Finite Models of Cyclic Concurrent Programs

By

David R. O'Hallaron
Paul F. Reynolds, Jr.

Computer Science Report # 85-02

March 22, 1985

Submitted to Journal of the ACM

Finite Models of Cyclic Concurrent Programs

DAVID R. O'HALLARON
PAUL F. REYNOLDS, JR.

University of Virginia, Charlottesville, Virginia

Static deadlock detection methods such as R. C. Holt's graph method and S. D. Carson's geometric method analyze a finite collection of concurrency states derived from the program text. Yet concurrent programs with cyclic processes can generate an infinite number of concurrency states. For such programs we must determine a finite set of concurrency states that is large enough to allow us to accurately predict the runtime behavior of the program. This problem is called the finite modeling problem. We solve the finite modeling problem for a class of cyclic consumable resource *PV* programs called *SI* programs. We use a weak form of the deadlock predicate to establish necessary conditions for deadlock and we use the geometric method to establish a sufficient condition. These results are then used to show that solving the finite modeling problem for *SI* programs requires time proportional to the sum of the number of processes and the number of semaphores.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; D.2.4 [**Software Engineering**]: Program Verification; D.4.5 [**Operating Systems**]: Process Management — *deadlocks*

General Terms: Languages, Theory, Verification

Additional Keywords and Phrases: Semaphore invariant method, static deadlock detection, deadlock predicate, geometric model, message passing

1. Introduction

Static deadlock detection methods such as R. C. Holt's graph method [Hol72] and S. D. Carson's geometric method [Car84] analyze a finite collection of concurrency states derived from the program text. Yet a concurrent program with cyclic processes can generate an infinite number of concurrency states. For such a program, we must select a finite set of concurrency states that is large enough to allow us to make accurate predictions about the

Authors' present address: Department of Computer Science, University of Virginia, Charlottesville, Virginia, 22901.

runtime behavior of the program. We call this problem the *finite modeling problem.* Our work solves the finite modeling problem for a class of cyclic consumable resource *PV* programs known as *SI* programs. *SI* programs are sufficiently powerful to express synchronous, asynchronous, and buffered message passing over statically defined and directly named communication channels [AnS83]. While *SI* programs are of a fairly restricted nature, they are able to model an important class of concurrent programs. First we discuss some helpful concepts, terminology, and notation.

The notion of a *concurrency state* varies with the static modeling technique. The graph model defines a concurrency state as a node in a directed graph, where each node is a directed bipartite graph modeling the resource requests and holdings at that state. The semaphore invariant method [Hab72, Hab75] defines a concurrency state as a collection of unique auxiliary variable values, while the geometric method defines a concurrency state as a point in an N-dimensional space. We can identify a concurrency state in the model of some N process program with a tuple $S(s_1, \ldots, s_N)$, where $s_i = k$ only if process i has completed the execution of k statements. We say that S is the concurrency state, whatever the definition of concurrency state, that results from executing s_i statements of each process i.

We can associate three sets with any concurrency state, S: the set D^S of processes that are blocked forever(deadlocked) at S, the set E^S of processes that are not deadlocked at S but which will eventually deadlock from S, and the set X^S comprising those processes not in the first two sets.

If $|D^S| > 0$, then S is a *deadlock state*. Let S be a deadlock state where $|D^S| > 0$. Then S is a *total* deadlock state if $|D^S| = N$, a *partial* deadlock state if $|D^S| < N$, a *stable* deadlock state if $|E^S| = 0$, and an *unstable* deadlock state if $|E^S| > 0$. Notice that total deadlock states are always stable, while partial deadlock states can be either stable or unstable. Unstable deadlock states are uninteresting in that they always lead to stable deadlock states. This notion of stable and unstable deadlock states is a way of distinguishing interesting partial deadlock states from uninteresting partial deadlock states and will help us derive necessary conditions for deadlock in SI programs in Section Three.

A deadlock state has two additional attributes. A state that violates the constraints imposed by the underlying synchronization primitives is called *infeasible*. For example, an infeasible state in a PV program is a concurrency state that violates one or more of the semaphore invariants. If there is some sequence of feasible states from the initial state that ends with a concurrency state S, then S is said to be *reachable*. Notice that all infeasible states are unreachable, while not all feasible states are reachable. Throughout this paper we will refer to a feasible and stable deadlock state as simply a deadlock state.

Let $S_1^{k_1}, \ldots, S_N^{k_N}$ be the statements associated with deadlock state S, and let $S_1^{k_1'}, \ldots, S_N^{k_N'}$ be the statements associated with deadlock state S'. We say that S and S' are *equivalent* if and only if

$$\forall_{i \in \{1 \ldots N\}} : S_i^{k_i} = S_i^{k_i'}$$

and the processes deadlocked at S are precisely the processes deadlocked at S'.

To perform static deadlock detection on a cyclic concurrent program, we must somehow represent the concurrency states that would result if the program were executed. This collection of concurrency states is called a *model*. The size of a model can be expressed in terms of the number of times that each process is expanded. A k*-cycle* model is the set of concurrency states generated by expanding the first k cycles of each process. If k is finite, then the model is called a *finite* model. Given a finite k-cycle model of a program, we must determine some ζ-cycle subset of the k-cycle model on which to perform the deadlock analysis.

We say that a k-cycle model is *accurate* if and only if there exists some $\zeta < k$ such that the ζ-cycle subset of the k-cycle model contains a reachable deadlock state if and only if the ∞-cycle model contains a reachable deadlock state. We say that a k-cycle model is *complete* if and only if it is accurate and the ζ–cycle subset contains all *unique* deadlocks, where a unique deadlock is a particular set of processes deadlocked at a particular set of statements. Thus, for every deadlock state, the ζ–cycle subset of a complete model contains an equivalent deadlock state.

We limit our discussion to static detection of deadlocks in cyclic straight–line *PV* programs, where a process is composed of a non–terminating outer loop, with no inner loops or conditionals. These programs can be divided into three classes [Hol72]: *reusable* resource programs, *consumable* resource programs, and *general* resource programs. In a reusable resource program, resources (semaphores for our purposes) are requested and released symmetrically within the same process. In a consumable resource program, resources are requested and released in different processes. General resource programs are combinations of reusable and consumable resource programs.

Regardless of the definition of concurrency state, there are some classes of cyclic concurrent programs that generate an infinite number of concurrency states (e.g. consumable resource programs). Although ∞–cycle models are reasonable in theory, methods such as the graph method and the geometric method require that the models be finite in practice. This motivates the finite modeling problem: how to efficiently determine the size of accurate and complete finite models of a cyclic general resource concurrent program.

Carson [Car84] has solved the finite modeling problem for total deadlocks in reusable resource programs by showing that 2–cycle models are complete. He has also shown that finding a solution to the finite modeling problem for total deadlocks in a particular general resource program is no more difficult than solving the deadlock predicate for all combinations of statements that could potentially deadlock, a problem requiring time exponential in the number of processes.

Our work solves the finite modeling problem for a class of consumable resource programs called *SI* programs. We show how the size of accurate and complete models for total and partial deadlocks in SI programs can be determined without resorting to solving the deadlock predicate. In particular, we use the semaphore invariant method and the geometric method to show that 2–cycle models are accurate and $(NM-2M+2)$–cycle models are complete, where N is the number of processes and M is the largest initial semaphore value plus unity.

In Section Two we formally define SI programs. In Sections Three and Four we derive necessary and sufficient conditions for deadlock in ∞–cycle models of SI programs. In Section Five we use these results to derive the size of accurate and complete models.

2. SI Programs

SI programs are a class of cyclic consumable–resource PV programs. For our purposes, a PV program is a collection of N cyclic processes

$$\textbf{var } L; \textbf{ cobegin } P_1 \,//\, P_2 \,//\, \cdots \,//\, P_N \textbf{ end}$$

where L is a list of S semaphores and where process P_i has the form

$$P_i : \textbf{cycle } S_i^1; S_i^2; \cdots ; S_i^{k_i} \textbf{ endcycle.}$$

Each S_i^k is a statement of the form

$$\textbf{when } \sigma_1 > 0 \wedge \cdots \wedge \sigma_k > 0 \textbf{ do } \sigma_1 \leftarrow \sigma_1 - 1; \cdots ; \sigma_k \leftarrow \sigma_k - 1$$

denoted in the program text by

$$P(\sigma_1, \ldots, \sigma_k)$$

or of the form

$$\textbf{when true do } \sigma_1 \leftarrow \sigma_1 + 1; \cdots ; \sigma_k \leftarrow \sigma_k + 1$$

denoted in the program text by

$$V(\sigma_1, \ldots, \sigma_k).$$

Let σ_{ij}^P be the number of times that semaphore σ_i is acquired during a single cycle of process j and let σ_{ij}^V be the number of times that semaphore σ_i is released during a single cycle of process j. Then a PV program is an SI program if and only if:

$P1 : \forall_{i \in \{1..S\}} : \forall_{j \in \{1..N\}} : \sigma_{ij}^P + \sigma_{ij}^V \leqslant 1,$

$P2 : \forall_{i \in \{1..S\}} : \sum_{j=1}^{N} \sigma_{ij}^P = 1,$ and

$P3 : \forall_{i \in \{1..S\}} : \sum_{j=1}^{N} \sigma_{ij}^V = 1.$

In other words, a semaphore appears at most once in any given process and in exactly one P statement and one V statement in the entire program.

3. Necessary Conditions for Deadlock in SI Programs

In this section we use the semaphore invariant method [Hab72, Hab75] to derive a number of necessary conditions for deadlock in SI programs. The semaphore invariant method uses auxiliary variables to construct the deadlock predicate [OwG76], the satisfiability of which is a necessary condition for deadlock. The deadlock predicate consists of auxiliary variables, semaphore invariants, preconditions of statements, and blocking predicates. We describe each of these in more detail.

Let σ be a semaphore initialized to σ_0, let σ_i^P and σ_i^V be ***semaphore auxiliary variables***, initially zero, such that σ_i^P is incremented when process i acquires σ, and σ_i^V is incremented when process i releases σ. Then the semaphore invariant for σ is

$$I_\sigma \equiv \sigma = \sigma_0 - \sum_{i=1}^{N} \sigma_i^P + \sum_{i=1}^{N} \sigma_i^V.$$

The semaphore invariant for SI programs is simpler. From $P2$ and $P3$ we see that σ is acquired by exactly one process and released by exactly one process. From $P1$ we see that σ is acquired and released by different processes. Let σ be acquired by process i and released by process j. Then the semaphore invariant for SI programs reduces to

$$I_\sigma \equiv \sigma = \sigma_0 + \sigma_j^V - \sigma_i^P.$$

Associated with each process i is a set of auxiliary variables V_i. Included in V_i are a *repetition auxiliary variable* [Car84], R_i, initially zero, that is incremented when process i

completes a cycle, and a set of semaphore auxiliary variables, where $\sigma_i^P \in V_i$ if process i requests σ, and where $\sigma_i^V \in V_i$ if process i releases σ. The *precondition* of the *jth* statement in the *ith* process, $pre(S_i^j)$, describes the relationships that exist among the auxiliary variables in V_i just before S_i^j is executed. In the most general case [Car84] these relationships are of the form

$$c_1 v_1 + c_2 = v_2$$

where v_i is a semaphore or repetition auxiliary variable, c_1 is a positive constant, and c_2 is a positive or negative constant.

For SI programs these relationships are simpler. From $P1$ and from the definition of repetition auxiliary variables, we see that v_1 is incremented exactly once per cycle, thus $c_1 = 1$. Auxiliary variable v_1 is either incremented before v_2 or after v_2, thus c_2 is either unity or zero. Thus, relationships among auxiliary variables in the preconditions of SI programs are limited to the form

$$v_1 - c_2 = v_2$$

where c_2 is a constant zero or unity.

The semaphore invariant method defines a concurrency state as a unique collection of non–negative semaphore auxiliary variable values that are consistent with the constraints imposed by the preconditions. If, in addition, these values are consistent with the constraints imposed by the semaphore invariants, then the state is feasible.

Associated with each statement S_i^j in an SI program is a *blocking predicate*, b_i^j, that describes the conditions under which S_i^j is blocked. If S_i^j is a V operation, then S_i^j never blocks and b_i^j is simply the predicate (*false*). If S_i^j is a P operation, then S_i^j is blocked if one or more of its semaphores is zero. Let B_i^j be the set of semaphores requested by S_i^j. Then b_i^j is the predicate

$$\exists_{\sigma \in B_i^j} : \sigma = 0.$$

Let S be the number of semaphores and let k_i be the number of statements in process i. Then the deadlock predicate is:

$$D = \bigwedge_{i=1}^{N} \left(\bigvee_{j=1}^{k_i} pre(S_i^j) \wedge b_i^j \right) \wedge \left(\bigwedge_{\zeta=1}^{S} I_\zeta \right)$$

Notice that this definition differs from the traditional definition of the deadlock predicate in two ways: First, we have introduced repetition auxiliary variables to the preconditions[Car84]. Second, we have extended the blocking predicate to accommodate our definition of *PV* programs, where a *P* or *V* operation may operate on more than one semaphore.

The semaphore invariant method is an attractive static deadlock detection method because the deadlock predicate is simply and compactly generated. It has some significant disadvantages, though. First, determining the satisfiability of the deadlock predicate is an exponential problem. Second, because of the possibility of feasible and unreachable deadlock states, satisfiability of the deadlock predicate is not a sufficient condition for deadlock. E. M. Clarke has developed an iterative procedure to strengthen the deadlock predicate so that satisfiability is a sufficient condition as well [Cla80]. However a final problem remains: the deadlock predicate does not allow for the possibility of partial deadlocks.

We can allow for partial deadlocks by defining a weaker form of the deadlock predicate called the ***weak deadlock predicate.*** Let P be the powerset of $\{1, \ldots, N\}$ less the empty set. Let P^i be the ith set in P, and let X^i be the complement of P^i. Then the weak deadlock predicate is

$$WD = \bigvee_{i=1}^{2^N-1} \left(\bigwedge_{j \in P^i} \left(\bigvee_{k=1}^{k_j} pre(S_j^k) \wedge b_j^k \right) \wedge \left(\bigwedge_{j \in X^i} \left(\bigvee_{k=1}^{k_j} pre(S_j^k) \right) \right) \right) \wedge \left(\bigwedge_{j=1}^{S} I_j \right)$$

This predicate is true for all concurrency states where processes are blocked, so satisfiability of the weak deadlock predicate is a necessary condition for total or partial deadlock. Satisfiability is not a sufficient condition because of the possibility of feasible and unreachable deadlock states, and because the predicate is satisfied at those concurrency states where processes are blocked but not deadlocked. Nonetheless, the weak deadlock predicate can help us reason about necessary conditions for deadlock in *PV* and *SI* programs. First we establish necessary conditions for stable deadlocks in *PV* programs and *SI* programs.

Lemma 3–1: If S is a stable deadlock state in a *PV* program and process $i \in D^S$ is blocked on semaphore σ, then for each process j that releases σ, $j \in D^S$.

Proof: Since process i is blocked forever waiting for σ to be released, process j is either deadlocked at S, or will eventually deadlock before releasing σ. If the former is true, then $j \in D^S$. If the latter is true, then $j \in E^S$, which contradicts the hypothesis of a stable deadlock state. Thus, $j \in D^S$.

□

Lemma 3–2: If S is a stable deadlock state in an *SI* program and processes $i \in D^S$ and $j \in X^S$ use a semaphore σ, then process i requests σ and process j releases σ.

Proof: Suppose the contrary, that process j requests σ and process i releases σ. From $P2$ in the definition of *SI* programs, process i is the only process that releases σ. Since process i is deadlocked, process j must eventually deadlock on σ, which implies that $j \in E^S$, which contradicts the hypothesis that S is a stable deadlock state.

□

We say that a deadlock state is a 1–cycle deadlock if at least one process is deadlocked on its first cycle. We can use the weak deadlock predicate and Lemmas 3–1 and 3–2 to show that the existence of a 1–cycle deadlock state in the ∞–cycle model is a necessary condition for deadlock in *SI* programs.

Lemma 3–3: The ∞–cycle model of an SI program contains a deadlock state only if it contains an equivalent 1–cycle deadlock state.

Proof: Suppose that S is a stable deadlock state in the ∞–cycle model of an SI program. Then there are N disjuncts, one from each process, that satisfy the weak deadlock predicate. Let A and B be disjoint sets of auxiliary variables where A is the set of auxiliary variables used in the preconditions of the deadlocked statements, and B is the set of auxiliary variables used in the preconditions of the statements that are not deadlocked. The smallest auxiliary variable in A is a repetition auxiliary variable, say R_i. Since auxiliary variables in SI programs are related by constants, subtracting R_i from each auxiliary variable in A gives a new concurrency state, S'.

To show that S' is feasible, we observe that the semaphore invariants for those semaphores used exclusively by processes in D^S and X^S are still satisfied. Further, the semaphore invariant for a semaphore σ shared by a process $i \in D^S$ and a process $j \in X^S$ is still satisfied because, from Lemma 3–2, we decreased σ_i^P while leaving σ_j^V unchanged.

To show that S' and S are equivalent, we observe that the relationships among the auxiliary variables in A are the same for S and S', as are the relationships between auxiliary variables in B. Thus S' involves the same N statements as S, and from Lemma 3–1, processes deadlocked at S are deadlocked at S', and from Lemma 3–2, processes not deadlocked at S are not deadlocked at S'.

Now, since $R_i' = 0$, we see that S' is a feasible 1–cycle deadlock state equivalent to S.

□

Lemma 3–3 says that the 1–cycle deadlock states are the unique deadlock states. We use this result in Section Five to derive the size of accurate and complete models for SI programs. We will also use this result to derive a sufficient condition for deadlock–freedom in SI programs.

We know from Lemma 3–3 that an SI program deadlocks only if there exists a 1–cycle deadlock state. This implies deadlock only if at least one process blocks during its first cycle. The contrapositive says that if no process blocks during its first cycle, then the program is deadlock–free. Now consider an *SI* program where all initial semaphore values are non–zero. It is clear from the definition of *SI* programs that no process will block during its first cycle. Thus the program is deadlock–free and we can say that the absence of any initially zero–valued semaphores in an *SI* program is a sufficient condition for deadlock–freedom.

We can also use the weak deadlock predicate to derive necessary conditions for the relationships between the number of cycles completed by each deadlocked process at a stable deadlock state. Suppose that S is a stable deadlock state in the ∞–cycle model of an *SI* program, where

$$D^S = \{1, \ldots, \zeta\}$$

is the set of processes deadlocked at S, and where

$$X^S = \{\zeta+1, \ldots, N\}$$

is the set of processes not deadlocked at S. Suppose also that the processes in D^S are deadlocked at statments

$$S_1^{k_1}, \ldots, S_\zeta^{k_\zeta}.$$

Associated with each $S_i^{k_i}$ for each $i \in D^S$ is a disjunct from the weak deadlock predicate of the form

$$pre(S_i^{k_i}) \wedge (\sigma = 0).$$

Let σ be requested by process i and released by process j. Since S is stable, we know from Lemma 3–1 that $j \in D^S$, and from the definition of *SI* programs that $i \neq j$. If we substitute the repetition auxiliary variables R_i and R_j from $pre(S_i^{k_i})$ and $pre(S_j^{k_j})$ into the semaphore invariant for σ, we get an equation of the form

$$\begin{aligned}\sigma = 0 &= \sigma_0 + \sigma_j^V - \sigma_i^P \\ &= \sigma_0 + R_j + c_i' - R_i \ , i \neq j\end{aligned}$$

where c_i' is a constant zero if σ is released after $S_j^{k_j}$ in the text of process j, and unity if σ is released before $S_j^{k_j}$. Rearranging terms gives

$$R_i = R_j + c_i \; ; i \neq j$$

where $c_i = \sigma_0 + c_{\cdot}$. This equation, called a *cyclic dependency equation*, describes the cyclic relationship between deadlock process i and deadlocked process j at deadlock state S. If we perform a similar substitution for all of the ζ processes in D^S, we get a system of ζ cyclic dependency equations in ζ unknowns of the form:

$$R_1 = R_i + c_1 \, , i \neq 1 \in \{1, \ldots, \zeta\}$$
$$R_2 = R_j + c_2 \, , j \neq 2 \in \{1, \ldots, \zeta\}$$
$$\ldots$$
$$R_\zeta = R_k + c_\zeta \, , k \neq \zeta \in \{1, \ldots, \zeta\}$$

where $0 \leqslant c_i \leqslant max(\sigma_0) + 1$. A consistent system of cyclic dependency equations associated with a stable deadlock state in the ∞–cycle model of an SI program has some simple properties that allow us to derive an additional necessary condition for deadlock: at a stable deadlock state at least two deadlocked processes are executing the same cycle such that no other deadlocked processes are executing an earlier cycle.

Lemma 3–4: For a consistent system of cyclic dependency equations associated with a stable deadlock state S where $D^S = 1, \ldots, \zeta$, then

$$\exists_{i \neq j \in \{1 \ldots \zeta\}} : R_i = R_j \tag{3–1}$$

where

$$\forall_{k \in \{1 \ldots \zeta\}} : R_i \leqslant R_k . \tag{3–2}$$

Proof: To show that Equation 3–1 is true, suppose the contrary, such that

$$\forall_{i \in \{1 \ldots \zeta\}} : c_i > 0.$$

This assumption leads, after some renumbering of subscripts, to an equation of the form

$$R_1 = R_2 + c_1 = R_3 + c_2 = \cdots = R_\xi + c_{\xi-1} = R_i + c_\xi \, , i \in \{1 \ldots xi\}$$

which is a contradiction. To show that Equation 3–2 is true, partition the system of equations into disjoint sets A and B with

$$A = \{R_i \mid R_i = R_j\}$$

$$B = \{R_i \mid R_i = R_j + c_i\}\ ,\ c_i > 0$$

and let $R_m = min(A)$. If $B = \emptyset$ we are done. If $B \neq \emptyset$, then consider some $R_1 \in B$ with $R_1 = R_2 + c_1$. If $R_2 \in A$ then

$$R_1 > R_2 \geqslant R_m.$$

On the other hand, if $R_2 \in B$ then $R_2 = R_3 + c_2$ and

$$R_1 > R_2 > R_3.$$

Similarly, if $R_3 \in B$ then $R_3 = R_4 + c_3$ and

$$R_1 > R_2 > R_3 > R_4.$$

Continuing in this fashion, we must encounter some $R_i \in A$ such that

$$R_1 > R_2 > R_3 > R_4 > \cdots > R_i \geqslant R_m.$$

Thus $R_m < min(B)$, which establishes the truth of Equation 3–2.

□

A simple example will illustrate the concepts we have discussed in this section. Consider the following *SI* program:

```
a,b,c,d,e = 0: semaphore
cobegin
        P1: cycle
                P(a); V(b)              -- deadlocks on the first cycle
        endcycle
        //
        P2: cycle
                V(c); P(b); V(a)        -- deadlocks on the first cycle
        endcycle
        //
        P3: cycle
                P(d); P(c)              -- deadlocks on the second cycle
        endcycle
        //
        P4: cycle
                V(d); V(e)              -- never deadlocks
        endcycle
        //
        P5: cycle
                P(e)                    -- never deadlocks
        endcycle
    end
```

Consider the infinite set of stable deadlock states where processes 1, 2, and 3 are deadlocked on semaphores a, b, and c, respectively, and where process 4 is ready to release d and where process 5 is ready to request e. The weak deadlock predicate for such states is

$$
\begin{aligned}
&(a_1^P = b_1^V = R_1) \wedge (a = 0) \wedge \\
&(c_2^V - 1 = b_2^P = a_s^V = R_2) \wedge (b = 0) \wedge \\
&(d_3^P - 1 = c_3^P = R_3) \wedge (c = 0) \wedge \\
&(d_4^V = e_4^V = R_4) \wedge \\
&(e_5^V = R_5) \wedge \\
&(a = a_2^V - a_1^P) \wedge \\
&(b = b_1^V - b_2^P) \wedge \\
&(c = c_2^V - c_3^P) \wedge \\
&(d = d_4^V - d_3^P) \wedge \\
&(e = e_4^V - e_5^P)
\end{aligned}
$$

Suppose that S is one of these stable deadlock states such that

$R_1 = 2$	$a_2^V = 2$	$a_1^P = 2$
$R_2 = 2$	$b_1^V = 2$	$b_2^P = 2$
$R_3 = 3$	$c_2^V = 3$	$c_3^P = 3$
$R_4 = 4$	$d_4^V = 4$	$d_3^P = 4$
$R_5 = 0$	$e_4^V = 4$	$e_5^P = 0$

We see that S is a deadlock state where processes 1 and 2 are deadlocked during the third cycle, process 3 is deadlocked on its fourth cycle, process 4 is starting its fifth cycle, and process 5 has yet to execute. While S is certainly a feasible deadlock state, inspection of the text shows that it is not reachable. In the next section we will show that the only reachable deadlock states in SI programs are the 1–cycle deadlock states. By subtracting R_1 from the auxiliary variables in processes 1, 2, and 3, we get an associated 1–cycle deadlock state, S':

$$\begin{array}{lll} R_1 = 0 & a_2^V = 0 & a_1^P = 0 \\ R_2 = 0 & b_1^V = 0 & b_2^P = 0 \\ R_3 = 1 & c_2^V = 1 & c_3^P = 1 \\ R_4 = 4 & d_4^V = 4 & d_3^P = 2 \\ R_5 = 0 & e_4^V = 4 & e_5^P = 0 \end{array}$$

We showed in Lemma 3–3 that 1–cycle deadlock states such as S' are feasible deadlocks involving the same set of statements. Inspection of the auxiliary variable values for S' shows that this is true.

In Lemma 4–4 we established some properties of the cyclic dependency equations associated with a stable deadlock state. The cyclic dependency equations for S and S' can be easily generated by substituting the disjuncts for processes 1, 2, and 3 into the sema–phore invariants for a, b, and c to get

$$R_1 = R_2$$

$$R_2 = R_1$$

$$R_3 = R_2 + 1$$

These equations tell us that at deadlock states S and S', process 1 and process 2 are deadlocked on the same cycle, and process 3 is deadlocked on the next cycle.

Let us review what we have accomplished in this section. We have introduced a weak form of the deadlock predicate whose satisfiability is a necessary condition for total and partial deadlock. We then used this predicate to show that the existence of a 1–cycle deadlock state in the ∞-cycle model of an *SI* program is a necessary condition for deadlock. We then showed, for *SI* programs, how a system of cyclic dependency equations can be derived from a set of disjuncts that satisfy the weak deadlock predicate. Finally, we used properties of consistent systems of cyclic dependency equations to derive necessary conditions on the number of cycles executed by the deadlocked processes of an *SI* program. These results will help us derive accurate and complete models of *SI* programs in Section Five.

4. A Sufficient Condition for Deadlock in SI Programs

In this section we derive a sufficient condition for deadlock in *SI* programs. Our proof uses results from Carson's recent work in geometric models of concurrent programs [Car84]. We present a brief discussion of the geometric model. The reader is encouraged to consult Carson's paper for a formal and complete treatment.

The geometric model is a formalization of the familiar notion of progress graphs [CoD73], where a concurrency state, or ordered set of process times, is modeled as a point in a continuous N dimensional space. The *ith* time axis is labeled with synchronization events(P and V operations) in the order in which they executed by process i. For convenience synchronization events are assigned integer times starting from unity. Thus, for two processes, the point $P(0,0)$ represents the initial concurrency state where neither process has completed a synchronization event and the point $P(3,1)$ represents the state where processes one and two have completed their third and first synchronization events respectively.

Sets of concurrency states with like synchronization properties are modeled as N dimensional rectangular regions bounded by $2N$ $N-1$ dimensional hyperplanes. The size and location of an N dimensional region can be represented with two points: the *vertex*, or point closest to the origin, and the *extent*, or point furthest from the origin. Regions of interest for static deadlock detection include *forbidden* regions, *nearness* regions, *deadlock* regions, and *unsafe* regions.

Forbidden regions model concurrency states that violate the constraints imposed by the the underlying synchronization primitives. For *PV* programs, points within forbidden regions model concurrency states that violate the semaphore invariants.

Definition 4–1: forbidden region

A forbidden region is a triple (σ, V, E), where σ is a semaphore associated with the region, V is the vertex of the region, and E is the extent of the region.

□

Carson has developed a technique for generating the vertices of forbidden regions from the source text of *PV* programs; the technique is based on the notion of *deficits*. Let S_i^j be the *jth* statement executed by the *ith* process of a *PV* program.

Definition 4–2: deficit

The deficit for semaphore σ at S_i^j, denoted $d(i,j,\sigma)$, is defined inductively to be

$$d(i,0,\sigma) = 0$$

$$d(i,j,\sigma) = d(i,j-1,\sigma) + \begin{cases} 1 & \text{if } S_i^j \textit{ requests } \sigma \\ -1 & \text{if } S_i^j \textit{ releases } \sigma \\ 0 & \textit{otherwise} \end{cases}$$

□

Definition 4–3: deficit at a point

The deficit at a point P for semaphore σ with initial value σ_0, denoted $D(P,\sigma)$, is defined to be

$$D(P,\sigma) = -\sigma_0 + \sum_{i=1}^{N} d(i,p_i,\sigma).$$

□

The deficit at a point P for a semaphore σ is simply minus the value of σ at P. A point P is feasible if and only if $D(P,sigma) \leqslant 0$ for all σ.

Let $use[\sigma]$ be the set of processes that either request or release semaphore σ. Let $stmt(p_i)$ be the statement associated with the *ith* coordinate at point $P(p_1,\ldots,p_N)$. Possible values for $stmt(p_i)$ are $P(\sigma)$, $V(\sigma)$, and I (initial statement at the origin).

Definition 4–4: vertex of a forbidden region

The vertex of a forbidden region for semaphore σ is a point $V(v_1,\ldots,v_N)$ such that

$$\exists_{i \in use[\sigma]} : stmt(v_i) = P(\sigma), \tag{4–1}$$

$$\forall_{i \in use[\sigma]} : stmt(v_i) \in \left\{ P(\sigma),V(\sigma),I \right\}, \tag{4–2}$$

$$\forall_{i \notin use[\sigma]} : v_i = 0, \text{ and} \tag{4–3}$$

$$D(V,\sigma) = 1. \tag{4–4}$$

□

Nearness regions model concurrency states where the progress of a single process is blocked by a hyperplane of a forbidden region.

Definition 4–5: nearness region

A nearness region is a triple (d, V, E), where d is the direction in which the system is blocked within the region, V is the vertex of the region, and E is the extent of the region.

□

Up to N nearness regions can be generated from a forbidden region.

Definition 4–6: vertex and extent of a nearness region

The *jth* nearness region, $R^j(d^j, V^j, E^j)$, formed from forbidden region $R'(\sigma, V', E')$ with vertex $V'(v_1', \ldots, v_N')$ and extent $E'(e_1', \ldots, e_N')$, is a region with

$$d^j = j$$

vertex $V^j(v_1^j, \ldots, v_N^j)$ such that

$$v_j^j = v_j' - 1$$

$$\forall_{k \neq j \in \{1 \ldots N\}} : v_k^j = v_k'$$

and extent $E^j(e_1^j, \ldots, e_N^j)$ such that

$$e_j^j = v_j'$$

$$\forall_{k \neq j \in \{1 \ldots N\}} : e_k^j = e_k'.$$

□

Nearness regions of degree ζ are the intersection of ζ distinct nearness regions. Points within nearness regions of degree ζ model concurrency states where ζ processes are blocked by the hyperplanes of ζ distinct forbidden regions. Nearness regions generated directly from forbidden regions are called nearness regions of degree 1. Associated with a nearness region of degree ζ is a *blocked set* B and an *invariant set* I such that $i \in B$ if and only if process i is blocked within the region, and $i \in I$ if and only if extent coordinate $e_i = \infty$. Since geometric models in practice are finite, infinity is defined to be some finite value such that the point $P(\infty_1, \ldots, \infty_N)$ lies outside the model.

A deadlock region is a nearness region of degree ζ where points within the region model concurrency states in which one or more processes are blocked forever.

Definition 4–7: deadlock region

A deadlock region is a tuple (I,B,V,E), where I is the invariant set, B is the blocked set, V is the vertex, E is the extent, such that

$$B \bigcup I = \{1, \ldots, N\}.$$

□

The definition of a deadlock region given above fails to capture certain unstable deadlock states. In practice this is not a problem because unstable deadlock states always lead to stable deadlock states.

The final region of interest is the unsafe region. Unsafe regions model concurrency states that ultimately lead to deadlock.

Definition 4–8: unsafe region

An unsafe region is a triple (D,V,E), where D is a deadlock region, V is the vertex, and E is the extent.

□

Carson has shown that the vertex of an unsafe region formed from a deadlock region D is easily computed.

Definition 4–9: vertex of an unsafe region

Let the nearness regions of degree 1 whose intersection is the deadlock region D be $R^j(d^j, V^j, E^j)$. Let $max(\varnothing) = 0$. Then the vertex of the unsafe region formed from deadlock region D is a point $V^U(v_1^U, \ldots, v_N^U)$ such that

$$\forall_{i \in \{1 \ldots N\}} : v_i^U = \max(v_i^k, k \in B^D \wedge k \neq i).$$

□

In Figure 4–1(a) we give an example of a reusable resource program that uses three semaphores. We assume a=1, b=1 and c=2 initially. Figure 4–2(a) is a 1–cycle geometric model of this program. Region $R1$, the vertically oriented solid rectangle, is a forbidden region for semaphore b. Region $R2$, the horizontally oriented solid rectangle, is a forbid–

```
a,b = 1; c = 2: semaphore
cobegin
        A: cycle P(a); P(c); P(b); V(c); V(b); V(a) endcycle
        //
        B: cycle P(b); P(c); P(a); V(c); V(a); V(b) endcycle
end.
```

(a)

```
a,b,c,d,e,f = 0: semaphore
cobegin
        A: cycle V(a); P(b); V(c); P(d); V(e); P(f) endcycle
        //
        B: cycle V(f); P(c); V(b); P(e); V(d); P(a) endcycle
end.
```

(b)

Fig. 4–1: *(a) Reusable Resource Program (b) SI Program*

den region for semaphore a. Regions $N1$ and $N4$ are nearness regions associated with forbidden region $R1$; similarly $N2$ and $N3$ for $R2$. Nearness regions $N1$ and $N3$ form deadlock region, D. The unsafe region, U, can be computed given the other regions. If a vector representing the progress of processes A and B were to contact a part of U then A and B would inevitably deadlock. Deadlock is inevitable only if processes A and B seize semaphores concurrently. This is reflected in the fact that vertex of the unsafe region is not at the origin. In Figure 4–1(b) we give an example of an SI program that uses six semaphores; Figure 4–2(b) is its 1–cycle geometric model. There are four forbidden regions of interest here, $R1 - R4$. The two deadlock regions in this example, $D1$ and $D2$, are formed from $R1$, $R2$ and $R3$, $R4$ respectively. Region $D1$ is reachable; region $D2$ is unreachable. We shall see that if a region such as $D2$ exists, then there must be a some

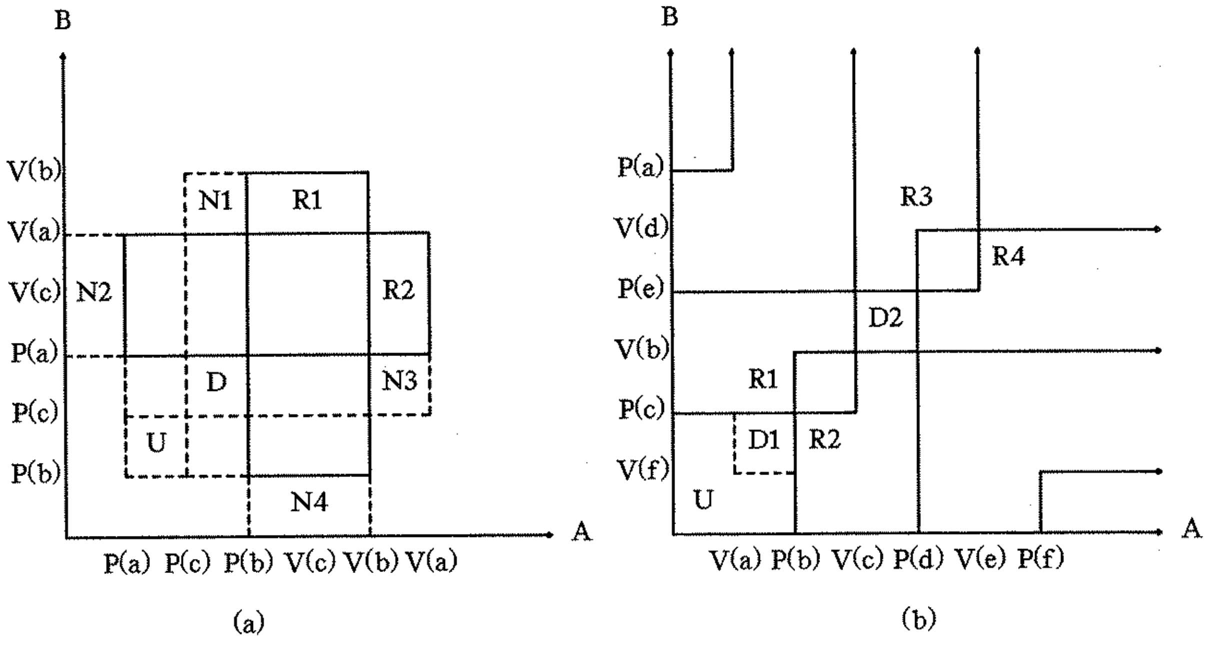

Fig. 4–2: *(a) Reusable Resource Program (b) SI Program*

reachable deadlock region, $D1$. Furthermore, since the unsafe regions for both $D1$ and $D2$ have their vertices at the origin, this program will inevitably deadlock. We shall see that this is a property shared by all SI programs.

Regions generated from SI programs have properties that allow us to derive a sufficient condition for deadlock. One such property is that a forbidden region formed by a semaphore request during the first cycle of some process has a vertex in the 1–cycle subset of the ∞–cycle model with exactly one non–zero coordinate.

Lemma 4–1: Let S_i^j be a request of semaphore σ. If R is a forbidden region formed by the first instance of statement S_i^j and $V(v_1,\ldots,v_N)$ is a point such that

$$0 < v_i = j < \infty \tag{4–5}$$

and

$$\forall_{k \neq i \in \{1 \ldots N\}} : v_k = 0. \tag{4–6}$$

then V is the vertex of R.

Proof: We show that point V satisfies the four conditions of a vertex point of a forbidden region given in Definition 4–4. Condition 4–1 is true by the assumption that S_i^j is a P operation. Condition 4–2 is true by the assumption the S_i^j is a P operation and by Equation 4–6. Condition 4–3 is true by Equation 4–6. From the definition of SI programs and Equation 4–6 we have

$$d(i,v_i,\sigma) = 1$$

and

$$\forall_{k \neq i \in \{1 \ldots N\}} : d(k,v_k,\sigma) = 0,$$

which imply that

$$D(V,\sigma) = \sum_{k=1}^{N} d(k,v_k,\sigma) = 1,$$

thereby satisfying Condition 4–4.

□

Another property of *SI* programs is that there is exactly one nearness region generated from the forbidden regions described in Lemma 4–1 and the vertex of this nearness region lies in the 1–cycle subset and contains exactly one non–zero coordinate.

Lemma 4–2: A forbidden region whose vertex lies in the 1–cycle subset of the ∞–cycle model of an *SI* program generates exactly one nearness region $R^j(d^j,V^j,E^j)$. The vertex of this nearness region is a point

$$V(v_1^j, \ldots, v_N^j)$$

where

$$0 \leqslant v_j^j < \infty$$

and

$$\forall_{k \neq j \in \{1 \ldots N\}} : v_k^j = 0.$$

Proof: From Lemma 4–1, there is exactly one non–zero coordinate in the vertex of the forbidden region, hence only one nearness region can be formed from a given forbidden region. The location of this nearness region follows directly from Definition 4–6.

□

The next lemma will help us derive the vertex of the unsafe region associated with a deadlock region formed from nearness regions whose vertices lie in the 1–cycle subset.

Lemma 4–3: Let D be a deadlock region formed from the intersection of ζ distinct nearness regions $R^j(d^j,V^j,E^j)$, $j = 1 \ldots \zeta$, whose vertices lie in the 1–cycle subset. Then

$$\forall_{k \neq j \in \{1 \ldots \zeta\}} : v_j^k = 0.$$

Proof: The proof follows directly from Lemma 4–2 and from the fact that the directions associated with ζ intersecting nearness regions are distinct.

□

Given these properties of the forbidden regions and nearness regions generated in the 1–cycle subsets of the ∞–cycle models of *SI* programs, we can show that the existence of a (perhaps unreachable) deadlock state in the ∞–cycle model is a sufficient condition for deadlock in *SI* programs.

Lemma 4–4: An *SI* program contains a deadlock state only if the program deadlocks.

Proof: Consider a deadlock state in the ∞–cycle model. From Lemma 3–3, there is an associated 1–cycle deadlock state, S', where processes $1, \ldots, \zeta$ are deadlocked in their first cycle. Then there is a deadlock state, S, in the 1–cycle subset where processes $1, \ldots, \zeta$ are deadlocked. Let S be contained in a deadlock region D formed from the intersection of ζ distinct nearness regions, $R^j(d^j, V^j, E^j)$, $j = 1 \ldots \zeta$, where v_i^j is the *ith* vertex coordinate in V^j. The *ith* vertex coordinate in the vertex of the unsafe region U formed from D is defined to be

$$\forall_{i \in \{1 \ldots N\}} : v_i^U = \max(v_i^k,\ k \in B^D \wedge k \neq i).$$

From Lemma 4–3 it follows that

$$\max(v_i^k,\ k \in \{1 \ldots \zeta\} \wedge k \neq i) = 0.$$

Hence, the vertex of U is at point $V^U(0, \ldots, 0)$. Since the origin is always reachable, deadlock is inevitable.

□

Lemma 4–4 says that the presence of a deadlock state S in the ∞–cycle model implies that deadlock is inevitable. It is a powerful result because it makes no assumptions about the reachability or feasibility of S. Lemma 4–4 also implies that the set X^S consists of processes that will never deadlock from S or from any concurrency state. Thus X^S is invariant over all concurrency states. These results will be useful when we derive accurate and complete models for *SI* programs in the next section.

5. Accurate and Complete Models of SI Programs

In this section we use the results from the previous two sections to derive the size of accurate and complete models of SI programs. Our work so far has been concerned with the properties of ∞–cycle models of SI programs. In practice though, static deadlock detection methods such as the geometric method require finite models to analyze. The next result establishes a relationship between deadlock states in finite and infinite models of PV programs.

Lemma 5–1: S is a deadlock state in the k–cycle subset of the $(k+1)$–cycle model of a PV program if and only if S is a deadlock state in the k–cycle subset of the ∞–cycle model.

Proof:

If: Processes deadlocked at S are blocked at every concurrency state in the ∞–cycle model reachable from S. The $(k+1)$–cycle model is a subset of the ∞–cycle model. This implies that processes deadlocked at S are blocked at every concurrency state in the $(k+1)$–cycle model reachable from S.

Only If: Carson has shown that a forbidden region ends in the current cycle of each process, the next cycle of each process, or never at all [Car84]. This implies that processes blocked in the k–cycle subset are released in the k–cycle subset, the $(k+1)$–cycle subset, or never at all. Thus, processes deadlocked in the k–cycle subset of the $(k+1)$–cycle model are deadlocked in the k–cycle subset of the ∞–cycle model.

□

Lemma 5–1 says that if we are somehow able to determine that the k–cycle subset of the ∞–cycle model of a PV program is accurate or complete, then the k–cycle subset of the $(k+1)$–cycle model is also accurate or complete. In other words, we model $k+1$ cycles of each process, but analyze only the concurrency states generated by the first k cycles.

Lemma 3–3 tells us that the unique deadlocks in SI programs are the 1–cycle deadlock states. Furthermore, Lemma 4–4 implies that 1–cycle deadlock states are the only reachable deadlock states. From this it is easy to show that 2–cycle models of *SI* programs are accurate.

Theorem 5–1: Two–cycle models of *SI* programs are accurate.

Proof: The 1–cycle deadlock states are the only reachable deadlock states. Thus we must show that the 1–cycle subset of the 2–cycle model contains a reachable 1–cycle deadlock state if and only if the ∞–cycle model contains a reachable 1–cycle deadlock state.

If: Let a reachable stable 1–cycle deadlock state in ∞–cycle model be

$$S'(s_1', \ldots, s_\zeta', s_{\zeta+1}', \ldots, s_N')$$

where processes $\{1, \ldots, \xi\}$, $\xi \leqslant \zeta \leqslant N$, are deadlocked on the first cycle, and where processes $\{\xi+1, \ldots, \zeta\}$ are deadlocked at later cycles. From Lemma 4–4, we know that processes $\{\zeta+1, \ldots, N\}$ will never deadlock. Then we have a deadlock state S in the 1–cycle subset at

$$S(s_1', \ldots, s_\xi', t_{\xi+1}, \ldots, t_\zeta, t_{\zeta+1}, \ldots, t_N)$$

where t_i is the last statement in the first cycle of process i. Clearly, S is reachable, which from Lemma 5–1 implies that S is a reachable deadlock state in the 1–cycle subset of the 2–cycle model.

Only If: See Lemma 5–1.

□

Theorem 5–1 says that in order to statically determine the deadlock potential of an *SI* program, we need only analyze the 1–cycle subset of the 2–cycle model. If the 1–cycle subset contains a deadlock state, then the program is guaranteed to deadlock. If the 1–cycle subset contains no deadlocks, then the program will never deadlock. Although 2–cycle models are accurate, they are not complete. Recall the example from Section Three where two processes deadlock on the first cycle and a third process deadlocks on its second cycle. In this case, the 1–cycle subset does not contain the partial deadlock state where all three processes are deadlocked.

The following result will help us derive the size of complete models of *SI* programs. It uses properties of consistent cyclic dependency equations to bound the number of cycles that a deadlocked process has completed at a 1–cycle deadlock state. Let $max(\sigma_0)$ be the largest initial semaphore value.

Lemma 5–2: If S is a 1–cycle deadlock state in an *SI* program, then each deadlocked process has completed at most $(N-2)(max(\sigma_0)+1)$ cycles.

Proof: Lemma 3–4 tells us that at a 1–cycle deadlock state at least two processes have not completed their first cycles and that all other processes are executing a later cycle. Thus, the largest difference in the number of cycles completed at a 1–cycle deadlock state involving $\zeta \leqslant N$ processes is given by an equation of the form:

$R_1 = R_2 + max(\sigma_0) + 1$

$R_2 = R_3 + max(\sigma_0) + 1$

. . .

$R_{\zeta-2} = R_{\zeta-1} + max(\sigma_0) + 1$

$R_{\zeta-1} = R_\zeta$

$R_\zeta = R_{\zeta-1}$

Since this is a 1–cycle deadlock state, we have $R_\zeta = 0$ and

$$R_1 = (\zeta - 2)(max(\sigma_0) + 1) \leqslant (N-2)(max(\sigma_0) + 1).$$

Thus, in the worst case no deadlocked process has completed more than $(N-2)(max(\sigma_0) + 1)$ cycles.

□

We can now derive the size of complete models of *SI* programs. Let $M = max(\sigma_0) + 1$.

Theorem 5–2: $(NM-2M+2)$–cycle models of *SI* programs are complete.

Proof: We have already shown in Theorem 5–1 that 2–cycle models are accurate. Since the unique deadlocks are the 1–cycle deadlock states and since 1–cycle deadlock states are the only reachable deadlock states, we must show that the $(NM-2M+1)$–cycle subset of the $(NM-2M+2)$–cycle model contains a reachable 1–cycle deadlock state if and only if the ∞–cycle model contains a reachable 1–cycle deadlock state.

If: See Lemma 5–1.

Only If: Let S' be a stable and reachable 1–cycle deadlock state

$$S'(s_1', \ldots, s_\zeta', s_{\zeta+1}', \ldots, s_N')$$

where $D^{S'} = \{1, \ldots, \zeta\}$ is the set of deadlocked processes and $X^{S'} = \{\zeta+1, \ldots, N\}$, is the set of processes that, from Lemma 4–4, will never deadlock. Now consider the 1–cycle deadlock state

$$S(s_1', \ldots, s_\xi', t_{\xi+1}, \ldots, t_\zeta, t_{\zeta+1}, \ldots, t_N)$$

where t_i is the last statement in the $(NM-2M+1)th$ cycle of process i. From Lemma 5–2 we know that S is contained in the $(NM-2M+1)$–cycle subset of the ∞–cycle model. Clearly S is reachable, which from Lemma 5–1 implies that S is a reachable 1–cycle deadlock state in the $(NM-2M+1)$–cycle subset of the $(NM-2M+2)$–cycle model.

□

Theorem 5–2 says that we can determine the size of a complete model for an *SI* program by simply counting the number of processes and determining the value of the largest initial semaphore.

Theorems 5–1 and 5–2 suggest a strategy for efficient static detection of deadlocks in an *SI* program. First we generate the 2–cycle model and analyze its 1–cycle subset for deadlocks. If the 1–cycle subset contains no deadlock states, then by Theorem 5–1 the program is deadlock free. If the 1–cycle subset contains a deadlock, then the program will inevitably deadlock. To identify all of the unique deadlocks in the program, then by Theorem 5–2 we generate the $(NM-2M+2)$–cycle model and analyze its $(NM-2M+1)$–cycle subset for deadlock states.

6. Conclusions

We have solved the finite modeling problem for a class of cyclic consumable *PV* programs known as *SI* programs. We showed that determining the size of accurate and complete models of *SI* programs requires time proportional to the sum of the number of processes and the number of semaphores. In particular we showed that 2–cycle models are accurate for *SI* programs and that $(NM-2M+2)$–cycle models are complete, where N is the number of processes, and M is the largest initial semaphore value plus unity.

Along the way, we introduced the the notion of stable and unstable partial deadlock states as a way of distinguishing between interesting and uninteresting partial deadlock states. We introduced a weak form of the deadlock predicate whose satisfiability is a necessary condition for deadlock in *PV* programs. We then used this weak deadlock predicate to derive a number of necessary conditions for deadlock in *SI* programs. We next used the geometric model of concurrent programs to show that the existence of a (perhaps unreachable) deadlock state is a sufficient condition for deadlock in *SI* programs. Finally, we used these results to derive the size of accurate and complete models. We are currently studying the finite modeling problem for more general classes of *PV* programs.

References

[AnS83] G. R. Andrews and F. B. Schneider, Concepts and Notations for Concurrent Programming, *Computing Surveys 15*, 1 (March 1983), 3–43.

[Car84] S. D. Carson, Geometric Models of Concurrent Programs, PhD Dissertation, University of Virginia, 1984.

[Cla80] E. M. Clarke, Synthesis of Resource Invariants for Concurrent Programs, *ACM Transactions on Programming Languages and Systems 2*, 3 (July 1980), 338–358.

[CoD73] E. G. Coffman and P. J. Denning, *Operating System Theory*, Prentice–Hall, Englewood Cliffs, New Jersey, 1973.

[Hab72] A. N. Habermann, Synchronization of Communicating Processes, *Communications of the ACM 15*, 3 (March 1972), 171–176.

[Hab75] A. N. Habermann, Path Expressions, Technical Report, Department of Computer Science, Carnegie–Mellon University, June 1975.

[Hol72] R. C. Holt, Some Deadlock Properties of Computer Systems, *Computing Surveys 4*, 3 (September 1972), .

[OwG76] S. Owicki and D. Gries, Verifying Properties of Parallel Programs: An Axiomatic Approach, *Communications of the ACM 19*, 5 (May 1976), 279–284.