Synthesis of Resource Invariants for Concurrent Programs

EDMUND MELSON CLARKE, JR.
Harvard University

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.

Methods are developed in this paper for automatically synthesizing resource invariants. Specifically, the resource invariants of a concurrent program are characterized as *least fixpoints* of a functional which can be obtained from the text of the program. By the use of this fixpoint characterization and a *widening operator* based on convex closure, good approximations may be obtained for the resource invariants of many concurrent programs.

Key Words and Phrases: concurrent program, conditional critical region, correctness proof, resource invariant
CR Categories: 4.32, 5.24

1. INTRODUCTION

Owicki and Gries [17] 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 [16] and Pneuli [18].

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 program-

This research was partially supported by the National Science Foundation under Grant MCS-7508146.
A preliminary version of this paper appeared in the Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages.

Author's address: Center for Research in Computing Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138.

ming language (SCL) in which processes access shared data via conditional critical regions. We consider only *invariance* [18] or *safety* properties [16] of SCL programs. This class of properties includes mutual exclusion and absence of deadlock and is analogous 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 into 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 [10], 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. This 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. 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 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

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

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 [6], however, we are able to speed up the convergence of the chain $F_C^j(false)$ and obtain a close approximation to $\mu(F_C)$ in a finite number of steps. The widening operator that we use represents a set of program states by its *convex closure* in the state space of the program. Although fixpoint techniques have been previously used in the study of resource invariants [9, 14], 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. 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 consists 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*,

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

which permits the simultaneous or interleaved execution of the statements in the *processes* $P1, \ldots, Pn$. 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

$$Pi\textbf{: cycle } S_1^i; S_2^i; \ldots; S_{k_i}^i \textbf{ end},$$

where

$$"S_1^i; S_2^i; \ldots; S_{k_i}^i"$$

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 is not 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* σ is an ordered list $(pc_1, pc_2, \ldots, pc_n; s)$, where

(1) pc_i is the *program counter* for process Pi and is in the range $1 \leq pc_i \leq k_i$.
(2) s maps the set of synchronization variables to the set Z of integers and is called the *program store*.

We 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 $\sigma_0, \sigma_1,$

$\ldots, \sigma_j, \ldots$. The *initial state* σ_0 has the form $(1, 1, \ldots, 1; s_0)$, where s_0 reflects the assignments made in the initialization part of C. Consecutive states

$$\sigma_j = (pc_1^j, \ldots, pc_n^j; s_j) \quad \text{and} \quad \sigma_{j+1} = (pc_1^{j+1}, \ldots, pc_n^{j+1}; s_{j+1})$$

are related as follows: There exists an m, $1 \leq m \leq n$, 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 $\sigma_0, \sigma_1, \ldots, \sigma_j, \ldots$ of program C, then we say that state σ_j is *reachable* from the initial state σ_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 $\sigma = (pc_1, \ldots, pc_n; s)$ is always $S^i_{pc_i}$. We say that program C is *blocked* in state σ if the condition of the next statement to be executed in each process is false in state σ. A state σ of C is a *deadlock state* if σ is reachable from the initial state of C and C is blocked in state σ. Two statements S_1 and S_2 in different processes of C are *mutually exclusive* if there does not exist a state σ, 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 Σ is the set of all program states, then 2^Σ will be the set of all possible predicates, *false* will correspond to the empty state set, and *true* will correspond to the set Σ of all program states. Also, logical operations on predicates can be interpreted as set-theoretic operations on subsets of Σ; i.e., "or" becomes "union," "and" becomes "intersection," "not" becomes "complement," and "implies" becomes "is a subset of."

SP$[A](U)$ denotes 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$[A](\cup_i U_i) = \cup_i$SP$[A](U_i)$.*

PROOF. See [5]. □

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 [12] 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 $\models \mathrm{SP}[A](P) \rightarrow Q$. Proof systems for partial correctness of sequential statements are not 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 $\rightarrow$ VC which give the precondition and postcondition for each statement C in the proof. To ensure that the proofs of sequential correctness for the individual processes are *interference free* [17], 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) $\models \bar{x} = e \rightarrow pre(S_1^i) \wedge \mathrm{IR}$;
(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 $\models \{pre(S_j^i) \wedge b_j^i \wedge \mathrm{IR}\}A_j^i\{post(S_j^i) \wedge \mathrm{IR}\}$.

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 [17]. □

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

$$M(\mathrm{RS}_C) = pre(S_1) \wedge pre(S_2) \wedge \mathrm{IR}$$

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

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

is unsatisfiable. Local deadlock, in which only a subset of the processes is 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 [11] 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 ith 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) $pre(S_1^i) \equiv post(S_{k_i}^i) \equiv \{d_1^i = d_2^i = \cdots = d_{k_i}^i\}$.

(2) If S_j^i is a conditional critical region with associated auxiliary variable d_j^i, then

$$post(S_j^i) = pre(S_j^i)[(d_j^i - 1)/d_j^i].$$

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 are treated as null statements in the analysis which follows. Annotating C as described in the previous paragraph, we obtain

```
{a_1^1 = 0 ∧ a_1^2 = 0 ∧ a_2^1 = 0 ∧ a_2^2 = 0 ∧ 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 $M(\mathrm{SI}_C) = pre(\mathrm{SA}) \wedge pre(\mathrm{SB}) \wedge I_a$ is unsatisfiable, it follows that statements SA and SB are mutually exclusive. Similarly, we see that C is free from deadlock, since the predicate $D(\mathrm{SI}_C)$ 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* [3]. 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 (see Figure 1). 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:

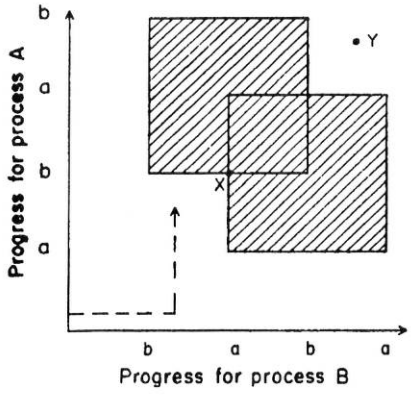

Figure 1

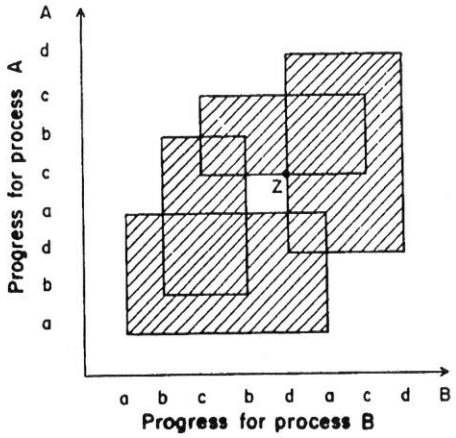

Figure 2

$a := b := c := d := 1$
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 in Figure 2. 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

$$\begin{aligned} I = \{ & a = 1 + a_1^2 + a_2^2 - a_1^1 - a_2^1 \wedge a \geq 0 \\ & \wedge b = 1 + b_1^2 + b_2^2 - b_1^1 - b_2^1 \wedge b \geq 0 \\ & \wedge c = 1 + c_1^2 + c_2^2 - c_1^1 - c_2^1 \wedge c \geq 0 \\ & \wedge d = 1 + d_1^2 + d_2^2 - d_1^1 - d_2^1 \wedge d \geq 0 \\ & \wedge a_1^2 \geq 0 \wedge a_2^2 \geq 0 \wedge a_1^1 \geq 0 \wedge a_2^1 \geq 0 \\ & \wedge b_1^2 \geq 0 \wedge b_2^2 \geq 0 \wedge b_1^1 \geq 0 \wedge b_2^1 \geq 0 \\ & \wedge c_1^2 \geq 0 \wedge c_2^2 \geq 0 \wedge c_1^1 \geq 0 \wedge c_2^1 \geq 0 \\ & \wedge d_1^2 \geq 0 \wedge d_2^2 \geq 0 \wedge d_1^1 \geq 0 \wedge d_2^1 \geq 0 \}. \end{aligned}$$

It is not difficult to show that the condition $D(\mathrm{SI}_C)$ for absence of deadlock is satisfied by the state Z in which

$$\begin{array}{lllll} a = 0, & a_1^2 = 1, & a_2^2 = 0, & a_1^1 = 1, & a_2^1 = 1, \\ b = 0, & b_1^2 = 0, & b_2^2 = 1, & b_1^1 = 1, & b_2^1 = 1, \\ c = 0, & c_1^2 = 0, & c_2^2 = 0, & c_1^1 = 0, & c_2^1 = 1, \\ d = 0, & d_1^2 = 0, & d_2^2 = 0, & d_1^1 = 1, & d_2^1 = 0. \end{array}$$

Thus absence of deadlock cannot be proved by means of the semaphore invariant

method. The state Z which satisfies $D(\mathrm{SI}_C)$ is an example of an unreachable feasible state in which each process of C is blocked; we 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 $D(\mathrm{SI}_C)$ is unsatisfiable *if and only if* deadlock is impossible for the program C. Clearly, if $D(\mathrm{SI}_C)$ is unsatisfiable, then deadlock is impossible. Thus assume that $D(\mathrm{SI}_C)$ is satisfied by some state σ. By construction of the predicate D all processes are blocked in state σ. Since σ 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? One possibility is to cover the "holes" in the unfeasible region of a progress graph by means of additional linear constraints. A technique for generating the new constraints is discussed 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 [10] 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 in [5] that the semaphore invariant method is as powerful as the *reduction method* of Lipton [15]: If a *PV* program has a reduction proof of deadlock freedom, then it also has a proof using 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

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

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 $a_1x_1 + a_2x_2 + \cdots + a_nx_n + a_{n+1} \leq 0$;
(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.,

$$\begin{aligned} x_1 &:= x_1 + b_1, \\ X_2 &:= x_2 + b_2, \\ &\vdots \\ X_n &:= x_n + b_n. \end{aligned}$$

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 smokers problem, can all be expressed as linear SCL programs. Arguments are given in [14] and [20] 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 $H_i(\bar{x}) = a_1^i x_1 + a_2^i x_2 + \cdots + a_n^i x_n + a_{n+1}^i$ be a linear form occurring in the condition of some critical region of C. We 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

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

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

$$dH_i = \sum_j \frac{\partial H_i}{\partial S_j} dS_j$$

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

In this section we show that the resource invariants of an SCL program C are fixpoints of a function F_C which can be obtained from the text of C. Before describing the functional F_C, we must introduce some additional terminology; as in Section 2 we identify predicates with subsets of the set Σ of all program states. Let F be a functional which maps the predicates into predicates; i.e., $F : 2^\Sigma \to 2^\Sigma$. 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 $U_0 \subseteq U_1 \subseteq \cdots \subseteq U_j \subseteq \cdots$ of subsets of Σ,

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

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

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

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

Let C be an SCL program having the form described in Section 2, where a single resource $R(\bar{x})$ is shared by n processes $P_1, \ldots, P_n$. We further assume that C contains K critical regions $S_1, \ldots, S_K$, that the ith critical region has the form **with** R **when** b_i **do** A_i **od**, and that pre and post functions for S_i are computed using the algorithm of Sections 4 and 6. The fixpoint functional $F_C : 2^{\Sigma} \rightarrow 2^{\Sigma}$ is defined by

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

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

THEOREM 7.1

(a) *The functional F_C is a continuous mapping on 2^{Σ}. Thus F_C has a least fixpoint $\mu(F_C)$ which is given by*

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

(b) *All resource invariants IR of C are fixpoints of F_C.*

(c) *The least fixpoint $\mu(F_C)$ is a resource invariant for C and can be characterized as the set of states which occur in valid computations of C starting from initial state σ_0; i.e.,*

$$\mu(F_C) = \{\sigma \mid \sigma_0 \xrightarrow{C} \sigma\}.$$

(d) *The resource invariant 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.*

(e) *If L is a predicate such that $L \subseteq \mu(F_C)$, then*

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

The proof is given in the appendix.

Part (d) of Theorem 7.1 shows that $\mu(F_C)$ is the "strongest" resource invariant for program C; part (e) is important because it gives a method for improving approximations to $\mu(F_C)$. To illustrate Theorem 7.1, 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 ∧ b = 0 ∧ a1 = 0 ∧ a2 = 0 ∧ b1 = 0 ∧ b2 = 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

$$\begin{aligned} F_C(J) = {} & a = 0 \wedge b = 0 \wedge a_1 = 0 \wedge a_2 = 0 \wedge b_1 = 0 \wedge b_2 = 0 \\ & \vee J \\ & \vee \mathrm{SP}[a := a + 1;\ a_1 := a_1 + 1](b = 0 \wedge a_1 = a_2 \wedge J) \\ & \vee \mathrm{SP}[a := a - 1;\ a_2 := a_2 + 1](\mathit{true} \wedge a_1 - 1 = a_2 \wedge J) \\ & \vee \mathrm{SP}[b := b + 1;\ b_1 := b_1 + 1](a = 0 \wedge b_1 = b_2 \wedge J) \\ & \vee \mathrm{SP}[b := b - 1;\ b_2 := b_2 + 1](\mathit{true} \wedge b_1 - 1 = b_2 \wedge J). \end{aligned}$$

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

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

It is easily checked that $F_C(\mathrm{IR}) = \mathrm{IR}$ so that IR is a fixpoint of F_C. Since $D = \{\mathit{pre}(A1) \wedge b = 0 \wedge \mathit{pre}(B1) \wedge a = 0 \wedge \mathrm{IR}\}$ 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 $M = \{\mathit{pre}(\mathrm{SA}) \wedge \mathit{pre}(\mathrm{SB}) \wedge \mathrm{IR}\}$ is satisfiable. By using Theorem 7.1(e) we may compute the strongest resource invariant $\mu(F_C)$. Let $L = \{\mathrm{IR} \wedge a_1 = a_2 \wedge b_1 =$

$b_2\}$; then $L \subseteq \mu(F_C)$. Since

$$F_C^3(L) = F_C^4(L) = \cdots = \{\mathrm{IR} \wedge a = 1 \rightarrow b = 0 \wedge b = 1 \rightarrow a = 0\},$$

we see that

$$\mu(F_C) = \bigcup_{i=0}^{\infty} F_C^i(L) = F_C^3(L)$$

$$= \{a = a_1 - a_2 \wedge b = b_1 - b_2 \wedge a = 1 \rightarrow b = 0$$

$$\wedge b = 1 \rightarrow a = 0 \wedge a_1 \geq 0 \wedge a_2 \geq 0 \wedge b_1 \geq 0 \wedge b_2 \geq 0\}.$$

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

Note that Theorem 7.1(e) 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 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 [6] may be used to speed up convergence to $\mu(F_C)$. The widening operator $*$ is characterized by the following two properties:

(a) For all *admissible* predicates U and V, $U \subseteq U * V$ and $V \leq U * V$.
(b) For any ascending chain of admissible predicates $U_0 \subseteq U_1 \subseteq U_2 \subseteq \cdots$, 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 *admissible* 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 [7]. Let U and V be polygonal convex sets. Then U and V can be represented as conjunctions

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

where each conjunct is a linear inequality of the form $a_1x_1 + \cdots + a_mx_m + a_{m+1} \leq 0$. 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 γ_j and δ_k are *equivalent* if they determine the same half space of Q^m. $U * V$ is the conjunction of all those γ_j in the representation of U for which there is an equivalent δ_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 $F_C^i(false)$ in the chain $F_C^0(false) \subseteq F_C^1(false) \subseteq \cdots$ may not be polygonal convex

sets, let $G_i = \mathrm{CV}[F_C^i(false)]$ where CV is the *convex hull operator*. The sequence $G_0 \subseteq G_1 \subseteq \cdots$ 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 Σ; i.e.,* $I^1 \supseteq I^2 \supseteq I^3 \cdots$.
(d) *Suppose that $\mu(F_C) = \Lambda L$, where L is a finite set of linear inequalities. Then there exists an $r \geq 0$ such that $I^t = \Lambda L$ for $t \geq r$.*

The proof is given in the appendix.

In practice, when computing I^t we stop generating the predicates $H_0^t, H_1^t, \ldots$ as soon as a predicate H_l^t is found such that $H_l^t = H_{l+1}^t$. If the limit $\bigcup_{k=0}^{l} H_k^t$ 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 $\mu(F_C)$.

Part (d) of Theorem 8.1 shows that the sequence of approximations produced by our method will converge exactly to the strongest resource invariant in a finite number of steps when the strongest invariant is the conjunction of a finite set L of inequalities. Since the inequalities in L may be arbitrarily complicated, this result proves that our method is strictly more powerful than methods like the semaphore invariant method in which all inequalities must have a particular form. However, since each I^t will be a conjunction of linear inequalities, it is possible to construct example SCL programs for which the sequence in Theorem 8.1(c) fails to converge. Thus, in general, there will still be programs for which the resource invariants provided by our method are insufficiently powerful to prove mutual exclusion and absence of deadlock. Nevertheless, our method is quite powerful in practice, because most real synchronization problems are counting problems with relatively simple resource invariants.

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

$$\begin{aligned} F_C(J) = {} & a_1^1 = 0 \wedge a_1^2 = 0 \wedge a_2^1 = \wedge\, a_2^2 = 0 \wedge a = 0 \\ & \vee J \\ & \vee \mathrm{SP}[a_1^1 := a_1^1 + 1;\ a := a - 1](a_1^1 = a_1^2 \wedge a > 0 \wedge J) \\ & \vee \mathrm{SP}[a_1^2 := a_1^2 + 1;\ a := a + 1](a_1^1 - 1 = a_1^2 \wedge J) \\ & \vee \mathrm{SP}[a_2^1 := a_2^1 + 1;\ a := a - 1](a_2^1 = a_2^2 \wedge a > 0 \wedge J) \\ & \vee \mathrm{SP}[a_2^2 := a_2^2 + 1;\ a := a + 1](a_2^1 - 1 = a_2^2 \wedge J). \end{aligned}$$

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 $F_C^0(\mathit{false}) \subseteq F_C^1(\mathit{false}) \subseteq F_C^2(\mathit{false}) \subseteq \cdots$. does not converge. By computing the sequence of approximations I^t however, we obtain

$$\begin{aligned} I^1 &= \{a_1^2 \geq 0 \wedge a_2^2 \geq 0\}, \\ I^2 &= \{a_1^2 \geq 0 \wedge a_2^2 \geq 0\}, \\ I^3 &= \{a_1^2 \geq 0 \wedge a_2^2 \geq 0 \wedge a + a_2^1 - a_2^2 \leq 1 \wedge a_2^1 \geq a_2^2 \wedge a \\ &\qquad - a_1^2 - a_2^2 + a_1^1 + a_2^1 = 1\}, \\ I^4 &= \{a_1^2 \geq 0 \wedge a_2^2 \geq 0 \wedge a \geq 0 \wedge a + a_2^1 - a_2^2 \leq 1 \wedge a_2^1 \\ &\qquad \geq a_2^2 \wedge a - a_1^2 - a_2^2 + a_1^1 + a_2^1 = 1\}, \\ I^4 &= I^5 = I^6 = \cdots. \end{aligned}$$

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* [2], 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**
  A_i: **with** R **when** $aw \leq 0$ **do** $a_i := a_i + 1$; $rr := rr + 1$ **od**;
    read;
  B_i: **with** R **when** *true* **do** $b_i := b_i + 1$; $rr := rr - 1$ **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** $rr \leq 0 \wedge rw \leq 0$ **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

(1) *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 [20].

(2) *Constructing 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 [14].

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 [19] and the reachability tree construction of Keller [13]. Also, it is not clear how the techniques of this paper generalize to synchronization methods, such as path expressions [11] for which linear restraints are not explicitly given, and to other properties of concurrent programs, such as absence of starvation for which more complicated proof techniques are required.

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.

APPENDIX

Proof of Theorem 7.1

(a) The continuity of F_C follows directly from the additivity of SP.

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

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

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

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

Hence

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

So

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

Thus every resource invariant IR is a fixpoint of F_C.

(c) Since $\mu(F_C)$ is a fixpoint of F_C, we have

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

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

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

By construction of the pre and post functions, we also have

$$\mathrm{SP}[A_i](pre(S_i)) \subseteq post(S_i).$$

By monotonicity,

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

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

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

or, equivalently, that

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

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

Next let $\mathrm{IR} = \{\sigma \mid \sigma_0 \xrightarrow{C} \sigma\}$. We show that IR is a fixpoint of F_C and that $\mathrm{IR} \subseteq \mu(F_C)$. Since $\mu(F_C)$ is the least fixpoint of F_C, it follows that $\mu(F_C) = \mathrm{IR}$.

(i) $F_C(\mathrm{IR}) = \mathrm{IR}$. Clearly $\mathrm{IR} \subseteq F_C(\mathrm{IR})$. Let

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

Then either $\sigma \in J_0 \subseteq \mathrm{IR}$ or $\sigma \in \mathrm{IR}$ or there exists i_0 such that

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

Only the third case is interesting. If

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

then there is a state $\sigma' \in pre(S_{i_0}) \wedge b_{i_0} \wedge \mathrm{IR}$ such that $A_{i_0}(\sigma') = \sigma$. Since $\sigma' \in \mathrm{IR}$, 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 $\sigma = A_{i_0}(\sigma')$, $\sigma_0, \ldots, \sigma_r, \sigma$ is also a computation of C and $\sigma \in \mathrm{IR}$.

(ii) $\mathrm{IR} \subseteq \mu(F_C)$. Let $\sigma \in \mathrm{IR}$. Then there exists a computation $\sigma_0, \sigma_1, \ldots, \sigma_r$ in which $\sigma_r = \sigma$. We prove by induction on r that $\sigma_r \in F_C^{r+1}(false)$. Since $F_C^1(false) = J_0$, the basis case $\sigma_0 \in F_C^1(false)$ is true. Assume that for all computations $\sigma_0, \sigma_1, \ldots, \sigma_{r-1}$ of C that $\sigma_{r-1} \in F_C^r(false)$. Let $\sigma_0, \sigma_1, \ldots, \sigma_{r-1}, \sigma_r$ be a computation of length r. Then there exists an 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

$$\sigma_r \in \mathrm{SP}[A_{i_0}](pre(S_{i_0}) \wedge b_{i_0} \wedge F_C^r(false)) \subseteq F_C^{r+1}(false).$$

It follows that

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

(d) We prove that for the resource invariant systems RS_C, the condition $D(\mathrm{RS}_C)$ is unsatisfiable *iff* deadlock is impossible. Clearly, if $D(\mathrm{RS}_C)$ is unsatisfiable, then deadlock is impossible. We must show that if $D(\mathrm{RS}_C)$ is satisfiable, then there exists a state σ_d which is reachable from the initial state of C in which every process of C is blocked. Let σ_d be a program state which satisfies $D(\mathrm{RS}_C)$. Since σ_d satisfies $D(\mathrm{RS}_C)$, it follows that $\sigma_d \in \mu(F_C)$ and also that each process of C is blocked in state σ_d. Since $\sigma_d \in \mu(F_C)$, σ_d is reachable from the initial state σ_0 of C. Thus σ_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.

(e) It is easy to show that for all $j \geq 0$,

$$F^j(false) \subseteq F^j(L) \subseteq F^j(\mu(F_C)).$$

Thus

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

PROOF OF THEOREM 8.1

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

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

(b) $$\mu(F_C) = \bigcup_{i=0}^{\infty} F^i(false) \subseteq \bigcup_{i=0}^{\infty} \mathrm{CV}[F^i(false)] \subseteq \bigcup_{j=0}^{\infty} G_j \subseteq \bigcup_{j=0}^{\infty} G_{t+j} \subseteq \bigcup_{j=0}^{\infty} H_j^t \subseteq I^t.$$

(c) Let U and V be polygonal convex sets. We write $U \sqsubset V$ if for every conjunct δ 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 that $H_k^{t+1} \sqsubset H_{k+1}^t$; 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 $H_k^{t+1} \subset H_{k+1}^t$. 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 $I^0 \supseteq I^1 \supseteq I^2 \supseteq \cdots$ is a decreasing chain in Σ.

(d) Assume that $\mu(F_C) = \wedge L$. Since program states are m-tuples of integers and every point of $\mu(F_C)$ is reachable by some computation of C, there must exist a

set of linear inequalities M and two index values r, s $(0 \leq r \leq s)$ which satisfy the following three conditions:

(1) $G_r = (\wedge L) \wedge (\wedge M)$;
(2) $G_t \sqsubset \wedge L$ for $t \geq r$; and
(3) no inequality of M is equivalent to any conjunct of G_s.

Next, consider $I^r = \bigcup_{j=0}^{\infty} H_j^r$. By induction on j and conditions (1) and (2) above, we see that $(\wedge L) \wedge (\wedge M) \sqsubset H_j^r$ and $H_j^r \sqsubset \wedge L$ for $j \geq 0$. Since $H_{s-r}^r = H_{s-r-1}^r * G_{s-r+r} = H_{s-r-1}^r * G_s$ and no inequality γ of M is equivalent to any conjunct of G_s, it follows that $H_{s-r}^r = \wedge L$. Since $H_j^r \subseteq \wedge L$ for $j \geq 0$, we see that

$$I^r = \bigcup_{j=0}^{\infty} H_j^r = \wedge L.$$

For $t \geq r$ we have $\wedge L \subseteq I^t$ and also $I^t \subseteq I^r \subseteq L$. Thus $I^t = \wedge L$ for $t \geq r$. □

REFERENCES

(Note. References [8, 12] are not cited in the text.)

1. AGERWALA, T. A complete model for representing the coordination of asynchronous processes. Computer Research Report 32, Johns Hopkins University, Baltimore, Md., 1974.
2. BRINCH HANSEN, P. *Operating System Principles*. Prentice-Hall, Englewood Cliffs, N.J., 1973.
3. COFFMAN, E.G. JR., ELPHICH, M.J., AND SHOSHANI, A. Systems deadlock. *Comput. Surv. 3*, 2 (June 1971), 67–78.
4. CLARKE, E.M. Program invariants as fixed points. 18th Ann. IEEE Symp. on Foundations of Computer Science, Providence, R.I., Nov. 1977, pp. 18–28.
5. CLARKE, E.M. Synthesis of Resource Invariants for Concurrent Programs. Proc. 6th ACM Symp. on Principles of Programming Languages, San Antonio, Texas, Jan. 1979, pp. 211–221.
6. COUSOT, P., AND COUSOT, R. Static determination of dynamic properties of programs. Proc. 2nd Int. Symp. on Programming, B. Robinet, Ed., Dunod, Paris, April 1976.
7. COUSOT, P., AND HALBWACHS, N. Automatic discovery of linear restraints among variables of a program. Proc. 5th ACM Symp. on Principles of Programming Languages, Tucson, Ariz., 1978, pp. 84–96.
8. DIJKSTRA, E.W. Cooperating sequential processes. In *Programming Languages*, G. Genuys, Ed., Academic Press, N.Y., 1968, pp. 43–112.
9. FLON, L., AND SUZUKI, N. Nondeterminism and the correctness of parallel programs. Tech. Rep., Dep. of Computer Science, Carnegie-Mellon Univ., May 1977.
10. HABERMANN, A.N. Synchronization of communicating processes. *Commun. ACM 15*, 3 (March 1972), 171–176.
11. HABERMANN, A.N. Path expressions. Tech. Rep., Dep. of Computer Science, Carnegie-Mellon Univ., June 1975.
12. HOARE, C.A.R. Towards a theory of parallel programming. In *Operating Systems Techniques*, C.A.R. Hoare and R.H. Perrot, Eds., Academic Press, N.Y., 1972, pp. 61–71.
13. KELLER, R.M. Generalized petri nets as models for system verification. Tech. Rep., Computer Science Dep., Univ. of Utah, 1977.
14. VAN LAMSVEERDE, A., AND SINTZOFF, M. Formal derivation of strongly correct parallel programs. MBLE Research Rep., Brussels, Belgium, 1976.
15. LIPTON, R.J. Reduction: A new method of proving properties of systems of processes. Proc. 2nd ACM Symp. on Principles of Programming Languages, Palo Alto, Calif., 1975, pp. 78–86.
16. LAMPORT, L. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng. 3*, 2 (1977), 125–143.
17. OWICKI, S., AND GRIES, D. Verifying properties of parallel programs: An Axiomatic approach. *Commun. ACM 19*, 5 (1976) 279–284.

18. PNEULI, A. The temporal logic of programs. 18th Ann. IEEE Symp. on Foundations of Computer Science, Providence, R.I., Nov. 1977, pp. 46–57.
19. ROSEN, B.K. Correctness of parallel programs: The Church-Rosser approach. *Theor. Comput. Sci. 2* (1976), 183–207.
20. SCHMID, H.A. On the efficient implementation of conditional critical regions and the construction of monitors. *Acta Inf. 6*, 3 (1976), 227–249.

Received March 1979; revised January 1980; accepted April 1980