Lecture Notes in
Computer Science

Edited by G. Goos and J. Hartmanis

85

Automata, Languages and Programming

Seventh Colloquium
Noordwijkerhout, the Netherlands
July 14–18, 1980

Edited by
J. W. de Bakker and J. van Leeuwen

Springer-Verlag
Berlin Heidelberg New York 1980

SEMANTIC ANALYSIS OF COMMUNICATING SEQUENTIAL PROCESSES

(Shortened Version)

Patrick Cousot* and Radhia Cousot**

1. *INTRODUCTION*

We present *semantic analysis techniques for concurrent programs* which are designed as networks of nondeterministic sequential processes, communicating with each other explicitly, by the sole means of synchronous, unbuffered message passing. The techniques are introduced using a version of Hoare[78]'s programming language CSP (*Communicating Sequential Processes*).

One goal is to propose an *invariance proof method* to be used in the development and verification of correct programs. The method is suitable to *partial correctness*, *absence of deadlock* and *non-termination proofs*. The design of this proof method is formalized so as to prepare the way to possible alternatives.

A complementary goal is to propose an *automatic technique for gathering information about CSP programs* that can be useful to both optimizing compilers and program partial verification systems.

2. *SYNTAX AND OPERATIONAL SEMANTICS*

2.1 *Syntax*

The set sCSP of syntactically valid programs is informally defined so as to capture the essential features of CSP.

- Programs $\underline{Pr}$: $[\underline{P}(1) \parallel \underline{P}(2) \parallel \ldots \parallel \underline{P}(\underline{\pi})]$ where $\underline{\pi} \geq 2$
  (A program consists of a single parallel command specifying concurrent execution of its constituent disjoint processes).
- Processes $\underline{P}(i)$, $i \in [1,\underline{\pi}]$: $\underline{P\ell}(i) :: \underline{D}(i);\underline{\lambda}(i,1):\underline{S}(i)(1);\ldots;\underline{\lambda}(i,\underline{\sigma}(i)):\underline{S}(i)(\underline{\sigma}(i))$
  where $\underline{\sigma}(i) \geq 1$
  (Each process $\underline{P}(i)$ has a unique name $\underline{P\ell}(i)$ and consists of a sequence of simple commands prefixed with declarations $\underline{D}(i)$ of local variables).
- Process labels $\underline{P\ell}(i)$, $i \in [1,\underline{\pi}]$.
- Declarations $\underline{D}(i)$, $i \in [1,\underline{\pi}]$: $\underline{x}(i)(1):\underline{t}(i)(1);\ldots;\underline{x}(i)(\underline{\delta}(i)):\underline{t}(i)(\underline{\delta}(i))$ where $\underline{\delta}(i) \geq 1$.
- Variables $\underline{x}(i)(j)$, $i \in [1,\underline{\pi}]$, $j \in [1,\underline{\delta}(i)]$.
- Types $\underline{t}(i)(j)$, $i \in [1,\underline{\pi}]$, $j \in [1,\underline{\delta}(i)]$.
- Program locations $\underline{\lambda}(i,j)$, $i \in [1,\underline{\pi}]$, $j \in [1,\underline{\sigma}(i)]$.
  (Each command has been labeled to ease future references).
- Simple commands $\underline{S}(i)(j)$, $i \in [1,\underline{\pi}]$, $j \in [1,\underline{\sigma}(i)]$:
  - . Null commands $\underline{S}(i)(j)$, $i \in [1,\underline{\pi}]$, $j \in \underline{N}(i)$: $\underline{\text{skip}}$
  - . Assignment commands $\underline{S}(i)(j)$, $i \in [1,\underline{\pi}]$, $j \in \underline{A}(i)$: $\underline{x}(i)(\underline{\alpha}(i,j)) := \underline{e}(i,j)(\underline{x}(i))$
    where $\underline{\alpha}(i,j) \in [1,\underline{\delta}(i)]$

---

* 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 INRIA (SESORI-78208) and by CNRS (ATP Intelligence Artif.).

(The pattern-matching feature introduced in Hoare[78] is treated using dynamic type checking. Multiple assignments or assignments to parts of structured variables are realized using global assignments to variables).

- Test commands $\underline{S}(i)(j)$, $i\in[1,\underline{\pi}]$, $j\in\underline{T}(i)$:
  $\underline{\text{if}}\ \underline{b}(i,j)(\underline{x}(\overline{i}))\ \underline{\text{go}}\ \underline{\text{to}}\ \underline{\lambda}(i,\overline{\underline{n}(i,j)})$ where $\underline{n}(i,j)\in[1,\underline{\sigma}(i)]$.
- Stop commands $\underline{S}(i)(j)$, $i\in[1,\underline{\pi}]$, $j\in\underline{H}(i)$: $\underline{\text{stop}}$
  (Specify termination of process $\overline{\underline{P}}(i)$).
- Communication commands $\underline{S}(i)(j)$, $i\in[1,\underline{\pi}]$, $j\in\underline{C}(i)$:
  $[\underline{G}(i,j,1)\rightarrow\underline{\lambda}(i,\underline{n}(i,j,1))\ \square\ \ldots\ \square\ \underline{G}(i,j,\underline{\gamma}(i,j))\rightarrow\underline{\lambda}(i,\underline{n}(i,j,\underline{\gamma}(i,j)))]$
  where $(\underline{\gamma}(i,j)\geq 1)\wedge(\forall k\in[1,\underline{\gamma}(i,j)],\ \underline{n}(i,j,k)\in[1,\underline{\sigma}(i)])$
  (The execution of the command $\underline{S}(i)(j)$ is delayed until one arbitrary but successfully executable input-output guard $\underline{G}(i,j,k)$ $(k\in[1,\underline{\gamma}(i,j)])$ is selected and executed. Next the command labeled $\underline{\lambda}(i,\underline{n}(i,j,k))$ is executed. If all input-output guards fail the process $\underline{P}(i)$ fails in deadlock).

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

- Input-Output guards $\underline{G}(i,j,k)$, $i\in[1,\underline{\pi}]$, $j\in\underline{C}(i)$, $k\in[1,\underline{\gamma}(i,j)]$:
  - Input guards $\underline{G}(i,j,k)$, $i\in[1,\underline{\pi}]$, $j\in\underline{C}(i)$, $k\in\underline{I}(i,j)$:
    $\underline{b}(i,j,k)(\underline{x}(i));\underline{P\ell}(\theta(i,j,k))?\underline{x}(i)(\underline{\alpha}(i,j,k))$
    where $(\underline{\theta}(i,j,k)\in([1,\underline{\pi}]-\{i\}))\wedge(\underline{\alpha}(i,j,k)\in[1,\underline{\delta}(i)])$.
  - Output guards $\underline{G}(i,j,k)$, $i\in[1,\underline{\pi}]$, $j\in\underline{C}(i)$, $k\in\underline{O}(i,j)$:
    $\underline{b}(i,j,k)(\underline{x}(i));\underline{P\ell}(\underline{\theta}(i,j,k))!\underline{e}(i,j,k)(\underline{x}(i))$ where $\underline{\theta}(i,j,k)\in([1,\underline{\pi}]-\{i\})$.

  $\{\underline{I}(i,j),\underline{O}(i,j)\}$ is a partition of $[1,\underline{\gamma}(i,j)]$.
  (Pure signals are transmitted using typed variables).

- Expressions $\underline{e}(i,j)(\underline{x}(i))$, $i\in[1,\underline{\pi}]$, $j\in\underline{A}(i)$
  $\underline{e}(i,j,k)(\underline{x}(i))$, $i\in[1,\underline{\pi}]$, $j\in\underline{C}(i)$, $k\in\underline{O}(i,j)$
  ($\underline{e}(i,j)$ maps $dom(\underline{e}(i,j))\subseteq\underline{t}(i)$ into $\underline{t}(i)(\underline{\alpha}(i,j))$ and $\underline{e}(i,j,k)$ maps $dom(\underline{e}(i,j,k))\subseteq\underline{t}(i)$ into $\cup\{\underline{t}(\underline{\theta}(i,j,k))(\ell):\ell\in[1,\underline{\delta}(\underline{\theta}(i,j,k))]\}$).
- Boolean expressions $\underline{b}(i,j)(\underline{x}(i))$, $i\in[1,\underline{\pi}]$, $j\in\underline{T}(i)$
  $\underline{b}(i,j,k)(\underline{x}(i))$, $i\in[1,\underline{\pi}]$, $j\in\underline{C}(i)$, $k\in[1,\underline{\gamma}(i,j)]$
  ($\underline{b}(i,j)$ (resp. $\underline{b}(i,j,k)$) maps $dom(\underline{b}(i,j))$ (resp. $dom(\underline{b}(i,j,k))$) into truth values).
- The following abbreviations will be used :
  $\underline{P\ell}(\underline{\theta}(i,j))?\underline{x}(i)(\underline{\alpha}(i,j)) = [\underline{\text{true}};\underline{P\ell}(\underline{\theta}(i,j,1))?\underline{x}(i)(\underline{\alpha}(i,j,1))\rightarrow\underline{\lambda}(i,j+1)]$
  $\underline{P\ell}(\underline{\theta}(i,j))!\underline{e}(i,j)(\underline{x}(i)) = [\underline{\text{true}};\underline{P\ell}(\underline{\theta}(i,j,1))!\underline{e}(i,j,1)(\underline{x}(i))\rightarrow\underline{\lambda}(i,j+1)]$

This syntax is not intended to be of practical use. The syntax of some examples freely deviates from the above definition when the correspondence is obvious.

2.2 *Operational Semantics*

Roughly an operational semantics defines for each syntactically valid program a set St of states and a transition relation $tr\in[[St\times St]\rightarrow B]$ which is *true* between each state and its possible successors. $B=\{true,false\}$ is the uniquely complemented complete lattice of truth values with ordering $false\Rightarrow true$, infimum *false*, supremum *true*, join $\vee$, meet $\wedge$, complement $\neg$.

2.2.1 *Operational Semantics of Individual Processes*

The semantics of each process $\underline{P}(i)$, $i\in[1,\underline{\pi}]$ can be defined independently of the other processes as long as no communication command is involved.

- Program locations : $\underline{L} = \Pi\{\{\underline{\lambda}(i,j):j\in[1,\underline{\sigma}(i)]\}:i\in[1,\underline{\pi}]\}$
  (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 $I\rightarrow\cup\{E(i):i\in I\}$ of all functions f for which $f(i)\in E(i)$ for all $i\in I$).

- States : $S(i) = \underline{t}(i) \times \underline{L}(i),\ i \in [1, \underline{\pi}]$.

- Transition relation :

$$\tau(i) \in [[S(i) \times S(i)] \rightarrow B],\ i \in [1, \pi]$$
$$\tau(i) = \lambda((xa, ca), (xb, cb)).[\exists j, k \in [1, \underline{\sigma}(i)] : ca = \underline{\lambda}(i,j) \wedge cb = \underline{\lambda}(i,k) \wedge [(Null(i,j)(xa,xb) \wedge k = j+1) \vee (Assign(i,j)(xa,xb) \wedge k = j+1) \vee (Test(i,j,k)(xa,xb))]]$$

$$Null(i,j) \in [[\underline{t}(i) \times \underline{t}(i)] \rightarrow B],\ i \in [1, \underline{\pi}],\ j \in [1, \underline{\sigma}(i)]$$
$$Null(i,j) = \lambda(xa, xb).[j \in \underline{N}(i) \wedge xa = xb]$$

$$Assign(i,j) \in [[\underline{t}(i) \times \underline{t}(i)] \rightarrow B],\ i \in [1, \underline{\pi}],\ j \in [1, \underline{\sigma}(i)]$$
$$Assign(i,j) = \lambda(xa, xb).[(j \in \underline{A}(i)) \wedge (\forall q \in ([1, \underline{\delta}(i)] - \{\underline{\alpha}(i,j)\}),\ xb(q) = xa(q)) \wedge (xa \in dom(\underline{e}(i,j))) \wedge xb(\underline{\alpha}(i,j)) = \underline{e}(i,j)(xa)]$$

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

2.2.2 *Characterization of the States that a Process can Reach after a Communication*

When process $\underline{P}(i)$ is at location c with values x of its local variables, the output guard $\underline{G}(i,j,k)$ is successfully executable only if $Ogse(i,j,k)(x,c)$ is *true* :

$$Ogse(i,j,k) \in [S(i) \rightarrow B],\ i \in [1, \underline{\pi}],\ j \in \underline{C}(i),\ k \in \underline{O}(i,j)$$
$$Ogse(i,j,k) = \lambda(x,c).[c = \underline{\lambda}(i,j) \wedge x \in dom(\underline{b}(i,j,k)) \wedge \underline{b}(i,j,k)(x) \wedge x \in dom(\underline{e}(i,j,k))].$$

If tr is a relation then tr^* denotes its reflexive transitive closure.

The states (xb,cb) that process $\underline{P}(i)$ can reach after execution of output guard $\underline{G}(i,j,k)$ in state (xa,ca) and before meeting a communication or stop command are such that $Rsao(i,j,k)((xa,ca),(xb,cb))$ is *true* :

$$Rsao(i,j,k) \in [[S(i) \times S(i)] \rightarrow B],\ i \in [1, \underline{\pi}],\ j \in \underline{C}(i),\ k \in \underline{O}(i,j)$$
$$Rsao(i\ j,k) = \lambda((xa,ca),(xb,cb)).[Ogse(i,j,k)(xa,ca) \wedge \tau[i]^*[(xa, \underline{\lambda}(i, \underline{n}(i,j,k))), (xb,cb)]]$$

When process $\underline{P}(i)$ is at location c with values x of its local variables the input guard $\underline{G}(i,j,k)$ is successfully executable only if $Igse(i,j,k)(x,c)$ is *true* :

$$Igse(i,j,k) \in [S(i) \rightarrow B],\ i \in [1, \underline{\pi}],\ j \in \underline{C}(i),\ k \in \underline{I}(i,j)$$
$$Igse(i,j,k) = \lambda(x,c).[c = \underline{\lambda}(i,j) \wedge x \in dom(\underline{b}(i,j,k)) \wedge \underline{b}(i,j,k)(x)]$$

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 $subst(x)(j_1/v_1, \ldots, j_n/v_n) = subst[subst(x)(j_1/v_1)][j_2/v_2, \ldots, j_n/v_n]$

The states (xb,cb) that process $\underline{P}(i)$ can reach after execution in state (xa,ca) of input guard $\underline{G}(i,j,k)$ which assigns the transmitted value $v \in \underline{t}(i)(\underline{\alpha}(i,j,k))$ to variable $\underline{x}(i)(\underline{\alpha}(i,j,k))$ and before meeting a communication or stop command are such that $Rsai(i,j,k)((xa,ca),(xb,cb))$ is *true* :

$$Rsai(i,j,k) \in [[S(i) \times \underline{t}(i)(\underline{\alpha}(i,j,k)) \times S(i)] \rightarrow B],\ i \in [1, \underline{\pi}],\ j \in \underline{C}(i),\ k \in I(i,j)$$
$$Rsai(i,j,k) = \lambda((xa,ca), v, (xb,cb)).[Igse(i,j,k)(xa,ca) \wedge \tau[i]^*[(subst(xa)(\underline{\alpha}(i,j,k)/v), \underline{\lambda}(i, \underline{n}(i,j,k))), (xb,cb)]]$$

2.2.3 *Operational Semantics of Communicating Processes*

We introduce the transition relations ι and μ which describe the cooperation of concurrently operating processes. Concurrency in the execution of a program is modeled by global nondeterminism in the selection of successor states. The resolution of the global nondeterminism is left unspecified since CSP definition specifies no scheduling policy whether fair or unfair.

2.2.3.1 *States*

$S = \underline{t} \times \underline{L}$

(When a process is willing to accept a rendez-vous, the states of all other processes may have to be checked in order to determine which processes are ready to communicate or have terminated and next which data are exchanged).

2.2.3.2 *Transition Relations*

- $\underline{C\ell}(i) = \underline{C}(i) \cup \underline{H}(i),\ i \in [1,\underline{\pi}]$

  (The only program locations relevant to cooperation between processes are those corresponding to communication or stop commands).

- $\iota \in [[S \times S] \to B]$

  $\iota = \lambda((xa,ca),(xb,cb)).[\forall i \in [1,\underline{\pi}],\ (ca(i)=cb(i)=\underline{\lambda}(i,1) \wedge xa(i)=xb(i)) \vee (ca(i)=\underline{\lambda}(i,1) \wedge$
  $\tau[i]^*[(xa(i),ca(i)),(xb(i),cb(i))] \wedge cb(i) \in \underline{\lambda}(i,\underline{C\ell}(i)))]$

  (If E, E1, E2 are sets, $f \in [E1 \to E2]$ and $E \subseteq E1$ then $f(E)$ is defined as $\{f(x) : x \in (dom(f) \cap E)\}$. The transition relation ι defines the "ready to communicate" or "stop" states which are possible successors of the "entry" states. As far as cooperation between processes is concerned, a process which is never willing to communicate and never terminates does not progress).

- $\underline{Ch} = \{<i,j,k \to \ell,m,n> : i,\ell \in [1,\underline{\pi}] \wedge j \in \underline{C}(i) \wedge k \in \underline{O}(i,j) \wedge m \in \underline{C}(\ell) \wedge n \in \underline{I}(\ell,m) \wedge i=\underline{\theta}(\ell,m,n) \wedge$
  $\ell=\underline{\theta}(i,j,k) \wedge \{\underline{e}(i,j,k)(x) : x \in dom(\underline{e}(i,j,k))\} \cap \underline{t}(\ell)(\underline{\alpha}(\ell,m,n)) \neq \emptyset\}$.

  (The set $\underline{Ch}$ of communication channels is isomorphic with the set of statically matching pairs of input-output guards).

- $\mu \in [[S \times S] \to B]$

  $\mu = \lambda((xa,ca),(xb,cb)).[\exists <i,j,k \to \ell,m,n> \in \underline{Ch} :$
  $[\forall q \in ([1,\underline{\pi}]-\{i,\ell\}),\ (ca(q)=cb(q)) \wedge (xa(q)=xb(q))]$
  $\wedge [Rsao(i,j,k)((xa(i),ca(i)),(xb(i),cb(i))) \wedge cb(i) \in \underline{\lambda}(i,\underline{C\ell}(i))]$
  $\wedge [\underline{e}(i,j,k)(xa(i)) \in \underline{t}(\ell)(\underline{\alpha}(\ell,m,n))]$
  $\wedge [Rsai(\ell,m,n)((xa(\ell),ca(\ell)),\underline{e}(i,j,k)(xa(i)),(xb(\ell),cb(\ell))) \wedge cb(\ell) \in \underline{\lambda}(\ell,\underline{C\ell}(\ell))]]$

  (The transition relation μ defines the "ready to communicate" or "stop" states which are the possible successors of "ready to communicate" states. The dynamic discrimination of input messages is modeled by dynamic type checking. When several rendez-vous are possible the selection is free. Hence μ specifies all possible orderings of the communications between processes).

3. FIXPOINT CHARACTERIZATION OF CORRECTNESS PROPERTIES

3.1 *Fundamental Theorem*

Let $\underline{Pr}$ be any syntactically correct sCSP program. Its operational semantics defines a set S of states and transition relations ι and μ. Let $P=[S \to B]$ be the set of predicates describing properties of initial, communication or termination states. It is a uniquely complemented complete lattice $P(\Rightarrow, \mathit{false}, \mathit{true}, \vee, \wedge, \neg)$ for the pointwise ordering $\Rightarrow$ (thus the meaning of symbols $\Rightarrow$, *false*, *true*, $\vee$, $\wedge$ and $\neg$ is context-dependent). Let E be the set of possible entry specifications for the program $\underline{Pr}$. The meaning of these specifications is described by $Init \in [E \to P]$ such that $Init(\phi)$ characterizes the set of possible initial states corresponding to the entry specification $\phi \in E$:

- $P = [S \to B]$

  $E = \Pi\{[\underline{t}(i) \to B] : i \in [1,\underline{\pi}]\}$

- $Init \in [E \to P]$

  $Init = \lambda\beta.[Post(\iota)(\lambda(x,c).[\forall i \in [1,\underline{\pi}],\ x(i) \in dom(\beta(i)) \wedge \beta(i)(x(i)) \wedge c(i)=\underline{\lambda}(i,1)])]$

  where the predicate transformer *Post* is defined as :

- $Post \in [[[S \times S] \to B] \to [P \to P]]$

  $Post = \lambda\theta.[\lambda\beta.[\lambda sb.[\exists sa \in S : \beta(sa) \wedge \theta(sa,sb)]]]$

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 $\phi\epsilon E$ is characterized by $Post(\mu^*)(Init(\phi))$. Notice that when programs are non-deterministic *Post* characterizes possible but not necessarily certain descendants of the entry states. The following fixpoint characterization of $Post(\mu^*)(Init(\phi))$ is the basis of our approach (Cousot[79]) :

- $f \in [E \to [P \to P]]$
  $f = \lambda\phi.[\lambda\beta.[Init(\phi) \vee Post(\mu)(\beta)]]$
- $Lfp \in [[P \to P] \to P]$ is the least fixpoint operator for isotone operators on the complete lattice P (Cousot & Cousot[79b]).

Theorem 3.1.1

$\forall\phi\epsilon E,\ Post(\mu^*)(Init(\phi)) = Lfp(f(\phi))$

The above fixpoint theorem leads to sound and complete invariance proof methods (Cousot[79]) and to automatic program analysis techniques (Cousot & Cousot[79a]). However in order to put these methods into practice one or several applications of the following step are required.

3.2 (Pre)homomorphic Image of the Predicate Algebra

Let $A(\Rightarrow, false, true, \vee, \wedge, \neg)$ be a uniquely complemented complete lattice of "assertions". The meaning of A is defined by a *false*-strict $\vee$-complete morphism from $P(\Rightarrow, false, true, \vee, \wedge, \neg)$ into $A(\Rightarrow, false, true, \vee, \wedge, \neg)$. $\rho(\beta)$ is the representation of a "Predicate" $\beta\epsilon P$ by an "assertion" belonging to A. Corresponding to f, let us introduce $F\epsilon[E \to [A \to A]]$ defined as $\lambda\phi.[\lambda\alpha.[INIT(\phi) \vee POST(\alpha)]]$ where $INIT\epsilon[E \to A]$ and $POST\epsilon[A \to A]$. F is said to be equivalent to (resp. an upper approximation of) f up to ρ if and only if $\forall\phi\epsilon E$, $F(\phi)$ is isotone and $\rho\circ f(\phi)$ equals (resp. implies) $F(\phi)\circ\rho$. Let $LFP\epsilon[[A \to A] \to A]$ be the least fixpoint operator. The following theorem shows that whenever F is equivalent to (resp. an upper approximation of) f, $\rho(Lfp(f(\phi)))$ equals (resp. implies) $LFP(F(\phi))$:

Theorem 3.2.1

Let f and F be respectively isotone operators on the complete lattices $P(\Rightarrow, false, true, \vee, \wedge)$ and $A(\Rightarrow, false, true, \vee, \wedge)$, ρ be a *false*-strict $\vee$-complete morphism from P into A such that $\rho\circ f = F\circ\rho$ (resp. $\rho\circ f \Rightarrow F\circ\rho$) then $\rho(Lfp(f)) = LFP(F)$ (resp. $\rho(Lfp(f)) \Rightarrow LFP(F)$).

The importance of this theorem is that it shows that whenever F is equivalent to (resp. an upper approximation of) f up to the $\vee$-morphism ρ, results about the considered program $\underline{Pr}$ obtained using P and f are equivalent to (resp. correctly approximated by) the results obtained using A and F. For example a set of assertions interleaved at appropriate places in the program can be used instead of a single global invariant.

3.3 Associating Assertions with Communication Channels

Let us introduce a *Proj* homomorphic image $Ag(false, \vee, INITg, POSTg)$ of $P(false, \vee, Init, Post(\mu))$:

- $Ag = [\underline{Ch} \to [S \to B]]$
- $Proj \in [P \to Ag]$
  $$Proj = \lambda\beta.[\lambda\langle i,j,k \to \ell,m,n\rangle.[\lambda(x,c).[Ogse(i,j,k)(x(i),c(i)) \wedge \wedge Igse(\ell,m,n)(x(\ell),c(\ell)) \wedge \beta(x,c)]]]$$

The following auxiliary definition is used for describing the behavior of process $\underline{P}(i)$ between locations $\underline{\lambda}(i,j)$ and $\underline{\lambda}(i,k)$ as long as no communication or stop command is encountered :

- $Tr\ell(i)(j,k) \in [[\underline{t}(i) \times \underline{t}(i)] \to B],\ i \in [1,\underline{\pi}],\ j \in [1,\underline{\sigma}(i)]$
  $Tr\ell(i)(j,k) = \lambda(xa,xb).[\tau[i]^*[(xa,\underline{\lambda}(i,j)),(xb,\underline{\lambda}(i,k))]]$

- $INIT\mathrm{g} \in [E \to Ag]$
  $$\begin{aligned} INIT\mathrm{g} = \lambda\phi.[\lambda<i,j,k \to \ell,m,n>.[\lambda(x,c).[\\ &(\exists y \in \underline{t}(i): \phi(i)(y) \wedge Tr\ell(i)(1,j)(y,x(i)) \wedge Ogse(i,j,k)(x(i),c(i))) \\ &\wedge (\exists z \in \underline{t}(\ell): \phi(\ell)(z) \wedge Tr\ell(\ell)(1,m)(z,x(\ell)) \wedge Igse(\ell,m,n)(x(\ell),c(\ell))) \\ &\wedge (\forall p \in ([1,\underline{\pi}]-\{i,\ell\}), (\phi(p)(x(p)) \wedge c(p)=\underline{\lambda}(p,1)) \vee (\exists u \in \underline{t}(p), \exists q \in \underline{C\ell}(p): \\ &\qquad \phi(p)(u) \wedge Tr\ell(p)(1,q)(u,x(p)) \wedge c(p)=\underline{\lambda}(p,q))))]]\end{aligned}$$

- $POST\mathrm{g} \in [Ag \to Ag]$
  $$\begin{aligned} POST\mathrm{g} = \lambda\alpha.[\lambda<i,j,k \to \ell,m,n>.[\lambda(x,c).[&Ogse(i,j,k)(x(i),c(i)) \wedge Igse(\ell,m,n)(x(\ell), \\ &\wedge [\exists <p,q,r \to s,t,u> \in \underline{Ch},\ y \in \underline{t}(p),\ z \in \underline{t}(s): \\ &\quad \alpha(<p,q,r \to s,t,u>)(subst(x)(p/y,s/z),subst(c)(p/\underline{\lambda}(p,q),s/\underline{\lambda}(s,t))) \\ &\quad \wedge \underline{e}(p,q,r)(y) \in \underline{t}(s)(\underline{\alpha}(s,t,u)) \\ &\quad \wedge (\exists v \in \underline{C\ell}(p): c(p)=\underline{\lambda}(p,v) \wedge Tr\ell(p)(\underline{\eta}(p,q,r),v)(y,x(p))) \\ &\quad \wedge (\exists w \in \underline{C\ell}(s): c(s)=\underline{\lambda}(s,w) \wedge \\ &\qquad Tr\ell(s)(\underline{\eta}(s,t,u),w)(subst(z)(\underline{\alpha}(s,t,u)/\underline{e}(p,q,r)(y)),x(s\end{aligned}$$

- $Fg \in [E \to [Ag \to Ag]]$
  $Fg = \lambda\phi.[\lambda\alpha.[INIT\mathrm{g}(\phi) \vee POST\mathrm{g}(\alpha)]]$

Lemma 3.3.1

$\forall \phi \in E,\ Proj \circ f(\phi) = Fg(\phi) \circ Proj$

Theorem 3.3.2

$\forall \phi \in E,\ Proj(Lfp(f(\phi))) = LFP(Fg(\phi))$

3.4 Analysis of the Behavior of Individual Processes

A global assertion about the states of process $\underline{P}(i)$ can be replaced by a set assertions about the values of the process variables preceding each command :

- $A\ell = \Pi\{\Pi\{[\underline{t}(i) \to B]: j \in [1,\underline{\sigma}(i)]\}: i \in [1,\underline{\pi}]\}$
- $Proj\ell(i) \in [[S(i) \to B] \to A\ell(i)],\ i \in [1,\underline{\pi}]$
  $Proj\ell(i) = \lambda\alpha.[\lambda j.[\lambda x.[\alpha(x,\underline{\lambda}(i,j))]]]$

3.4.1 Analysis of the Behavior of Individual Processes Independently of Communica

By definition $Tr\ell(i)(j,k)(xa,xb)$ is *true* if and only if execution of process $\underline{P}$ starting from location $\underline{\lambda}(i,j)$ with the initial state xa of the variables $\underline{x}(i)$ can r location $\underline{\lambda}(i,k)$ with $\underline{x}(i)$=xb and without encountering communication commands. The following characterization of $Tr\ell(i)(j,k)$ as a fixpoint together with Cousot & Cousot[77b] shows that the computation of $Tr\ell(i)(j,k)$ looks like symbolic execution with this difference that all paths are followed simultaneously and infinite paths handled by induction.

- $Post\ell(i) \in [[[\underline{t}(i) \times \underline{t}(i)] \to B] \to [[\underline{t}(i) \to B] \to [\underline{t}(i) \to B]]],\ i \in [1,\underline{\pi}]$
  $Post\ell(i) = \lambda\theta.[\lambda\beta.[\lambda xb.[\exists xa \in \underline{t}(i): \beta(xa) \wedge \theta(xa,xb)]]]$

- $F\ell(i) \in [A\ell(i) \to [A\ell(i) \to A\ell(i)]],\ i \in [1,\underline{\pi}]$
  $$\begin{aligned} F\ell(i) = \lambda\beta.[\lambda\alpha.[\lambda j.[&\beta(j) \vee Post\ell(i)(Null(i,j-1))(\alpha(j-1)) \vee Post\ell(i)(Assign(i,j-1))(\alpha(j \\ &\vee (\exists k \in \underline{T}(i): Post\ell(i)(Test(i,k,j))(\alpha(k))))]]\end{aligned}$$

Theorem 3.4.1.1

$\forall i \in [1,\underline{\pi}],\ j,k \in [1,\underline{\sigma}(i)],$
$Tr\ell(i)(j,k) = \lambda(xa,xb).[LFP[F\ell(i)(\lambda m.[\lambda x.[(m=j) \wedge x=xa]])][k][xb]]$

3.4.2 *Analysis of the Behavior of Individual Processes taking Communications into Account*

We define $\mathit{Desc\ell}(\phi)(i)(j)$ characterizing the possible values that local variables $\underline{x}(i)$ can possess at run-time when location $\underline{\lambda}(i,j)$ of process $\underline{P}(i)$ is reached during an execution of the program starting from an initial state of the local variables $\underline{x}(k)$, $k\in[1,\underline{\pi}]$ satisfying the entry specification $\phi\in E$:

- $\mathit{Desc\ell} \in [E \to A\ell]$

$$\begin{aligned}\mathit{Desc\ell} = \lambda\phi.[\lambda i.[\lambda j.[\lambda x.[&\{\mathit{Post\ell}(i)(\tau(i)^*)(\lambda(y,c).[\phi(i)(y) \wedge c=\underline{\lambda}(i,1)])(x,\underline{\lambda}(i,j))\}\\ &\vee\{\exists\langle\ell,m,n \to p,q,r\rangle\in\underline{Ch}, y\in\underline{t}, c\in\underline{L}: \mathit{Proj}[\mathit{Post}(\mu^*)(\mathit{Init}(\overline{\phi}))][\langle\ell,m,n\to \overline{p},q,r\rangle][y,c]\\ &\quad\wedge[\{(\ell=i)\wedge \mathit{Rsao}(i,\overline{m},n)((\overline{y(i)},c(i)),(x,\underline{\lambda}(i,j)))\}\vee\\ &\qquad\{(p=i)\wedge(\underline{e}(\ell,m,n)(y(\ell))\in\underline{t}(i)(\underline{\alpha}(i,q,\overline{r}))\\ &\qquad\qquad\wedge \mathit{Rsai}(i,q,r)((\overline{y(i)},\overline{c(i)}),\ \underline{e}(\ell,m,n)(y(\ell)),(x,\underline{\lambda}(i,j)))\}]\}]]]]\end{aligned}$$

Since the local descendants of the entry states are either direct descendants of the entry states or the descendants of the states following either an output or an input, the meaning of each separate process can be completely determined only when an initial state for the local variables is provided and the input-output requests from other processes (as determined at paragraph 3.3) are known.

If a state of the program is known before a communication then the corresponding state of the communicating processes after this communication is given by :

- $\mathit{Postoc} \in [Ag \to A\ell]$

$$\begin{aligned}\mathit{Postoc} = \lambda\alpha.[\lambda i.[\lambda j.[\lambda x.[&\exists\langle\ell,m,n\to p,q,r\rangle\in\underline{Ch}, y\in\underline{t}, c\in\underline{L}: (\ell=i)\wedge(m+1=j)\wedge\\ &\alpha(\langle i,m,n\to p,q,r\rangle)(\mathit{subst}(y)(i/\overline{x}),\mathit{subst}(\overline{c})(i/\underline{\lambda}(i,m)))\wedge\\ &\mathit{Ogse}(i,m,n)(x,\underline{\lambda}(i,m))]]]]\end{aligned}$$

- $\mathit{Postic} \in [Ag \to A\ell]$

$$\begin{aligned}\mathit{Postic} = \lambda\alpha.[\lambda i.[\lambda j.[\lambda x.[&\exists\langle\ell,m,n\to p,q,r\rangle\in\underline{Ch}, y\in\underline{t}, c\in\underline{L}: (p=i)\wedge(q+1=j)\wedge\\ &\alpha(\langle\ell,m,n\to i,q,r\rangle)(y,\mathit{subst}(c)(\overline{i/\lambda(i,q)}))\wedge\underline{e}(\ell,m,n)(y(\ell))\in\underline{t}(i)(\underline{\alpha}(i,q,r))\wedge\\ &\mathit{Igse}(i,q,r)(x,\underline{\lambda}(i,q))\ \wedge\ x=\mathit{subst}[y(i)][\underline{\alpha}(i,q,r)/\underline{e}(\ell,m,\overline{n})(y(\overline{\ell}))]]]]]\end{aligned}$$

The following theorem gives a fixpoint characterization of the local descendants of the entry states :

Theorem 3.4.2.1

$\forall\phi\in E,\ \forall i\in[1,\underline{\pi}]$,

$$\mathit{Desc\ell}(\phi)(\overline{i}) = \mathit{LFP}[F\ell[i][\lambda m.(\lambda x.((m=1)\wedge\phi(i)(x)))\vee\mathit{Postoc}(\mathit{LFP}(Fg(\phi)))(i)\vee\mathit{Postic}(\mathit{LFP}(Fg(\phi)))(i)]]$$

3.5 *Example*

```
[P1 :: x:integer; 11: x:=10;
        12: loop 13: exit when x≤0; 14: P2!x; 15: P2?x; 16: end loop;
        17: P2!x; 18: stop
||P2 :: y:integer;
        21: loop 22: P1?y; 23: exit when y=0; 24: P1!y-1; 25: end loop;
        26: stop
]
```

3.5.1 *Analysis of the Behavior of Individual Processes*

The systems of equations $X = F\ell(i)(\beta)(X)$, $i=1,2$ are the following :

$$\left[\begin{array}{l}
X(11)=\beta(1)\\
X(12)=\lambda x.[\beta(2)(x)\vee\exists x'\in integer:X(11)(x')\wedge x=10]\\
X(13)=\beta(3)\vee X(12)\vee X(16)\\
X(14)=\lambda x.[\beta(4)(x)\vee(X(13)(x)\wedge x>0)]\\
X(15)=\beta(5)\\
X(16)=\beta(6)\\
X(17)=\lambda x.[\beta(7)(x)\vee(X(13)(x)\wedge x\leq 0)]\\
X(18)=\beta(8)
\end{array}\right.
\qquad
\left[\begin{array}{l}
X(21)=\beta(1)\\
X(22)=\beta(2)\vee X(21)\vee X(25)\\
X(23)=\beta(3)\\
X(24)=\lambda y.[\beta(4)(y)\vee(X(23)(y\\
X(25)=\beta(5)\\
X(26)=\lambda y.[\beta(6)(y)\vee(X(23)(y)
\end{array}\right.$$

Using theorem 3.4.1.1 we compute :

$$\left[\begin{array}{l}
\mathit{Tr\ell}(1)(1,4)(xa,xb)=(xb=10)\\
\mathit{Tr\ell}(1)(5,5)(xa,xb)=(xb=xa)\\
\mathit{Tr\ell}(1)(6,4)(xa,xb)=(xa>0\wedge xb=xa)\\
\mathit{Tr\ell}(1)(6,7)(xa,xb)=(xa\leq 0\wedge xb=xa)\\
\mathit{Tr\ell}(1)(8,8)(xa,xb)=(xb=xa)
\end{array}\right.
\qquad
\left[\begin{array}{l}
\mathit{Tr\ell}(2)(1,2)(ya,yb)=(yb=ya)\\
\mathit{Tr\ell}(2)(3,4)(ya,yb)=(ya\neq 0\wedge yb=\\
\mathit{Tr\ell}(2)(3,6)(ya,yb)=(ya=0\wedge yb=\\
\mathit{Tr\ell}(2)(5,2)(ya,yb)=(yb=ya)
\end{array}\right.$$

3.5.2 *Analysis of the Communications*

The channels of communications are $<14\rightarrow 22>$, $<24\rightarrow 15>$, $<17\rightarrow 22>$. Assume $\phi=(\lambda x.[\mathit{true}],\lambda y.[\mathit{true}])$ then the system of equations $X=Fg(\phi)(X)$ is constructed the results of 3.5.1 :

$$\left[\begin{array}{lll}
X(<14\rightarrow 22>) & = & \lambda((x,y),(c1,c2)).[c1=14\wedge c2=22\wedge[(x=10)\vee(\exists x'\in integer:\\
& & \qquad X(<24\rightarrow 15>)(x',y,15,24)\wedge x=y-1\wedge x>0)]]\\
X(<24\rightarrow 15>) & = & \lambda((x,y),(c1,c2)).[c1=15\wedge c2=24\wedge(\exists y'\in integer:\\
& & \qquad X(<14\rightarrow 22>)(x,y',14,22)\wedge y=x\wedge y\neq 0)]\\
X(<17\rightarrow 22>) & = & \lambda((x,y),(c1,c2)).[c1=17\wedge c2=22\wedge(\exists x'\in integer:\\
& & \qquad X(<24\rightarrow 15>)(x',y,24,15)\wedge x=y-1\wedge x\leq 0)]
\end{array}\right.$$

The least solution P is computed by successive approximations (Cousot[77]) :

$$\left[\begin{array}{lll}
P(<14\rightarrow 22>) & = & \lambda((x,y),(c1,c2)).[c1=14\wedge c2=22\wedge[(x=10)\vee(1\leq x\leq 9\wedge y=x+1)]]\\
P(<24\rightarrow 15>) & = & \lambda((x,y),(c1,c2)).[c1=15\wedge c2=24\wedge 1\leq x=y\leq 10]\\
P(<17\rightarrow 22>) & = & \lambda((x,y),(c1,c2)).[c1=17\wedge c2=22\wedge x=0\wedge y=1]
\end{array}\right.$$

3.5.3 *Local descendants of the entry States*

The systems of equations $X=F\ell[i][\lambda m.(\lambda z.(m=1))\vee \mathit{Postoc}(P)(i)\vee \mathit{Postic}(P)(i)][X]$ $i=1,2$ are now :

$$\left[\begin{array}{l}
X(11)=\lambda x.[\mathit{true}]\\
X(12)=\lambda x.[\exists x':X(11)(x')\wedge x=10]\\
X(13)=X(12)\vee X(16)\\
X(14)=\lambda x.[X(13)(x)\wedge x>0]\\
X(15)=\lambda x.[1\leq x\leq 10]\\
X(16)=\lambda x.[0\leq x\leq 9]\\
X(17)=\lambda x.[X(13)(x)\wedge x\leq 0]\\
X(18)=\lambda x.[x=0]
\end{array}\right.
\qquad
\left[\begin{array}{l}
X(21)=\lambda y.[\mathit{true}]\\
X(22)=X(21)\vee X(25)\\
X(23)=\lambda y.[0\leq y\leq 10]\\
X(24)=\lambda y.[X(23)(y)\wedge y\neq 0]\\
X(25)=\lambda y.[1\leq y\leq 10]\\
X(26)=\lambda y.[X(23)(y)\wedge y=0]
\end{array}\right.$$

$\mathit{Desc\ell}(\phi)(i)$, $i=1,2$ equals the least solutions to the above equations (Th.3.4.2.

$$\left[\begin{array}{l}
P(11)=\lambda x.[\mathit{true}]\\
P(12)=\lambda x.[x=10]\\
P(13)=\lambda x.[0\leq x\leq 10]\\
P(14)=P(15)=\lambda x.[1\leq x\leq 10]\\
P(16)=\lambda x.[0\leq x\leq 9]\\
P(17)=P(18)=\lambda x.[x=0]
\end{array}\right.
\qquad
\left[\begin{array}{l}
P(21)=P(22)=\lambda y.[\mathit{true}]\\
P(23)=\lambda y.[0\leq y\leq 10]\\
P(24)=P(25)=\lambda y.[1\leq y\leq 10]\\
P(26)=\lambda y.[y=0]
\end{array}\right.$$

In general the least solutions to the systems of equations associated with non-trivial programs are not mechanically computable. Even by hand such calculations cannot be worked out since they are amazingly complex. The solution to this intricacy is the idea of approximation which is central to proof methods and automatic program analysis techniques.

4. INVARIANCE PROOF METHODS

4.1 Outline of Our Approach

$\psi \in P$ 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 $\overline{Post}(\mu^*)(Init(\phi)) \Rightarrow \psi$.

4.1.1 Fundamental Invariance Proof Method

The fixpoint characterization of $Post(\mu^*)(Init(\phi))$ given by theorem 3.1.1 leads to a *sound* and *complete* invariance proof method :

Theorem 4.1.1.1

$\forall \phi \in E, \forall \psi \in P, [\exists I \in P: (f(\phi)(I) \Rightarrow I) \wedge (I \Rightarrow \psi)] \Leftrightarrow [Post(\mu^*)(Init(\phi)) \Rightarrow \psi]$

This invariance proof method fits for use in *partial correctness*, *absence of deadlock* and *non-termination proofs* (the only difference is with respect to the choice of ψ).

4.1.2 (Pre)homomorphic Variants of the Fundamental Proof Method

(Pre)homomorphic images of the predicate algebra (as described at paragraph 3.2) have to be introduced in order to put the fundamental invariance proof method into practice. The soundness or soundness and completeness of these variants of the fundamental proof method follow from the following :

Theorem 4.1.2.1

Let f and F be respectively isotone operators on the complete lattices $P(\Rightarrow, false, true, \vee, \wedge)$ and $A(\Rightarrow, false, true, \vee, \wedge)$, ρ be a *false*-strict $\vee$-complete morphism from P into A such that $\rho \circ f \Rightarrow F \circ \rho$ (resp. $\rho \circ f = F \circ \rho$), $\overline{F}$ be an isotone operator on A such that $F \Rightarrow \overline{F}$ (resp. $F = \overline{F}$) then $\forall \psi \in A$, $[\exists I \in A: \overline{F}(I) \Rightarrow I \wedge I \Rightarrow \psi]$ implies (resp. implies and reciprocally) that $\rho(Lfp(f)) \Rightarrow \psi$.

4.2 A Sound and Complete Invariance Proof Method

For example using the homomorphic image of the predicate algebra described at paragraph 3.3 we get the following particular invariance proof method :

Corollary 4.2.1

$$\forall \phi \in E, \forall \psi \in Ag, [\exists I \in Ag: (INITg(\phi) \Rightarrow I) \wedge (POSTg(I) \Rightarrow I) \wedge (I \Rightarrow \psi)] \Leftrightarrow [Proj(Post(\mu^*)(Init(\phi))) \Rightarrow \psi]$$

Theorem 4.1.2.1 also shows that we can replace $Tr\ell(i)(j,k)$, $i \in [1,\underline{\pi}]$, $j,k \in [1,\underline{\sigma}(i)]$ by upper bounds in the definitions of *INITg* and *POSTg*. By theorems 3.4.1.1 and 4.1.2.1 the correctness of these upper bounds can be shown using an invariance proof method. Hence, proof method 4.2.1 follows the lines drawn by example 3.5 (except that no fixpoint computation is involved and equations are replaced by inequations understood as verification rules).

4.3 *A Sound Variant Without Program Location Counters*

For the sake of completeness in the invariance proof method 4.2.1 the assertions $I \in \mathrm{Ag}$ may have to take the values of the program location counters into account. Yet, on grounds of methodology, reasonings about program location counters are usually ruled out. In order to get rid of program location counters let us introduce :

- $\mathrm{Ags} = [\underline{\mathrm{Ch}} \to [\underline{t} \to \mathrm{B}]]$, $\mathit{Epcg} \in [\mathrm{Ag} \to \mathrm{Ags}]$, $\mathit{Apcg} \in [\mathrm{Ags} \to \mathrm{Ag}]$

- $\mathit{Projs} \in [\mathrm{P} \to \mathrm{Ags}]$
  $\mathit{Projs} = \mathit{Epcg} \circ \mathit{Proj}$

- $\mathrm{Fs} \in [\mathrm{E} \to [\mathrm{Ags} \to \mathrm{Ags}]]$
  $\mathrm{Fs} = \lambda\phi.[\mathit{Epcg} \circ \mathrm{Fg}(\phi) \circ \mathit{Apcg}]$

For example one can choose :

- $\mathit{Epcg} = \lambda\alpha.[\lambda \mathrm{ch}.[\lambda x.[\exists c \in \mathrm{L}: \alpha(\mathrm{ch})(x,c)]]]$
  $\mathit{Apcg} = \lambda\alpha.[\lambda \mathrm{ch}.[\lambda(x,c).\overline{[}\alpha(\mathrm{ch})(x)]]]$

The soundness of the corresponding invariance proof method is shown in a general setting by the following :

<u>Theorem</u> 4.3.1

If Epcg is a *false*-strict $\vee$-complete morphism, Apcg is upper semi-continuous, $\mathrm{Es} \subseteq \mathrm{E}$, $\mathrm{It} = \mathit{Lfp}(\lambda X.[\{\mathit{false}\} \cup \{\mathit{Init}(\phi) \vee \mathit{Post}(\mu)(\alpha) : \phi \in \mathrm{Es} \wedge \alpha \in X\}])$ and $(\forall \alpha \in \mathrm{It},\ \mathit{Proj}(\alpha) \Rightarrow \mathit{Apcg}(\mathit{Projs}(\alpha)))$ then :
$\forall \phi \in \mathrm{Es},\ \mathit{Projs}[\mathit{Post}(\mu^*)(\mathit{Init}(\phi))] \Rightarrow \mathit{LFP}(\mathrm{Fs}(\phi))$

<u>Corollary</u> 4.3.2

If moreover $\forall \phi \in \mathrm{Es}$, $\overline{\mathrm{F}}\underline{\mathrm{s}}(\phi) \in [\mathrm{Ags} \to \mathrm{Ags}]$ is isotone and such that $\mathrm{Fs}(\phi) \Rightarrow \overline{\mathrm{F}}\mathrm{s}(\phi)$, then :
$\forall \phi \in \mathrm{Es},\ \forall \psi s \in \mathrm{Ags}, [\exists I \in \mathrm{Ags}: (\overline{\mathrm{F}}\mathrm{s}(\phi)(I) \Rightarrow I) \wedge (I \Rightarrow \psi s)] \Rightarrow [\mathit{Projs}[\mathit{Post}(\mu^*)(\mathit{Init}(\phi))] \Rightarrow \psi s]$

Completeness can only be established for programs involving only one or two processes (i.e. when $\underline{\pi}=2$).

4.4 *Introducing Auxiliary Variables for Completeness*

In general the reciprocal of theorem 4.3.2 is not true for programs involving more than two processes. A completeness result can nevertheless be obtained using history variables. The use of history variables (Apt, Francez & de Roever[79b], Clarke[78], Clint[73], Owicki & Gries[76]) has several drawbacks. For example, it is difficult to guess which auxiliary variables must be introduced. Only partial solutions are known (i.e. either incomplete or for restricted classes of programs (Clarke[79])) and this may be a major disadvantage for program verification systems. Another drawback is that for some proofs the number of auxiliary variables is greater than the number of variables in the original program (e.g. Gries[79]). Our invariance proof method for sCSP avoids these difficulties. A completeness result is obtained by adding a single local auxiliary variable to each process which is assigned a different value at initialization and prior to reach each communication or stop command.

4.4.1 *Soundness*

Let <u>Prx</u> be a program augmented with auxiliary variables which can only appear in assignments to auxiliary variables. "x" will be postfixed to the definitions of paragraph 2 and 3 when concerning <u>Prx</u>. The set <u>tx</u> of values of the program variables will be understood as the product of the sets <u>tv</u> of values of the main variables and <u>ta</u> of values of the auxiliary variables. Assignments to auxiliary variables only involve total functions. Therefore whenever $\alpha = \iota x$ or $\alpha = \mu x$ we have :

$$(\forall x1, x2 \in \underline{tv},\ c1, c2 \in \underline{Lx},$$
$$[\exists a1, a2 \in \underline{ta} : \alpha(((x1,a1),c1),((x2,a2),c2))] \Rightarrow [\forall a1 \in \underline{ta}, \exists a2 \in \underline{ta} : \alpha(((x1,a1),c1),((x2,a2),c2))])]$$

Assume the program $\underline{Prx}$ is transformed into a program $\underline{Pr}$ by replacing all assignments to auxiliary variables by null statements and by deleting all declarations of auxiliary variables. The operational semantics of $\underline{Pr}$ defines a set of program locations $\underline{L} = \underline{Lx}$, a set of states $S = (\underline{tv} \times \underline{L})$ and transition relations $\iota, \mu \in [S \to B]$ such that :

- $\iota = \lambda((x1,c1),(x2,c2)).[\exists a1, a2 \in \underline{ta} :\ \iota x(((x1,a1),c1),((x2,a2),c2))]$
- $\mu = \lambda((x1,c1),(x2,c2)).[\exists a1, a2 \in \underline{ta} :\ \mu x(((x1,a1),c1),((x2,a2),c2))]$

A correspondance can be established between the entry specifications $\phi x \in Ex$ for $\underline{Prx}$ and $\phi \in E$ for $\underline{Pr}$ by elimination of the auxiliary variables :

- $Eave \in [Ex \to E]$ $\quad Eave = \lambda \phi x.[\lambda i.[\lambda x.[\exists a \in \underline{ta}(i) : \phi x(i)(x,a)]]]$
- $Aave \in [E \to Ex]$ $\quad Aave = \lambda \phi.[\lambda i.[\lambda(x,a).[\overline{\phi(i)}(x)]]]$

The same way assertions $\psi x \in Agx$ interleaved in $\underline{Prx}$ can be connected to assertions $\psi \in Ag$ interleaved in $\underline{Pr}$ by eliminating auxiliary variables :

- $Eavg \in [Agx \to Ag]$ $\quad Eavg = \lambda \alpha.[\lambda ch.[\lambda(x,c).[\exists a \in \underline{ta} : \alpha(ch)((x,a),c)]]]$
- $Aavg \in [Ag \to Agx]$ $\quad Aavg = \lambda \alpha.[\lambda ch.[\lambda((x,a),c).[\overline{\alpha(ch)}(x,c)]]]$

Proofs about $\underline{Prx}$ using Fsx (hence refering to auxiliary variables but not to program counters) can safely be used for proving invariance properties of $\underline{Pr}$:

Theorem 4.4.1.1

If $Epcgx$ is a *false*-strict $\vee$-complete morphism, $Apcgx$ is upper semi-continuous, $Es \subseteq E$, $Itx = Lfp(\lambda X.[\{false\} \cup \{Initx(\underline{Aave}(\phi)) \vee Postx(\mu x)(\alpha) : \phi \in Es \wedge \alpha \in X\}])$, $(\forall \alpha \in Itx, Projx(\alpha) \Rightarrow Apcg\underline{x}(Projsx(\alpha)))$, $(\forall \phi \in Es, \overline{F}sx(Aave(\phi)) \in [Agsx \to Agsx]$ is isotone and such that $Fsx(Aave(\phi)) \Rightarrow \overline{F}sx(Aave(\phi))$ then :

$\forall \phi \in Es, \psi x \in A\underline{g}sx,$
$[\exists I \in Agsx : (\overline{F}sx(Aave(\phi))(I) \Rightarrow I) \wedge (I \Rightarrow \psi x)] \Rightarrow [Proj(Post(\mu^*)(Init(\phi))) \Rightarrow Eavg(Apcgx(\psi x))]$

4.4.2 *Completeness*

The reciprocal of theorem 4.4.1.1 is not true in general. Nevertheless the completeness of the invariance proof method employing auxiliary variables transformations can be understood in the following sense :

Theorem 4.4.2.1

Given an arbitrary program, let $\underline{Pr}$ be the equivalent transformed program such that every command is preceded by a null command. Assume $\phi \in E$, $\psi \in Ag$ and $\underline{Pr}$ is such that $[Proj(Post(\mu^*)(Init(\phi))) \Rightarrow \psi]$.

Then it is possible to transform $\underline{Pr}$ into $\underline{Prx}$ by adding auxiliary simple variables and assignments to $\underline{Pr}$ such that there exist $Epcgx$ and $Apcgx$ verifying $[\exists \phi x \in Ex, \psi sx \in Agsx : (Eave(\phi x) = \phi) \wedge (Eavg(Apcgx(\psi sx)) = \psi) \wedge (\exists I \in Agsx : (Fsx(\phi x)(I) \Rightarrow I) \wedge (I \Rightarrow \psi sx))]$.

We do not have to worry about assertion languages and interpretations which fail to be expressive since "assertions" are regarded as functions into $\{true, false\}$. The proof of theorem 4.4.2.1 clearly indicates that auxiliary variables in $\underline{Prx}$ exactly correspond to program location counters in $\underline{Pr}$: the declaration $\underline{a}(i):\underline{ta}(i)$ of a single auxiliary variable is added to each process $\underline{P}(i)$ of $\underline{Pr}$. $\underline{ta}(i)$ is such that there exists a one to one map $\underline{a\ell}(i)$ from $\underline{ta}(i)$ onto the set $\{\underline{\lambda}(i,1)\} \cup \underline{\lambda}(i, \underline{C\ell}(i))$ of locations preceding an entry, communication or stop command. Each null command "$\underline{\lambda}(i,j-1):\underline{skip}$" preceding a communication or stop command "$\underline{\lambda}(i,j):\underline{S}(i)(j)$" is replaced by an assignment "$\underline{\lambda}(i,j-1):\underline{a}(i):=\underline{a\ell}(i)^{-1}(\underline{\lambda}(i,j))$". No other command is changed (although the functions with domain $\underline{tv}(i)$ in $\underline{P}(i)$ are extended to domain $\underline{tv}(i) \times \underline{ta}(i)$ in $\underline{Px}(i)$). The fact that auxiliary variables simulate program counters is clear from the definitions of :

- $Epcgx = \lambda\alpha.[\lambda ch.[\lambda(x,a).[\alpha(ch)((x,a),\underline{a\ell}(a))]]]$
- $Apcgx = \lambda\alpha.[\lambda ch.[\lambda((x,a),c).[\alpha(ch)(x,\overline{a}) \wedge c=\underline{a\ell}(a)]]]$
- $\phi x \quad = \lambda i.[\lambda(x,a).[\phi(i)(x) \wedge a=\underline{a\ell}(i)^{-1}(\underline{\lambda}(i,1))]]$
- $\psi sx \quad = \lambda ch.[\lambda(x,a).[\psi(ch)(x,\underline{a\ell}(a))]]$

4.5 *Examples of Proofs*

Given two disjoint sets of integers S_0 and T_0, $S_0 \cup T_0$ has to be partitionned into two subsets S and T such that $|S|=|S_0|$, $|T|=|T_0|$ and every element of S is smaller than any element of T :

- Entry specifications : $\phi_1(x,mx,S)=[S=S_0 \wedge S_0 \cap T_0=\emptyset \wedge S_0 \neq \emptyset]$, $\phi_2(y,mn,T)=[T=T_0 \wedge S_0 \cap T_0=\emptyset]$ (To cut it short the conventions that $min(\emptyset)=+\infty$ and $+\infty \notin (S_0 \cup T_0)$ have not been incorporated in the entry specifications).
- Exit specifications : $\psi(x,mx,S,y,mn,T)=[|S|=|S_0| \wedge |T|=|T_0| \wedge S \cup T=S_0 \cup T_0 \wedge max(S)<min(T$

The following version of a program given by Apt, Francez & de Roever[79b] uses two parallel processes P1 and P2 which exchange the current maximum of S with the current minimum of T until $max(S)<min(T)$.

```
P1 ::    mx, x:portion; S:set of portion;
     11: repeat 12: mx:=max(S); 13: S:=S-{mx}; 14: P2!mx; 15: P2?x; 16: S:=Su{x};
     17: until mx=x;
     18: stop
```

--- We first obtain the following descriptions $Tr\ell(1)(i,j)$ of the transformation of the values of the variables of P1 when P1 is executed starting at entry point or after communication point i and ending at stop point or before communication point j without intermediate communications. Since no loop is involved no inductive assertion is necessary and the equations of theorem 3.4.1.1 can be solved exactly.

$$\begin{cases} Tr\ell(1)(1,4)(mx1,x1,S1,mx2,x2,S2)=[mx2=max(S1) \wedge x2=x1 \wedge S2=S1-\{mx2\}] \\ Tr\ell(1)(5,5)(mx1,x1,S1,mx2,x2,S2)=[mx2=mx1 \wedge x2=x1 \wedge S2=S1] \\ Tr\ell(1)(6,4)(mx1,x1,S1,mx2,x2,S2)=[mx2=max(S1 \cup \{x1\}) \wedge x2=x1 \wedge x1 \neq mx1 \wedge S2=((S1 \cup \{x1\})-\{mx2\})] \\ Tr\ell(1)(6,8)(mx1,x1,S1,mx2,x2,S2)=[mx2=mx1=x2=x1 \wedge S2=(S1 \cup \{x1\})] \end{cases}$$

The independent analysis of P2 is similar :

```
P2 ::    mn, y:portion; T:set of portion;
     21: loop 22: mn:=min(T); 23: P1?y; 24: exit when y<mn; 25: T:=(T-{mn})u{y};
              26: P1!mn;
     27: end loop;
     28: P1!y; 29: stop
```

$$\begin{cases} Tr\ell(2)(1,3)(mn1,y1,T1,mn2,y2,T2)=[mn2=min(T1) \wedge y2=y1 \wedge T2=T1] \\ Tr\ell(2)(4,6)(mn1,y1,T1,mn2,y2,T2)=[mn2=mn1 \wedge y2=y1 \wedge y1 \geq mn1 \wedge T2=((T1-\{mn1\}) \cup \{y1\})] \\ Tr\ell(2)(4,8)(mn1,y1,T1,mn2,y2,T2)=[mn2=mn1 \wedge y2=y1 \wedge y1<mn1 \wedge T2=T1] \\ Tr\ell(2)(7,3)(mn1,y1,T1,mn2,y2,T2)=[mn2=min(T1) \wedge y2=y1 \wedge T2=T1] \\ Tr\ell(2)(9,9)(mn1,y1,T1,mn2,y2,T2)=[mn2=mn1 \wedge y2=y1 \wedge T2=T1] \end{cases}$$

--- The assertions I(ch1), I(ch2), I(ch3) respectively associated with the communication channels ch1=<1,4,1 → 2,3,1>, ch2=<2,6,1 → 1,5,1> and ch3=<2,8,1 → 1,5,1> of program [P1 || P2] are the following :

$$\begin{cases} I(ch1)(mx,x,S,mn,y,T) =[|S|+1=|S_0| \wedge |T|=|T_0| \wedge mx>max(S) \wedge S \cup \{mx\} \cup T=S_0 \cup T_0 \\ \qquad\qquad \wedge (S \cup \{mx\}) \cap T=\emptyset \wedge mn=min(T)] \\ I(ch2)(mx,x,S,mn,y,T) =[|S|+1=|S_0| \wedge |T|=|T_0| \wedge mn \notin T \wedge S \cup T \cup \{mn\}=S_0 \cup T_0 \\ \qquad\qquad \wedge (S \cap (T \cup \{mn\}))=\emptyset \wedge y=mx] \\ I(ch3)(mx,x,S,mn,y,T) =[|S|+1=|S_0| \wedge |T|=|T_0| \wedge max(S)<mx=y<min(T) \wedge S \cup \{y\} \cup T=S_0 \cup T_0] \end{cases}$$

The verification conditions given by theorem 4.2.1 are the following (universal quantification is implicit) :

$$\left[\begin{array}{ll}
I(ch1)(mx,x,S,mn,y,T) \Leftarrow & [[(\exists mx',x',S':\phi_1(mx',x',S') \wedge \mathit{Tr\ell}(1)(1,4)(mx',x',S',mx,x,S)) \wedge \\
 & \quad (\exists mn',y',T':\phi_2(mn',y',T') \wedge \mathit{Tr\ell}(2)(1,3)(mn',y',T',mn,y,T))] \vee \\
 & [\exists mx',x',S',mn',y',T':I(ch2)(mx',x',S',mn',y',T') \wedge \\
 & \quad \mathit{Tr\ell}(1)(6,4)(mx',mn',S',mx,x,S) \wedge \mathit{Tr\ell}(2)(7,3)(mn',y',T',mn,y,T)]] \\
I(ch2)(mx,x,S,mn,y,T) \Leftarrow & [\exists mx',x',S',mn',y',T':I(ch1)(mx',x',S',mn',y',T') \wedge \\
 & \mathit{Tr\ell}(1)(5,5)(mx',x',S',mx,x,S) \wedge \mathit{Tr\ell}(2)(4,6)(mn',mx',T',mn,y,T)] \\
I(ch3)(mx,x,S,mn,y,T) \Leftarrow & [\exists mx',x',S',mn',y',T':I(ch1)(mx',x',S',mn',y',T') \wedge \\
 & \mathit{Tr\ell}(1)(5,5)(mx',x',S',mx,x,S) \wedge \mathit{Tr\ell}(2)(4,8)(mn',mx',T',mn,y,T)]
\end{array}\right.$$

--- Finally the verification of the exit specification consists in proving that :

$$\begin{array}{ll}
\psi(mx,x,S,mn,y,T) \Leftarrow & [\exists mx',x',S',mn',y',T': I(ch3)(mx',x',S',mn',y',T') \wedge \\
 & \mathit{Tr\ell}(1)(6,8)(mx',y',S',mx,x,S) \wedge \mathit{Tr\ell}(2)(9,\underline{9})(mn',y',T',mn,y,T)]
\end{array}$$

5. *AUTOMATIC PROGRAM ANALYSIS TECHNIQUES*

Automatic program analysis techniques can be used to gather information about programs in order to enhance program efficiency and reliability.

5.1 *Outline of Our Approach*

The design of a variety of automatic program analysis techniques (Cousot & Cousot[79a]) is tantamount to defining for each program $\underline{Pr}\in sCSP$ a prehomomorphic image $A(\bot,\sqcup,F)$ of $P(\mathit{false},\vee,f)$ using an approximation operator $\overline{\rho}\in[P\rightarrow A]$ which is a strict join-complete morphism. In addition to the hypotheses of paragraph 3.2, the elements of A are chosen to be computer-representable and F must be such that $\forall\phi\in E$, $\mathit{LFP}(F(\phi))$ is either iteratively computable (using any chaotic iteration strategy, Cousot[77]) or approximable from above (using a widening operator which speeds up the convergence of the iterates, Cousot & Cousot[77a]).

Once understood in this way the numerous global flow analysis algorithms which have been developped in the literature for sequential programs can be generalized to sCSP. The main difficulty of these generalizations has been solved at paragraphs 3.3 and 3.4 which show how to use a system of equations obtained in a natural manner from the program text. Program locations counters can be dispensed with, but for programs involving more than two communicating processes more accurate results are obtained by introducing auxiliary variables as indicated at paragraph 4.4.2. When the approximate assertions of A can describe relationships among the values of the program variables the analysis schema can follow the three steps of paragraph 3.5 since information can be propagated through individual processes at step 1 once for all (it is sound and sometimes more accurate to intersect the results of step one with the ones obtained by a preliminary analysis of individual processes viewed as classical sequential programs by treating inputs as assignments of unknown values and outputs as null commands). When the approximate assertions cannot describe relationships among the values of variables, the first step based on theorem 3.4.1.1 is ineffective. Then during the evaluation of $\mathit{LFP}(F(\phi))$, information must be propagated through individual processes by fixpoint computations as many times as F is evaluated.

5.2 *Example*

The analysis of Hoare[78]'s bounded buffer example (modified using output guards and explicit termination signals) considering only linear equality or inequality relationships among integer variables (as in Cousot & Halbwachs[78]) is the following (booleans are mapped onto $\{0,1\}$, n is a global symbolic constant greater than 0) :

```
{n≥1}
[Producer :: output:(0..n-1)portion; i:integer; 1:i:=0;
       while i≠n do {0≤i<n} 3:X!output(i);i:=i+1 od; 6:X!true; 7:stop

||X ::  buffer:(0..9)portion; in,out:integer; Pe:boolean; 1:in:=0;out:=0;Pe:=false;
        {n≥1 ∧ in=out=Pe=0}
      2:*[¬Pe ∧ in<out+10 ∧ Producer?buffer(in mod 10) →
            {in<n ∧ 0≤out≤in<out+10 ∧ Pe=0} in:=in+1;
         []out<in; Consumer!buffer(out mod 10) →
            {in≤n ∧ 0≤out<in≤out+10 ∧ 0≤Pe≤1} out:=out+1
         []¬Pe; Producer?Pe →
            {1≤in≤n ∧ 0≤out≤in≤out+10 ∧ Pe=1} skip
         []Pe ∧(in=out); Consumer!true →
            {1≤in=out≤n ∧ Pe=1} 7:stop]

||Consumer :: input:(0..n-1)portion; j:integer; Xe:boolean; 1:j:=0; Xe:=false;
        {n≥1 ∧ j=Xe=0}
      2:*[¬Xe; X?input(j) → {0≤j<n ∧ Xe=0} j:=j+1
         []¬Xe; X?Xe → {1≤j≤n ∧ Xe=1} 5:stop]                                    ]
```

The assertions I(ch1),...,I(ch4) respectively associated with the communication channels ch1=<1,3,1→2,2,1>, ch2=<1,6,1→2,2,3>, ch3=<2,2,2→3,2,1> and ch4=<2,2,4→3,2,2> are the following :

```
[I(ch1)=λ(n,i,in,out,Pe,j,Xe,c1,c2,c3).[i<n ∧ 0≤out=j≤i=in<out+10 ∧ Pe=0 ∧ Xe=0 ∧ c1=3 ∧
          c2=2 ∧ c3=2]
|I(ch2)=λ(n,i,in,out,Pe,j,Xe,c1,c2,c3).[1≤i≤n ∧ 0≤out=j≤i=in≤out+10 ∧ Pe=0 ∧ Xe=0 ∧
          c1=6 ∧ c2=2 ∧ c3=2]
|I(ch3)=λ(n,i,in,out,Pe,j,Xe,c1,c2,c3).[0≤out=j<i=in≤out+10 ∧ Pe≥0 ∧ Xe=0 ∧
          3n-3i+c1-6-Pe≥0 ∧ c1-4Pe≥3 ∧ c1-Pe≤6 ∧ 3≤c1≤7 ∧ c2=2 ∧ c3=2]
|I(ch4)=λ(n,i,in,out,Pe,j,Xe,c1,c2,c3).[1≤i=in=out=j≤n ∧ Pe=1 ∧ Xe=0 ∧ c1=7 ∧ c2=2 ∧ c3=2]
```

6. *CONCLUSIONS*

Our proof method differs from the ones introduced by Apt, Francez & de Roever[79a] (who use trees to record communications and account for nondeterminism as in Francez, Hoare, Lehmann & de Roever[78]), Apt, Francez & de Roever[79b] (who use a global invariant) and by Chandy & Misra[79] (who use assertions over message sequences). More interestingly the design of our proof method has been formalized so as to open the way to alternatives. For example one variant we have considered consists in choosing :

- $Ag = \Pi\{[\underline{t}(i)\times\underline{t}(\ell) \to B] : \langle i,j,k \to \ell,m,n\rangle \in \underline{Ch}\}$
- $Proj = \lambda\overline{\beta}.[\lambda\langle\overline{i},j,k \to \ell,m,n\rangle.[\lambda(xi,x\ell).[\mathit{Ogse}(i,j,k)(xi,\underline{\lambda}(i,j)) \wedge \mathit{Igse}(\ell,m,n)(x\ell,\underline{\lambda}(\ell,m)) \wedge (\exists x\in\underline{t}, c\in\underline{L}: \beta(\mathit{subst}(x)(i/xi,\ell/x\ell),\mathit{subst}(c)(i/\underline{\lambda}(i,j),\ell/\underline{\lambda}(\ell,m))))]]]$.

so that the assertion associated with any channel between two processes involve only the values of the local variables of these two processes. This is sound and leads to simpler verification conditions than the ones of paragraphs 3.3 and 3.4 but unfortunately this is not complete (e.g. for blocking states involving more than two processes).

We have said little about program analysis techniques since paragraph 3 together with Cousot & Cousot[79a] allow for an easy generalization of the now classical methods for sequential programs. By the way, the particular problem of data flow analysis for concurrent processes has also been considered by Reif[79] in the different context of asynchronous communications via buffers. His algorithm cannot be adapted to CSP since the size and number of elements in the buffers are not taken into account. In particular, CSP rendez-vous concept cannot be modeled. Another annoying thing is that communication histories are completely ignored. For example, his algorithm cannot discover that x and y are constants in the following trivial program :

```
[P1 :: P2!1; P2!2 || P2 :: x,y:integer; x:=1; y:=2; P1?x; P1?y ]
```

This is easily determined using constant propagation generalized along the lines of paragraph 3.

Several generalizations are in progress. On one hand, we are considering other homomorphic images of the predicate algebra leading to different (although formally equivalent) verification conditions and richer language constructs such as distributed termination of repetitive commands, families of processes and nested parallel commands. It might also be interesting to investigate other language features (such as automatic buffering, unbounded process activation, process valued variables, shared variables) which have not been incorporated to CSP. We are curious to know if these constructions have properties which are sufficiently simple to prove for justifying their inclusion in a programming language. On the other hand, we are considering other correctness properties of CSP that can be characterized using fixpoints and which are more difficult to prove than the invariance properties considered here.

7. REFERENCES

Apt K.,Francez N. & de Roever W.P.[1979a], *Semantics for concurrently communicating finite sequential processes, based on predicate transformers*, Progress Report, Vakgroep inf., Rijksuniversiteit Utrecht, The Netherlands, (June 1979).

Apt K., Francez N. & de Roever W.P.[1979b], *A proof system for communicating sequential processes*, Tech. Rep. RUU-CS-79-8, Vakgroep Inf., Rijkuniv. Utrecht, NL., (Aug. 79).

Chandy K.M. & Misra J.[1979], *An axiomatic proof technique for networks of communicating processes*, TR 98, Univ. of Texas at Austin, (May 79).

Clarke E.M.Jr[1978], *Proving correctness of coroutines without history variables*, TR-CS-1978-4, Dept. of Comp. Sci., Duke Univ., USA, (1978).

Clarke E.M.Jr[1979], *Synthesis of resource invariants for concurrent programs*, 6th ACM-POPL, (Jan. 1979), 211-221.

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

Cousot P.[1977], *Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice*, Rap. de Recherche n°88, Laboratoire IMAG, Univ. Grenoble, (Sept. 1977).

Cousot P.[1979], *Analysis of the behavior of dynamic discrete systems*, Rapport de Recherche n°161, Laboratoire IMAG, Univ. Grenoble, (Jan. 1979).

Cousot P. & Cousot R.[1977a], *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, 4th ACM-POPL, (Jan. 1977), 238-252.

Cousot P. & Cousot R.[1977b], *Automatic synthesis of optimal invariant assertions : mathematical foundations*, ACM-Symp. on Artificial Int. & Prog. Languages, SIGPLAN Notices 12, 8(Aug. 1977), 1-12.

Cousot P. & Cousot R.[1979a], *Systematic design of program analysis frameworks*, 6th ACM-POPL, (Jan. 1979), 269-282.

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

Cousot P. & Halbwachs N.[1978], *Automatic discovery of linear restraints among variables of a program*, 5th ACM-POPL, (Jan. 1978), 84-97.

Francez N., Hoare C.A.R., Lehmann D.J. & de Roever W.P.[1978], *Semantics of nondeterminism, concurrency and communication*, Lect. Notes Comp. Sci. 64, Springer Verlag, Extended abstract, (Sept. 1978), 191-200.

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

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

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

Reif J.H.[1979], *Data flow analysis of communicating processes*, 6th ACM-POPL, (Jan. 1979), 257-268.