CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

chateau du montet
54500 vandœuvre les nancy

REASONING ABOUT
PROGRAM INVARIANCE PROOF METHODS

Patrick COUSOT and Radhia COUSOT

CRIN-80-P050 July 1980

université de nancy I
université de nancy 2
institut national polytechnique de lorraine

REASONING ABOUT PROGRAM INVARIANCE PROOF METHODS

Patrick Cousot and Radhia Cousot

ABSTRACT

We present a mathematical model based on dynamic discrete transition systems for constructing, explaining, verifying, comparing, optimizing and transforming program proof methods. We are particularly interested in invariance properties of parallel programs such as partial correctness, absence of global deadlock or non-termination which consist in characterizing a super-set of the possible descendents of the valid initial states.

We first characterize invariance properties using fixpoints and next derive the fundamental sound and complete invariance proof method. Proving the invariance of an assertion ψ consists in showing that some global invariant I stronger than ψ is true for the valid entry states and remains true after any transition. Known invariance proof methods (Floyd, Hoare, Naur,... for sequential deterministic programs; Clarke, Clint,... for coroutines; Howard,... for monitors; Ashcroft, Ashcroft & Manna, Hoare, Keller, Lamport, Mazurkiewicz, Newton, Owicki & Gries,... for parallel programs with shared variables; Apt,Francez & de Roever, Cousot & Cousot, Levine,... for communicating sequential processes à la Hoare;...) all reduce to the fundamental invariance proof method and differ only because they use different decompositions of the global invariant into local assertions attached to program control points. We formalize this notion of decomposition by means of Galois connections, the mathematical properties of which are reminded. We next show that a given decomposition of the global invariant into local assertions associated with program control points can lead to a complete lattice of distinct verification conditions. In order to illustrate this point we propose several variants of the Lamport-Owicki proof rules. All these variants are sound and complete althougth some of them require stronger invariants than others. Invariance proof methods which have been designed for very different programming languages can nevertheless be compared by showing that they rely on similar decompositions of the global invariant. It is also remarked that some decompositions lead to sound but intrinsically incomplete proof methods when the correspondence between the global invariant and its decomposition is not one-to-one for all programs. We next introduce the notion of reduction/extension of dynamic discrete transition systems in order to formalize the transformation of a program by introduction of auxiliary variables so that the assertions interleaved in the program text need not explicitly mention program control points. We finally compare our approach with other formalisms for reasoning about program proof methods in particular program logics.

RESUME

Nous présentons un modèle mathématique basé sur les systèmes dynamiques à transitions discrètes permettant de construire, expliquer, vérifier, comparer, optimiser et transformer les méthodes de preuve de propriétés des programmes. Nous nous intéressons particulièrement aux propriétés d'invariance des programmes parallèles comme la correction partielle, l'absence d'interblocage global ou la non-terminaison, qui consistent à caractériser un sur-ensemble des descendants possibles des états d'entrée.

Après avoir caractérisé les propriétés d'invariance par point-fixe, nous montrons que la méthode fondamentale de preuve d'invariance d'une assertion ψ consiste à découvrir une assertion plus forte I qui est vraie pour les états initiaux et reste vraie après une transition quelconque. Les méthodes connues de preuve d'invariance (Floyd, Hoare, Naur,... pour les programmes séquentiels déterministes; Clarke, Clint,... pour les coroutines; Howard,... pour les moniteurs; Ashcroft, Ashcroft & Manna, Hoare, Keller, Lamport, Mazurkiewicz, Newton, Owicki & Gries,... pour les programmes parallèles partageant des variables communes; Apt,Francez & de Roever, Cousot & Cousot, Levine,... pour les processus séquentiels communicants à la Hoare;...) se ramènent à la méthode fondamentale et ne diffèrent que par des décompositions différentes de l'invariant global en des assertions locales associées aux différents points de contrôle du programme. Après avoir formalisé cette décomposition au moyen de connections de Galois dont nous rappelons les propriétés mathématiques, nous montrons qu'une même décomposition de l'invariant global en des assertions associées aux points de contrôle du programme peut donner lieu à des règles de vérification distinctes formant un treillis complet. Nous illustrons ce résultat en proposant quelques variantes de la méthode de Lamport-Owicki, qui sont toutes correctes et complètes bien que certaines d'entre-elles nécessitent l'emploi d'invariants plus forts que d'autres. Des méthodes de preuve d'invariance ayant été conçues pour des langages très différents peuvent également être comparées en montrant qu'elles sont fondées sur des décompositions similaires de l'invariant global. Nous remarquons également que certaines décompositions conduisent à des méthodes de preuve qui sont correctes mais intrinsèquement incomplètes quand la correspondance entre l'invariant global et sa décomposition n'est pas injective pour tous les programmes. Nous introduisons ensuite la notion de réduction/extension de systèmes dynamiques à transitions discrètes qui permet de formaliser la transformation d'un programme par introduction de variables auxiliaires afin d'éviter de mentionner explicitement les points de contrôle du programme dans les assertions. Finalement, nous comparons notre appoche pour raisonner sur les méthodes de preuve de programes avec d'autres formalismes, notamment les logiques de programmes.

REASONING ABOUT PROGRAM INVARIANCE PROOF METHODS

Patrick Cousot* and Radhia Cousot**

1. INTRODUCTION

The essence of program proof methods such as those described by Naur[66], Floyd[67], Hoare[69] and their followers may be difficult to explain concisely. The main difficulty is the diversity of syntactic formalisms which are used for representing algorithms. This difficulty is even more evident when concurrent programs are concerned. For example, although similarities and differences are intuitively observed between methods for proving invariance properties of parallel programs such as Ashcroft[75], Hoare[75], Howard[76], Keller[76], Lamport[77], Mazurkiewicz[77], Newton[75], Owicki & Gries[76a,76b], it is difficult to compare them in abstracto. Until now the design of invariance proof methods has been ad-hoc and informal. The correctness of these methods is often established a posteriori (references in Apt[79]), rather it is more advisable to formally construct a priori correct program verification methods. Although soundness and completeness proofs rely on simple and similar principles they are often long and wearisome. Instead of always restarting from scratch the verification of program proof methods should be obtained from general purpose theorems. The manipulation of program proof methods should also be considered. For example can't Clarke[80]'s method for proving correctness of coroutines be considered as an optimization of Clint[73]'s method since simpler invariants are needed? The transformation of a program proof method in order to adapt it to

---

* Université de Metz, Faculté des Sciences, Ile du Saulcy, 57000 Metz, France
** CRIN-Nancy, Laboratoire Associé au CNRS n°262
This work was supported by CNRS (ATP Intelligence Artificielle)

different language features is not always easier than the original and may be error-proned (e.g. Apt, Francez & de Roever[79]). Rather program verification methods should be manipulated using correctness preserving transformations.

By analogy with what happened for programs the above arguments advocate in favor of a mathematical formalism for reasoning about program proof methods in particular for constructing, explaining, verifying, comparing, optimizing and transforming them. We present such a mathematical model based on dynamic discrete transition systems and illustrate it by means of an application to invariance proof techniques for parallel programs sharing global variables and synchronized by conditional critical sections. Finally other approaches are discussed.

2. OPERATIONAL SEMANTICS

We assume that the set La of syntactically correct programs can be defined by any context-sensitive syntax definition method. Defining the operational semantics of the programming language La consists in defining for each program $Pr \in La$ a set of states S(Pr) and a transition relation $t(Pr) \in [[S(Pr) \times S(Pr)] \rightarrow B]$ between any state and its possible successors. $B=\{true, false\}$ is the set of truth values, it is a complete boolean lattice $\langle B; \Rightarrow, false, true, \vee, \wedge, \neg \rangle$ with ordering $false \Rightarrow true$.

Example 2.0.1. : Let us consider a language La where each program $Pr \in La$ consists of parallel processes sharing global variables and synchronized by conditional critical regions. The abstract syntax is as follows :

- Programs Pr : $D;[P(1) \parallel P(2) \parallel \ldots \parallel P(\pi)]$ where $\pi \geq 1$.
- Declarations D : $x(1):t(1);\ldots;x(\delta):t(\delta)$ where $\delta \geq 1$.
- Variables $x(i), i \in [1,\delta]$.
- Types $t(i), i \in [1,\delta]$.
- Processes $P(i), i \in [1,\pi]$: $\lambda(i,1):C(i,1);\ldots;\lambda(i,\sigma(i)):C(i,\sigma(i));\lambda(i,\sigma(i)+1):stop$ where $\sigma(i) \geq 1$.
- Labels $\lambda(i,j), i \in [1,\pi], j \in [1,\sigma(i)+1]$.
- Commands $C(i,j), i \in [1,\pi], j \in [1,\sigma(i)]$:
  - Null commands $C(i,j), i \in [1,\pi], j \in N(i)$: skip.
  - Assignment commands $C(i,j), i \in [1,\pi], j \in A(i)$: $x(\alpha(i,j)) := e(i,j)(x)$ where $\alpha(i,j) \in [1,\delta]$ and $e(i,j)$ maps $dom(e(i,j)) \subseteq t$ into $t(\alpha(i,j))$.

. Test commands $\underline{C}(i,j), i\in[1,\underline{\pi}], j\in\underline{T}(i)$: if $\underline{b}(i,j)(\underline{x})$ go to $\underline{\lambda}(i,\underline{\eta}(i,j))$ where $\underline{b}(i,j)$ maps $dom(\underline{b}(i,j))\subseteq\underline{t}$ into truth values and $\underline{\eta}(i,j)\in[1,\underline{\sigma}(i)+1]$.

. Await commands $\underline{C}(i,j), i\in[1,\underline{\pi}], j\in\underline{W}(i)$: await $\underline{b}(i,j)(\underline{x})$ then $\underline{x}:=\underline{f}(i,j)(\underline{x})$ where $\underline{b}(i,j)$ maps $dom(\underline{b}(i,j))\subseteq\underline{t}$ into truth values and $\underline{f}(i,j)$ maps $dom(\underline{f}(i,j))\subseteq\underline{t}$ into $\underline{t}$.

. Stop commands $\underline{C}(i,j), i\in[1,\underline{\pi}], j\in\underline{H}(i)$: stop.

$\{\underline{N}(i),\underline{A}(i),\underline{T}(i),\underline{W}(i),\underline{H}(i)\}$ is a partition of $[1,\underline{\sigma}(i)]$.

The concrete syntax of some examples will freely deviate from the above abstract syntax when the correspondence is obvious.

Let us now define the operational semantics. Concurrency in the execution of a program is modeled by global nondeterminism in the selection of successor states :

- Program locations : $\underline{L}=\Pi\{\{\underline{\lambda}(i,j):j\in[1,\underline{\sigma}(i)+1]\}:i\in[1,\underline{\pi}]\}$.
  (If $\{E(i):i\in I\}$ is a family of sets the cartesian product $\Pi\{E(i):i\in I\}$ is defined as the subset of $I\to\cup\{E(i):i\in I\}$ of all functions f for which $f(i)\in E(i)$ for all $i\in I$).
- States : $S=\underline{t}\times\underline{L}$
- Transition relation :

$t\in[[S\times S]\to B]$

$t=\lambda((xa,ca),(xb,cb)).[\exists i\in[1,\underline{\pi}], j\in[1,\underline{\sigma}(i)], k\in[1,\underline{\sigma}(i)+1]:(\forall\ell\in[1,\underline{\pi}]-\{i\}, cb(\ell)=ca(\ell))\wedge(ca(i)=\underline{\lambda}(i,j))\wedge(cb(i)=\underline{\lambda}(i,k))\wedge Com(i,j,k)(xa,xb)]$

$Com(i,j,k)\in[[\underline{t}\times\underline{t}]\to B], i\in[1,\underline{\pi}], j\in[1,\underline{\sigma}(i)], k\in[1,\underline{\sigma}(i)+1]$

$Com(i,j,k)=\lambda(xa,xb).[(Null(i,j)(xa,xb)\wedge k=j+1)\vee(Assign(i,j)(xa,xb)\wedge k=j+1)\vee Test(i,j,k)(xa,xb)\vee(Await(i,j)(xa,xb)\wedge k=j+1)]$

$Null(i,j)\in[[\underline{t}\times\underline{t}]\to B], i\in[1,\underline{\pi}], j\in[1,\underline{\sigma}(i)]$

$Null(i,j)=\lambda(xa,xb).[j\in\underline{N}(i)\wedge xb=xa]$

$Assign(i,j)\in[[\underline{t}\times\underline{t}]\to B], i\in[1,\underline{\pi}], j\in[1,\underline{\sigma}(i)]$

$Assign(i,j)=\lambda(xa,xb).[j\in\underline{A}(i)\wedge xa\in dom(\underline{e}(i,j))\wedge xb=subst(xa)(\underline{\alpha}(i,j)/\underline{e}(i,j)(xa))]$

(If $\{E(i):i\in I\}$ is a family of sets and $x\in\Pi\{E(i):i\in I\}, j\in I, v\in E(j)$ then $subst(x)(j/v)$ equals y such that $y(j)=v$ whereas $y(k)=x(k)$ for all $k\in I$ such that $k\neq j$. If $n>1, j_1,\ldots,j_n\in I$ and $v_1\in E(j_1),\ldots,v_n\in E(j_n)$ then $subst(x)(j_1/v_1,\ldots,j_n/v_n)=subst[subst(x)(j_1/v_1)][j_2/v_2,\ldots,j_n/v_n]$).

$Test(i,j,k)\in[[\underline{t}\times\underline{t}]\to B], i\in[1,\underline{\pi}], j\in[1,\underline{\sigma}(i)], k\in[1,\underline{\sigma}(i)+1]$

$Test(i,j,k)=\lambda(xa,xb).[j\in\underline{T}(i)\wedge xb=xa\wedge xa\in dom(\underline{b}(i,j))\wedge[(\underline{b}(i,j)(xa)\wedge k=\underline{\eta}(i,j))\vee(\neg\underline{b}(i,j)(xa)\wedge k=j+1)]]$

$Await(i,j) \in [[\underline{t} \times \underline{t}] \rightarrow B], i \in [1, \underline{\pi}], j \in [1, \underline{\sigma}(i)]$

$Await(i,j) = \lambda(xa,xb).[j \in \underline{W}(i) \wedge xa \in (dom(\underline{b}(i,j)) \wedge \underline{b}(i,j)(xa) \wedge xa \in dom(\underline{f}(i,j)) \wedge xb = \underline{f}(i,j)(xa)]$

The above semantics does require that statements be executed as indivisible actions. This convention can be lifted for assignments and boolean expressions when memory reference is indivisible and each assignment or boolean expression refers at most once to at most one variable which can be changed by another process while this statement or expression is beeing evaluated. *End of Example.*

3. FIXPOINT CHARACTERIZATION OF INVARIANCE PROPERTIES

Invariance properties of programs include *partial correctness, absence of deadlocks* and *non-termination* that is properties of programs that can be proved by characterizing super-sets of the sets of states that can be reached during execution of these programs.

We assume that $E(\underline{Pr})$ is the set of possible entry specifications for the program $\underline{Pr} \in \underline{La}$ the meaning of which is given by $\varepsilon(\underline{Pr}) \in [E(\underline{Pr}) \rightarrow P(\underline{Pr})]$ where $P(\underline{Pr}) = [S(\underline{Pr}) \rightarrow B]$. $\varepsilon(\underline{Pr})(\phi)$ is a predicate over states which characterizes the set of possible initial states corresponding to the entry specification $\phi \in E(\underline{Pr})$ of program $\underline{Pr}$.

Let $Post(\underline{Pr}) \in [[[S(\underline{Pr}) \times S(\underline{Pr})] \rightarrow B] \rightarrow [P(\underline{Pr}) \rightarrow P(\underline{Pr})]]$ be the "before-after" predicate transformer defined as :

$Post(\underline{Pr}) = \lambda\theta.[\lambda\omega.[\lambda sa.[\exists sb \in S(\underline{Pr}) : \omega(sb) \wedge \theta(sb,sa)]]]$

By definition the set of states which may be reached during any execution of program $\underline{Pr}$ starting with an initial value of the variables satisfying the entry specification $\phi \in E(\underline{Pr})$ is characterized by $Post(\underline{Pr})(t(\underline{Pr})^*)(\varepsilon(\underline{Pr})(\phi))$ where $t(\underline{Pr})^*$ is the reflexive transitive closure of $t(\underline{Pr})$.

Let $f(\underline{Pr}) \in [E(\underline{Pr}) \rightarrow [P(\underline{Pr}) \rightarrow P(\underline{Pr})]]$ be $\lambda\phi.[\lambda\omega.[\varepsilon(\underline{Pr})(\phi) \vee Post(\underline{Pr})(t(\underline{Pr}))(\omega)]]$ and lfp be the least fixpoint operator for isotone operators on complete lattices (Cousot & Cousot[79b]). Invariance proof methods (Cousot[79]) and automatic program analysis techniques (Cousot & Cousot[79a]) are founded on the following :

THEOREM 3.0.1

$\forall \underline{Pr} \in \underline{La}, \forall \phi \in E(\underline{Pr}), \quad Post(\underline{Pr})(t(\underline{Pr})^*)(\varepsilon(\underline{Pr})(\phi)) = lfp(f(\underline{Pr})(\phi))$

For the sake of conciseness the parameter $\underline{Pr}$ will be omitted and we will write $\forall \underline{Pr} \in \underline{La}, \forall \phi \in E, \; Post(t^*)(\varepsilon(\phi)) = lfp(f(\phi))$.

4. FUNDAMENTAL SOUND AND COMPLETE INVARIANCE PROOF METHOD

$\psi \in P$ is said to be invariant during execution of program Pr starting with any state satisfying the entry specification $\phi \in E$ if and only if $Post(t^*)(\varepsilon(\phi)) \Rightarrow \psi$.

Example 4.0.1. : Assume that $\xi \in P$ characterizes exit states, $\phi \in E$ is an entry specification and $\psi \in P$ is an exit specification. A partial correctness proof consists in showing the invariance of $(\xi \Rightarrow \psi)$ that is $Post(t^*)(\varepsilon(\phi)) \Rightarrow (\xi \Rightarrow \psi)$. Let us define $\beta = \lambda s.[\forall s' \in S, \neg t(s,s')]$ so that β characterizes blocking states. An absence of deadlock proof consists in showing the invariance of $(\beta \Rightarrow \xi)$ that is $Post(t^*)(\varepsilon(\phi)) \Rightarrow (\beta \Rightarrow \xi)$. Finally a non-termination proof is also an invariance proof since it consists in showing that $Post(t^*)(\varepsilon(\phi)) \Rightarrow (\neg\xi \wedge \neg\beta)$. *End of Example.*

The above fixpoint characterization of $Post(t^*)(\varepsilon(\phi))$ and Tarski's fixpoint theorem lead to a sound and complete invariance proof method :

THEOREM 4.0.2

$$\forall Pr \in La,\ \forall \phi \in E,\ \forall \psi \in P,\ [Post(t^*)(\varepsilon(\phi)) \Rightarrow \psi] \Leftrightarrow [\exists I \in P : (f(\phi)(I) \Rightarrow I) \wedge (I \Rightarrow \psi)]$$

Example 4.0.3. : The global assertion $\psi = \lambda(x,c).[(c(1) = \underline{2} \wedge c(2) = \underline{2}) \Rightarrow (x=2)]$ is invariant for the program :

```
x:integer;
[1: await true then x:=x+1; 2: stop
||1: await true then x:=x+1; 2: stop]
```

and the entry specification :

$\phi = (x=0)$ such that $\varepsilon(\phi) = \lambda(x,c).[c(1) = \underline{1} \wedge c(2) = \underline{1} \wedge x=0]$.

This can be proved using the assertion :

$$I = \lambda(x,c).[(c(1) = \underline{1} \wedge c(2) = \underline{1} \wedge x=0) \vee (c(1) = \underline{1} \wedge c(2) = \underline{2} \wedge x=1) \vee (c(1) = \underline{2} \wedge c(2) = \underline{1} \wedge x=1) \vee (c(1) = \underline{2} \wedge c(2) = \underline{2} \wedge x=2)]$$

such that $\varepsilon(\phi) \Rightarrow I$ and $Post(t)(I) \Rightarrow I$ and $I \Rightarrow \psi$. *End of Example.*

Example 4.0.4. : Ashcroft[75] justifies his method for proving assertions about parallel programs by a verification condition theorem which, using the above notations, can be stated as :

$$\forall Pr \in La, \forall \phi \in E, \forall \psi \in P,$$
$$[\varepsilon(\phi) \Rightarrow \psi \wedge Post(t)(Post(t^*)(\varepsilon(\phi)) \wedge \psi) \Rightarrow \psi] \Leftrightarrow [Post(t^*)(\varepsilon(\phi)) \Rightarrow \psi].$$

Ashcroft next remarks that we don't know exactly which states satisfy

Post(t*)(ε(ϕ)) so that "we could have left this term out of the verification condition entirely". He adds "however, if the impossibility of reaching certain states is crucial for certain properties of a program to hold" then we can "explicitly incorporate the impossibility into the assertions we wish to prove valid and check the above condition for all states". This informally introduces the induction principle :

$\forall \underline{Pr} \in \underline{La}$, $\forall \phi \in E$, $\forall \psi \in P$,

$[\exists I \in P: (\varepsilon(\phi) \Rightarrow I) \wedge (\mathit{Post}(t)(I) \Rightarrow I) \wedge (I \Rightarrow \psi)] \Rightarrow [\mathit{Post}(t^*)(\varepsilon(\phi)) \Rightarrow \psi]$

later proved by Keller[76]. *End of Example.*

5. DECOMPOSITION OF A GLOBAL ASSERTION ABOUT PROGRAM STATES INTO A SET OF LOCAL ASSERTIONS

According to theorem 4.0.2 an invariance proof consists in showing that $[\exists I \in P: (\varepsilon(\phi) \Rightarrow I) \wedge (\mathit{Post}(t)(I) \Rightarrow I) \wedge (I \Rightarrow \psi)]$. A single global assertion I about values of variables and program locations is used in order to describe sets of program states. This verification condition can be decomposed into a conjonction of verification conditions $[\exists I \in P: (\varepsilon(\phi) \Rightarrow I) \wedge \wedge \{\mathit{Post}(t(k))(I) \Rightarrow I : k \in T\} \wedge (I \Rightarrow \psi)\}$ whenever the transition relation is of the form $t = \vee\{t(k) : k \in T\}$ (e.g. Ashcroft[75], Keller[76]).

Another type of decomposition consists in formulating this global assertion using a set of local assertions interspread at appropriate places in the program text. Without loss of generality the set of local assertions used in the proof of a program $\underline{Pr}$ will be assumed to belong to a complete boolean lattice $\langle A; \Rightarrow, \mathit{false}, \mathit{true}, \vee, \wedge, \neg \rangle$ (the symbols $\Rightarrow$, *false*, *true*, $\vee$, $\wedge$, $\neg$ are overloaded). The meaning of A will be formally specified by maps $\rho \in [P \to A]$ and $\overline{\rho} \in [A \to P]$ which define a correspondence with P.

Example 5.0.1. : Naur[66], Floyd[67] and Hoare[69]'s invariance proof methods are applicable to programs $\underline{Pr}$ involving a single process ($\underline{\pi}=1$). A local assertion about values of variables is associated with each program point, hence $A = \Pi\{[\underline{t} \to B] : j \in [1, \underline{\sigma}(1)+1]\}$. The meaning of $\Omega \in A$ is given by $\overline{\rho}(\Omega) = \lambda(x,c).[\vee\{c = \underline{\lambda}(1,j) \wedge \Omega(j)(x) : j \in [1, \underline{\sigma}(1)+1]\}]$. The set of local assertions corresponding to a global assertion $\omega \in P$ is $\rho(\omega) = \lambda j.[\lambda x.[\omega(x, \underline{\lambda}(1,j))]]$. ρ is a total complete isomorphism and $\overline{\rho}$ is its inverse. *End of Example.*

Since global assertions of P and sets of local assertions of A play equivalent rôles one is tempted to require $\rho \in [P \to A]$ to be a total complete

lattice isomorphism. A weaker hypothesis turns out to be sufficient : ρ will only be required to be a complete join-morphism (i.e. $\forall X \subseteq P$, $\rho(\vee X)=\vee\rho(X)$. In particular when X is empty, ρ is *false*-strict).

6. GALOIS CONNECTIONS

Complete join-morphisms between posets correspond in a one-to-one manner with Galois connections (Pickert[52], Dubreil & Croisot[54]).

In what follows $\cup X$ (respectively $\cap X$) denotes the least upper bound (respectively greatest lower bound) of a subset X of a poset $\langle P;\leq\rangle$ when it exists. We define a Galois connection between posets $\langle P;\leq\rangle$ and $\langle A;\leq\rangle$ as a pair $(\rho,\overline{\rho})$ of total maps $\rho\in[P\to A]$ and $\overline{\rho}\in[A\to P]$ which are isotone and such that $\rho\circ\overline{\rho}\leq 1$ and $1\leq\overline{\rho}\circ\rho$ where 1 is the identity map. (This definition is equivalent to the one of Birkhoff[40] and Ore[44] when considering the dual of $\langle A;\leq\rangle$). The partial maps $\partial\in[[A\to P]\to[P\to A]]$ and $\overline{\partial}\in[[P\to A]\to[A\to P]]$ are defined as

(6.0.1) $\partial=\lambda\overline{\rho}.[\lambda x.[\cap\{y\in A:\ x\leq\overline{\rho}(y)\}]]$

(6.0.2) $\overline{\partial}=\lambda\rho.[\lambda y.[\cup\{x\in P:\ \rho(x)\leq y\}]]$

Let $\langle P;\leq\rangle$ and $\langle A;\leq\rangle$ be posets and $\rho\in[P\to A]$, $\overline{\rho}\in[A\to P]$ be total isotone maps. Then it follows from Schmidt[53] and Pickert[52] that the following conditions are equivalent :

(6.0.3) ρ is a complete $\cup$-morphism and $\overline{\rho}=\overline{\partial}(\rho)$

(6.0.4) $\rho\circ\overline{\rho}\leq 1$ and $1\leq\overline{\rho}\circ\rho$

(6.0.5) $\forall x\in P,\ \forall y\in A,\ (\rho(x)\leq y)\Longleftrightarrow(x\leq\overline{\rho}(y))$

(6.0.6) $\partial(\overline{\rho})=\rho$ and $\overline{\partial}(\rho)=\overline{\rho}$

(6.0.7) $\rho\leq\partial(\overline{\rho})$ and $\overline{\partial}(\rho)\leq\overline{\rho}$

(6.0.8) $\overline{\rho}$ is a complete $\cap$-morphism and $\rho=\partial(\overline{\rho})$

If $(\rho,\overline{\rho})$ is a Galois connection between $\langle P;\leq\rangle$ and $\langle A;\leq\rangle$ then the above results together with Zervos[71] imply :

(6.0.9) $(\rho\circ\overline{\rho}=1) \Longleftrightarrow$ (ρ is onto) $\Longleftrightarrow$ ($\overline{\rho}$ is one-to-one)

(6.0.10) $(\overline{\rho}\circ\rho=1) \Longleftrightarrow$ (ρ is one-to-one) $\Longleftrightarrow$ ($\overline{\rho}$ is onto)

Let $\langle P;\leq,\perp,\top,\cup,\cap,\neg\rangle$ and $\langle A;\leq,\perp,\top,\cup,\cap,\neg\rangle$ be boolean lattices. If f is a function of a boolean lattice into a boolean lattice, we define $\tilde{f}$ as $\lambda x.[\neg(f(\neg(x)))]$, then :

(6.0.11) $(\rho,\overline{\rho})$ is a Galois connection between P and A if and only if $(\tilde{\overline{\rho}},\tilde{\rho})$ is a Galois connection between A and P

(6.0.12) f is onto (one-to-one) iff $\tilde{f}$ is onto (one-to-one)

Example 6.0.13. : If $t \in [[S \times S] \to B]$ is a transition relation then its inverse is $t^{-1} = \lambda(s_1, s_2).[t(s_2, s_1)]$. Define *Pre* as $\lambda\theta.[Post(\theta^{-1})]$. It is observed in Cousot[79] that $(Post(t), \tilde{Pre}(t))$ is a Galois connection between P and P. This is used to prove that Floyd-Naur 's invariance proof method :

$$\forall \underline{Pr} \in \underline{La}, \phi \in E, \psi \in P, \ [\exists I \in P: \ \varepsilon(\phi) \Rightarrow I \wedge Post(t)(I) \Rightarrow I \wedge I \Rightarrow \psi] \Leftrightarrow [Post(t^*)(\varepsilon(\phi)) \Rightarrow \psi]$$

is equivalent to Hoare's method :

$$\forall \underline{Pr} \in \underline{La}, \phi \in E, \psi \in P, \ [\exists I \in P: \ \varepsilon(\phi) \Rightarrow I \wedge I \Rightarrow \tilde{Pre}(t)(I) \wedge I \Rightarrow \psi] \Leftrightarrow [Post(t^*)(\varepsilon(\phi)) \Rightarrow \psi]$$

It is also observed that $\forall I \in P$, $\tilde{Pre}(t)(I) \Rightarrow (Pre(t)(I) \vee \neg Pre(t)(true))$ and equality holds when t is deterministic (i.e. $\forall s_1, s_2, s_3 \in S$, $(t(s_1, s_2) \wedge t(s_1, s_3)) \Rightarrow (s_2 = s_3)$). In that case $(I \Rightarrow \tilde{Pre}(t)(I))$ is equivalent to $(I \wedge Pre(t)(true)) \Rightarrow Pre(t)(I)$. *End of Example.*

7. THE COMPLETE LATTICE OF VARIANTS OF THE FUNDAMENTAL INVARIANCE PROOF METHOD

Given $\underline{Pr} \in \underline{La}$, let ρ be a complete join-morphism from the complete boolean lattice $\langle P; \Rightarrow, false, true, \vee, \wedge, \neg \rangle$ of "global assertions" into the complete boolean lattice $\langle A; \Rightarrow, false, true, \vee, \wedge, \neg \rangle$ of "sets of local assertions". Define $\overline{\rho}$ as $\overline{\partial}(\rho)$.

A set $\Psi \in A$ of assertions is said to be invariant during execution of program $\underline{Pr}$ starting with any state satisfying the entry specification $\phi \in E$ if and only if :

(7.0.1) $\rho(Post(t^*)(\varepsilon(\phi))) \Rightarrow \Psi$

or equivalently (3.0.1) :

$$\rho(lfp(f(\phi))) \Rightarrow \Psi$$

This property can be proved using any of the sound or sound and complete variants of the fundamental invariance proof method 4.0.2 which are introduced by the following :

THEOREM 7.0.2

Let f be an isotone operator on the complete lattice $\langle P; \Rightarrow, false, true, \vee, \wedge \rangle$ ρ be a complete $\vee$-morphism from P into the complete lattice $\langle A; \Rightarrow, false, true, \vee, \wedge \rangle$, $\overline{\rho}$ be $\overline{\partial}(\rho)$ and $\underline{F}$ be $\rho \circ f \circ \overline{\rho}$. Then :

(1) $\rho(lfp(f)) \Rightarrow lfp(\underline{F})$ and the inequality can be strict.

(2) If ρ is one-to-one (but this is not necessary) then $\rho \circ f = \underline{F} \circ \rho$ so that $\rho(lfp(f)) = lfp(\underline{F})$.

If moreover, there exists an isotone operator $\overline{F}$ on A such that $\rho \circ f = \overline{F} \circ \rho$ then :

(3) $\underline{F} \Rightarrow \overline{F}$, the inequality can be strict and equality holds if (but not only if) ρ is onto.

(4) Let F be any isotone operator on A such that $\underline{F} \Rightarrow F \Rightarrow \overline{F}$ then : $lfp(\underline{F}) = lfp(F) = lfp(\overline{F}) = \rho(lfp(f))$.

(5) Further if F is any isotone operator on A such that $\rho(lfp(f)) \Rightarrow lfp(F)$ (respectively $\rho(lfp(f)) = lfp(F)$) then $\forall \Psi \in A$, $\rho(lfp(f)) \Rightarrow \Psi$ if (respectively if and only if) $[\exists I \in A : (F(I) \Rightarrow I) \wedge (I \Rightarrow \Psi)]$.

Given an operational semantics, the design of an invariance proof method consists in choosing A and ρ that is how sets of program states are going to be described by assertions. Then theorem 7.0.2 leads to sound and complete verification rules. If ρ is not onto, the complete lattice of $F \in [A \rightarrow A]$ such that $\underline{F} \Rightarrow F \Rightarrow \overline{F}$ is not reduced to a single element and several program verification rules can be considered. In order to prove the invariance of $\Psi \in A$, $\overline{F}$ may require to guess a stronger invariant I than $\underline{F}$ since $(\overline{F}(I) \Rightarrow I)$ implies $(F(I) \Rightarrow I)$ which implies $(\underline{F}(I) \Rightarrow I)$ but the reciprocal may not be true. This argument in favor of $\underline{F}$ must be balanced by the fact that $\overline{F}$ leads to simpler verification rules. In practice a balance must be found between weaker invariants and more complicated verification rules or stronger invariants and simpler verification rules.

8. CONSTRUCTING AN INVARIANCE PROOF METHOD

We study invariance proof techniques for parallel programs consisting of concurrent processes sharing global variables and synchronized by conditional critical sections as defined at example 2.0.1. Several well-known methods are shortly reviewed and some of the possible alternatives are constructively introduced and discussed.

8.1 USING A SINGLE GLOBAL INVARIANT

In the invariance proof methods of Ashcroft[75] and Keller[76] a single invariant is used to indicate correctness (see paragraph 4). As remarked at paragraph 5 the ability to decompose the verification conditions is not lost. However the use of a single invariant is inadequate for large programs.

8.2 ASSOCIATING AN ASSERTION ABOUT VARIABLES WITH EACH CONTROLE STATE

One of the early attempts toward the decomposition of the global invariant can be found in Ashcroft& Manna[70]. Converting a parallel program to a nondeterministic serial program is one way of applying the idea of Naur, Floyd and Hoare that is associating a local assertion about the values of the variables with each possible value of the control state. Another way to apply the same idea consists in defining :

(8.2.0.1) $A = [\underline{L} \to [\underline{t} \to B]]$

$\rho = \lambda\omega.[\lambda c.[\lambda x.[\omega(x,c)]]]$

Since ρ is bijective the corresponding invariance proof method is sound and complete (th.7.0.2). Unfortunately this leads to $(\underline{\sigma}(1)+1)x...x(\underline{\sigma}(\underline{\pi})+1)$ local assertions so that proofs tend to be very long.

8.3 ASSOCIATING AN ASSERTION ABOUT VARIABLES WITH EACH PROGRAM POINT

The above exponential explosion can be avoided by associating about $(\underline{\sigma}(1)+1)+...+(\underline{\sigma}(\underline{\pi})+1)$ local assertions with a program. For example one can associate with each program point an assertion about the values of the variables. Otherwise stated this consists in choosing :

(8.3.0.1) $A = \Pi\{\Pi\{[\underline{t} \to B] : j \in [1, \underline{\sigma}(i)+1]\} : i \in [1, \underline{\pi}]\}$

$\rho = \lambda\omega.[\lambda i.[\lambda j.[\lambda x.[\exists c \in \underline{L} : \omega(x, subst(c)(i/\underline{\lambda}(i,j)))]]]]$

$\overline{\rho} = \lambda\Omega.[\lambda(x,c).[\forall i \in [1,\underline{\pi}],\ \exists j \in [1,\underline{\sigma}(i)+1] : c(i)=\underline{\lambda}(i,j) \wedge \Omega(i)(j)(x)]]$

Keller[76] and Owicki & Gries[76b] have shown that the corresponding proof method is not complete using informal arguments about an example. Theorem 7.0.2 gives a formal basis to such a reasoning : the proof method corresponding to ρ is not complete if a program $\underline{Pr}$ can be found such that $\rho(lfp(f(\phi))) \neq lfp(\underline{F}(\phi))$.

8.4 ASSOCIATING AN ASSERTION ABOUT VARIABLES AND PROGRAM LOCATION COUNTERS WITH EACH PROGRAM POINT

An alternative to 8.3.0.1 consists in associating with each program point an assertion about the memory state and the control state. This idea can be extirpated out of Newton[75]'s ISIPA model and is clear in Lamport[77] (it is also implicitly present in Owicki & Gries[76b], Mazurkiewicz[77] although the

control state is simulated by history variables). Otherwise stated this consists in choosing :

(8.4.0.1) $A = \Pi\{\Pi\{[\underline{t}x\underline{L} \to B]:j\in[1,\underline{\sigma}(i)+1]\}:i\in[1,\underline{\pi}]\}$

$\rho = \lambda\omega.[\lambda i.[\lambda j.[\lambda(x,c).[c(i)=\underline{\lambda}(i,j) \wedge \omega(x,c)]]]]$

$\overline{\rho} = \lambda\Omega.[\lambda(x,c).[\forall i\in[1,\underline{\pi}],\exists j\in[1,\underline{\sigma}(i)+1]: c(i)=\underline{\lambda}(i,j) \wedge \Omega(i)(j)(x,c)]]$

$(\rho,\overline{\rho})$ is a Galois connection between P and A and ρ is one-to-one so that according to theorem 7.0.2.(2)-(5) the corresponding invariance proof method is sound and complete. Let us introduce :

$Post(m)(n,p) \in[[[\underline{t}x\underline{L}]\to B] \to[[\underline{t}x\underline{L}]\to B]],\ m\in[1,\underline{\pi}],n\in[1,\underline{\sigma}(m)],p\in[1,\underline{\sigma}(m)+1]$

$Post(m)(n,p) = \lambda\psi.[\lambda(x,c).[c(m)=\underline{\lambda}(m,p)\wedge(\exists x'\in\underline{t}:\psi(x',subst(c)(m/\underline{\lambda}(m,n)))\wedge Com(m,n,p)(x',x))]]$

so that if ψ is an assertion about variables and control states which is true before execution of command $\underline{C}(m,n)$, then $Post(m)(n,p)(\psi)$ is true after execution of $\underline{C}(m,n)$ and going to $\underline{\lambda}(m,p)$. According to theorem 7.0.2, let us also introduce :

(8.4.0.2) $\underline{F} \in [E\to[A\to A]]$

$\underline{F} = \lambda\phi.[\rho\circ f(\phi)\circ\overline{\rho}]$

$= \lambda\phi.[\lambda\Psi.[\lambda i.[\lambda j.[\lambda(x,c).[[j=1 \wedge (\forall k\in[1,\underline{\pi}],c(k)=\underline{\lambda}(k,1))\wedge \phi(x)]$

$\vee[\exists n\in[1,\underline{\sigma}(i)]:Post(i)(n,j)(\Psi(i)(n)\wedge Cont\{i\}(\Psi))(x,c)]$

$\vee[c(i)=\underline{\lambda}(i,j) \wedge \exists m\in[1,\underline{\pi}]-\{i\},n\in[1,\underline{\sigma}(m)],p\in[1,\underline{\sigma}(m)+1]:$

$Post(m)(n,p)(\Psi(i)(j)\wedge \Psi(m)(n) \wedge Cont\{i,m\}(\Psi))(x,c)]]]]]]$

where

$Cont \in [2^{[1,\underline{\pi}]}\to[A\to[[\underline{t}x\underline{L}]\to B]]]$

$Cont = \lambda X.[\lambda\Psi.[\lambda(x,c).[\forall k\in[1,\underline{\pi}]-X,\exists\ell\in[1,\underline{\sigma}(k)+1]:c(k)=\underline{\lambda}(k,\ell) \wedge \Psi(k)(\ell)(x,c)]]]$

Let us define :

(8.4.0.3) $\overline{F} \in [E\to[A\to A]]$

$\overline{F} = \lambda\phi.[\lambda\Psi.[\lambda i.[\lambda j.[\lambda(x,c).[[j=1 \wedge (\forall k\in[1,\underline{\pi}],c(k)=\underline{\lambda}(k,1)) \wedge \phi(x)]$

$\vee[\exists n\in[1,\underline{\sigma}(i)]:Post(i)(n,j)(\Psi(i)(n))(x,c)]$

$\vee[c(i)=\underline{\lambda}(i,j) \wedge \exists m\in[1,\underline{\pi}]-\{i\},n\in[1,\underline{\sigma}(m)],p\in[1,\underline{\sigma}(m)+1]:$

$Post(m)(n,p)(\Psi(m)(n))(x,c)]]]]]]$

such that $\forall\phi\in E,\ \rho\circ f(\phi)=\overline{F}(\phi)\circ\rho$.

The invariance proof method corresponding to $\overline{F}$ is similar to the one introduced by Newton[75] using a quite different definition of concurrent programs. We have not found in the literature any invariance proof method similar to the one corresponding to $\underline{F}$. Since $\underline{F}\neq\overline{F}$ we have in fact introduced a complete

lattice of $F \in [E \to [A \to A]]$ such that $\underline{F} \Rightarrow F \Rightarrow \overline{F}$ each one corresponding to a particular invariance proof method. For example, Lamport[77]'s method as well as Owicki & Gries[76a]'s method (except for the use of auxiliary variables) correspond to :

(8.4.0.4) $\ddot{F} \in [E \to [A \to A]]$

$$\ddot{F} = \lambda\phi.[\lambda\Psi.[\lambda i.[\lambda j.[\lambda(x,c).[[j=1 \wedge (\forall k \in [1,\underline{\pi}], c(k)=\underline{\lambda}(k,1)) \wedge \phi(x)]$$
$$\vee[\exists n \in [1,\underline{\sigma}(i)]: \mathit{Post}(i)(n,j)(\Psi(i)(n))(x,c)]$$
$$\vee[c(i)=\underline{\lambda}(i,j) \wedge \exists m \in [1,\underline{\pi}]-\{i\}, n \in [1,\underline{\sigma}(m)], p \in [1,\underline{\sigma}(m)+1]:$$
$$\mathit{Post}(m)(n,p)(\Psi(i)(j) \wedge \Psi(m)(n))(x,c)]]]]]]$$

The first term corresponds to initialization, the second term corresponds to a sequential proof and the third term to an interference-freeness proof. This last term disappears when considering monoprocess programs ($\underline{\pi}=1$) or multiprocess programs with assertions about parts of the store such that only operations acting on separate parts may be performed concurrently (e.g. Hoare[75] Mazurkiewicz[77]).

Example 8.4.0.5 : Let us consider the following program with assertions about program counters c and variables x :

x:integer;
{$\phi(x)$}
[1: {$\Psi(1)(1)(x,c)$} await *true* then x:=x+1; 2: {$\Psi(1)(2)(x,c)$} stop
||1: {$\Psi(2)(1)(x,c)$} await *true* then x:=x+2; 2: {$\Psi(2)(2)(x,c)$} stop]

The verification condition $(\overline{F}(\phi)(\Psi) \Rightarrow \Psi)$ requires the verification of the following formulas for all $c \in \{\underline{1},\underline{2}\} \times \{\underline{1},\underline{2}\}$, $x \in \underline{\text{integer}}$:

Initialization :

$\Psi(1)(1)(x,c) \Leftarrow [c(1)=\underline{1} \wedge c(2)=\underline{1} \wedge \phi(x)]$

$\Psi(2)(1)(x,c) \Leftarrow [c(1)=\underline{1} \wedge c(2)=\underline{1} \wedge \phi(x)]$

Sequential proof :

$\Psi(1)(2)(x,c) \Leftarrow \mathit{Post}(1)(1,2)(\Psi(1)(1))(x,c)$

$\Psi(2)(2)(x,c) \Leftarrow \mathit{Post}(2)(1,2)(\Psi(2)(1))(x,c)$

(If the precondition of a statement is true and this statement is executed then its postcondition must be true).

Interference-freeness proof :

$\Psi(1)(1)(x,c) \Leftarrow [c(1)=\underline{1} \wedge \mathit{Post}(2)(1,2)(\Psi(2)(1))(x,c)]$

$\Psi(1)(2)(x,c) \Leftarrow [c(1)=\underline{2} \wedge \mathit{Post}(2)(1,2)(\Psi(2)(1))(x,c)]$

$\Psi(2)(1)(x,c) \Leftarrow [c(2)=\underline{1} \wedge \mathit{Post}(1)(1,2)(\Psi(1)(1))(x,c)]$

$\Psi(2)(2)(x,c) \Leftarrow [c(2)=\underline{2} \wedge \mathit{Post}(1)(1,2)(\Psi(1)(1))(x,c)]$

(Any assertion Ψ used in the proof of a process $\underline{P}(i)$ must be true after executing one step of any other process $\underline{P}(k)$).

Notice that this differs from the interference freeness proof of Lamport [77] and Owicki & Gries[76a] (i.e. $\ddot{F}(\phi)(\Psi)\Rightarrow\Psi$) which would be of the form :

$\Psi(1)(1)(x,c) \Leftarrow [c(1)=\underline{1} \wedge \mathit{Post}(2)(1,2)(\Psi(2)(1) \wedge \Psi(1)(1))(x,c)]$

...

(Any assertion ψ of process $\underline{P}(i)$ must be invariant under execution of any other process $\underline{P}(k)$. To prove this, it suffices to show that if control in $\underline{P}(k)$ is at some point whose assertion is true, *and* ψ *is true*, then executing the next statement in $\underline{P}(k)$ will leave ψ true).

These verification conditions also differ from the ones corresponding to $(\underline{F}(\phi)(\Psi)\Rightarrow\Psi)$. In that last case the sequential proof would be of the form :

$\Psi(1)(2)(x,c) \Leftarrow \mathit{Post}(1)(1,2)(\Psi(1)(1) \wedge \lambda(x,c).[(c(2)=\underline{1} \wedge \Psi(2)(1)(x,c))$
$\vee (c(2)=\underline{2} \wedge \Psi(2)(2)(x,c))](x,c)$

...

A final remark is that although all invariance proof methods are complete, it may be the case for some programs that some assertions can be proved invariant using $\underline{F}$ but cannot be proved invariant using $\ddot{F}$ (i.e. Lamport[77] and Owicki & Gries[76a] method) without having to guess stronger assertions. For the above example this would be the case of :

$\phi(x)=(x=0)$

$\Psi(1)(1)(x,c)=(\mathrm{even}(x))$ $\qquad$ $\Psi(1)(2)(x,c)=(x=1 \vee x=3)$

$\Psi(2)(1)(x,c)=(x=0 \vee x=1)$ $\qquad$ $\Psi(2)(2)(x,c)=(x=2 \vee x=3)$

End of Example.

8.5 USING A GLOBAL INVARIANT TOGETHER WITH LOCAL ASSERTIONS

Various invariance proof methods use a global invariant (e.g. the resource invariance of Hoare[72], Owicki & Gries[76b], the monitor invariant of Howard [76]) together with assertions interspread in the program text. Such a decomposition is defined by :

(8.5.0.1) $A = ([[\underline{t}x\underline{L}] \to B] \times \Pi\{\Pi\{[[\underline{t}x\underline{L}] \to B] : j \in [1,\underline{\sigma}(i)+1]\} : i \in [1,\underline{\pi}]\})$

$\rho = \lambda\omega.[(\omega, \lambda i.[\lambda j.[\lambda(x,c).[c(i)=\underline{\lambda}(i,j) \wedge \omega(x,c)]]])]$

$\overline{\rho} = \lambda(\omega,\Omega).[\lambda(x,c).[\omega(x,c) \wedge$
$(\forall i \in [1,\underline{\pi}], \exists j \in [1,\underline{\sigma}(i)+1]: c(i)=\underline{\lambda}(i,j) \wedge \Omega(i)(j)(x,c))]]$

Since ρ is one-to-one the corresponding proof method using $\underline{F}$ is complete (th. 7.0.2.(2-5)). Notice that all properties of the program can be expressed using only the global invariant. This disadvantage would disappear if in addition to the global variables of domain $\underline{t}$, each process $\underline{P}(i)$, $i\epsilon[1,\underline{\pi}]$ could have access to local variables of domain $\underline{t\ell}(i)$. A possible decomposition would then be :

(8.5.0.2) $A = [[\underline{t}\times\underline{L}]\rightarrow B]\times\Pi\{\Pi\{[[\underline{t}\times\underline{t\ell}(i)]\rightarrow B]:j\epsilon[1,\underline{\sigma}(i)+1]\}:i\epsilon[1,\underline{\pi}]\}$

$\rho \epsilon [[[\underline{t}\times\underline{t\ell}\times\underline{L}]\rightarrow B]\rightarrow A]$

$\rho = \lambda\omega.[(\lambda(x,c).[\exists x\ell\epsilon\underline{t\ell}: \omega(x,x\ell,c)],$

$\lambda i.[\lambda j.[\lambda(x,x\ell i).[\exists x\ell\epsilon\underline{t\ell}, c\epsilon\underline{L}: \omega(x,\mathit{subst}(x\ell)(i/x\ell i),c)]]])]$

so that the global assertion would be about global variables and program location counters whereas local assertions would be about global variables and visible local variables.

8.6 ADAPTING AN INVARIANCE PROOF METHOD TO NEW LANGUAGE FEATURES

All invariance proof methods amount to the same fundamental principle (th. 4.0.2) but variants can be introduced by different decompositions of the assertions about program states (th. 7.0.2). Hence in order to compare proof methods for different languages or to adapt known methods to new language features, it is sufficient to study the various possible decompositions. Let us consider for example Hoare[78]'s communicating sequential processes. The proof method of Apt, Francez & de Roever[79] is an adaptation of 8.5.0.1 whereas Levin[79] is an adaptation of 8.4.0.1. Both differ from Cousot & Cousot[80] which is based on yet another decomposition taking communication channels into account.

9. EXPRESSIVENESS

At paragraph 7 we have stated a sound and complete method $[\exists I\epsilon A : F(\phi)(I)\Rightarrow I \wedge I\Rightarrow\psi]$ for proving the invariance of $\Psi\epsilon A$ for any program $\underline{Pr}$. Invariance of Ψ was defined as $\rho(\mathit{Post}(t^*)(\varepsilon(\phi)))\Rightarrow\Psi$ or according to 6.0.5 equivalently as $\mathit{Post}(t^*)(\varepsilon(\phi))\Rightarrow\overline{\rho}(\Psi)$. This does not mean that using the above invariance proof method any $\psi\epsilon P$ can be proved invariant in the sense of paragraph 4 that is $\mathit{Post}(t^*)(\varepsilon(\phi))\Rightarrow\psi$. If A has not been chosen expressive enough there may exist no $\Psi\epsilon A$ such that $\overline{\rho}(\Psi)=\psi$.

Example 9.0.1 : The power of automatic program verification systems for proving the partial correctness of programs is limited by their fundamental incapability of guessing the invariant I and the fact that $\Rightarrow$ is not computable. The fact that I can always be chosen as $lfp(F(\phi))$ is of no help since $lfp(F(\phi))$ is not computable for all programs. In order to circumvent these undecidability results, the idea behind automatic program flow analysis (e.g. Cousot & Cousot [77], Cousot & Cousot[79a]) is to consider weak computer representable properties of programs for which $lfp(F(\phi))$ is either computable or approximable from above. This means that not all properties $\psi \in P$ of program $\underline{Pr}$ can be expressed by $\Psi \in A$ i.e. $\overline{\rho}$ is not onto. *End of Example.*

The scope of applicability of the various invariance proof methods discussed so far may be limited by the fact that some assertions of P cannot be expressed in A. This cannot be the case when $\overline{\rho}$ is onto. These results are better summarized by the following :

THEOREM 9.0.2

$\forall \underline{Pr} \in \underline{La}$, let $\langle A; \Rightarrow, \mathit{false}, \mathit{true}, \vee, \wedge \rangle$ be a complete lattice, ρ be a complete join-morphism of P into A, $\overline{\rho}$ be $\overline{\partial}(\rho)$, $\underline{F} \in [E \to [A \to A]]$ be $\lambda\phi.[\rho \circ f(\phi) \circ \overline{\rho}]$, $F, \overline{F} \in [E \to [A \to A]]$ be isotone operators such that $\forall \phi \in E$, $\rho \circ f(\phi) = \overline{F}(\phi) \circ \rho$ and $\underline{F} \Rightarrow F \Rightarrow \overline{F}$, then :

(1) $\forall \phi \in E, \forall \Psi \in A$, $[\rho(\mathit{Post}(t^*)(\varepsilon(\phi))) \Rightarrow \Psi] \Leftrightarrow [\exists I \in A : F(\phi)(I) \Rightarrow I \wedge I \Rightarrow \Psi]$

(2) If moreover ρ is one-to-one or $\overline{\rho}$ is onto or $\overline{\rho} \circ \rho = 1$ then $\forall \phi \in E$, $\forall \psi \in P$, $[\mathit{Post}(t^*)(\varepsilon(\phi)) \Rightarrow \psi] \Leftrightarrow [\exists I \in A : F(\phi)(I) \Rightarrow I \wedge I \Rightarrow \rho(\psi)]$

10. REDUCTION - EXTENSION

A common idea for proving properties of a program consists in reasoning about a slightly transformed but equivalent program. This is the case when using auxiliary variables in invariance proofs. This notion can be formalized as a reduction of dynamic discrete transition systems.

(10.0.1) Let Sr, Se be sets and $tr \in [[Sr \times Sr] \to B]$, $te \in [[Se \times Se] \to B]$ be transition relations. We say that (Sr,tr) is a σ-reduction of (Se,te) if and only if there exists a total map σ from Sr into $(2^{Se} - \emptyset)$ such that $(\forall x, y \in Sr, (x \neq y) \Rightarrow (\sigma(x) \cap \sigma(y) \neq \emptyset))$ and $tr^* = R(te^*)$.

Where

(10.0.2) $R \in [[[Se \times Se] \to B] \to [[Sr \times Sr] \to B]]$

$R = \lambda\theta.[\lambda(x,y).[\exists X,Y \in Se: (X \in \sigma(x)) \wedge (Y \in \sigma(y)) \wedge \theta(X,Y)]]$

$\overline{R} \in [[[Sr \times Sr] \to B] \to [[Se \times Se] \to B]]$

$\overline{R} = \lambda\theta.[\lambda(X,Y).[\forall x,y \in Sr, (X \in \sigma(x) \wedge Y \in \sigma(y)) \Rightarrow \theta(x,y)]]$

Notice that $(R,\overline{R})$ is a Galois connection between $[[Se \times Se] \to B]$ and $[[Sr \times Sr] \to B]$ and R is onto. Let us define $Pr=[Sr \to B]$ and $Pe=[Se \to B]$ and introduce :

(10.0.3) $r \in [Pe \to Pr]$

$r = \lambda\Omega.[\lambda x.[\exists X \in Se: (X \in \sigma(x)) \wedge \Omega(X)]]$

$\overline{r} \in [Pr \to Pe]$

$\overline{r} = \lambda\omega.[\lambda X.[\forall x \in Sr, (X \in \sigma(x)) \Rightarrow \omega(x)]]$

$(r,\overline{r})$ is a Galois connection between Pe and Pr and r is onto. Therefore $(\tilde{\overline{r}},\tilde{r})$ is a galois connection between Pr and Pe and $\tilde{r}$ is onto (6.0.11-12). The following theorem states that reachability properties are preserved by reduction whence one can prove invariance properties of (Sr,tr) by reasoning about its extension (Se,te) :

THEOREM 10.0.4

If (Sr,tr) is a reduction of (Se,te) then :

(1) $\forall \phi \in Pr,\ Post(tr^*)(\phi) = r(Post(te^*)(\tilde{\overline{r}}(\phi)))$

(2) $\forall \phi \in Pr,\ \forall \Psi \in Pe,\ [Post(te^*)(\tilde{\overline{r}}(\phi)) \Rightarrow \Psi] \Rightarrow [Post(tr^*)(\phi) \Rightarrow r(\Psi)]$

Example 10.0.5 : "Soundness of the auxiliary variables transformation" Let Pr be a program and Pra be the same program augmented with auxiliary variables which can only appear in assignments to auxiliary variables. This means that $\forall i \in [1,\pi]$, there exists a total one-to-one map $\lambda(i)$ from the set $L(i)$ of labels of process $P(i)$ into the set $La(i)$ of labels of the corresponding process $Pa(i)$ such that commands of $Pa(i)$ designated by a label belonging to $La(i)-\lambda(i)(L(i))$ are assignments to auxiliary variables whereas commands labeled $\lambda(i,j)$ in $P(i)$ and commands labeled $\lambda(i)(\lambda(i,j))$ in $Pa(i)$ are the same (in particular a test command $\lambda(i,j)$:if $b(i,j)(x)$ go to $\lambda(i,\eta(i,j))$ of $P(i)$ is transformed into $\lambda(i)(\lambda(i,j))$:if $b(i,j)(x)$ go to $\lambda(i)(\lambda(i,\eta(i,j)))$. Let t (resp. ta) be the domain of the main (resp. auxiliary) variables. Let S be $t \times L$ and Sa be $t \times ta \times La$. Define $\sigma \in [S \to (2^{Sa}-\emptyset)]$ as $\lambda(x,c).[\{(x,xa,\lambda(c)) : xa \in ta\}]$ and call $te \in [[S \times S] \to B]$ and $ta \in [[Sa \times Sa] \to B]$ the respective transition relations of the original and the transformed program

as defined at 2.0.1. Then (S,t) is a σ-reduction of (Sa,ta) so that, according to theorem 10.0.4, one can prove invariance properties of the original program by reasoning about the transformed program.

In particular, given a program Pr, an entry specification ϕ and assertions $\Psi \in A$ (where $A=\Pi\{\Pi\{[t \to B]: j \in [1,\sigma(i)+1]\}: i \in [1,\pi]\}$), Owicki & Gries[76a] invariance proof method consists in guessing a transformed program Pra and assertions $\Psi a \in Aa$ (where $Aa=\Pi\{\Pi\{[t \times ta \to B]: j \in [1,\sigma a(i)+1]\}: i \in [1,\pi]\}$) such that $e(\Psi a) \Rightarrow \Psi$ where $e=\lambda \Psi a.[\lambda i.[\lambda j.[\lambda x.[\exists xa \in ta, k \in [1,\sigma a(i)+1]: \lambda(i)(\lambda(i,j))=\lambda(i,k) \wedge \Psi a(i)(k)(x,xa)]]]]$ and which can be proved invariant for Pra using theorem 9.0.2.1 and decomposition 8.3.0.1 (hence without refering to program location counters).

This invariance proof method is sound since $\rho a[Post(ta^*)(\tilde{\overline{r}}(\varepsilon(\phi)))] \Rightarrow \Psi a$ implies $Post(t^*)(\varepsilon(\phi)) \Rightarrow r \circ \overline{\rho a}(\Psi a) \Rightarrow \overline{\rho}(\Psi)$ by 6.0.5, 10.0.4.1, $\rho \circ r \circ \overline{\rho a} \Rightarrow e$, 6.0.5 and therefore $\rho(Post(t^*)(\varepsilon(\phi))) \Rightarrow \Psi$. *End of Example.*

Example 10.0.6 : "Completeness of the auxiliary variables transformation" The proof consists in showing that given $Pr \in La$, $\phi \in E$, $\Psi \in A$ such that $\rho(Post(t^*)(\varepsilon(\phi))) \Rightarrow \Psi$, there exist $Pra \in La$, $\Psi a \in Aa$ such that Ψa can be shown invariant for Pra without refering to program location counters (i.e. $\rho a(\tilde{\overline{r}}(\varepsilon(\phi)) \vee Post(ta)(\overline{\rho a}(\Psi a))) \Rightarrow \Psi a$), $e(\Psi a)$ is invariant for the original program (i.e. $Post(t^*)(\varepsilon(\phi)) \Rightarrow \overline{\rho}(e(\Psi a))$ and $e(\Psi a) \Rightarrow \Psi$. Pra is constructed by adding one auxiliary variable xa(i) to each process P(i) taking its values in $[1,\sigma(i)+1]$ (or any other domain isomorphic with L(i)). To any command $\lambda(i,j):C(i,j)$ of Pr correspond commands $\lambda(i,2j-1):C(i,j);\lambda(i,2j):xa(i):=j$ of Pra (in particular $\lambda(i,j):$ **if** $b(i,j)(x)$ **go to** $\lambda(i,\eta(i,j))$ is transformed into $\lambda(i,2j-1):$ **if** $b(i,j)(x)$ **go to** $\lambda(i,2\eta(i,j)-1);\lambda(i,2j):xa(i):=j$) so that auxiliary variables simulate program location counters. Ψa is chosen as $\rho a(Post(ta^*)(\overline{r}(\varepsilon(\phi))))$. *End of Example.*

Example 10.0.7 : Let us propose answers to the open problems of Clarke [80]. First "is there a proof system similar to the one originally described by Owicki that does not require the use of history variables?". The answer is clearly yes when choosing the decomposition 8.4.0.1 instead of 8.3.0.1 i.e. when using program location counters. Although this alternative may be found inelegant, this result at least shows that proliferation of auxiliary variables (e.g. Gries[79]) can be avoided. More generally "are history variables necessary for formal verification of concurrent programs?". As shown by theorem 9.0.2 the answer only depends on the decomposition ρ which is chosen.

There is always one for which the answer is no (e.g. 8.2.0.1) and one for which the answer is yes (e.g. 8.3.0.1). *End of Example.*

11. CONCLUSION

We have studied program proof methods in abstracto, using a mathematical model based on dynamic discrete transition systems, a fixpoint characterization of correctness properties, a decomposition of assertions using Galois connections and a reduction of transition systems. This approach is quite general and was illustrated by means of an application to invariance proof techniques for parallel programs sharing global variables and synchronized by conditional critical sections (the case of concurrent programs designed as networks of nondeterministic sequential processes, communicating with each other explicitly, by the sole means of synchronous unbuffered message passing is treated along the same lines in Cousot & Cousot[80]). Although only invariance properties have been considered, other properties of programs such as liveness are amenable to the same formalization.

The model of dynamic discrete transition systems has already been used in Keller[76], Pnuelli[77], Rosen[77], Abrial & Schuman[79] but the connection between this highly abstract conceptual model and the means of presenting programs is left unspecified, in particular the idea of decomposition is absent. This idea appears in Cousot[79] (as partitionned dynamic discrete systems) and in Cousot & Cousot[79a] (in a different although related context). The observation that correctness properties of programs can be characterized as fixpoints (Clarke[77], Cousot & Cousot[77], Flon & Suzuki[78], Van Lamsweerde & Sintzoff[79]) can be traced back to Park[69]. Our notion of reduction/extension of dynamic discrete transition systems is related to Lipton[75]'s reduction of parallel programs. Lipton's intuitive notion of reduction has been formalized by Kwong[77] but his definitions are too restrictive for formalizing the use of auxiliary variables.

Program logics are alternate mathematical formalisms for reasoning about program proof methods (e.g. Pratt[76], Hoare[78a], Salwicki[78]). They are well-suited for decidability and computability results. However, these logics are tailored to particular languages (e.g. assignment, test, sequence, iteration, non-deterministic choice) and syntax-directed. Therefore a particular decomposition of the assertions about program states is chosen once for all.

There is always one for which the answer is no (e.g. 8.2.0.1) and one for which the answer is yes (e.g. 8.3.0.1). *End of Example.*

11. CONCLUSION

We have studied program proof methods in abstracto, using a mathematical model based on dynamic discrete transition systems, a fixpoint characterization of correctness properties, a decomposition of assertions using Galois connections and a reduction of transition systems. This approach is quite general and was illustrated by means of an application to invariance proof techniques for parallel programs sharing global variables and synchronized by conditional critical sections (the case of concurrent programs designed as networks of nondeterministic sequential processes, communicating with each other explicitly, by the sole means of synchronous unbuffered message passing is treated along the same lines in Cousot & Cousot[80]). Although only invariance properties have been considered, other properties of programs such as liveness are amenable to the same formalization.

The model of dynamic discrete transition systems has already been used in Keller[76], Pnuelli[77], Rosen[77], Abrial & Schuman[79] but the connection between this highly abstract conceptual model and the means of presenting programs is left unspecified, in particular the idea of decomposition is absent. This idea appears in Cousot[79] (as partitionned dynamic discrete systems) and in Cousot & Cousot[79a] (in a different although related context). The observation that correctness properties of programs can be characterized as fixpoints (Clarke[77], Cousot & Cousot[77], Flon & Suzuki[78], Van Lamsweerde & Sintzoff[79]) can be traced back to Park[69]. Our notion of reduction/extension of dynamic discrete transition systems is related to Lipton[75]'s reduction of parallel programs. Lipton's intuitive notion of reduction has been formalized by Kwong[77] but his definitions are too restrictive for formalizing the use of auxiliary variables.

Program logics are alternate mathematical formalisms for reasoning about program proof methods (e.g. Pratt[76], Hoare[78a], Salwicki[78]). They are well-suited for decidability and computability results. However, these logics are tailored to particular languages (e.g. assignment, test, sequence, iteration, non-deterministic choice) and syntax-directed. Therefore a particular decomposition of the assertions about program states is chosen once for all.

Hence our study of decompositions would be very difficult with this formalism. Moreover reasonings by induction on the syntactical structure of programming languages have several drawbacks when context-dependencies are involved. For example Milne[77] shows that continuations may be necessary in predicate-transformers. Notice that these context-dependencies don't only come from spaghetti-like languages. For example Owicki's interference-freeness is a context-sensitive notion that might be difficult to formalize using program logics as they now stand.

12. REFERENCES

Abrial J.R. & Schuman S.A. [1979], *Non-deterministic system specification*, Lectures Notes in Comp. Sci. 70, Springer-Verlag, Berlin, (1979), 34-50.

Apt K.R. [1979], *Ten years of Hoare logic, a survey*, Faculté of Economics, Erasmus Univ., Rotterdam, The Netherlands, (April 1979).

Apt K.R., Francez N. & de Roever [1979], *A proof system for communicating sequential processes*, Techn. Report RUU-CS-79-8, Vakgroep Informatica, Rijksuniversiteit Utrecht, The Netherlands, (Aug. 1979).

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

Ashcroft E.A. & Manna Z. [1970], *Formalization of properties of parallel programs*, Machine Intelligence, 6(1970), 17-41.

Birkhoff G. [1940], *Lattice Theory*, AMS Colloquium Pub., vol.25, First Ed., Providence, R.I., USA, (1940).

Clarke E.M. Jr. [1977], *Program invariants as fixpoints*, Proc. 18th annual Symp. on Foundations of Comp. Sci., Providence, R.I., USA, (Oct.31-Nov.2 1977), 18-29.

Clarke E.M. Jr. [1980], *Proving correctness of coroutines without history variables*, Acta Informatica, 13(1980), 169-188.

Clint M. [1973], *Program proving : coroutines*, Acta Informatica, 2(1973), 50-63.

Cousot P. [1979], *Analysis of the behavior of dynamic discrete systems*, Research Rep. 161, Lab. IMAG, Univ. of Grenoble, France, (Jan.1979).

Cousot P. & Cousot R. [1977], *Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Conf. Rec. of the 4th ACM Symp. on Principles of Prog. Lang., Los Angeles, Calif., USA, (Jan.1977), 238-252.

Cousot P. & Cousot R. [1979a], *Systematic design of program analysis frame-works*, Conf. Rec. of the 6th ACM Symp. on Principles of Prog. Lang., San Antonio, Texas, USA, (Jan.1979), 269-282.

Cousot P & Cousot R. [1979b], *Constructive versions of Tarski's fixed point theorems*, Pacific Journal of Math., 82(1979), 43-57.

Cousot P. & Cousot R. [1980], *Semantic analysis of communicating sequential processes*, Proc. 7th Int. Colloquium on Automata, Languages and Programming, ICALP80, Noordwijkerhout, The Netherlands, (July 14-18 1980), Lectures Notes in Comp. Sci., Springer Verlag.

Dubrueil P. & Croisot R. [1954], *Propriétés générales de la résiduation en liaison avec les correspondances de Galois*, Collect. Math., 7(1954), 193-203.

Flon L. & Suzuki N. [1978], *Consistent and complete proof rules for the total correctness of parallel programs*, CSL-78-6, Xerox-Parc, Palo Alto, CA., USA, (Nov.1978).

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

Gries D. [1979], *Yet another exercise : using two shared variables in two processes to provide starvation-free mutual exclusion*, TR-79-372, Dept. of Comp. Sci., Cornell Univ., N.Y., USA, (1979).

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

Hoare C.A.R. [1972], *Toward a theory of parallel programming*, in 'Operating Systems Techniques', Hoare and Perott Eds., Academic Press, N.Y., (1972).

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

Hoare C.A.R. [1978a], *Some properties of predicate transformers*, JACM 25, 3(July 1978), 461-480.

Hoare C.A.R. [1978b], *Communicating sequential processes*, CACM 21, 8(Aug.1978) 666-677.

Howard J.H. [1976], *Proving monitors*, CACM 19, 5(May 1976), 273-279.

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

Kwong Y.S. [1977], *On reduction of asynchronous systems*, Theoretical Comp. Sci. 5(1977), 25-50.

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

Levin G.M. [1979], *A proof technique for communicating sequential processes (with an example)*, TR79-401, Comp. Sci. Dept., Cornell Univ., N.Y.,(1979).

Lipton R.J. [1975], *Reduction : a method of proving properties of parallel programs*, CACM 18, 12(Dec. 1978), 717-721.

Mazurkiewicz A.[1977], *Invariants of concurrent programs*, Int. Conf. on Information Processing, IFIP-INFOPOL-76, J. Madey Ed., North-Holland Pub. Co., (1977), 353-372.

Milne R. [1977], *Transforming predicate transformers*, IFIP WG.2.2 working conf. on Formal Description of Programming Concepts, St-Andrews, NB, Canada North-Holland Pub. Co., (Aug.1977).

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

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

Ore O. [1944], *Galois connections*, Trans. Amer. Math. Soc., 55(1944), 493-513.

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

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

Park D. [1969], *Fixpoint induction and proofs of program properties*, Machine Intelligence, 5(1969), 59,78.

Pickert G. [1952], *Bemerkungen über Galois-Verbindungen*, Arch. Math. J., 3(1952), 285-289.

Pnueli A. [1977], *The temporal logic of programs*, Proc. 18th Annual Symp. on Foundations of Comp. Sci., Providence, R.I., USA, (Oct.31-Nov.2 1977), 46-57.

Pratt V.R. [1976], *Semantical considerations on Floyd-Hoare logic*, Proc. 17th IEEE Symp. on Foundations of Comp. Sci., (Oct.1976), 109-121.

Rosen B.K. [1976], *Correctness of parallel programs : the Church-Rosser approach*, Theoretical Comp. Sci., 2(1976), 183-207.

Salwicki A. [1978], *On algorithmic logic and its applications*, Math. Inst., Polish Acad. of Sci., Warsaw, Poland, (Nov.1978).

Schmidt J. [1953], *Beiträge zur filtertheorie II*, Math. Nachr., 10(1953), S.205.

Van Lamsweerde A. & Sintzoff M. [1979], *Formal derivation of strongly correct concurrent programs*, Acta Informatica, 12(1979), 1-31.

Zervos S.P. [1971], *Lattices and topology*, General topology and its relations to modern analysis and algebra III, Proc. 3rd Prague Topological Symp., (1971), 475-484.

13. APPENDIX

13.1 Proof of Theorem 7.0.2

(1) - By fixpoint property $\rho\circ f\circ\overline{\rho}(lfp(\underline{F}))=lfp(\underline{F})$ hence $f\circ\overline{\rho}(lfp(\underline{F}))\Rightarrow\overline{\rho}(lfp(\underline{F}))$ by 6.0.5 so that $lfp(f)\Rightarrow\overline{\rho}(lfp(\underline{F}))$ by Tarski's fixpoint theorem proving that $\rho(lfp(f))\Rightarrow lfp(\underline{F})$ by 6.0.5.

- Let P={a,b,c,d} be a complete lattice with strict ordering a<b<d and a<c<d. Let A={e,g,h} be a complete lattice with strict ordering e<g<h. Define $\rho(a)=\rho(c)=e$, $\rho(b)=\rho(d)=g$, $\overline{\rho}(e)=c$, $\overline{\rho}(g)=\overline{\rho}(h)=d$, f(a)=a and f(b)=f(c)=f(d)=b. We have $\underline{F}(e)=\underline{F}(g)=\underline{F}(h)=g$ so that $\rho(lfp(f))=e<g=lfp(\underline{F})$.

(2) - If ρ is one-to-one then $\overline{\rho}\circ\rho=1$ and $\underline{F}\circ\rho=\rho\circ f$ so that $\rho(lfp(f))=lfp(\underline{F})$ by (4) below.

-If P=A={a,b} with a<b and $f=\underline{F}=\rho=\lambda x.[a]$, $\overline{\rho}=\lambda x.[b]$ then $\rho(lfp(f))=lfp(\underline{F})$ although ρ is not one-to-one.

(3) - $\underline{F}=\rho\circ f\circ\overline{\rho}=\overline{F}\circ\rho\circ\overline{\rho}\leq\overline{F}$ since $\rho\circ\overline{\rho}\leq 1$. The inequality can be strict as shown by the following example : P=A={a,b}, a<b, $\rho(a)=\rho(b)=a$, $\overline{\rho}(a)=\overline{\rho}(b)=b$, $f=\overline{F}=1$, $\underline{F}(a)=\underline{F}(b)=a$. If ρ is onto then $\rho\circ\overline{\rho}=1$ so that $\underline{F}=\overline{F}$ but the reciprocal is not true as shown by example (2) when $\overline{F}=\underline{F}$.

(4) - Let X (resp. Y) be the upper iteration sequence for $\overline{F}$ (resp. f) starting with *false* i.e. $X(0)=false$, $X(\delta)=\overline{F}(\delta-1))$ for every successor ordinal δ and $X(\delta)=\vee\{X(\alpha):\alpha<\delta\}$ for every limit ordinal δ (Cousot & Cousot[79b]). ρ is *false*-strict whence $X(0)=\rho(Y(0))$. Assume $X(\alpha)=\rho(Y(\alpha))$ for every ordinal $\alpha<\delta$. If δ is a successor ordinal then $X(\delta)=\overline{F}(X(\delta-1))=\overline{F}(\rho(Y(\delta-1)))=\rho(f(Y(\delta-1)))=\rho(Y(\delta))$. If δ is a limit ordinal then $X(\delta)=\vee\{X(\alpha):\alpha<\delta\}=\vee\{\rho(Y(\alpha)):\alpha<\delta\}=\rho(\vee\{Y(\alpha):\alpha<\delta\})=\rho(Y(\delta))$. By transfinite induction $X(\delta)=\rho(Y(\delta))$ for every ordinal δ. According to Cousot & Cousot[79b,th.3.3] there exist ordinals $\varepsilon 1$ and $\varepsilon 2$ such that $lfp(\underline{F})=X(\varepsilon 1)=X(max(\varepsilon 1,\varepsilon 2))=\rho(Y(max(\varepsilon 1,\varepsilon 2)))=\rho(Y(\varepsilon 2))=\rho(lfp(f))$. Moreover $\underline{F}\Rightarrow F\Rightarrow\overline{F}$ implies $lfp(\underline{F})\Rightarrow lfp(F)\Rightarrow lfp(\overline{F})=\rho(lfp(f))$ and also $\rho(lfp(f))\Rightarrow lfp(\underline{F})$ by (1) so that equality holds.

(5) - If $\rho(lfp(f))\Rightarrow lfp(F)$ and $\Psi\in A$ such that $[\exists I\in A:F(I)\Rightarrow I\wedge I\Rightarrow\Psi]$ then $\rho(lfp(f))\Rightarrow lfp(F)=\wedge\{I\in A:F(I)\Rightarrow I\}\Rightarrow\Psi$ by Tarski's fixpoint theorem. Reciprocally if $\rho(lfp(f))=lfp(F)$ and $\rho(lfp(f))\Rightarrow\Psi$ then $F(lfp(F))=lfp(F)$ and $lfp(F)\Rightarrow\Psi$.

13.2 Proof of Theorems 3.0.1, 4.0.2 and 9.0.2

- If $\theta 1,\ \theta 2\in[[S\times S]\to B]$ then $\theta 1\circ\theta 2=\lambda(sa,sb).[\exists s\in S:\theta 1(sa,s)\wedge\theta 2(s,sb)]$. Let $eq\in[[S\times S]\to B]$ be $\lambda(sa,sb).[sa=sb]$. We have $t^*=lfp(\lambda\theta.[eq\vee\theta\circ t])$. $\forall\phi\in E$, $\lambda\theta.[Post(\theta)(\varepsilon(\phi))]$ is a complete $\vee$-morphism from the complete lattice $[[S\times S]\to B]$ into the complete lattice P and $\lambda\theta.[Post(\theta)(\varepsilon(\phi))]\circ\lambda\theta.[eq\vee\theta\circ t]=f(\phi)\circ\lambda\theta.[Post(\theta)(\varepsilon(\phi))]$ proving that $Post(t^*)(\varepsilon(\phi))=lfp(f(\phi))$ by theorem 7.0.2.(4).

- Theorems 4.0.2 and 9.0.2.(1) follow from 3.0.1 and 7.0.2.(5).

- By Tarski's fixpoint theorem and theorem 7.0.2.(4), $[\exists I\in\underline{A}:F(\phi)(I)\Rightarrow I\wedge I\Rightarrow\rho(\psi)]\Leftrightarrow[lfp(F(\phi))\Rightarrow\rho(\psi)]\Leftrightarrow[\rho(lfp(f(\phi)))\Rightarrow\rho(\psi)]\Leftrightarrow[lfp(f(\phi))=\rho\circ\rho(lfp(f(\phi)))\Rightarrow\rho\circ\rho(\psi)=\psi]$ by isotony and 6.0.10.

13.3 Proof of Theorem 10.0.4

$\forall\phi\in Pr$, $\underline{Post}(tr^*)(\phi)=Post(R(te^*))(\phi)=\lambda x.[\exists y\in Sr:\phi(y)\wedge\{\exists X\in\sigma(x),Y\in\sigma(y):te^*(Y,X)\}]=\lambda x.[\exists Y\in Se:\widetilde{r}(\phi)(Y)\wedge\{\exists X\in\sigma(x):te^*(Y,X)\}]=r(Post(te^*)(\widetilde{r}(\phi)))$.