_1,M>,<\overline{L}_2,M'>))$$

where :

$$\underline{\text{await-proof}}(P) =$$

$$[[M \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = tt \wedge P \wedge \tilde{I}(L'_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M',\bar{M})$$

$$\wedge \bigwedge_{\substack{j=1 \\ j\neq i}}^{n} \tilde{I}(L_j)(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_{j-1},L_{j+1},\ldots,L_n,M',\bar{M})]$$

$$\Rightarrow \bigwedge_{k=1}^{n} \tilde{I}(L_k)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M,\bar{M})]$$

As it is the case for all invariance proofs, it is not always necessary to exactly characterize the relationship $ts[\![CL]\!]^*(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle)$ between the input and output states of the body CL of the critical section. An approximation Ψ can be used instead because the above formula is equal to :

$$[\exists \Psi \in (S[\![CL]\!]^2 \to \{tt,ff\}) \mid$$

$$\wedge\ [ts[\![CL]\!]^*(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle) \Rightarrow \Psi(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle)]$$

$$[\underline{\text{await-proof}}(\Psi(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle))]]$$

Since we have to prove that Ψ is invariant, we can apply theorem 2.2.3-1 and we get :

$$[\exists \Psi \in (S[\![CL]\!]^2 \to \{tt,ff\}) \mid$$

$$[\exists J \in (S[\![CL]\!]^2 \to \{tt,ff\}) \mid$$

$$(\forall M' \in (V \to M), J(\langle \bar{L}_2,M'\rangle,\langle \bar{L}_2,M'\rangle))$$

$$\wedge\ (\forall (L_1,L_2) \in L\alpha[\![CL]\!]^2,\ \forall (M_1,M_2,M') \in (V \to M)^3,$$

$$(ts[\![CL]\!](\langle L_1,M_1\rangle,\langle L_2,M_2\rangle) \wedge J(\langle L_2,M_2\rangle,\langle \bar{L}_2,M'\rangle)) \Rightarrow J(\langle L_1,M_1\rangle,\langle \bar{L}_2,M'\rangle))$$

$$\wedge\ (\forall (M,M') \in (V \to M)^2, J(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle) \Rightarrow \Psi(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle))]$$

$$\wedge\ [\underline{\text{await-proof}}(\Psi(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle))]]$$

Simplifying by elimination of Ψ we get equivalently :

$$[\exists J \in (S[\![CL]\!]^2 \to \{tt,ff\}) \mid$$

$$[\forall M' \in (V \to M),\ J(\langle \bar{L}_2,M'\rangle,\langle \bar{L}_2,M'\rangle)]$$

$$\wedge\ [\forall (L_1,L_2) \in \{L \mid CL \Xi \alpha L:\beta\}^2,\ \forall (M_1,M_2,M') \in (V \to M)^3,$$

$$(ts[\![CL]\!](\langle L_1,M_1\rangle,\langle L_2,M_2\rangle) \wedge J(\langle L_2,M_2\rangle,\langle \bar{L}_2,M'\rangle)) \Rightarrow J(\langle L_1,M_1\rangle,\langle \bar{L}_2,M'\rangle)]$$

$$\wedge\ [\underline{\text{await-proof}}(J(\langle \underline{L}_1,M\rangle,\langle \bar{L}_2,M'\rangle))]]$$

Applying 3.1.2-1 and 3.1.2-2 this is equivalent to :

$$[\exists \tilde{J} \in \prod_{L \in \{L \mid CL \Xi \alpha L:\beta\}} ((V \to M)^2 \to \{tt,ff\}) \mid$$

$$[\forall M' \in (V \to M),\ \tilde{J}(\bar{L}_2)(M',M')]$$

$$\wedge\ [\forall L \in L \mid CL \Xi \alpha L:\beta,\ \forall (M,M') \in (V \to M)^2,$$

$$[\exists L' \in L \mid \underline{\text{cond}}[\![CL]\!](L,L')(M) \wedge \tilde{J}(L')(\underline{\text{succ}}[\![CL]\!](L)(M),M')] \Rightarrow \tilde{J}(L)(M,M')]]$$

$$\wedge\ [\underline{\text{await-proof}}(\tilde{J}(\underline{L}_1)(M,M'))]]$$

Thus the verification condition for await commands has been divided into two subproblems. Firstly, the body CL of the conditional critical section has to be analyzed independently of its context in order to invent at each point L a relationship $\tilde{J}(L)(M,M')$ between the current and final memory states M and M'.

$\tilde{J}$ must be proved invariant using the backward proof method recapitulated at paragraph 3.1.4. This is followed by a proof await-proof($\tilde{J}(\underline{L}_1)(M,M')$) where the await command is considered as atomic and its semantics is defined by $\tilde{J}(\underline{L}_1)(M,M')$. As for other commands this proof can be divided into :

. a sequential proof (omitting term context(i,i)) :

$$[[M \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = tt \wedge \tilde{J}(\underline{L}_1)(M,M') \wedge \tilde{I}(L'_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M',\overline{M})] \Rightarrow \tilde{I}(L_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M,\overline{M})]$$

. and interference freeness checks (omitting term context(i,k)) :

$$\forall k \in [1,n]-\{i\},$$
$$[[M \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = tt \wedge \tilde{J}(\underline{L}_1)(M,M') \wedge \tilde{I}(L'_i)(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M',\overline{M}) \wedge \tilde{I}(L_k)(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_{k-1},L_{k+1},\ldots,L_n,M',\overline{M})] \Rightarrow \tilde{I}(L_k)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M,\overline{M})]$$

3.2.3 Completeness Check

Let us define $\rho[\![P]\!] \in (A[\![P]\!] \to \tilde{A}[\![P]\!])$ which specifies how any induction hypothesis $I \in A[\![P]\!]$ can be coded by a vector of invariants $\tilde{I}(L)$ associated with each point L of program P such that :

$P \equiv \underline{L}:\underline{\text{cobegin}}\ CL_1 \| \ldots \| CL_i \| \ldots \| CL_n\ \underline{\text{coend}};\overline{L}:$

by cases :

- $\rho[\![P]\!](I)(\underline{L}) = \lambda(\underline{M},\overline{M}).[\forall \overline{L} \in L, (P \equiv \alpha;\overline{L}) \Rightarrow I(<\underline{L},\underline{M}>,<\overline{L},\overline{M}>)]$
- $\forall i \in [1,n],\ \forall L \in L\alpha[\![CL_i]\!],$
  $\rho[\![P]\!](I)(i)(L) = \lambda(L_1,\ldots,L_{i-1},L_{i+1},\ldots,L_n,M,\overline{M}).[\forall \overline{L} \in L,$
  $(P \equiv \alpha;\overline{L}:) \Rightarrow I(<(L_1,\ldots,L_{i-1},L,L_{i+1},\ldots,L_n),M>,<\overline{L},\overline{M}>)]$
- $\rho[\![P]\!](I)(\overline{L}) = \lambda(M,\overline{M}).[I(<\overline{L},M>,<\overline{L},\overline{M}>]$

The completeness proof shows that it is complete to omit the terms context(i,i) in the sequential proof and context(i,k) in the completeness check. It also shows that completeness checks could have been further simplified by omitting the term $\tilde{I}(L_K)$ on the left hand side of the implication. (However in practice the term is important since $\tilde{I}(L'_i)$ can be given a simpler form).

We have to prove that :

$$\forall I \in A[\![P]\!],\ C[\![P]\!](\varepsilon,\sigma)(\Psi)(I) \Rightarrow \tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\rho[\![P]\!](I))$$

The proof follows the cases considered at paragraph 3.2.2. We only treat case 3.2.1 (since case 3.2.2 is similar whereas the proof for the remaining cases is like the one given at paragraph 3.1.2).

The term corresponding to $\tilde{C}[\![P]\!](\varepsilon,\sigma)(\Psi)(\rho[\![P]\!](I))$ in case 3.2.1 is the conjunction of a sequential proof and interference freeness checks :

$$[[[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M)\wedge(\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L}:)\Rightarrow I(<(L_1,\ldots L_{i-1},L'_i,L_{i+1},\ldots,L_n),$$
$$\underline{succ}[\![CL_i]\!](L_i)(M)>,<\overline{L},\overline{M}>))]\Rightarrow(\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L}:)\Rightarrow I(<(L_1,\ldots,L_n),M>,<\overline{L},\overline{M}>)]$$

$$\wedge_{\substack{k=1\\k\neq i}}^{n}[[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M)$$
$$\wedge(\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L}:)\Rightarrow I(<(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_n),\underline{succ}[\![CL_i]\!](L_i)(M)>,<\overline{L},\overline{M}>))$$
$$\wedge(\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L}:)\Rightarrow I(<(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_n),\underline{succ}[\![CL_i]\!](L_i)(M)>,<\overline{L},\overline{M}>))]$$
$$\Rightarrow(\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L}:)\Rightarrow I(<(L_1,\ldots,L_n),M>,<\overline{L},\overline{M}>))]]$$

which is implied by :

$$[[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M)$$
$$\wedge(\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L})\Rightarrow I(<(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_n),\underline{succ}[\![CL_i]\!](L_i)(M)>,<\overline{L},\overline{M}>))]$$
$$\Rightarrow(\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L}:)\Rightarrow I(<(L_1,\ldots,L_n),M>,<\overline{L},\overline{M}>))]$$

which is implied by :

$$[\forall\overline{L}\in\mathcal{L},(P\equiv\alpha;\overline{L})\Rightarrow$$
$$[[\underline{cond}[\![CL_i]\!](L_i,L'_i)(M)\wedge I(<(L_1,\ldots,L_{i-1},L'_i,L_{i+1},\ldots,L_n),\underline{succ}[\![CL_i]\!](L_i)(M)>,<\overline{L},\overline{M}>)]$$
$$\Rightarrow I(<(L_1,\ldots,L_n),M>,<\overline{L},\overline{M}>)]]$$

which is the term of $C[\![P]\!](\varepsilon,\sigma)(\Psi)(I)$ corresponding to the transition considered in case 3.2.1 of paragraph 3.2.2.

3.2.4 Summary of the Verification Conditions for the Partial Correctness Proof of a Parallel Program by Backward Induction

$P\equiv l_0:\underline{cobegin}\ CL_1\|\ldots\|CL_{i-1}\|\ldots\|\alpha l_{ij}:\beta\|CL_{i+1}\|\ \ldots\|CL_n\ \underline{coend};l_1:$

The entry assertion $P_0(\underline{x},\overline{x})$ relates the initial and final values $\underline{x}$ and $\overline{x}$ of variables.

The assertion $P_{ij}(c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_n,x,\overline{x})$ associated with point j of process i relates the program location counters $c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_n$ of the other processes, the current value x of the variables and their final value $\overline{x}$.

The exit assertion $P_1(\overline{x},\overline{x})$ is on the final value $\overline{x}$ of the variables.

- *Sequential Proof*

The verification conditions are the same than for sequential programs (3.1.4) plus

. Finalization of parallelism :

$$\alpha\underline{cobegin}\ \beta_1;l_1:\|\ldots\|\beta_n;l_n:\underline{coend};l_f:$$
$$P_f(x,\overline{x})\Rightarrow\mathop{\wedge}_{i=1}^{n}P_i(l_1,\ldots,l_{i-1},l_{i+1},\ldots,l_n,x,\overline{x})$$

. Await command :

αl_{i_1} : await B then
l_{i_2} : β
l_{i_3} :
end;
l_{i_4} : γ

At each point l_{ij} of the body $l_{i_2}:\beta l_{i_3}:$, an assertion $P_{ij}(x,x')$ relates the current value x at l_{ij} to the final value x' at l_{i_3} of the variables. The corresponding verification conditions are those of sequential programs (except for the initialization rule).

$$[B(x) \wedge P_{i_2}(x,x') \wedge P_{i_4}(c_1,\dots,c_{i-1},c_{i+1},\dots,c_n,x',\overline{x})] \Rightarrow P_{i_1}(c_1,\dots,c_{i-1},c_{i+1},\dots,c_n,x,\overline{x})$$

. Initialization of parallelism :

l_0 : cobegin $l_1:\alpha_1 \| \dots \| l_n:\alpha_n$ coend β

$$[\bigwedge_{i=1}^{n} P_i(l_1,\dots,l_{i-1},l_{i+1},\dots,l_n,x,\overline{x})] \Rightarrow P_0(x,\overline{x})$$

- *Interference freeness checks*

For each label l_{kj} of each process CL_k, $k \in [1,n]-\{i\}$

. Null command :

αl_{i_1} : skip; $l_{i_2}:\beta$

$$(P_{i_2}[l_{kj}/c_k] \wedge P_{kj}[l_{i_2}/c_i]) \Rightarrow P_{kj}[l_{i_1}/c_i]$$

. Assignment command :

αl_{i_1} : V := E; $l_{i_2}:\beta$

$$(P_{i_2}[l_{kj}/c_k, E/V] \wedge P_{kj}[l_{i_2}/c_i, E/V]) \Rightarrow P_{kj}[l_{i_1}/c_i]$$

. Conditional command :

αl_{i_1} : if B then
l_{i_2} : β
l_{i_3} :
else
l_{i_4} : γ
l_{i_5} :
fi;
l_{i_6} : δ

$$((P_{i_2}[l_{kj}/c_k] \wedge B \wedge P_{kj}[l_{i_2}/c_i]) \vee (P_{i_4}[l_{kj}/c_k] \wedge \neg B \wedge P_{kj}[l_{i_4}/c_i])) \Rightarrow P_{kj}[l_{i_1}/c_i]$$

$$(P_{i_6}[l_{kj}/c_k] \wedge P_{kj}[l_{i_6}/c_i]) \Rightarrow (P_{kj}[l_{i_3}/c_i] \wedge P_{kj}[l_{i_5}/c_i])$$

. Iteration command :

αl_{i_1} : while B do
l_{i_2} :
β
l_{i_3} :
od;
l_{i_4} : γ

$((P_{i_2}[l_{kj}/c_k] \wedge B \wedge P_{kj}[l_{i_2}/c_i]) \vee (P_{i_4}[l_{kj}/c_k] \wedge \neg B \wedge P_{kj}[l_{i_4}/c_i]))$
$\Rightarrow (P_{kj}[l_{i_1}/c_i] \wedge P_{kj}[l_{i_3}/c_i])$

. Await command :

αl_{i_1} : await B then
l_{i_2} :
β
l_{i_3} :
end;
l_{i_4} : γ

$[B(x) \wedge P_{i_2}(x,x')$
$\wedge P_{i_4}(c_1,\ldots,c_{k-1},l_{kj},c_{k+1},\ldots,c_{i-1},c_{i+1},\ldots,c_n,x',\bar{x})$
$\wedge P_{kj}(c_1,\ldots,c_{k-1},c_{k+1},\ldots,c_{i-1},l_{i_4},c_{i+1},\ldots,c_n,x',\bar{x})]$
$\Rightarrow P_{kj}(c_1,\ldots,c_{k-1},c_{k+1},\ldots,c_{i-1},l_{i_1},c_{i+1},\ldots,c_n,x,\bar{x})$

3.2.5 Examples

Example 3.2.5.1 : A very simple example taken from Owicki & Gries [76b].
We have to prove that the following program :

```
0:  cobegin
       11: await true then
              12:
              13: x:=x+1;
       14: end;
    ||
       21: await true then
              22:
              23: x:=x+1;
       24: end;
    3: coend;
```

is partially correct with respect to :

$\phi(\underline{x}) = (\underline{x}=0)$
$\psi(\underline{x},\bar{x}) = (\bar{x}=2)$

The verification conditions are the following :

- Finalization :

$P_3(\bar{x},\bar{x})$
$P_3(x,\bar{x}) \Rightarrow [P_{24}(14,x,\bar{x}) \wedge P_{14}(24,x,\bar{x})]$

- Sequential proof for process 2 :

  $P_{23}(x',x')$

  $P_{23}(x+1,x') \Rightarrow P_{22}(x,x')$

  $[P_{22}(x,x') \wedge P_{24}(c_1,x',\overline{x})] \Rightarrow P_{21}(c_1,x,\overline{x})$

- Absence of interference of process 2 with the proof of process 1 :

  $[P_{22}(x,x') \wedge P_{24}(14,x',\overline{x}) \wedge P_{14}(24,x',\overline{x})] \Rightarrow P_{14}(21,x,\overline{x})$

  $[P_{22}(x,x') \wedge P_{24}(11,x',\overline{x}) \wedge P_{11}(24,x',\overline{x})] \Rightarrow P_{11}(21,x,\overline{x})$

- Likewise, we have a sequential proof for process 1 and a check that process process 1 does not interfere with the proof of process 2.

- Initialization :

  $[P_{11}(21,x,\overline{x}) \wedge P_{21}(11,x,\overline{x})] \Rightarrow P_0(x,\overline{x})$

  $[\phi(x) \wedge P_0(x,\overline{x})] \Rightarrow \psi(x,\overline{x})$

The proof outline is the following :

$\{(x=0 \wedge x=\overline{x}-2) \Rightarrow (\overline{x}=2)\}$

0: $\{x=\overline{x}-2\}$

cobegin

11: $\{(c_2=21 \wedge x=\overline{x}-2) \vee (c_2=24 \wedge x=\overline{x}-1)\}$

await true then

12: $\{x=x'-1\}$

x:=x+1;

13: $\{x=x'\}$

end;

14: $\{(c_2=21 \wedge x=\overline{x}-1) \vee (c_2=24 \wedge x=\overline{x})\}$

||

21: $\{(c_1=11 \wedge x=\overline{x}-2) \vee (c_1=14 \wedge x=\overline{x}-1)\}$

await true then

22: $\{x=x'-1\}$

x:=x+1;

23: $\{x=x'\}$

end;

24: $\{(c_1=11 \wedge x=\overline{x}-1) \vee (c_1=14 \wedge x=\overline{x})\}$

coend;

3: $\{x=\overline{x}\}$ □

Example 3.2.5.2 : Factorial

In order to compute f=n! where n>1, one can imagine a program with two processes, the first process computing the product $f_1=1*2*3*...$ of small integers and the second process computing the product $f_2=n*(n-1)*(n-2)*...$ of large integers. If the two processes stop at some limits l and l+1 respectively, we have $f=f_1*f_2=(1*...*l)*((l+1)*...*n)$. Since the relative speed of the processes is unknown the limit l cannot be fixed in advance without risking that one process be inactive while the other terminates its computation. This leads to a program such as :

```
n1:=1; n2:=n;
cobegin
      f1:=1;while (n1+1)<n2 do n1:=n1+1;f1:=f1*n1; od;
   ||
      f2:=n;while (n1+1)<n2 do n2:=n2-1;f2:=f2*n2; od;
coend;
f:=f1*f2;
```

However this program is incorrect since when $n_1=1$ and $n_2=1+2$ both processes might test at the same time that $(n_1+1)<n_2$ which leads to $n_1=n_2=1+1$ and $f=n!*(1+1)$. One solution is to synchronize the processes in order to avoid this interference, but synchronizations are costly. The other solution is to take the phenomenon into account, a posteriori correcting its effect when necessary. This leads to the following asynchronous program :

```
0:
1: n1:=1; n2:=n;
   cobegin
          11:  f1:=1;
          12:  while (n1+2)<n2 do
          13:  n1:=n1+1;
          14:  f1:=f1*n1;
          15:  od;
          16:
      ||
          21:  f2:=n2;
          22:  while (n1+2)<n2 do
          23:  n2:=n2-1;
          24:  f2:=f2*n2;
          25:  od;
          26:
3: coend;
4: if (n1+1)=n2 then f:=f1*f2; else f:=f1*f2*(n1+1); fi;
```

The verification conditions are the following :

- Finalization :

$$\forall \bar{n},\bar{n}_1,\bar{n}_2,\bar{f}_1,\bar{f}_2,\bar{f},\ P_4(\bar{n},\bar{n}_1,\bar{n}_2,\bar{f}_1,\bar{f}_2,\bar{f},\bar{n},\bar{n}_1,\bar{n}_2,\bar{f}_1,\bar{f}_2,\bar{f})$$
$$[(P_4[f_1*f_2/f]\wedge(n_1+1)=n_2)\vee(P_4[f_1*f_2*(n_1+1)/f]\wedge(n_1+1)\neq n_2)]\Rightarrow P_3$$
$$P_3\Rightarrow(P_{26}[16/c_1]\wedge P_{16}[26/c_2]$$

- Sequential proof of process 2 :

$$[(P_{23}\wedge(n_1+2)<n_2)\vee(P_{26}\wedge(n_1+2)\geq n_2)]\Rightarrow[P_{22}\wedge P_{25}]$$
$$P_{25}[f_2*n_2/f_2]\Rightarrow P_{24}$$
$$P_{24}[n_2-1/n_2]\Rightarrow P_{23}$$
$$P_{22}[n_2/f_2]\Rightarrow P_{21}$$

- Proof of absence of interference of process 2 with the proof of process 1:

For $j=1,\ldots,6$:

$$[(P_{23}[1j/c_1]\wedge(n_1+2)<n_2\wedge P_{1j}[23/c_2])\vee(P_{26}[1j/c_1]\wedge(n_1+2)\geq n_2\wedge P_{1j}[26/c_2])$$
$$\Rightarrow (P_{1j}[22/c_2]\wedge P_{1j}[25/c_2])$$

$$(P_{25}[1j/c_1,f_2*n_2/f_2]\wedge P_{1j}[25/c_2,f_2*n_2/f_2])\Rightarrow P_{1j}[24/c_2]$$
$$(P_{24}[1j/c_1,n_2-1/n_2]\wedge P_{1j}[24/c_2,n_2-1/n_2])\Rightarrow P_{1j}[23/c_2]$$
$$(P_{22}[1j/c_1,n_2/f_2]\wedge P_{1j}[22/c_2,n_2/f_2])\Rightarrow P_{1j}[21/c_2]$$

- The sequential proof of process 1 and proof of absence of interference of process 1 with the proof of process 2 are similar.

- Initialization :

$(P_{11}[21/c_2] \wedge P_{21}[11/c_1]) \Rightarrow P_1$
$P_1[1/n_1, n/n_2] \Rightarrow P_0$
$(n>1 \wedge P_0) \Rightarrow (f=\overline{n}! \wedge \overline{n}=n)$

In the following proof outline, we have :

$\pi(a,b)$=if $a \leq b$ then $a*(a+1)*\ldots*b$ else 1 fi
$I=(n=\overline{n} \wedge [((\overline{n}_1+1)=\overline{n}_2 \wedge \overline{f}=\overline{f}_1*\overline{f}_2) \vee ((\overline{n}_1+1)\neq\overline{n}_2 \wedge \overline{f}=\overline{f}_1*\overline{f}_2*(\overline{n}_1+1))])$

0: $\{(n>1) \Rightarrow (\overline{f}=n! \wedge \overline{n}=n)\}$
$n_1:=1;\ n_2:=n;$
1: $\{(n_1<n_2) \Rightarrow (1 \leq \overline{n}_2-\overline{n}_1 \leq 2 \wedge \overline{f}_1=\pi(n_1+1,\overline{n}_1) \wedge \overline{f}_2=\pi(\overline{n}_2,n_2) \wedge I)\}$
cobegin
11: $\{[(c_2 \in \{21,22\} \wedge n_1<n_2) \vee (c_2=23 \wedge n_1+2<n_2) \vee (c_2 \in \{24,25\} \wedge n_1+1<n_2) \vee (c_2=26 \wedge n_1<n_2 \leq n_1+2)] \Rightarrow [\overline{f}_1=\pi(n_1+1,\overline{n}_1) \wedge \max(n_1,\overline{n}_2-2) \leq \overline{n}_1<\overline{n}_2 \wedge I]\}$
$f_1:=1;$
12: $\{[(c_2 \in \{21,22\} \wedge n_1<n_2) \vee (c_2=23 \wedge n_1+2<n_2) \vee (c_2 \in \{24,25\} \wedge n_1+1<n_2) \vee (c_2=26 \wedge n_1<n_2 \leq n_1+2)] \Rightarrow [\overline{f}_1=f_1*\pi(n_1+1,\overline{n}_1) \wedge \max(n_1,\overline{n}_2-2) \leq \overline{n}_1<\overline{n}_2 \wedge I]\}$
while $(n_1+2)<n_2$ do
13: $\{[(c_2 \in \{21,22,23\} \wedge n_1+2<n_2) \vee (c_2 \in \{24,25,26\} \wedge n_1+1<n_2)] \Rightarrow [\overline{f}_1=f_1*\pi(n_1+1,\overline{n}_1) \wedge \max(n_1+1,\overline{n}_2-2) \leq \overline{n}_1<\overline{n}_2 \wedge I]\}$
$n_1:=n_1+1;$
14: $\{[(c_2 \in \{21,22,23\} \wedge n_1+1<n_2) \vee (c_2 \in \{24,25,26\} \wedge n_1<n_2)] \Rightarrow [\overline{f}_1=f_1*\pi(n_1,\overline{n}_1) \wedge \max(n_1,\overline{n}_2-2) \leq \overline{n}_1<\overline{n}_2 \wedge I]\}$
$f_1:=f_1*n_1;$
15: $\{[(c_2 \in \{21,22,23\} \wedge n_1+1<n_2) \vee (c_2 \in \{24,25,26\} \wedge n_1<n_2)] \Rightarrow [\overline{f}_1=f_1*\pi(n_1+1,\overline{n}_1) \wedge \max(n_1,\overline{n}_2-2) \leq \overline{n}_1<\overline{n}_2 \wedge I]\}$
od;
16: $\{[1 \leq n_2-n_1 \leq 2] \Rightarrow [\overline{n}_2-2 \leq n_1=\overline{n}_1<\overline{n}_2 \wedge f_1=\overline{f}_1 \wedge I]\}$
||
21: $\{[(c_1 \in \{11,12\} \wedge n_1<n_2) \vee (c_1=13 \wedge n_1+2<n_2) \vee (c_1 \in \{14,15\} \wedge n_1+1<n_2) \vee (c_1=16 \wedge n_1<n_2 \leq n_1+2)] \Rightarrow [\overline{f}_2=\pi(\overline{n}_2,n_2) \wedge \overline{n}_1<\overline{n}_2 \leq \min(\overline{n}_1+2,n_2) \wedge I]\}$
$f_2:=n_2;$
22: $\{[(c_1 \in \{11,12\} \wedge n_1<n_2) \vee (c_1=13 \wedge n_1+2<n_2) \vee (c_1 \in \{14,15\} \wedge n_1+1<n_2) \vee (c_1=16 \wedge n_1<n_2 \leq n_1+2)] \Rightarrow [\overline{f}_2=\pi(\overline{n}_2,n_2-1)*f_2 \wedge \overline{n}_1<\overline{n}_2 \leq \min(\overline{n}_1+2,n_2) \wedge I]\}$
while $(n_1+2)<n_2$ do
23: $\{[(c_1 \in \{11,12,13\} \wedge n_1+2<n_2) \vee (c_1 \in \{14,15,16\} \wedge n_1+1<n_2)] \Rightarrow [\overline{f}_2=\pi(\overline{n}_2,n_2-1)*f_2 \wedge \overline{n}_1<\overline{n}_2 \leq \min(\overline{n}_1+2,n_2-1) \wedge I]\}$
$n_2:=n_2-1;$
24: $\{[(c_1 \in \{11,12,13\} \wedge n_1+1<n_2) \vee (c_1 \in \{14,15,16\} \wedge n_1<n_2)] \Rightarrow [\overline{f}_2=\pi(\overline{n}_2,n_2)*f_2 \wedge \overline{n}_1<\overline{n}_2 \leq \min(\overline{n}_1+2,n_2) \wedge I]\}$
$f_2:=f_2*n_2;$
25: $\{[(c_1 \in \{11,12,13\} \wedge n_1+1<n_2) \vee (c_1 \in \{14,15,16\} \wedge n_1<n_2)] \Rightarrow [\overline{f}_2=\pi(\overline{n}_2,n_2-1)*f_2 \wedge \overline{n}_1<\overline{n}_2 \leq \min(\overline{n}_1+2,n_2) \wedge I]\}$
od;
26: $\{[1 \leq n_2-n_1 \leq 2] \Rightarrow [\overline{f}_2=f_2 \wedge \overline{n}_1<\overline{n}_2=n_2 \leq \overline{n}_1+2 \wedge I]\}$
coend;
3: $\{n_1=\overline{n}_1 \wedge n_2=\overline{n}_2 \wedge f_1=\overline{f}_1 \wedge f_2=\overline{f}_2 \wedge I\}$
if $(n_1+1)=n_2$ then $f:=f_1*f_2;$ else $f:=f_1*f_2*(n_1+1);$ fi;
4: $\{n=\overline{n} \wedge n_1=\overline{n}_1 \wedge n_2=\overline{n}_2 \wedge f_1=\overline{f}_1 \wedge f_2=\overline{f}_2 \wedge f=\overline{f}\}$ □

3.2.5.2 Methodological remarks

- The assertions associated with each point of each process are of the form $A \Rightarrow R$ where A depends upon the current values of the program location counters

of the other processes and the current values of the variables and R is a relationship between the current and final values of the variables. A describes what has been achieved until now and R describes what remains to be done. This would have been even more clear if we had used redondant assertions which describe more precisely the behavior of the program such as :

4: $\{[n>1 \wedge f=n!] \Rightarrow [n=\overline{n} \wedge n_1=\overline{n}_1 \wedge n_2=\overline{n}_2 \wedge f_1=\overline{f}_1 \wedge f_2=\overline{f}_2 \wedge f=\overline{f}]\}$

23: $\{[[(c_1=11 \wedge n_1=1 \wedge n_1+2<n_2) \vee (c_1 \in \{12,13\} \wedge n_1 \geq 1 \wedge f_1=n_1! \wedge n_1+2<n_2) \vee (c_1=14 \wedge n_1>1 \wedge f_1=(n_1-1)! \wedge n_1+1<n_2) \vee (c_1 \in \{15,16\} \wedge n_1>1 \wedge f_1=n_1! \wedge n_1+1<n_2)] \wedge [n_2 \leq n \wedge f_2=\pi(n_2,n)]] \Rightarrow [\overline{f}_2=\pi(\overline{n}_2,n_2-1)*f_2 \wedge \overline{n}_1<\overline{n}_2 \leq \underline{\min}(\overline{n}_1+2,n_2) \wedge I]\}$

This possibility offered by backward induction should be contrasted with forward induction (Lamport[77], Owicki & Gries[76a]). Using forward induction one must specify what has been done (i.e. A) but what remains to be done (i.e. R) cannot be specified. Therefore in order to understand the program, the reader must invent what remains to be done from what has been done and the output specification.

This observation for the factorial program is in fact general. The proof is that $J=(A \Rightarrow R)$ where $A(s)=(\exists \underline{s} \in S[\![P]\!] \mid \varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},s))$ and $R(s,\overline{s})=(t[\![P]\!]^*(s,\overline{s}) \wedge \sigma(\overline{s}))$ is always an invariant for backward positive induction I^{-1} (i.e. satisfies conditions 2.2.4-1 when Ψ is invariant for P, ε and σ). Notice that we could have proved the completeness of forward induction i using A. Therefore using the backward induction principle I^{-1} one can specify for the program reader the maximum information A about the current program state which could be specified using the forward induction principle i, plus some information R on what remains to be done.

- For completeness, we use program location counters. We could equivalently have used auxiliary variables, since we have shown using a very general model (Cousot & Cousot[80]) that both approaches are equivalent. In Owicki & Gries[76a, 76b], auxiliary variables are only used in order to designate program points. The method consists in implicitly defining a map f from program points to some finite domain, introducing as many assignments to auxiliary variables as necessary so that when control is at points $l_1,\ldots,l_n$ the auxiliary variables have value $f(l_1,\ldots,l_n)$ and using the auxiliary variables in the assertions. Exactly the same methodology can be applied by defining explicitly the map f and using it directly in the assertions.

4. CONSTRUCTION OF A METHOD FOR PROVING FREEDOM FROM DEADLOCK BY BACKWARD INDUCTION

A program is blocked if not all processes have terminated and all of its processes that have not yet terminated are delayed at an await. Formally if

$P \equiv \underline{L}:\underline{\text{cobegin}}\ CL_1 \| \ldots \| CL_n\ \underline{\text{coend}};\overline{L}:$

then P is blocked in state s if and only if $\beta[\![P]\!](s)$ is true, where

$$\beta[\![P]\!](s) = [\exists L_1 \in La[\![CL_1]\!], \ldots \exists L_n \in La[\![CL_n]\!], \exists M \in (V \to M) \mid s = \langle (L_1, \ldots, L_n), M\rangle \wedge$$
$$[[\forall i \in [1,n], \underset{\vee}{(CL_i \equiv \alpha L_i : \underline{\text{await}}\ B\ \underline{\text{then}}\ \gamma \wedge B \in dom(\mathbb{B}[\![B]\!]) \wedge \neg \mathbb{B}[\![B]\!](M))} (CL_i \equiv \alpha; L_i :)]$$
$$\wedge [\exists j \in [1,n] \mid \neg(CL_j \equiv \alpha; L_j :)]]]$$

A program P if free from deadlock if no execution of P can lead to a state where P is blocked, that is :

$$\forall (\underline{s}, \overline{s}) \in S[\![P]\!]^2, (\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s}, \overline{s})) \Rightarrow \neg\beta[\![P]\!](\overline{s})]$$

This is an invariance property where $\sigma(\overline{s}) = tt$ and $\Psi(\underline{s}, \overline{s}) = \neg\beta[\![P]\!](\overline{s})$ does not depend upon initial states. According to paragraph 2.2.6 freedom from deadlock can be proved by backward induction using the contrapositive principle $\overline{\iota}$.

The choice of an assertion language and its semantics is similar to the one for partial correctness (3.2.1) except that at point L_i of process i we have an assertion of the form :

$$I(L_i)(L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n, M)$$

instead of :

$$I(L_i)(L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n, M, \overline{M})$$

since it is no longer necessary to relate the current and final states.

The determination of the verification conditions corresponding to 2.2.4-3(b) is quite similar to paragraphs 3.2.2 and 3.2.3 except that final memory states are omitted. The remaining cases are :

- Initialization :

  $$[\forall \underline{s},\ \varepsilon(\underline{s}) \Rightarrow \neg\tilde{\rho}[\![P]\!](\overline{I})(\underline{s})]$$
  $$= [\forall \underline{L}, \underline{M},\ (P \equiv \underline{L}:\alpha \wedge \phi(\underline{M})) \Rightarrow \neg\overline{I}(\underline{L})(\underline{M})]$$

- Finalization :

  $$[\forall \overline{s},\ \beta[\![P]\!](\overline{s}) \Rightarrow \tilde{\rho}[\![P]\!](\overline{I})(\overline{s})]$$

  When $\overline{s}$ is not of the form $\langle (L_1, \ldots, L_n), M\rangle$, $\beta[\![P]\!](\overline{s})$ is false so that the condition is trivially verified, otherwise this is equivalent to :

$$[\forall L_1,\ldots,L_n,M,$$
$$[(P \equiv L:\underline{\text{cobegin}}\ CL_1 \| \ldots \| CL_n\ \underline{\text{coend}};\overline{L}:) \wedge (\exists j \in [1,n] \mid \neg CL_j \equiv \alpha;L_j:) \wedge$$
$$(\forall i \in [1,n],((CL_i \equiv \alpha;L_i:) \vee (CL_i \equiv \alpha L_i:\underline{\text{await}}\ B\ \underline{\text{then}}\ \gamma \wedge M \in dom(\mathbb{B}[\![B]\!]) \wedge$$
$$\mathbb{B}[\![B]\!](M)=tt))]$$
$$\Rightarrow \bigwedge_{k=1}^{n} \overline{I}(L_k)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M)$$

Informally, for any tuple $L_1,\ldots,L_n$ of labels such that :

- L_i designates a conditional critical section L_i:await B then α
  (in which case $\underline{\text{blocked}}[\![P]\!](i,L_i)(M)=[M \in dom(\mathbb{B}[\![B]\!])) \wedge \mathbb{B}[\![B]\!](M)=tt]$)

or

- L_i designates the exit point of process CL_i
  (in which case $\underline{\text{blocked}}[\![P]\!](i,L_i)(M)=tt$)

and such that not all labels designate exit points we must prove that :

$$\bigwedge_{i=1}^{n} \underline{\text{blocked}}[\![P]\!](i,L_i)(M) \Rightarrow \bigwedge_{i=1}^{n} \overline{I}(L_i)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M)$$

Example 4.1 : Let us consider the following very simple example :

```
0: cobegin                11: while true do 12: P(m); 13: V(m); 14: od; 15:
   ||                     21: while true do 22: P(m); 23: V(m); 24: od; 25:
3: coend;
```

where :

P(m)≡await m>0 then m:=m-1 end
V(m)≡m:=m+1

The verification conditions are :

- Initialization :

  $\phi(m) \Rightarrow \neg P_0(m)$

- Induction :

  . $[P_{11}(21,m) \wedge P_{21}(11,m)] \Rightarrow P_0(m)$

  . Sequential proof of process 1 :

  $P_{12}(c_2,m) \Rightarrow [P_{11}(c_2,m) \wedge P_{14}(c_2,m)]$
  $[m>0 \wedge m'=m-1 \wedge P_{13}(c_2,m')] \Rightarrow P_{12}(c_2,m)$
  $P_{14}(c_2,m+1) \Rightarrow P_{13}(c_2,m)$

  . Absence of interference of process 1 with the proof of process 2 :

  For j=1,...,5,
  $[P_{12}(2j,m) \wedge P_{2j}(12,m)] \Rightarrow [P_{2j}(11,m) \wedge P_{2j}(14,m)]$
  $[m>0 \wedge m'=m-1 \wedge P_{13}(2j,m') \wedge P_{2j}(13,m)] \Rightarrow P_{2j}(12,m)$
  $[P_{14}(2j,m+1) \wedge P_{2j}(14,m+1)] \Rightarrow P_{2j}(13,m)$

  . The sequential proof of process 2 and the checks of absence of interference of process 2 with the proof of process 1 are similar.

  . $P_3(m) \Rightarrow [P_{15}(25,m) \wedge P_{25}(15,m)]$

- Finalization :
  - Possible deadlock at 12 and 22 :
    $m \leq 0 \Rightarrow [P_{12}(22,m) \wedge P_{22}(12,m)]$
  - Possible deadlock at 12 and 25 :
    $m \leq 0 \Rightarrow [P_{12}(25,m) \wedge P_{25}(12,m)]$
  - Possible deadlock at 15 and 22 :
    $m \leq 0 \Rightarrow [P_{15}(22,m) \wedge P_{22}(15,m)]$

A proof outline is :

$\{m \geq 1 \Rightarrow \neg(m<1)\}$
0: $\{m<1\}$
cobegin
11: $\{[m \leq 0 \wedge c_2 \in \{21,22,24\}] \vee [m<0 \wedge c_2=23] \vee [c_2=25]\}$
while true do
12: $\{[m \leq 0 \wedge c_2 \in \{21,22,24\}] \vee [m<0 \wedge c_2=23] \vee [c_2=25]\}$
P(m);
13: $\{[m<0 \wedge c_2 \in \{21,22,24\}] \vee [m<-1 \wedge c_2=23] \vee [c_2=25]\}$
V(m);
14: $\{[m \leq 0 \wedge c_2 \in \{21,22,24\}] \vee [m<0 \wedge c_2=23] \vee [c_2=25]\}$
od;
15: $\{tt\}$
||
21: $\{[m \leq 0 \wedge c_1 \in \{11,12,14\}] \vee [m<0 \wedge c_1=13] \vee [c_1=15]\}$
while true do
22: $\{[m \leq 0 \wedge c_1 \in \{11,12,14\}] \vee [m<0 \wedge c_1=13] \vee [c_1=15]\}$
P(m);
23: $\{[m<0 \wedge c_1 \in \{11,12,14\}] \vee [m<-1 \wedge c_1=13] \vee [c_1=15]\}$
V(m);
24: $\{[m \leq 0 \wedge c_1 \in \{11,12,14\}] \vee [m<0 \wedge c_1=13] \vee [c_1=15]\}$
od;
25: $\{tt\}$
coend;
3: $\{tt\}$

Informally, at each program point we give a condition on m which is necessary (but may be not sufficient) for the program to be later blocked. Since this condition is not satisfied by the entry states when $m \geq 1$ the program cannot end in deadlock. □

5. CONSTRUCTION OF A METHOD FOR PROVING MUTUAL EXCLUSION BY BACKWARD INDUCTION

Two program sections are mutually exclusive if they do not contain statements which can be executed at the same time. Assume cs_i (cs_j) is a characteristic predicate of the set of labels belonging to the critical section in process CL_i (CL_j) of program :

$P \equiv L$: cobegin $CL_1 \| \ldots \| CL_n$ coend; $\bar{L}$:

The critical sections are mutually exclusive if and only if :

$$\forall(\underline{s},\overline{s})\in S[\![P]\!]^2,[\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s})]\Rightarrow \underline{me}[\![P]\!](i,j)(cs_i,cs_j)(\overline{s})]$$

where :

$$\underline{me}[\![P]\!](i,j)(cs_i,cs_j)(\overline{s}) =$$

$$[\forall(L_1,\ldots,L_n)\in\mathcal{L}^n, M\in(V\to M),\ (\overline{s}=<(L_1,\ldots,L_n),M>)\Rightarrow\neg(cs_i(L_i)\wedge cs_j(L_j))]$$

Mutual exclusion is an invariance property which can be proved using the contrapositive backward induction $\overline{\iota}$. The verification conditions are therefore similar to absence of deadlocks, except for finalization which is :

$$[\neg\underline{me}[\![P]\!](i,j)(cs_i,cs_j)(\overline{s})\Rightarrow\tilde{\rho}[\![P]\!](\overline{I})(\overline{s})]$$

$$= [(cs_i(L_i)\wedge cs_j(L_j))\Rightarrow \bigwedge_{k=1}^{n}\overline{I}(L_k)(L_1,\ldots,L_{k-1},L_{k+1},\ldots,L_n,M)]$$

Informally for all labels L_i of CL_i such that $cs_i(L_i)$ and L_j of CL_j such that $cs_j(L_j)$,

$$\overline{I}(L_i)(c_1,\ldots,c_{j-1},L_j,c_{j+1},\ldots,c_{i-1},c_{i+1},\ldots,c_n,M)$$

and

$$\overline{I}(L_j)(c_1,\ldots,c_{j-1},c_{j+1},\ldots,c_{i-1},L_i,c_{i+1},\ldots,c_n,M)$$

must hold. Moreover for all labels L_k of process CL_k, $k\in[1,n]-\{i,j\}$, we must have :

$$[cs_i(c_i)\wedge cs_j(c_j)]\Rightarrow\overline{I}(L_k)(c_1,\ldots,c_{k-1},c_{k+1},\ldots,c_n,M)$$

Example 5.1 : The program points 13 and 23 in program 4.1 are mutually exclusive when m≤1. The verification conditions are those of 4.1 except for finalization which is :

$P_{13}(23,m)$
$P_{23}(13,m)$

The proof outline is :

```
      {(m≤1)⇒¬(m>1)}
0:    {m>1}
      cobegin
                  11: {[m>0 ∧ c2=23]∨[m>1 ∧ c2∈{21,22,24}]}
                      while true do
                  12:     {[m>0 ∧ c2=23]∨[m>1 ∧ c2∈{21,22,24}]}
                          P(m);
                  13:     {[c2=23]∨[m>0 ∧ c2∈{21,22,24}]}
                          V(m);
                  14:     {[m>0 ∧ c2=23]∨[m>1 ∧ c2∈{21,22,24}]}
                      od;
                  15: {ff}
      ||
                  21: {[m>0 ∧ c1=13]∨[m>1 ∧ c1∈{11,12,14}]}
                      while true do
                  22:     {[m>0 ∧ c1=13]∨[m>1 ∧ c1∈{11,12,14}]}
                          P(m);
                  23:     {[c1=13]∨[m>0 ∧ c1∈{11,12,14}]}
                          V(m);
                  24:     {[m>0 ∧ c1=13]∨[m>1 ∧ c1∈{11,12,14}]}
                      od;
                  25: {ff}
      coend;
3:    {ff}
```

□

6. NON-TERMINATION PROOF BY BACKWARD INDUCTION

A program P does not terminate if and only if :

$[\forall(\underline{s},\overline{s})\in S[\![P]\!]^2,\ (\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}))\Rightarrow\Psi(\overline{s})]$

where

$\Psi(\overline{s}) = \neg[\exists\overline{L}\in\mathcal{L},\overline{M}\in(V\to M)\mid \overline{s}=<\overline{L},\overline{M}> \wedge P\Xi\alpha;\overline{L}:]$

This invariance property can be proved by backward induction $\overline{i}$. The verification conditions are those of paragraph 5 except for finalization which is :

$\forall\overline{M},\ I(\overline{L})(\overline{M})$ where $P\Xi\alpha;\overline{L}:$

Example 6.1 : The verification conditions are those of example 4.1 except for the finalization condition which is $P_3(m)$. The proof outline is :

```
0: {ff}
   cobegin
      11:{ff}while true do 12:{ff}P(m);13:{ff}V(m);14:{ff}od;15:{tt}
   ||
      21:{ff}while true do 22:{ff}P(m);23:{ff}V(m);24:{ff}od;25:{tt}
   coend;
3: {tt}
```

□

7. CONCLUSION

We have shown that invariance proof methods can be formally constructed rather than experimentally invented and later, if ever, proved sound and complete. This approach was illustrated by the construction of new backward invariance proof methods for a simple parallel language. Since the construction of invariance proof methods has been shown to consist in a few human choices (operational semantics, induction principle, assertion language and its semantics) and a lot of algebraic manipulations, our approach introduces the possibility of using interactive mechanical aids. Then complicated languages could be considered for which the construction of invariance proof methods would involve symbolic computations that might turn out to be formidable (e.g. ADA).

The backward partial correctness proof method which we have introduced (thus generalizing Morris & Wegbreit's subgoal induction) can be advantageously compared with the forward methods à la Lamport-Owicki & Gries. This is because the assertion A which is used in the forward method at each program point can be used for the backward method in the form A==>R. When the proof is used as comments this is useful to the program reader since A describes what has been

done up to that point and R what remains to be done. R must be invented by the program reader when forward proof methods are used.

An advantage of forward proof methods is that the same assertions can be used for partial correctness, absence of deadlocks, mutual exclusion, clean behavior, non-termination, etc. This is because in the case of forward induction, the same induction principle ($\mathcal{I}$) can be used for all these invariance properties. This is not the case for backward proof methods for which two induction principles must be used ($\mathcal{I}^{-1}$ and $\bar{\mathcal{I}}$) which lead to somewhat different verification conditions.

A successfull compromise might consist in using an intelligent combination of forward and backward induction principles. This is, for example, what is implicitly done in the informal proofs given by Ricart & Agrawala[81].

ACKNOWLEDGMENTS : I am very grateful to Professor Patrick Cousot for his advices and help.

8. REFERENCES

Apt K.R., Francez N. & de Roever W.P.[80], *A proof system for communicating sequential processes*, TOPLAS 2, 3(1980), 359-385.

Ashcroft E.A.[75], *Proving assertions about parallel programs*, J. of Comp. and System Sci. 10, (1975), 110-135.

Cousot P. & Cousot R.[80a], *Semantic analysis of communicating sequential processes*, Automata, Languages and Programming, 7th Colloq., Noordwijkerhout, Lect. Notes in Comp. Sci. 85, Springer Verlag, (July 1980), 119-133.

Cousot P. & Cousot R.[80b], *Reasoning about invariance proof methods*, Proc. Int. Workshop on Program Construction, Château de Bonas, France, Tome 1, INRIA Ed., (Sept. 1980).

Floyd R.W.[67], *Assigning meaning to programs*, Proc. Symp. in Applied Math., Vol.19 AMS, Providence R.I. USA, (1967), 19-32.

Hoare C.A.R.[69], *An axiomatic basis for computer programming*, CACM 12, 10(Oct. 1969), 576-580, 583.

Hoare C.A.R.[75], *Parallel programming : an axiomatic approach*, Computer Languages, 1(1975), 151-160.

Keller R.M.[76], *Formal verification of parallel programs*, CACM 19, 7(July 1976) 371-384.

King J.C.[79], *Program correctness : on inductive assertion methods*, RJ2525, IBM Research, San Jose, CA, USA, (Aug. 1979).

Lamport L.[77], *Proving the correctness of multiprocess programs*, IEEE Trans. on Soft. Eng., SE3, 2(March 1977), 125-143.

Levin G.M.[79], *A proof technique for communicating sequential processes (with an example)*, TR79-401, Comp. Sci. Dept., Cornell U., Ithaca NY, USA, (1979).

Manna Z.[70], *Mathematical theory of partial correctness*, JCSS 5, 3(June 1970) 238-253.

Misra J.[78], *Some aspects of the verification of loop computations*, IEEE Trans. on Soft. Eng., SE-4, 6(Nov. 1978), 478-486.

Morris J.H. & Wegbreit B.[77], *Subgoal induction*, CACM 20, 4(April 1977), 209-222.

Naur P.[66], *Proof of algorithms by general snapshots*, BIT 6, (1966), 310-316.

Newton G.[75], *Proving properties of interacting processes*, Acta Informatica 4 (1975), 117-126.

Owicki S. & Gries D.[76a], *An axiomatic proof technique for parallel programs I*, Acta Informatica, 6(1976), 319-340.

Owicki S. & Gries D.[76b], *Verifying properties of parallel programs : an axiomatic approach*, CACM 19, 5(May 1976), 279-285.

Ricart G. & Agrawala A.K.[81], *An optimal algorithm for mutual exclusion in computer networks*, CACM 24, 1(Jan. 1981), 9-17.