# Synthesis of Resource Invariants for Concurrent Programs

Edmund Melson Clarke, Jr.†

Aiken Computation Laboratory
Harvard University
Cambridge, Mass. 02138

## Abstract

Owicki and Gries have developed a proof system for *conditional critical regions*. In their system logically related variables accessed by more than one process are grouped together as *resources*, and processes are allowed access to a resource only in a critical region for that resource. Proofs of synchronization properties are constructed by devising predicates called *resource invariants* which describe relationships among the variables of a resource when no process is in a critical region for the resource. In constructing proofs using the system of Owicki and Gries, the programmer is required to supply the resource invariants.

We show that *convexity* plays a key role in the derivation of strong resource invariants. We also develop methods for automatically synthesizing resource invariants. Specifically, we characterize the resource invariants of a concurrent program as *least fixpoints* of a functional which can be obtained from the text of the program. By using this fixpoint characterization and a *widening operator* which exploits our observation on the importance of convexity, good approximations may be obtained for the resource invariants of many concurrent programs.

## 1. Introduction

Owicki and Gries [OW76] have developed a proof system for *conditional critical regions*. In their system logically related variables accessed by more than one process are grouped together as *resources*, and processes are allowed access to a resource only in a critical region for that resource. Proofs of synchronization properties are constructed by devising predicates called *resource invariants*. These predicates describe relationships among the variables of a resource when no process is in a critical region for the resource. Related methods for verifying concurrent programs have been discussed by Lamport [LM77] and Pneuli [PN77].

In constructing proofs using the system of Owicki and Gries, the programmer is required to supply the resource invariants. We investigate the possibility of automatically synthesizing resource invariants for a simple concurrent programming language (SCL) in which processes access shared data via conditional critical regions. We consider only *invariance* [PN77] or *safety* properties [LM77] of SCL programs. This class of properties includes mutual exclusion and absence of deadlock and is analoguous to partial correctness for sequential programs. Correctness proofs of SCL programs are expressed in a proof system similar to that of Owicki and Gries.

To gain insight on the synthesis of resource invariants we restrict the SCL language so that all processes are nonterminating loops, and the only statements allowed in a process are P and V operations on semaphores. We call this class of SCL programs *PV programs*. For PV programs there is a simple method for generating resource invariants, i.e., the *semaphore invariant method* of Habermann [HA72] which expresses the current value of a semaphore in terms of its initial value and the number of P and V operations which have been executed. Although the semaphore invariant method is simple to state, it is a powerful technique for proving PV programs; we show that it is as powerful as the *reduction method* of Lipton [LI75] for proving freedom from deadlock.

The semaphore invariant method, however, is not complete for proving either absence of deadlock or mutual exclusion of PV programs. We show that there exist PV programs for which deadlock (mutual exclusion) is impossible, but the semaphore invariant method is insufficiently powerful to establish this fact. This incompleteness result is important because it demonstrates the role of *convexity* in the generation of powerful resource invariants. We also give a characterization of the class of PV programs for which the semaphore invariant method is complete for proving absence of deadlock (mutual exclusion).

The semaphore invariant method is generalized to the class of *linear SCL programs* in which solutions to many synchronization problems can be expressed. Although the generalized semaphore invariant also fails to be complete, it is sufficiently powerful to permit proofs of mutual exclusion and absence of deadlock for a significant class of concurrent programs. When the generalized semaphore invariant is insufficiently powerful to prove some desired property of an SCL program, is

†This research has been partially supported by National Science Foundation Grant No. MCS-7508146.

it possible to synthesize a stronger resource invariant? We argue that resource invariants are *fixpoints*, and that by viewing them as fixpoints it is possible to generate invariants which are stronger than the semaphore invariants previously described. We show that the resource invariants of an SCL program C are fixpoints of a functional $F_C$ which can be obtained from the text of program C and that the *least fixpoint* $\mu(F_C)$ of $F_C$ is the "strongest" such resource invariant. Since the functional $F_C$ is continuous, the least fixpoint $\mu(F_C)$ may be expressed as the limit

$$\mu(F_C) = \bigcup_{j=1}^{\infty} F_C^j(\text{false}) \quad .$$

Clearly, this characterization of $\mu(F_C)$ cannot be used directly to compute $\mu(F_C)$ unless C has only a finite number of different states or unless a good initial approximation is available for $\mu(F_C)$.

By using the notion of *widening* of Cousot [CO77] however, we are able to speed up the convergence of the chain $F_C^j(\text{false})$ and obtain a close approximation to $\mu(F_C)$ in a finite number of steps. The widening operator which we use exploits our observation on the importance of convexity in the generation of resource invariants. Although fixpoint techniques have been previously used in the study of resource invariants ([LA76], [FL77]), we believe that this is the first research on methods for speeding up the convergence of the sequence of approximations to $\mu(F_C)$. Examples are given in the text to illustrate the power of this new technique.

The SCL language and its semantics are discussed in Sections 2 and 3 of this paper. Sections 4 and 5 contain a description of the semaphore invariant method and a discussion of why it is incomplete. Section 6 introduces the class of linear SCL programs and briefly describes how the semaphore invariant can be generalized to this class of programs. The fixpoint theory of resource invariants is presented in Section 7. Section 8 contains an account of Cousot's widening operator and how it can be used in approximating resource invariants. The paper concludes with a discussion of the results and some remaining open problems.

## 2. A Simple Concurrent Programming Language (SCL)

An SCL program will consist of two parts: (1) an *initialization part* "$\bar{x}:=\bar{e}$" in which initial values are assigned to the synchronization variables $\bar{x}$, and (2) a *concurrent execution part*

resource R($\bar{x}$): cobegin P1//P2//...Pn coend

which permits the simultaneous or interleaved execution of the statements in the *processes* P1,...,Pn. All variables accessed by more than one process must appear in the prefix R($\bar{x}$) of the concurrent execution part. Processes have the form

Pi: cycle $S_1^i$; $S_2^i$;...;$S_{k_i}^i$ end

where "$S_1^i$; $S_2^i$;...;$S_{k_i}^i$" is a list of conditional critical regions. The cycle construct is a nonterminating loop with the property that the next statement to be executed after $S_{k_i}^i$ is the first statement $S_1^i$ of the loop. Although the cycle statement simplifies the generation of loop invariants, the results of this paper also apply to terminating loops (e.g., while loops). The extension of SCL to allow multiple resources is straightforward and will not be treated in this paper.

*Conditional critical regions* have the form with R when b do A od. Only variables listed in R can appear in the boolean expression b and the body A of the conditional critical region. When execution of a process reaches the conditional critical region with R when b do A od, the process is delayed until no other process is using R and the condition b is satisfied. Then the statement A is executed as an indivisible action.

Let C be an SCL program with the format described above; a *program state* $\sigma$ is an ordered list $(pc_1, pc_2,...,pc_n; s)$, where

(1) $pc_i$ is the *program counter* for process Pi and is in the range $1 \le pc_i \le k_i$.

(2) s maps the set of synchronization variables to the set Z of integers and is called the *program store*.

We will write b(s) to denote the value of predicate b in store s; A(s) will be the new store resulting when the sequential statement A is executed in store s.

A *computation* of an SCL program C is a sequence of program states $\sigma_0, \sigma_1,...,\sigma_j,...$ . The *initial state* $\sigma_0$ has the form $(1, 1,...,1; s_0)$ where $s_0$ reflects the assignments made in the initialization part of C. Consecutive states

$$\sigma_j = (pc_1^j,...,pc_n^j; s_j)$$

and

$$\sigma_{j+1} = (pc_1^{j+1},...,pc_n^{j+1}; s_{j+1})$$

are related as follows: There exists an m, $1 \le m \le n$ such that

(1) $pc_i^{j+1} = pc_i^j$ if $i \neq m$ .

(2) $$pc_m^{j+1} = \begin{cases} pc_m^j + 1 & \text{if } pc_m^j < k_m \\ 1 & \text{otherwise} \end{cases}$$

(3) if statement $pc_m^j$ in process m is with R when b do A od, then $b(s_j)$ = true and $s_{j+1} = A(s_j)$.

Note that concurrency in the execution of an SCL program is modeled by nondeterminism in the selection of successor states.

If there exists a computation $\sigma_0, \sigma_1,...\sigma_j,...$ of program C, then we say that state $\sigma_j$ is *reachable* from the initial state $\sigma_0$ of C and write $\sigma_0 \overset{C}{\Rightarrow} \sigma_j$. Note that the *next statement to be executed by* process Pi in state $\sigma = (pc_1,...,pc_n;s)$ is always $S^i_{pc_i}$. We say that program C is *blocked* in state $\sigma$ if the condition of the next statement to be executed in each process is false in state $\sigma$. A state $\sigma$ of C is a *deadlock state* if $\sigma$ is reachable from the initial state of C and C is blocked in state $\sigma$. Two statements $S_1$ and $S_2$ in different processes of C are *mutually exclusive* if there does not exist a state $\sigma$, reachable from the initial state of C, in which $S_1$ and $S_2$ are next to execute in their respective processes.

Frequently it will be convenient to identify a predicate U with the set of program states which make U true. If $\Sigma$ is the set of all program states, then $2^{\Sigma}$ will be the set of all possible predicates, *false* will correspond to the empty state set, and *true* will correspond to the set $\Sigma$ of all program states. Also logical operations on predicates can be interpreted as set theoretic operations on subsets of $\Sigma$ i.e., "or" becomes "union", "and" becomes "intersection", "not" becomes "complement", and "implies" becomes "is a subset of".

SP[A](U) will denote the *strongest postcondition* corresponding to the *sequential statement* A and the *precondition* U. If the predicate U is identified with the set of states which satisfy it, then SP[A](U) may be defined by SP[A](U) = $\{(pc_1,\ldots,pc_n;\ A(s)) \mid (pc_1,\ldots,pc_n;\ s) \in U\}$

THEOREM 2.1. *Let* A *be a sequential statement.*

(A) *(Monotonicity) if* $U, V \subseteq \Sigma$ *and* $U \subseteq V$, *then* $SP[A](U) \subseteq SP[A](V)$.

(B) *(Additivity) if* $\{U_i\}$, $i \geq 0$ *is a family of predicates, then*
$SP[A](\bigcup_i U_i) = \bigcup_i SP[A](U_i)$ .

Proof: See [CL77]. □

## 3. Resource Invariant Proofs

In this section we adapt the proof system of Owicki and Gries to SCL programs. We use the standard notation $\{P\}A\{Q\}$ of Hoare to express the *partial correctness* of the sequential statement A with respect to the *precondition* P and *postcondition* Q. The triple $\{P\}A\{Q\}$ is *true* ($\models \{P\}A\{Q\}$) iff $\models SP[A](P) \rightarrow Q$. Proof systems for partial correctness of sequential statements will not be discussed in this paper.

Let C be an SCL program and let ST be the set of statements occurring within the processes of C. A *resource invariant system* $RS_C$ for C will consist of two parts:

(1) A predicate IR called the *resource invariant*. All free variables of IR must appear in the resource prefix $R(\bar{x})$ of the program C.

(2) Proofs of sequential correctness for each of the individual processes of C.

For our purposes, these correctness proofs are represented by a set VC of assertions called *verification conditions* and two functions *pre*, *post*: ST → VC which give the precondition and postcondition for each statement C in the proof. To insure that the proofs of sequential correctness for the individual processes are *interference free* [OW76], we require that the free variables in the verification conditions for process i do not appear as free variables in the verification conditions for any process j with $j \neq i$. If C is an SCL program with the format described in Section 2, then the functions pre and post for process Pi must also satisfy the following conditions:

(a) $\models \bar{x}=\bar{e} \rightarrow pre(S_1^i) \wedge IR$

(b) $\models post(S_{k_i}^i) \rightarrow pre(S_1^i)$

(c) $\models post(S_j^i) \rightarrow pre(S_{j+1}^i)$ for $1 \leq j \leq k_i - 1$

(d) if $S_j^i$ is the conditional critical region
with R when $b_j^i$ do $A_j^i$ od
then $\models \{pre(S_j^i) \wedge b_j^i \wedge IR\} A_j^i \{post(S_j^i) \wedge IR\}$.

THEOREM 3.1. *Let* $RS_C$ *be a resource invariant system for the SCL program C. If* σ *is reachable in* C *and* $S_j^i$ *is the next statement of process* Pi *to execute in state* σ, *then* $\sigma \in pre(S_j^i)$.

Proof: See [OW76]. □

Resource invariant systems may be used to prove absence of deadlock and mutual exclusion of an SCL program C. To prove *mutual exclusion* of statements $S_1$ and $S_2$ in C it is sufficient to give a resource invariant system $RS_C$ for C such that

$$M(RS_C) = pre(S_1) \wedge pre(S_2) \wedge IR$$

is unsatisfiable. To prove that it is impossible for C to become *deadlocked* it is sufficient to exhibit a resource invariant system $RS_C$ such that the predicate

$$D(RS_C) = \bigwedge_{i=1}^{n} \left( \bigvee_{j=1}^{k_i} pre(S_j^i) \wedge \neg b_j^i \right) \wedge IR$$

is unsatisfiable. Local deadlock, in which only a subset of the processes are blocked, may be handled in a similar manner.

## 4. The Semaphore Invariant Method

P and V operations on a semaphore a can be treated as conditional critical regions: P(x) is equivalent to with R when x>0 do x:=x-1 od and V(x) to with R when true do x:=x+1 od. In this section we restrict the class of SCL programs so that the only statements allowed within processes are P and V operations on semaphores; we call such programs *PV programs*.

The semaphore invariant method [HA72] is based on the use of *auxiliary variables*. Let a be a semaphore with initial value m occurring in a PV program C. For each statement $S_i$ corresponding to a P operation we introduce an auxiliary variable $a_i^1$ which is incremented each time the P operation is executed. Similarly for each statement $S_i$ corresponding to a V operation we introduce a variable $a_i^2$. All auxiliary variables are initialized to zero at the beginning of the program C. The *semaphore invariant* states that the predicate $I_a \equiv \{a = m + \Sigma a_i^2 - \Sigma a_i^1 \wedge a \geq 0\}$ must be satisfied by C whenever C is not executing a P or V operation on the semaphore a.

When auxiliary variables are added to C in this manner, there is a simple method of generating appropriate pre and post functions for C. Let "Pi: cycle $S_1^i;\ldots;S_{k_i}^i$ end" be the $i^{th}$ process in the program C, and let $d_1^i,\ldots,d_{k_i}^i$ be the auxiliary variables for this process. The *pre* and *post* functions for process Pi will be defined inductively:

(1) $pre(S_1^i) \equiv post(S_{k_i}^i) \equiv \{d_1^i = d_2^i = \ldots = d_{k_i}^i\}$

(2) If $S_j^i$ is a conditional critical region with associated auxiliary variable $d_j^i$ then
$$post(S_j^i) = pre(S_j^i)\left[(d_j^i - 1)/d_j^i\right]$$

We refer to the resource invariant system consisting of the conjunction of the semaphore invariants $I_a$

and the annotation obtained by the above procedure as the *semaphore invariant system* ($SI_C$) corresponding to C.

Consider, for example, the PV program C

```
a:=1
cobegin
   A: cycle P(a); SA; V(a) end
   //
   B: cycle P(a); SB; V(a) end
coend
```

SA and SB represent the bodies of the critical regions established by the P and V operations and will be treated as null statements in the analysis which follows. Annotating C as described in the previous paragraph we obtain:

$\{a_1^1 = 0 \wedge a_1^2 = 0 \wedge a_2^1 = 0 \wedge a_2^2 = 0 \wedge a = 1\}$
resource R(a, $a_1^1$, $a_1^2$, $a_2^1$, $a_2^2$):
cobegin
   A: cycle
      $\{a_1^1 = a_1^2\}$
      with R when $a > 0$ do $a_1^1 := a_1^1 + 1$; $a := a - 1$ od;
      $\{a_1^1 - 1 = a_1^2\}$
      SA;
      $\{a_1^1 - 1 = a_1^2\}$
      with R when true do $a_1^2 := a_1^2 + 1$; $a := a + 1$ od;
      end
   //
   B: cycle
      $\{a_2^1 = a_2^2\}$
      with R when $a > 0$ do $a_2^1 := a_2^1 + 1$; $a := a - 1$ od;
      $\{a_2^1 - 1 = a_2^2\}$
      SB;
      $\{a_2^1 - 1 = a_2^2\}$
      with R when true do $a_2^2 := a_2^2 + 1$; $a := a + 1$ od;
      end
coend

The invariant $I_a$ for semaphore a is

$$I_a = \{a = 1 + a_1^2 + a_2^2 - a_1^1 - a_2^1 \wedge a \geq 0\} .$$

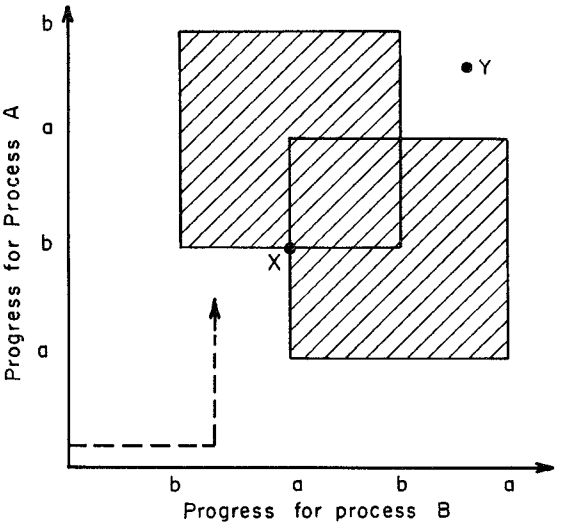
Since the predicate $M(SI_C) = pre(SA) \wedge pre(SB) \wedge I_a$ is unsatisfiable, it follows that statements SA and SB are mutually exclusive. Similarly we see that C is free from deadlock, since the predicate $D(SI_C)$ is also unsatisfiable.

## 5. Incompleteness of the Semaphore Invariant Method

The incompleteness of the semaphore invariant method is best explained by means of *progress graphs* [DI76]. The progress graph is a graphical method for representing the feasible states of a PV program. Consider, for example, the program C:

```
a:=1; b:=1
cobegin
   A: cycle P(a); P(b); V(a); V(b) end
   //
   B: cycle P(b); P(a); V(b); V(a) end
coend
```

Feasible computations of this program can be represented by a graph in which the number of instructions executed by a process is used as a measure of the progress of the process, e.g.,



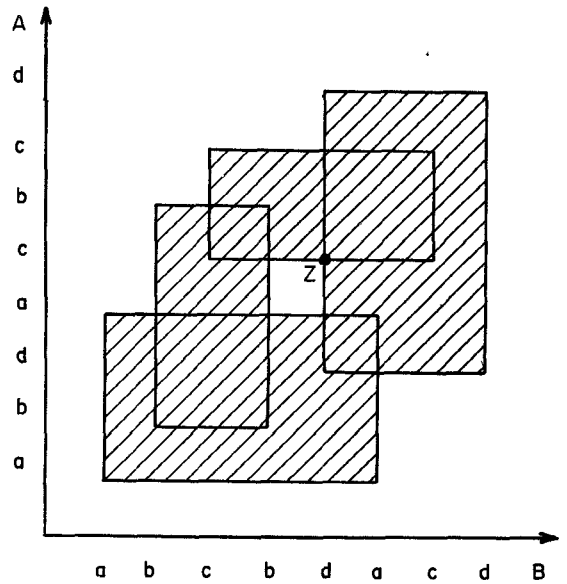

The dashed line represents a computation of the program C in which process B executes P(b) and process A executes P(a). The shaded region of the graph represents those program states which fail to satisfy the semaphore invariants for a or b; such states are called *unfeasible states*. The point labeled X in the graph is a *deadlock state*; the state X is reachable from the initial state of C but further progress for either process A or process B would violate one of the semaphore invariants (i.e., both processes are blocked). Those points in the graph (states of C) which are not reachable from the origin (initial state) by a polygonal path composed of horizontal and vertical line segments which never cross an unfeasible region (by a valid computation sequence of C) are called *unreachable points* (states). All unfeasible points are unreachable. The point labeled Y in the graph is an example of an *unreachable feasible point*; if the program C were started in state Y, the semaphore invariants would not be violated.

Next consider the PV program C:

```
a:=b:=c:=d:=1
cobegin
   A: cycle P(a); P(b); P(d); V(a); P(c); V(b); V(c);
      V(d) end
   //
   B: cycle P(a); P(b); P(c); V(b); P(d); V(a); V(c);
      V(d) end
coend
```

The progress graph for C is shown on the next page.

Note that deadlock can never occur during execution of program C. Let $SI_C$ be the semaphore invariant system for the program C. Thus, if auxiliary variables are added as described in Section 4,




the invariant I will be given by

$$I = \{a = 1 + a_1^2 + a_2^2 - a_1^1 - a_2^1 \wedge a \geq 0$$

$$\wedge b = 1 + b_1^2 + b_2^2 - b_1^1 - b_2^1 \wedge b \geq 0$$

$$\wedge c = 1 + c_1^2 + c_2^2 - c_1^1 - c_2^1 \wedge c \geq 0$$

$$\wedge d = 1 + d_1^2 + d_2^2 - d_1^1 - d_2^1 \wedge d \geq 0$$

$$\wedge a_1^2 \geq 0 \wedge a_2^2 \geq 0 \wedge a_1^1 \geq 0 \wedge a_2^1 \geq 0$$

$$\wedge b_1^2 \geq 0 \wedge b_2^2 \geq 0 \wedge b_1^1 \geq 0 \wedge b_2^1 \geq 0$$

$$\wedge c_1^2 \geq 0 \wedge c_2^2 \geq 0 \wedge c_1^1 \geq 0 \wedge c_2^1 \geq 0$$

$$\wedge d_1^2 \geq 0 \wedge d_2^2 \geq 0 \wedge d_1^1 \geq 0 \wedge d_2^1 \geq 0\}$$

It is not difficult to show that the condition $D(SI_C)$ for absence of deadlock is satisfied by the state Z in which

$$a = 0,\ a_1^2 = 1,\ a_2^2 = 0,\ a_1^1 = 1,\ a_2^1 = 1$$

$$b = 0,\ b_1^2 = 0,\ b_2^2 = 1,\ b_1^1 = 1,\ b_2^1 = 1$$

$$c = 0,\ c_1^2 = 0,\ c_2^2 = 0,\ c_1^1 = 0,\ c_2^1 = 1$$

$$d = 0,\ d_1^2 = 0,\ d_2^2 = 0,\ d_1^1 = 1,\ d_2^1 = 0$$

Thus absence of deadlock cannot be proven by means of the semaphore invariant method. The state Z which satisfies $D(SI_C)$ is an example of an unreachable feasible state in which each process of C is blocked; we will call such states *trap states*.

THEOREM 5.1. *The semaphore invariant method is complete for proving deadlock freedom for those PV programs whose progress graphs do not contain any* trap *states.*

Proof: Let C be a PV program whose progress graph does not contain any trap states. Thus any state of C in which all processes are blocked must be reachable from C's initial state. Let $SI_C$ be the semaphore invariant system for C. We show that the condition $D(SI_C)$ is unsatisfiable *if and only if* deadlock is impossible for the program C. Clearly, if $D(SI_C)$ is unsatisfiable than deadlock is impossible. Thus assume that $D(SI_C)$ is satisfied by some state $\sigma$. By construction of the predicate D all processes are blocked in state $\sigma$. Since $\sigma$ is reachable from the initial state of C, it is a deadlock state. □

A similar characterization may be given for mutual exclusion. How can the semaphore invariant method be strengthened to handle trap states? Since trap states correspond to "holes" in the unfeasible region of a progress graph, a method based on convexity is worthy of investigation. We return to this question in Section 8.

Although the semaphore invariant method is not complete for proving absence of deadlock or mutual exclusion of PV programs, it is a powerful tool for proving correctness of PV programs which occur in practice as the examples of [HA72] demonstrate. Additional evidence for the power of the semaphore invariant method may be obtained by comparing it to other methods which have been proposed for proving deadlock freedom of PV programs. We prove, for example, that the semaphore invariant method is as powerful as the *reduction method* of Lipton [LI74]: If a PV program has a reduction proof of deadlock freedom, then it also has a proof using the semaphore invariant method.

*Reduction* is a technique for decreasing the number of interleavings of statements which must be considered in the proof of a concurrent program. Let C be a concurrent program and S a statement contained in C. The reduction C/S is the concurrent program obtained from C by making S into a single indivisible (i.e., uninterruptable) action. To prove that program C has some property U, it is sufficient to produce a sequence of programs $C = C_1, C_2, \ldots, C_n$ and statements $S_1, \ldots, S_{n-1}$ such that

(1) $C_{i+1} = C_i/S_i$ for $i = 1, \ldots, n-1$.

(2) If $C_i$ has property U, then $C_i/S_i$ also has property U.

(3) $C_n$ trivially has property U.

Lipton gives a class of statements called *D-reductions* with the property that if C is a concurrent program and S is a D-reduction, then C is deadlock-free iff C/S is deadlock free. For PV programs D-reductions have the form $S = S_1, S_2, \ldots, S_n$ where $S_1, \ldots, S_{i-1}$ are P operations, $S_{i+1}, \ldots, S_n$ are V operations and $S_2, \ldots, S_n$ can always execute. In view of condition (3) above we will assume that the progress graph of the final program $C_n$ in a reduction proof of deadlock freedom does not contain any trap states.

LEMMA 5.2. *If the progress graph of a PV program C contains a trap state, then the progress graph of the D-reduction* C/S *also contains a trap state.*

Proof: Let $\sigma$ be a trap state for the PV program C. By definition, $\sigma$ is an unreachable feasible state of C in which every process is blocked. Let C/S be a D-reduction where $S = S_1; S_2; \ldots; S_n$ has the form described above. Assume that $\sigma$ is not a trap state for C/S. If

$\sigma$ were a feasible state of C/S, then it would be a trap state of C/S since every computation of C/S is a computation of C. Thus, $\sigma$ must not be a feasible state of C/S. Since $S = S_1; S_2; \ldots; S_n$ must be executed as an atomic statement in C/S, the process containing S must be blocked in state $\sigma$ while attempting to execute one of the statements $S_2, S_3, \ldots, S_n$. Since by definition of a D-reduction the statements $S_2, S_3, \ldots, S_n$ can always execute, this is a contradiction. □

THEOREM 5.3. *If a PV program has a proof of deadlock freedom using the reduction method, then it also has a proof of deadlock freedom using the semaphore invariant method.*

Proof: Assume that $C_0$ has a proof of deadlock freedom using the reduction method. Then there exists a sequence of D-reductions $C_1 = C_0/S_0, \ldots, C_n = C_{n-1}/S_{n-1}$ where $C_n$ is free from deadlock and does not contain any trap states. By Lemma 5.2 $C_0$ must also be free from deadlock and not contain any trap states. By Theorem 5.1, it is possible to prove that $C_0$ is free from deadlock by means of the semaphore invariant method. □

## 6. Generalization of the Semaphore Invariant Method

Since a large class of synchronization techniques can be modeled by counting operations on shared variables, the class of *linear SCL programs* is of particular interest. The conditional critical regions of a linear SCL program have the form

with R when $B(x_1, x_2, \ldots, x_n)$ do $A(x_1, x_2, \ldots, x_n)$ od

where

(1) the variables $x_1, x_2, \ldots, x_n$ belong to resource R.

(2) the condition $B(x_1, x_2, \ldots, x_n)$ is a truth functional combination of atomic formulas of the form $a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_{n+1} \leq 0$

(3) the body $A(x_1, x_2, \ldots, x_n)$ is a series of assignment statements which increment the shared variables $x_1, \ldots, x_n$ e.g.,

$$x_1 := x_1 + b_1$$
$$x_2 := x_2 + b_2$$
$$\vdots$$
$$x_n := x_n + b_n$$

Note that semaphores are special cases of linear SCL programs. Many other standard synchronization problems including the dining philosophers problem, the readers and writers problem, and the cigarette smoker's problem can all be expressed as linear SCL programs. Arguments are given in [SC76] and [AG74] that linear SCL programs are universal in their power to express synchronization constraints for concurrent programs. It is also possible to prove that mutual exclusion and deadlock freedom are undecidable for this class of programs.

We briefly outline how the semaphore invariant can be generalized to linear SCL programs. Let C be an SCL program. For each conditional critical region $S_i$ in C we introduce a new auxiliary variable $dS_i$ which counts the number of times $S_i$ has been executed. Thus, the algorithm of Section 4 may be used to generate pre and post functions for C; the resulting annotation of C will be called the *canonical annotation*.

Let $H_i(\bar{x}) = a_1^i x_1 + a_2^i x_2 + \ldots a_n^i x_n + a_{n+1}^i$ be a linear form occurring in the condition of some critical region of C. We will use the notation $\partial H_i/\partial S_j$ to denote the change in value of $H_i$ caused by the execution of statement $S_j$; note that $\partial H_i/\partial S_j$ is given by

$$\partial H_i/\partial S_j = \sum_{r=1}^{n} a_r^i b_r^j \quad .$$

Let $dH_i = H(\bar{x}) - H(\bar{x}_0)$ where $\bar{x}_0$ gives the initial values of the synchronization variables. Then, the relationship

$$dH_i = \sum_j \partial H_i/\partial S_j \, dS_j$$

must hold if no process is executing a critical region for R.

Although the generalized semaphore invariant is sufficiently powerful to permit proofs of mutual exclusion and absence of deadlock for a significant class of linear SCL programs, it fails to be complete for exactly the same reason as the original semaphore invariant.

## 7. A Fixpoint Theory of Resource Invariants

Let $\Sigma$ be the set of program states and let $F: 2^\Sigma \to 2^\Sigma$. If $U \subseteq \Sigma$ and $F(U) = U$, then U is a *fixpoint* for the functional F. If U is a fixpoint of F and $U \subseteq V$ for all other fixpoints V of F, then U is the *least fixpoint* of F. F is *continuous* if for every ascending chain $U_0 \subseteq U_1 \subseteq \ldots \subseteq U_j \subseteq \ldots$ of subsets of $\Sigma$,

$$F\left(\bigcup_{j=0}^{\infty} U_j\right) = \bigcup_{j=0}^{\infty} F(U_j) \quad .$$

If F is continuous, then F has a least fixpoint $\mu(F)$ which is given by

$$\mu(F) = \bigcup_{j=0}^{\infty} F^j(\text{false})$$

where $F^0(U) = U$ and $F^{j+1} = F(F^j(U))$.

The resource invariants of an SCL program C are fixpoints of a functional $F_C$ which can be obtained from the text of C. Let C be the SCL program

$\bar{x} := \bar{e}$;
resource $R(\bar{x})$
cobegin P1//P2//...Pn coend

We assume that C contains K critical regions $S_1, S_2, \ldots, S_K$ and that the $i^{th}$ critical region has the form $S_i \equiv$ with R when $b_i$ do $A_i$ od. The algorithm of Sections 4 and 6 will be used to generate the pre and post functions for C. Let $\Sigma$ be the set of possible states of C and let $F_C: 2^\Sigma \to 2^\Sigma$ be defined by

$$F_C(J) = J_0 \vee J \vee \bigvee_{i=1}^{K} SP[A_i](pre(S_i) \wedge b_i \wedge J)$$

where the predicate $J_0 = \{\bar{x} = \bar{e}\}$ describes the initial state of C.

THEOREM 7.1. *The function* $F_C$ *is a continuous mapping on* $2^{\Sigma}$. *Thus,* $F_C$ *has a least fixpoint* $\mu(F_C)$ *which is given by*

$$\mu(F_C) = \bigcup_{j=0}^{\infty} F_C^j(\text{false})$$

Proof: The continuity of $F_C$ follows directly from the additivity of SP. □

THEOREM 7.2. *All resource invariants* IR *of* C *are fixpoints of* $F_C$. *Also* $\mu(F_C)$ *is a resource for* C.

Proof: Let IR be a resource invariant for C. Clearly, $IR \subseteq F_C(IR)$. We must show that $F_C(IR) \subseteq IR$. By condition (a) in the definition of a resource invariant system $\{\bar{x}=\bar{e}\} \subseteq IR$. By condition (e) we see that for $1 \leq i \leq K$,

$$\models \{pre(S_i) \wedge b_i \wedge IR\}\ A_i\ \{post(S_i) \wedge IR\} .$$

It follows that for $1 \leq i \leq K$

$$SP[A_i](pre(S_i) \wedge b_i \wedge IR) \subseteq post(S_i) \wedge IR .$$

Hence,

$$\bigvee_{i=1}^{K} SP[A_i](pre(S_i) \wedge b_i \wedge IR) \subseteq IR .$$

So,

$$F_C(IR) = J_0 \vee IR \vee \bigvee_{i=1}^{K} SP[A_i](pre(S_i) \wedge b_i \wedge IR) \subseteq IR .$$

Thus, every resource invariant IR is a fixpoint of $F_C$.
Since $\mu(F_C)$ is a fixpoint of $F_C$, we have

$$\mu(F_C) = J_0 \vee \mu(F_C) \vee \bigvee_{i=1}^{K} SP[A_i](pre(S_i) \wedge b_i \wedge \mu(F_C))$$

Thus, $J_0 \subseteq \mu(F_C)$ and for $1 \leq i \leq K$

$$SP[A_i](pre(S_i) \wedge b_i \wedge \mu(F_C)) \subseteq \mu(F_C) .$$

By construction of the pre and post functions, we also have $SP[A_i](pre(S_i)) \subseteq post(S_i)$.
By monotonicity

$$SP[A_i](pre(S_i) \wedge b_i \wedge \mu(F_C)) \subseteq post(S_i) .$$

It follows that for $1 \leq i \leq K$

$$SP[A_i](pre(S_i) \wedge b_i \wedge \mu(F_C)) \subseteq post(S_i) \wedge \mu(F_C)$$

or, equivalently that

$$\models \{pre(S_i) \wedge b_i \wedge \mu(F_C)\}\ A_i\ \{post(S_i) \wedge \mu(F_C)\} .$$

Thus, $\mu(F_C)$ is a resource invariant corresponding to the canonical annotation given in Section 6. □

THEOREM 7.3. $\mu(F_C) = \{\sigma \mid \sigma_0 \overset{C}{\Rightarrow} \sigma\}$ .

Proof: Let $IR = \{\sigma \mid \sigma_0 \overset{C}{\Rightarrow} \sigma\}$. We show that IR is a fixpoint of $F_C$ and that $IR \subseteq \mu(F_C)$. Since $\mu(F_C)$ is the least fixpoint of $F_C$, it follows that $\mu(F_C) = IR$.

(A) $F_C(IR) = IR$: Clearly $IR \subseteq F_C(IR)$. Let $\sigma \in F_C(IR) = J_0 \vee IR \vee \bigvee_{i=1}^{K} SP[A_i](pre(S_i) \wedge b_i \wedge IR)$ then either $\sigma \in J_0 \subseteq IR$ or $\sigma \in IR$ or there exists $i_0$ such that

$$\sigma \in SP[A_{i_0}](pre(S_{i_0}) \wedge b_{i_0} \wedge IR) .$$

Only the third case is interesting. If

$$\sigma \in SP[A_{i_0}](pre(S_{i_0}) \wedge b_{i_0} \wedge IR),$$

then there is a state $\sigma' \in pre(S_{i_0}) \wedge b_{i_0} \wedge IR$ such that $A_{i_0}(\sigma') = \sigma$.
Since $\sigma' \in IR$, there is a computation $\sigma_0, \sigma_1, \ldots, \sigma_r$ of C with $\sigma_r = \sigma'$. Because $\sigma' \in pre(S_{i_0}) \wedge b_{i_0}$ and $\sigma = A(\sigma')$, $\sigma_0, \ldots, \sigma_r, \sigma$ is also a computation of C and $\sigma \in IR$.
(B) $IR \subseteq \mu(F_C)$: Let $\sigma \in IR$ then there exists a computation $\sigma_0, \sigma_1, \ldots, \sigma_r$ in which $\sigma_r = \sigma$. We prove by induction on r that $\sigma_r \in F_C^{r+1}(\text{false})$. Since $F_C^1(\text{false}) = J_0$, the basis case $\sigma_0 \in F_C^1(\text{false})$ is true. Assume that for all computations $\sigma_0, \sigma_1, \ldots, \sigma_{r-1}$ of C, $\sigma_{r-1} \in F_C^r(\text{false})$. Let $\sigma_0, \sigma_1, \ldots, \sigma_{r-1}, \sigma_r$ be a computation of length r, then there exists and $i_0$ such that $\sigma_{r-1} \in pre(S_{i_0}) \wedge b_{i_0}$ and $\sigma_r = A_{i_0}(\sigma_{r-1})$. Thus $\sigma_r \in SP[A_{i_0}](pre(S_{i_0}) \wedge b_{i_0} \wedge F_C^r(\text{false})) \subseteq F_C^{r+1}(\text{false})$.
It follows that

$$IR \subseteq \bigcup_{i=0}^{\infty} F_C^i(\text{false}) = \mu(F_C) .$$

□

THEOREM 7.4. *The resource system* $RS_C$ *consisting of* $\mu(F_C)$ *and the canonical annotation is relatively complete for proving absence of deadlock and mutual exclusion of SCL programs.*

Proof: We prove that for the resource invariant system $RS_C$, the condition $D(RS_C)$ is unsatisfiable *iff* deadlock is impossible. Clearly, if $D(RS_C)$ is unsatisfiable then deadlock is impossible. We must show that if $D(RS_C)$ is satisfiable, then there exists a state $\sigma_d$ which is reachable from the initial state of C in which every process of C is blocked. Let $\sigma_d$ be a program state which satisfies $D(RS_C)$. Since $\sigma_d$ satisfies $D(RS_C)$, it follows that $\sigma_d \in \mu(F_C)$ and also that each process of C is blocked in state $\sigma_d$. Since $\sigma_d \in \mu(F_C)$, $\sigma_d$ is reachable from the initial state $\sigma_0$ of C. Thus, $\sigma_d$ is a deadlock state for the program C. The proof of completeness for mutual exclusion is similar and will be left to the reader. □

Theorem 7.4 shows that $\mu(F_C)$ is the "strongest" resource invariant for program C. The next theorem is important because it gives a method for improving approximations to $\mu(F_C)$.

THEOREM 7.5. *If* L *is a predicate such that* $L \subseteq (F_C)$, *then*

$$\mu(F_C) = \bigcup_{j=0}^{\infty} F^j(L) .$$

Proof: It is easy to show that for all $j \geq 0$

$$F^j(\text{false}) \subseteq F^j(L) \subseteq F^j(\mu(F_C))$$

Thus

$$\mu(F_C) = \bigcup_{j=0}^{\infty} F^j(\text{false}) \subseteq \bigcup_{j=0}^{\infty} F^j(L) \subseteq \bigcup_{j=0}^{\infty} F^j(\mu(F_C)) = \mu(F_C)$$

□

To illustrate theorems 7.1 - 7.5 we consider the following solution to the mutual exclusion problem:

```
a:=0; b:=0;
resource R(a,b):
cobegin
   A: cycle A1: with R when b=0 do a:=a+1 od;
              SA;
              A2: with R when true do a:=a-1 od
      end
   //
   B: cycle B1: with R when a=0 do b:=b+1 od;
              SB;
              B2: with R when true do b:=b-1 od
      end
coend
```

Adding auxiliary variables and using the algorithm of Section 6 to generate pre and post functions we obtain:

$\{a=0 \wedge b=0 \wedge a_1=0 \wedge a_2=0 \wedge b_1=0 \wedge b_2=0\}$

```
resource (a, b, a1, a2, b1, b2):
cobegin
   A: cycle {a1=a2}
      A1: with R when b=0 do a:=a+1; a1:=a1+1
          od;
          {a1-1=a2}
          SA;
          {a1-1=a2}
      A2: with R when true do a:=a-1; a2:=a2+1
          od
      end
   //
   B: cycle {b1=b2}
      B1: with R when a=0 do b:=b+1; b1:=b1+1
          od;
          {b1-1=b2}
          SB;
          {b1-1=b2}
      B2: with R when true do b:=b-1; b2:=b2+1
          od
      end
coend
```

In this case the function $F_C$ is

$$F_C(J) = a=0 \wedge b=0 \wedge a_1=0 \wedge a_2=0 \wedge b_1=0 \wedge b_2=0 \vee J \vee SP[a:=a+1;\ a_1:=a_1+1](b=0 \wedge a_1=a_2 \wedge J) \vee SP[a:=a-1;\ a_2:=a_2+1](\text{true} \wedge a_1-1=a_2 \wedge J) \vee SP[b:=b+1;\ b_1:=b_1+1](a=0 \wedge b_1=b_2 \wedge J) \vee SP[b:=b-1;\ b_2:=b_2+1](\text{true} \wedge b_1-1=b_2 \wedge J)$$

Since a(b) is incremented in statement A1(B1) and decremented in statement A2(B2), an obvious guess for a resource invariant is

$$IR = \{a=a_1-a_2 \wedge b=b_1-b_2 \wedge a_1 \geq 0 \wedge a_2 \geq 0 \wedge b_1 \geq 0 \wedge b_2 \geq 0\}$$ .

It is easily checked that $F_C(IR) = IR$ so that IR is a fixpoint of $F_C$.

Since $D = pre(A1) \wedge b=0 \wedge pre(B1) \wedge a=0 \wedge IR$ is unsatisfiable, the invariant IR may be used to prove absence of deadlock for C. The invariant IR is not strong enough, however, to prove mutual exclusion of statements SA and SB, since the predicate $M = pre(SA) \wedge pre(SB) \wedge IR$ is satisfiable.

By using Theorem 7.5 we may compute the strongest resource invariant $\mu(F_C)$. Let

$L = \{IR \wedge a_1=a_2 \wedge b_1=b_2\}$ then $L \subseteq \mu(F_C)$. Since $F_C^3(L) = F_C^4(L) = \ldots = \{IR \wedge a=1 \rightarrow b=0 \wedge b=1 \rightarrow a=0\}$, we see that

$$\mu(F_C) = \bigcup_{i=0}^{\infty} F_C^i(L) = F_C^3(L) = \{a=a_1-a_2 \wedge b=b_1-b_2 \wedge a=1 \rightarrow b=0 \wedge b=1 \rightarrow a=0 \wedge a_1 \geq 0 \wedge a_2 \geq 0 \wedge b_1 \geq 0 \wedge b_2 \geq 0\}$$

Using the resource invariant $\mu(F_C)$ it is easy to show that the predicate $M' = pre(SA) \wedge pre(SB) \wedge \mu(F_C)$ is unsatisfiable; thus the statements SA and SB are mutually exclusive.

Note that Theorem 7.5 can only be used to obtain $\mu(F_C)$ if program C has a finite number of different possible states or unless a good approximation is already available to $\mu(F_C)$. In the next section we will examine more powerful techniques for obtaining strong resource invariants.

## 8. Speeding up the Convergence of Fixpoint Techniques for Approximating Resource Invariants

For linear SCL programs the notion of *widening* of Cousot [CO77] may be used to speed up convergence to $\mu(F_C)$. The widening operator * is characterized by the following two properties:

(A) for all *admissable* predicates U and V, $U \subseteq U*V$ and $V \subseteq U*V$

(B) for any ascending chain of admissable predicates $U_0 \subseteq U_1 \subseteq U_2 \subseteq \ldots$ the ascending chain defined by $V_0 = U_0$, $V_{i+1} = V_i * U_{i+1}$ is *eventually stable* i.e., there exists a $k \geq 0$ such that for $i \geq k$, $V_i = V_k$ .

In this paper the *admissable* predicates are the polygonal convex sets of $Q^m$ where Q is the set of rational numbers and m is the number of resource variables belonging to R. The widening operator * that we use is a modification of the one used by Cousot [CO78]. Let U and V be polygonal convex sets. Then U and V can be represented as conjunctions

$$U = \bigwedge_{j=1}^{g} \gamma_j \quad \text{and} \quad V = \bigwedge_{k=1}^{h} \delta_k$$

where each conjunct is a linear inequality of the form $a_1x_1 + \ldots + a_mx_m + a_{m+1} \leq 0$. We further assume that the representation of U and V is minimal, i.e., no conjunct can be dropped without changing U or V. We say that two linear inequalities $\gamma_j$ and $\delta_k$ are *equivalent* if they determine the same

half space of $Q^m$. $U*V$ is the conjunction of all those $\gamma_j$ in the representation of $U$ for which there is an equivalent $\delta_k$ in the representation of $V$. Thus the widening operator "throws out" all those constraints in the representation of $U$ which do not occur in the representation of $V$.

We now describe the strategy for approximating $\mu(F_C)$. Since the predicates $F_C^j(\text{false})$ in the chain $F_C^0(\text{false}) \subseteq F_C^1(\text{false}) \subseteq \ldots$ may not be polygonal convex sets, let $G_i = CV[F_C^i(\text{false})]$ where $CV$ is the *convex hull operator*. The sequence $G_0 \subseteq G_1 \subseteq \ldots$ is a chain of polygonal convex sets. The sequence $I^t$ will be used in obtaining a good approximation to the strongest resource invariant for $R$ and is defined by

$$I^t = \bigcup_{j=0}^{\infty} H_j^t$$

where $H_0^t = G_t$ and $H_{j+1}^t = H_j^t * G_{t+j+1}$

THEOREM 8.1.

*(A) Each $I^t$ can be computed in a finite number of steps.*

*(B)* $\mu(F_C) \subseteq I^t$ for $t \geq 1$.

*(C) The sequence $I^t$ is a decreasing chain in $\Sigma$ i.e.,* $I^1 \supseteq I^2 \supseteq I^3 \ldots$

Proof of (A): By condition 2 in the definition of a widening operator the sequence $H_0^t, H_1^t, H_2^t, \ldots$ must eventually stabilize. Thus there exists a $k$ such that

$$I^t = \bigcup_{j=0}^{k} H_j^t .$$

Proof of (B):

$$\mu(F_C) = \bigcup_{i=0}^{\infty} F^i(\text{false})$$

$$\subseteq \bigcup_{i=0}^{\infty} CV[F^i(\text{false})]$$

$$\subseteq \bigcup_{j=0}^{\infty} G_j$$

$$\subseteq \bigcup_{j=0}^{\infty} G_{t+j}$$

$$\subseteq \bigcup_{j=0}^{\infty} H_j^t$$

$$\subseteq I^t$$

Proof of (C): Let $U$ and $V$ be polygonal convex sets, we write $U \sqsubset V$ if for every conjunct $\delta$ in $V$, there is an equivalent conjunct in $U$. We prove by induction on $k$ that

$$H_k^{t+1} \sqsubset H_{k+1}^t .$$

(1) Basis step:

$$H_0^{t+1} = G_{t+1} \sqsubset H_0^t * G_{t+1} = H_1^t$$

(2) Induction step:

Assume $H_k^{t+1} \sqsubset H_{k+1}^t$ then

$$H_{k+1}^{t+1} = H_k^{t+1} * G_{t+k+2} \sqsubset H_{k+1}^t * G_{t+k+2} = H_{k+2}^t$$

Note that $H_k^{t+1} \sqsubset H_{k+1}^t$ implies $H_k^{t+1} \subseteq H_{k+1}^t$. Thus $I^{t+1} = \bigcup_{k=0}^{\infty} H_k^{t+1} \subseteq \bigcup_{k=0}^{\infty} H_{k+1}^t = I^t$ and the sequence $I^0 \supseteq I^1 \supseteq I^2 \supseteq \ldots$ is a decreasing chain in $\Sigma$. □

In practice when computing $I^t$ we stop generating the predicates $H_0^t, H_1^t, \ldots$ as soon as a predicate $H_\ell^t$ is found such that $H_\ell^t = H_{\ell+1}^t$. If the limit

$$\bigcup_{k=0}^{\ell} H_k^t$$

of the truncated chain fails to be a resource invariant for $C$, then additional predicates in the sequence $H_j^t$ may have to be computed. Thus the construction of $I^t$ provides a procedure which may be used to obtain successively better approximations to the strongest resource invariant $\mu(F_C)$.

We demonstrate this method of synthesizing resource invariants by considering the program C

```
a:=1
cobegin
   A: cycle P(a); SA; V(a) end
   //
   B: cycle P(a); SB; V(a) end
coend
```

discussed in Section 4. The function $F_C$ in this case is given by

$$F_C(J) = a_1^1=0 \wedge a_1^2=0 \wedge a_2^1=0 \wedge a_2^2=0 \wedge a=0$$
$$\vee J$$
$$\vee SP[a_1^1 := a_1^1+1;\ a := a-1](a_1^1 = a_1^2 \wedge a>0 \wedge J)$$
$$\vee SP[a_1^2 := a_1^2+1;\ a := a+1](a_1^1 - 1 = a_1^2 \wedge J)$$
$$\vee SP[a_2^1 := a_2^1+1;\ a := a-1](a_2^1 = a_2^2 \wedge a>0 \wedge J)$$
$$\vee SP[a_2^2 := a_2^2+1;\ a := a+1](a_2^1 - 1 = a_2^2 \wedge J)$$

While C is quite simple and can be handled by the methods of Section 4, there are potentially an infinite number of states and the chain $F_C^0(\text{false}) \subseteq F_C^1(\text{false}) \subseteq F_C^2(\text{false}) \subseteq \ldots$ does not converge. By computing the sequence of approximations $I^t$ however, we obtain:

$$I^1 = \{a_1^2 \geq 0 \wedge a_2^2 \geq 0\}$$

$$I^2 = \{a_1^2 \geq 0 \wedge a_2^2 \geq 0\}$$

$$I^3 = \{a_1^2 \geq 0 \wedge a_2^2 \geq 0 \wedge a + a_2^1 - a_2^2 \leq 1 \wedge a_2^1 \geq a_2^2$$
$$\wedge a + a_1^2 + a_2^2 - a_1^1 - a_2^1 = 1\}$$

$$I^4 = \{a_1^2 \geq 0 \wedge a_2^2 \geq 0 \wedge a \geq 0 \wedge a + a_2^1 - a_2^2 \leq 1 \wedge a_2^1 \geq a_2^2$$
$$\wedge a + a_1^2 + a_2^2 - a_1^1 - a_2^1 = 1\}$$

$$I^4 = I^5 = I^6 = \ldots$$

Note that $I^4$ is a resource invariant for C and that $I^4$ implies the semaphore invariant $I_a$ used in the proof of absence of deadlock and mutual exclusion in Section 4.

For the PV program used in Section 5 to illustrate the incompleteness of the semaphore invariant method, $I^{16}$ is strong enough to permit a proof of deadlock freedom. The trap state Z no longer causes a problem since $I^{16}$ contains the restraint $(b_1^2 - d_1^1) + (d_2^2 - b_2^2) \leq 1$ which is not satisfied by the unreachable feasible points in the progress graph of the program.

As a final example we consider the standard solution to the *readers and writers problem with writer priority* [BH73] where there are two reader processes and one writer process, e.g.,

```
rr:=0; rw:=0; aw:=0;
a1:=0; b1:=0;
a2:=0; b2:=0;
c:=0; d:=0; e:=0;
resource (rr, rw, aw, a1, b1, a2, b2, c, d, e):
cobegin
   reader_1
   //
   reader_2
   //
   writer
coend
```

Each reader process has the form:

```
reader_i: cycle
   Ai: with R when aw≤0 do ai:=ai+1;
         rr:=rr+1 od;
       read;
   Bi: with R when true do bi:=bi+1;
         rr:=rr-1 od;
       end
```

The writer process is:

```
writer: cycle
   C: with R when true do c:=c+1; aw:=aw+1
        od;
   D: with R when rr≤0 ∧ rw≤0 do d:=d+1;
        rw:=rw+1 od;
   E: with R when true do e:=e+1; aw:=aw-1;
        rw:=rw-1 od;
   end
```

Note that auxiliary variables $a_1$, $b_1$, $a_2$, $b_2$, c, d, and e have been added to the program to count the number of times critical regions $A_1$, $B_1$, $A_2$, $B_2$, C, D, and E are executed. The predicate $I^5$ generated by our approximation procedure is:

$$
\begin{aligned}
I^5 = \{ & aw - c + e = 0 \\
& \wedge rw - d + e = 0 \\
& \wedge rr - a_1 + b_1 - a_2 + b_2 = 0 \\
& \wedge a_1 - b_1 + d - e \leq 1 \\
& \wedge a_2 - b_2 + d - e \leq 1 \\
& \wedge c - e \leq 1 \\
& \wedge a_1 \geq b_1 \geq 0 \\
& \wedge a_2 \geq b_2 \geq 0 \\
& \wedge c \geq d \geq e \geq 0 \}
\end{aligned}
$$

This predicate is a resource invariant for the program and is sufficiently strong to prove absence of deadlock and mutual exclusion of read and write statements.

## 9. Open Problems

If a concurrent program contains a large number of critical regions, then the combinatorial explosion in the number of possible states which must be considered by the approximation procedure of Section 8 may prevent convergence to a suitable resource invariant. We are currently investigating techniques for minimizing this combinatorial explosion. Two techniques which seem promising are:

(A) Preprocessing the program to obtain information about which states can follow a given state during a computation of the program. For example, in the readers and writers problem, assume that reader_1 is waiting for entry into critical region A1 and that aw>0. If reader_2 executes critical region B2, it is unnecessary to check whether reader_1 is enabled to enter A1 since execution of B2 does not affect the value of aw. A similar analysis is currently used in obtaining efficient implementations of conditional critical regions [SC76].

(B) Construct the program and its correctness proof simultaneously. Although the programmer may not precisely know the resource invariant for the program he is writing, he may be able to deduce a first approximation to the invariant from the problem specification. In this case the technique of Sections 7 and 8 may be used to strengthen the approximation. Techniques for deriving correct concurrent programs have been investigated by van Lansweerde and Sintzoff [LA76].

A number of additional questions arise regarding the power of the generalized semaphore invariant of Section 6 and the fixpoint methods for generating resource invariants in Sections 7 and 8. It would be interesting to compare these proof techniques with other techniques which do not use resource invariants, e.g., the Church-Rosser approach of Rosen [RO76] and the reachability tree construction of Keller [KE77]. Also it is not clear how the techniques of this paper generalize to synchronization methods such as path expressions [HA75] for which linear restraints are not explicitly given.

Currently the author is building an automatic verification system for concurrent programs based on the ideas in this paper. This system will extract the "synchronization skeleton" of a concurrent program and use the techniques of Sections 6 and 8 to generate the appropriate resource invariants. The examples of Section 8 were all obtained with the aid of this system.

References


[AG74] Agerwala, T. A complete model for representing the coordination of asynchronous processes. Computer Research Report 32, John Hopkins University, Baltimore, MD.

[BH73] Brinch Hansen, P. Operatin System Principles, Prentice-Hall, NJ, 1973.

[CL77] Clarke, E. M. Program invariants as fixedpoints. 18th Annual Symposium of Foundations of Computer Science, Nov. 1977.

[CO76] Cousot, P. and Cousot, R. Static determination of dynamic properties of programs. Proc. 2nd International Symposium on Programming, B. Robinet, Ed., Dunod, Paris, April 1976.

[CO78] Cousot, P. and Halbwachs, N. Automatic discovery of linear restraints among variables of a program. Proceedings of 5th ACM Symposium on Principles of Programming Languages, 84-96, 1978.

[DI67] Dijkstra, E. W. Cooperating sequential processes. Programming Languages, G. Genuys, Ed., Academic Press, NY, 1968.

[FL77] Flon, L. and Suzuki, N. Nondeterminism and the correctness of parallel programs. Department of Computer Science, Carnegie Mellon University, 1977.

[HA72] Habermann, A. N. Synchronization of communicating processes. CACM, 15(3):171-176, 1972.

[HA75] Habermann, A. N. Path expressions. Department of Computer Science, Carnegie Mellon University, 1975.

[HO72] Hoare, C. A. R. Towards a theory of parallel programming. Operating Systems Techniques, C. A. R. Hoare, R. H. Perrot, Ed., Academic Press, 1972.

[KE77] Keller, R. M. Generalized petri nets as models for system verification. Computer Science Department Technical Report, University of Utah, 1977.

[LA76] van Lamsveerde, A. and Sintzoff, M. Formal derivation of strongly correct parallel programs. MBLE Research Report, Brussels, Belgium, 1976.

[LI75] Lipton, R. J. Reduction: A new method of proving properties of systems of processes. Proceedings of 2nd ACM Symposium on Principles of Programming Lang ages, 78-86, 1975.

[LM77] Lamport, L. Proving the correctness of multiprocess programs. IEEE Transactions on Software Engineering, 3(2):125-143, 1977.

[OW76] Owicki, S. and Gries, D. Verifying properties of parallel programs: An axiomatic approach. CACM 19(5):279-284, 1976.

[PN77] Pneuli, A. The temporal logic of programs. 18th Annual Symposium on Foundations of Computer Science, November 1977.

[RO76] Rosen, B. K. Correctness of parallel programs: The Church-Rosser Approach. Theoretical Computer Science, 2183-207, 1976.

[SC76] Schmid, H. A. On the efficient implementation of conditional critical regions and the construction of monitors. Acta Informatica, 6:227-249, 1976.