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 fixedpoints 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 Pnueli [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.
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. Correct-
ness 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 Haber-
mann [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 character-
ization 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 FC, which can be obtained from the text of program
C, and that the least fixpoint μ(FC) of FC is the "strongest" such resource
invariant. Since the functional FC is continuous, the least fixpoint μ(FC) may be
expressed as the limit

μ(FC) = \bigcup_{i=1}^{\infty} F_{C}^{i}(false).

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

By using the notion of widening of Cousot [6], however, we are able to speed
up the convergence of the chain F_{C}(false) and obtain a close approximation to
μ(FC) 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 μ(FC). 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,

\[
\text{resource } R(\bar{x}):
\text{cobegin } P1//P2// \cdots Pn \text{ coend},
\]

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

\[
P_i: \text{cycle } S_{i1}; S_{i2}; \ldots; S_{ik} \text{ end},
\]

where

“\(S_{i1}; S_{i2}; \ldots; S_{ik}\)”

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 \(S_{ik}\) is the first statement \(S_{i1}\) 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 \(\sigma\) is an ordered list \((pc_1, pc_2, \ldots, pc_k; s)\), where

1. \(pc_i\) is the program counter for process \(P_i\) 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 \(\sigma_0, \sigma_1, \ldots\).
The initial state $\sigma_0$ has the form $(1, 1, \ldots, 1; s_0)$, where $s_0$ reflects the assignments made in the initialization part of $C$. Consecutive states

$$\sigma_i = (pc_1, \ldots, pc_n; s_i) \quad \text{and} \quad \sigma_{i+1} = (pc_1^{i+1}, \ldots, pc_n^{i+1}; s_{i+1})$$

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

1. $pc_i^{i+1} = pc_i^i$ if $i \neq m$;
2. $pc_m^{i+1} = \begin{cases} pc_m^i + 1 & \text{if } pc_m^i < k_m, \\ 1 & \text{otherwise} \end{cases}$
3. if statement $pc_m^i$ in process $m$ is with $R$ when $b$ do $A$ od, then $b(s_i) = \text{true}$ and $s_{i+1} = A(s_i)$.

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 $\sigma_0, \sigma_1, \ldots, \sigma_i, \ldots$ of program $C$, then we say that state $\sigma_i$ is reachable from the initial state $\sigma_0$ of $C$ and write $\sigma_0 \rightarrow^* \sigma_i$. Note that the next statement to be executed by process $pi$ in state $\sigma = (pc_1, \ldots, pc_n; s)$ is always $S_{pi}$. 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 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.”

$SP[A](U)$ 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 $SP[A](U) = \{ (pc_1, \ldots, pc_n; A(s)) | (pc_1, \ldots, pc_n, s) \in U \}$.

**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]. \qed

**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 $P$.
postcondition \( Q \). The triple \( \{ P \} A \{ Q \} \) is true \( \iff \{ P \} A \{ Q \} \) iff \( \{ \text{SP} \} A \{ P \} \to Q \).

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 \( RSC \) 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 \( \text{pre}, \text{post}: ST \to VC \) 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 \( \text{pre} \) and \( \text{post} \) for process \( Pi \) must also satisfy the following conditions:

\begin{align*}
(a) & \vdash x = e \to \text{pre}(S^i_j) \land \text{IR}; \\
(b) & \vdash \text{post}(S^i_k) \to \text{pre}(S^i_j); \\
(c) & \vdash \text{post}(S^j_i) \to \text{pre}(S^j_j) \land \text{IR} \text{ for } 1 \leq j \leq k_i - 1; \\
(d) & \text{if } S^j_j \text{ is the conditional critical region}
\end{align*}

\begin{align*}
& \text{with } R \text{ when } b^j \text{ do } A \text{ od,} \\
& \text{then } \vdash \{ \text{pre}(S^j_j) \land b^j \land \text{IR} \} A^j_j \{ \text{post}(S^j_j) \land \text{IR} \}.
\end{align*}

**Theorem 3.1.** Let \( RSC \) be a resource invariant system for the SCL program \( C \). If \( a \) is reachable in \( C \) and \( S^j_j \) is the next statement of process \( Pi \) to execute in state \( a \), then \( a \in \text{pre}(S^j_j) \).

**Proof.** See [17]. \( \Box \)

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 \( S^i_j \) and \( S^i_k \) in \( C \), it is sufficient to give a resource invariant system \( RSC \) for \( C \) such that

\[ M(RSC) = \text{pre}(S^i_1) \land \text{pre}(S^i_2) \land \text{IR} \]

is unsatisfiable. To prove that it is impossible for \( C \) to become deadlocked, it is sufficient to exhibit a resource invariant system \( RSC \) such that the predicate

\[ D(RSC) = \bigwedge_{i=1}^{n} \left( \bigvee_{j=1}^{k_i} \text{pre}(S^j_j) \land \neg b^j_j \right) \land \text{IR} \]

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 \texttt{with R when x > 0 do x := x - 1 od} and V(x) to \texttt{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 \textit{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 \( a_i \) 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 \( \alpha_i \). All auxiliary variables are initialized to zero at the beginning of the program C. The \textit{semaphore invariant} states that the predicate \( I_a = (a = m + \Sigma a_i - \Sigma \alpha_i \land a \geq 0) \) 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 \( S_1; \ldots; S_k \) end" be the ith process in the program C, and let \( d_1, \ldots, d_k \) be the auxiliary variables for this process. The pre and post functions for process Pi will be defined inductively:

1. \( \text{pre}(S_i) = \text{post}(S_i) = \{d_1 = d_1 = \ldots = d_k\} \).
2. If \( S_i \) is a conditional critical region with associated auxiliary variable \( d_i \), then
\[
\text{post}(S_i) = \text{pre}(S_i)[(d_i - 1)/d_i] .
\]

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 \textit{semaphore invariant system} (\( \text{SIC} \)) corresponding to C.

Consider, for example, the PV program C:
\[
\begin{align*}
& a := 1 \\
& \text{cobegin} \\
& \phantom{\text{cobegin}} A: \text{cycle P(a); SA; V(a) end} \\
& \phantom{\text{cobegin}} // \\
& \phantom{\text{cobegin}} B: \text{cycle P(a); SB; V(a) end} \\
& \text{coend}
\end{align*}
\]

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
\[
\{a = 0 \land a_1 = 0 \land a_2 = 0 \land a_3 = 0 \land a = 1\}
\]

resource \( R(a, a_1, a_2, a_3) \):
\[
\begin{align*}
& \text{cobegin} \\
& \phantom{\text{cobegin}} \text{A: cycle} \\
& \phantom{\text{cobegin}} \quad \{a_1 = a_1\} \\
& \phantom{\text{cobegin}} \quad \text{with R when} \ a > 0 \ \text{do} \ a_1 := a_1 + 1; \ a := a - 1 \ \text{od}; \\
& \phantom{\text{cobegin}} \quad (a_1 - 1 = a_1) \\
& \phantom{\text{cobegin}} \quad \text{SA;} \\
& \phantom{\text{cobegin}} \quad \{a_1 - 1 = a_1\} \\
& \phantom{\text{cobegin}} \quad \text{with R when} \ \text{true do} \ a_3 := a_3 + 1; \ a := a + 1 \ \text{od}; \\
& \phantom{\text{cobegin}} \text{end} \\
& \phantom{\text{cobegin}} //
\end{align*}
\]
B: cycle
{a_i = a^j}
with R when a > 0 do a^i := a^i + 1; a := a - 1 od;
{a^i - 1 = a^j}
SB;
{a_i - 1 = a^j}
with R when true do a^j := a^j + 1; a := a + 1 od;
end
coend

The invariant $I_a$ for semaphore $a$ is

$$I_a = \{ a = 1 + a^i + a^j - a^i - a^j \land a \geq 0 \}.$$ 

Since the predicate $M(SI_C) = pre(SA) \land pre(SB) \land I_a$ 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 INARIANT 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 $C$:

$$c := 1; b := 1$$

cobegin
  $$A: \text{cycle } P(a); P(b); V(a); V(b) \text{ end}$$

  $$B: \text{cycle } P(b); P(a); V(b); V(a) \text{ end}$$

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 unreachable 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 $C$:

\[ a := b := c := d := 1 \]
\[
\text{cobegin}
\begin{align*}
  A & : \text{cycle } P(a); P(b); P(d); V(a); P(c); V(b); V(c); V(d) \text{ end} \\
  & // \\
  B & : \text{cycle } P(a); P(b); P(c); V(b); P(d); V(a); V(c); V(d) \text{ end} \\
\text{coend}
\end{align*}
\]

The progress graph for \( C \) is shown in Figure 2. Note that deadlock can never occur during execution of program \( C \). Let \( \text{SI}_c \) 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
\[
I = (a = 1 + a^1 + a^2 - a^1 - a^2 \land a \geq 0 \\
\land b = 1 + b^1 + b^2 - b^1 - b^2 \land b \geq 0 \\
\land c = 1 + c^1 + c^2 - c^1 - c^2 \land c \geq 0 \\
\land d = 1 + d^1 + d^2 - d^1 - d^2 \land d \geq 0 \\
\land a^1 \geq 0 \land a^2 \geq 0 \land a^1 \geq 0 \land a^2 \geq 0 \\
\land b^1 \geq 0 \land b^2 \geq 0 \land b^1 \geq 0 \land b^2 \geq 0 \\
\land c^1 \geq 0 \land c^2 \geq 0 \land c^1 \geq 0 \land c^2 \geq 0 \\
\land d^1 \geq 0 \land d^2 \geq 0 \land d^1 \geq 0 \land d^2 \geq 0).
\]

It is not difficult to show that the condition \( D(\text{SI}_c) \) for absence of deadlock is satisfied by the state \( Z \) in which
\[
\begin{align*}
  a &= 0, \quad a^1 = 1, \quad a^2 = 0, \quad a^1 = 1, \quad a^2 = 1, \\
  b &= 0, \quad b^1 = 0, \quad b^2 = 1, \quad b^1 = 1, \quad b^2 = 1, \\
  c &= 0, \quad c^1 = 0, \quad c^2 = 0, \quad c^1 = 0, \quad c^2 = 1, \\
  d &= 0, \quad d^1 = 0, \quad d^2 = 0, \quad d^1 = 1, \quad d^2 = 0.
\end{align*}
\]

Thus absence of deadlock cannot be proved by means of the semaphore invariant.
method. The state \( Z \) which satisfies \( D(\text{SIL}_C) \) is an example of an unreachable feasible state in which each process of \( C \) is blocked; we call such states \textit{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 \( \text{SIL}_C \) be the semaphore invariant system for \( C \). We show that the condition \( D(\text{SIL}_C) \) is unsatisfiable \textit{if and only if} deadlock is impossible for the program \( C \). Clearly, if \( D(\text{SIL}_C) \) is unsatisfiable, then deadlock is impossible. Thus assume that \( D(\text{SIL}_C) \) 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. \( \square \)

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

\[
\text{with } R \text{ when } B(x_1, x_2, \ldots, x_n) \text{ do } A(x_1, x_2, \ldots, x_n) \text{ od,}
\]

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 \( a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + a_{n+1} \leq 0 \);
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.,

\[
\begin{align*}
  x_1 &:= x_1 + b_1, \\
  x_2 &:= x_2 + b_2, \\
  \vdots \\
  x_n &:= x_n + b_n.
\end{align*}
\]
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 \( H_i(\vec{x}) = a_1^i x_1 + a_2^i x_2 + \cdots + a_n^i x_n + a_{n+1}^i \) be a linear form occurring in the condition of some critical region of \( C \). We use the notation \( \frac{\partial H_i}{\partial S_j} \) to denote the change in value of \( H_i \) caused by the execution of statement \( S_j \); note that \( \frac{\partial H_i}{\partial S_j} \) is given by

\[
\frac{\partial H_i}{\partial S_j} = \sum_{j=1}^{n} a_j^i b_j.
\]

Let \( dH_i = H(\vec{x}) - H(\vec{x}_0) \), where \( \vec{x}_0 \) 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 \( \Sigma \) of all program states. Let \( F \) be a functional which maps the predicates into predicates; i.e., \( F : 2^\Sigma \rightarrow 2^\Sigma \). 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 \( \Sigma \),

\[
F\left( \bigcup_{j=0}^{\infty} U_j \right) = \bigcup_{j=0}^{\infty} F(U_j).
\]
If $F$ is continuous, then $F$ has a least fixpoint $\mu(F)$ which is given by

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

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 $R(x)$ 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 $i$th critical region has the form

\begin{align*}
\text{with } R \text{ when } b, \text{ do } A_i, \text{ od,}
\end{align*}

and that pre and post functions for $S_i$ are computed using the algorithm of Sections 4 and 6. The fixpoint functional $F_C : 2^Z \rightarrow 2^Z$ is defined by

$$
F_C(J) = J_0 \lor J \lor \bigvee_{i=1}^{K} \text{SP}[A_i](\text{pre}(S_i) \land b_i \land J),
$$

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

**Theorem 7.1**

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

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

(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 $a_0$; i.e.,

$$
\mu(F_C) = \{ a | a_0 \xrightarrow{C} a \}.
$$

(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

cobegin
A: cycle \{a_1 = a_2\}
A1: with R when b = 0 do a := a + 1; a_1 := a_1 + 1 od;
   (a_1 - 1 = a_2)
   SA;
   (a_1 - 1 = a_2)
A2: with R when true do a := a - 1; a_2 := a_2 + 1 od
end

//
B: cycle \{b_1 = b_2\}
B1: with R when a = 0 do b := b + 1; b_1 := b_1 + 1 od;
   (b_1 - 1 = b_2)
   SB;
   (b_1 - 1 = b_2)
B2: with R when true do b := b - 1; b_2 := b_2 + 1 od
end
cobegin

In this case the function $F_C$ is

$$F_C(J) = a = 0 \land b = 0 \land a_1 = 0 \land a_2 = 0 \land b_1 = 0 \land b_2 = 0$$

$$\lor J$$

$$\lor SP[a := a + 1; a_1 := a_1 + 1](b = 0 \land a_1 = a_2 \land J)$$

$$\lor SP[a := a - 1; a_2 := a_2 + 1](true \land a_1 - 1 = a_2 \land J)$$

$$\lor SP[b := b + 1; b_1 := b_1 + 1](a = 0 \land b_1 = b_2 \land J)$$

$$\lor SP[b := b - 1; b_2 := b_2 + 1](true \land b_1 - 1 = b_2 \land J).$$

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

$$IR = \{a = a_1 - a_2 \land b = b_1 - b_2 \land a_1 \geq 0 \land a_2 \geq 0 \land b_1 \geq 0 \land b_2 \geq 0\}.$$
350    Edmund Melson Clarke, Jr.

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

\[
F_C^0(L) = F_C^1(L) = \cdots = (IR \land a = 1 \rightarrow b = 0 \land \overrightarrow{b} = 1 \rightarrow a = 0),
\]

we see that

\[
\mu(F_C) = \bigcup_{i=0}^{\infty} F_C^i(L) = F_C^0(L)
\]

\[
= \{ a = a_1 - a_2 \land b = b_1 - b_2 \land a = 1 \rightarrow b = 0
\]

\[
\land b = 1 \rightarrow a = 0 \land a_1 \geq 0 \land a_2 \geq 0 \land b_1 \geq 0 \land b_2 \geq 0 \}\).
\]

By using the resource invariant \( \mu(F_C) \) it is easy to show that the predicate \( M = (\text{pre}(SA) \land \text{pre}(SB) \land \mu(F_C)) \) 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 \( \ast \) is characterized by the following two properties:

(a) For all admissible predicates \( U \) and \( V \), \( U \subseteq U \ast V \) and \( V \subseteq U \ast V \).

(b) For any ascending chain of admissible predicates \( U_0 \subseteq U_1 \subseteq U_2 \subseteq \cdots \), the ascending chain defined by \( V_0 = U_0 \), \( V_{i+1} = V_i \ast U_{i+1} \) 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 \( \ast \) 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

\[
U = \bigwedge_{j=1}^{\infty} \gamma_j \quad \text{and} \quad V = \bigwedge_{k=1}^{\infty} \delta_k,
\]

where each conjunct is a linear inequality of the form \( a_1 x_1 + \cdots + a_m x_m + a_{m+1} \leq 0 \). 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 \( Q^m \). \( U \ast V \) 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 \( F_C(\text{false}) \) in the chain \( F_C(\text{false}) \subseteq F_C(\text{false}) \subseteq \cdots \) may not be polygonal convex
sets, let \( G_i = CV(F_{CV}^{ik}(false)) \) 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

\[
I^t = \bigcup_{j=0}^{\infty} H^t_j,
\]

where \( H_0^t = G_t \) and \( H_{j+1}^t = H_j^t \cdot G_{t+j+1} \).

**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., \( I^1 \supseteq I^2 \supseteq I^3 \cdots \).
(d) Suppose that \( \mu(F_C) = NL \), where \( L \) is a finite set of linear inequalities. Then there exists an \( r \geq 0 \) such that \( I^r = NL \) 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 \( H_j^t \) is found such that \( H_j^t = H_{j+1}^t \). If the limit \( \bigcup_{k=0}^{\infty} H_k^t \) 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 \( C \),

\[
a := 1\]
\[
\text{cobegin}\]
\[
A: \text{cycle } P(a); \ SA; \ V(a) \ \text{end}\]
\[
/\]
\[
B: \text{cycle } P(a); \ SB; \ V(a) \ \text{end}\]
\[
\text{coend}\]
discussed in Section 4. The function \( F_C \) in this case is given by

\[
F_C(J) = a^1 = 0 \land a^2 = 0 \land a^3 = 0 \land a = 0 \\
\lor J \\
\lor \text{SP}[a^1 := a^1 + 1; a := a - 1](a^1 = a^2 \land a > 0 \land J) \\
\lor \text{SP}[a^2 := a^2 + 1; a := a + 1](a^1 - 1 = a^2 \land J) \\
\lor \text{SP}[a^3 := a^3 + 1; a := a - 1](a^2 = a^3 \land a > 0 \land J) \\
\lor \text{SP}[a := a + 1](a^1 - 1 = a^2 \land J).
\]

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 \( F^0_C(\text{false}) \subseteq F^1_C(\text{false}) \subseteq F^2_C(\text{false}) \subseteq \ldots \) does not converge. By computing the sequence of approximations \( I^i \) however, we obtain

\[
I^1 = \{a^1 \geq 0 \land a^2 \geq 0\}, \\
I^2 = \{a^1 \geq 0 \land a^2 \geq 0\}, \\
I^3 = \{a^1 \geq 0 \land a^2 \geq 0 \land a + a^1 - a^2 \leq 1 \land a^2 \geq a^3 \land a \\
- a^2 + a^3 + a^1 = 1\}, \\
I^4 = \{a^1 \geq 0 \land a^2 \geq 0 \land a = 0 \land a + a^1 - a^2 \leq 1 \land a^3 \geq a^2 \land a \\
\leq a^2 \land a - a^3 - a^2 + a^1 + a^1 = 1\}, \\
I^5 = I^6 = I^7 = \ldots .
\]

Note that \( I^4 \) is a resource invariant for \( C \) and that \( I^i \) 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 \((b^2 - a^1) + (d^2 - b^2) \leq 1\) 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.,

\[
rr := 0; rw := 0; aw := 0; \\
a_1 := 0; b_1 := 0; a_2 := 0; b_2 := 0; \\
c := 0; d := 0; e := 0; \\
\text{resource (rr, rw, aw, a_1, b_1, a_2, b_2, c, d, e):} \\
\text{cobegin} \\
\quad \text{reader_1} \\
\qquad // \\
\quad \text{reader_2} \\
\qquad // \\
\quad \text{writer} \\
\text{coend}
\]

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 \( \text{true} \) do \( b_i := b_i + 1; \) \( rr := rr - 1 \) od;
end

The writer process is

writer: cycle
  \( C \): with \( R \) when \( \text{true} \) do \( c := c + 1; \) \( aw := aw + 1 \) od;
  \( D \): with \( R \) when \( rr \leq 0 \land rw \leq 0 \) do \( d := d + 1; \) \( rw := rw + 1 \) od;
  \( E \): with \( R \) when \( \text{true} \) 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

\[
I^5 = \{ aw - c + e = 0 \\
\land rw - d + e = 0 \\
\land rr - a_1 + b_1 - a_2 + b_2 = 0 \\
\land a_1 - b_1 + d - e \leq 1 \\
\land a_2 - b_2 + d - e \leq 1 \\
\land c - e \leq 1 \\
\land a_1 \geq b_1 \geq 0 \\
\land a_2 \geq b_2 \geq 0 \\
\land c \geq d \geq e \geq 0 \}.
\]

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 Lessewerde 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 IR ⊆ \( F_C(\text{IR}) \). We must show that \( F_C(\text{IR}) \subseteq \text{IR} \). By condition (a) in the definition of a resource invariant system, \((\bar{x} = \emptyset) \subseteq \text{IR}\). By condition (e) we see that for \( 1 \leq i \leq K \),

\[
(\text{pre}(S_i) \land b_i \land \text{IR})A_i(\text{post}(S_i) \land \text{IR})
\]

It follows that for \( 1 \leq i \leq K \),

\[
\text{SP}[A_i](\text{pre}(S_i) \land b_i \land \text{IR}) \subseteq \text{post}(S_i) \land \text{IR}
\]

Hence

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

So

\[
F_C(\text{IR}) = J_0 \lor \text{IR} \lor \bigvee_{i=1}^{K} \text{SP}[A_i](\text{pre}(S_i) \land b_i \land \text{IR}) \subseteq \text{IR}
\]

Thus every resource invariant IR is a fixpoint of \( F_C \).

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

\[
\mu(F_C) = J_0 \lor \mu(F_C) \lor \bigvee_{i=1}^{K} SP[A_i](\text{pre}(S_i) \land b_i \land \mu(F_C)).
\]

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

\[
SP[A_i](\text{pre}(S_i) \land b_i \land \mu(F_C)) \subseteq \mu(F_C).
\]

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

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

By monotonicity,

\[
SP[A_i](\text{pre}(S_i) \land b_i \land \mu(F_C)) \subseteq \text{post}(S_i).
\]

It follows that for \( 1 \leq i \leq K \),

\[
SP[A_i](\text{pre}(S_i) \land b_i \land \mu(F_C)) \subseteq \text{post}(S_i) \land \mu(F_C),
\]

or, equivalently, that

\[
\vdash (\text{pre}(S_i) \land b_i \land \mu(F_C)) A_i(\text{post}(S_i) \land \mu(F_C)).
\]

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

Next let \( IR = \{ \sigma \mid \sigma \rightarrow_C \sigma \} \). 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 \( \mu(F_C) = IR \).

(i) \( F_C(\mu(F_C)) = IR \). Clearly \( IR \subseteq F_C(\mu(F_C)) \). Let

\[
\sigma \in F_C(\mu(F_C)) = J_0 \lor IR \lor \bigvee_{i=1}^{K} SP[A_i](\text{pre}(S_i) \land b_i \land IR).
\]

Then either \( \sigma \in J_0 \subseteq IR \) or \( \sigma \in IR \) or there exists \( i_0 \) such that

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

Only the third case is interesting. If

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

then there is a state \( \sigma' \in \text{pre}(S_{i_0}) \land b_{i_0} \land IR \) 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 \( \sigma' \in \text{pre}(S_{i_0}) \land b_{i_0} \land \mu(F_C) \) and \( \sigma = A_{i_0}(\sigma') \), \( \sigma_0, \ldots, \sigma_r \) is also a computation of \( C \) and \( \sigma \in IR \).

(ii) \( IR \subseteq \mu(F_C) \). 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 \( \sigma_r \in FC^{r+1}(\text{false}) \). Since \( FC^{\text{false}}(\text{false}) = J_0 \), the basis case \( \sigma_0 \in FC^{\text{false}}(\text{false}) \) is true. Assume that for all computations \( \sigma_0, \sigma_1, \ldots, \sigma_{r-1} \) of \( C \) that \( \sigma_{r-1} \in FC^{r}(\text{false}) \). 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 \( \sigma_r \in \text{pre}(S_{i_0}) \land b_{i_0} \land \sigma_r = A_{i_0}(\sigma_{r-1}) \). Thus

\[
\sigma_r \in SP[A_{i_0}](\text{pre}(S_{i_0}) \land b_{i_0} \land FC^{r}(\text{false})) \subseteq FC^{r+1}(\text{false}).
\]
It follows that
\[
\text{IR} \subseteq \bigcup_{j=0}^{\infty} F^j(\text{false}) = \mu(F_C).
\]

(d) We prove that for the resource invariant systems \(RS_C\), 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.

(e) 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
\[
\mu(F_C) = \bigcup_{j=0}^{\infty} F^j(\text{false}) \subseteq \bigcup_{j=0}^{\infty} F^j(L) \subseteq \bigcup_{j=0}^{\infty} F^j(\mu(F_C)) = \mu(F_C).
\]

**Proof of Theorem 8.1**

(a) By condition 2 in the definition of a widening operator, the sequence \(H_0, H_1, H_2, \ldots\) must eventually stabilize. Thus there exists a \(k\) such that
\[
I' = \bigcup_{j=0}^{k} H_j.
\]

(b) \(\mu(F_C) = \bigcup_{j=0}^{\infty} F^j(\text{false}) \subseteq \bigcup_{k=0}^{\infty} CV[F^k(\text{false})] \subseteq \bigcup_{j=0}^{\infty} G_j \subseteq \bigcup_{j=0}^{\infty} G_{t+\delta} \subseteq \bigcup_{j=0}^{\infty} H_j \subseteq I'.
\]

(c) Let \(U\) and \(V\) be polygonal convex sets. We write \(U \subseteq V\) if for every conjunct \(\delta\) in \(V\) there is an equivalent conjunct in \(U\). We prove by induction on \(k\) that
\[
H_{k+1}^{t+1} \subseteq H_{k+1}.
\]

(1) **Basis step**
\[
H_0^{t+1} = G_{t+1} \subset H_0 \ast G_{t+1} = H_1.
\]

(2) **Induction step.** Assume that \(H_k^{t+1} \subset H_{k+1}^{t+1}\); then
\[
H_{k+1}^{t+1} = H_k^{t+1} \ast G_{t+k+2} \subset H_{k+1} \ast G_{t+k+2} = H_{k+2}.
\]

Note that \(H_k^{t+1} \subset H_{k+1}\) implies \(H_k^{t+1} \subset H_{k+1}\). Thus
\[
I^{t+1} = \bigcup_{k=0}^{\infty} H_k^{t+1} \subseteq \bigcup_{k=0}^{\infty} H_{k+1}^{t+1} = I',
\]

and the sequence \(I^0 \supseteq I^1 \supseteq I^2 \supseteq \cdots\) is a decreasing chain in \(\Sigma\).

(d) Assume that \(\mu(F_C) = \Lambda P\). 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. $G_r = (\forall L) \land (\exists M)$;
2. $G_t \subseteq AL$ for $t \geq r$; and
3. no inequality of $M$ is equivalent to any conjunct of $G_s$.

Next, consider $I' = \bigcup_{j=0}^{\infty} H_j^c$. By induction on $j$ and conditions (1) and (2) above, we see that $(\forall L) \land (\exists M) \subseteq H_j^c$ and $H_j^c \subseteq AL$ for $j \geq 0$. Since $H_{r-s}^c = H_{r-1-s}^c \cdot G_{s-r} = H_{r-s}^c \cdot G_s$ and no inequality of $M$ is equivalent to any conjunct of $G_s$, it follows that $H_r^c = AL$. Since $H_j^c \subseteq AL$ for $j \geq 0$, we see that

$$I' = \bigcup_{j=0}^{\infty} H_j^c = AL.$$

For $t \geq r$ we have $AL \subseteq I'$ and also $I' \subseteq I' \subseteq L$. Thus $I' = AL$ for $t \geq r$. □

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


Received March 1979; revised January 1980; accepted April 1980