

# The Geometry of Semaphore Programs

SCOTT D. CARSON and PAUL F. REYNOLDS, JR.

University of Virginia

---

Synchronization errors in concurrent programs are notoriously difficult to find and correct. Deadlock, partial deadlock, and unsafeness are conditions that constitute such errors.

A model of concurrent semaphore programs based on multidimensional, solid geometry is presented. While previously reported geometric models are restricted to two-process mutual exclusion problems, the model described here applies to a broader class of synchronization problems. The model is shown to be exact for systems composed of an arbitrary, yet fixed number of concurrent processes, each consisting of a straight line sequence of arbitrarily ordered semaphore operations.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; D.3.3 [**Programming Languages**]: Language Constructs—*concurrent programming structures*; D.4.1 [**Operating Systems**]: Process Management—*concurrency; deadlocks; multiprocessing/multiprogramming; mutual exclusion; synchronization*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*; H.1.1 [**Models and Principles**]: System and Information Theory—*general systems theory*

General Terms: Algorithms, Reliability, Theory, Verification

Additional Key Words and Phrases: Static deadlock detection, semaphore programs

---

## 1. INTRODUCTION

Deadlock avoidance, detection, and recovery were among the first topics in operating systems to receive formal treatment in the literature. Early studies were primarily concerned with deadlock *avoidance* during resource allocation in large, multiprogrammed systems such as OS/360 [6]. More recently, studies have treated the problem of statically proving the deadlock freedom of a set of communicating, concurrent processes.

This paper presents a geometry of semaphore programs based on Dijkstra's *progress graphs* [3]. Starting with a small set of postulates, we prove conditions under which blocking, full and partial deadlock, unsafeness, and unreachability of states occur. The model is provably exact for concurrent programs in which each task is composed of a straight-line sequence of semaphore operations. Algorithms for generating and analyzing the model are presented in [1].

Sections 2 and 3 of this paper present background material relevant to our study. In Section 4 we develop the progress graph as a formal geometrical object

---

Authors' addresses: S. D. Carson, Department of Computer Science, University of Maryland, College Park, MD 20742; P. F. Reynolds, Jr., Department of Computer Science, University of Virginia, Charlottesville, VA 22903.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 0164-0925/87/0100-0025 \$00.75

ACM Transactions on Programming Languages and Systems, Vol. 9, No. 1, January 1987, Pages 25-53.

and present the set of postulates that are used throughout the paper. Section 5 presents the theory of the hypersolid objects that constitute a progress graph, known as “forbidden regions.” We derive the conditions under which a point in a progress graph represents a state at which a process is blocked by introducing a property known as “nearness.” Next, in Section 6, we show that regions within which processes are blocked, known as “nearness regions,” can be derived from forbidden regions, and we prove the conditions under which the intersection of multiple nearness regions models deadlock in the original system. In Section 7 we derive techniques for computing the intersection of nearness regions. We use these results in Section 8 to derive the coordinates of regions of points at which deadlock is inevitable, known as “unsafe regions.” Section 9 presents the rather surprising result that the reachability problem described in [2] reduces to the problem of finding unsafe regions in a transformed progress graph. Finally, Section 10 summarizes the paper.

## 2. BACKGROUND

A *progress graph* is a multidimensional, Cartesian graph in which the progress of each of a set of concurrent processes is measured along an independent time axis. Each point in the graph represents a set of process times. Areas of relative progress disallowed by synchronization primitives invoked by the processes are represented as rectangular regions; these areas are known as *forbidden regions*. Two-dimensional progress graphs were first used by Dijkstra [3] to illustrate the action of his synchronization primitives, *P* and *V* [4].

More recent works by Papadimitriou, Yannakakis, Lipski, and Kung [8, 10, 11] have attempted to exploit the geometric properties of forbidden regions to find deadlocks in locked database-transaction systems. These studies have met with limited success in two senses: First, the algorithms for deadlock detection presented in [8] do not generalize to systems composed of more than two processes. Second, the studies are only concerned with progress graphs that represent systems composed of binary semaphore operations. The model developed in this paper contains neither of these restrictions.

## 3. FUNDAMENTALLY RELATED CONCEPTS

The geometric model developed in this paper is fundamentally related to two other models of synchronization: Holt’s directed graph model [6] and Habermann’s semaphore invariant model [2, 5]. In this section we present a brief discussion of both models. A more elaborate treatment can be found in [1].

Holt’s directed graph model is conceptually straightforward. A concurrent system is represented as a set of nodes (states) and directed arcs (state transitions). Each arc is labeled with the name of the process that causes the associated state transition. In Holt’s model, as in other models, the system is modeled using single-process state transitions; thus, no arc can have more than one label.

A process *P* is blocked at a node (state) *S* if there is no arc leaving node *S* with label *P*. Total deadlock, in which every process in the system is blocked, is represented as a node with no outgoing arcs. Partial deadlock, in which a subset of the processes is blocked, occurs when a process is blocked at a node and at

all nodes reachable from that node. Thus, total deadlock can be discovered by examining single nodes, whereas partial deadlock must be discovered by examining many nodes.

Habermann's semaphore invariant provides an interesting contrast to Holt's model. The semaphore invariant is based on Habermann's observation that the value of a semaphore is always equal to its initial value, minus the number of times that it has been successfully requested (Dijkstra's  $P$  operation), plus the number of times it has been produced (Dijkstra's  $V$  operation). Additionally, the semaphore invariant states that the value of a semaphore must always be greater than or equal to zero. This condition is denoted  $I_\sigma$ , for a semaphore  $\sigma$ . The semaphore invariant is typically expressed in terms of auxiliary variables that are (conceptually) added to the program to count semaphore operations on each semaphore.

The semaphore invariant can be used as a *resource invariant* in Owicki and Gries's proof system [9], as shown by Clarke [2]. A predicate that is true if the concurrent system can deadlock is formed. This predicate, called the *deadlock predicate*, is expressed in terms of the preconditions of the statements in each process, and in terms of  $I_\sigma$ . Informally, the deadlock predicate states that "each process has reached a statement beyond which it cannot proceed because the value of the semaphore it awaits is zero."

The semaphore invariant model is more compact than the graph model. Rather than representing a system explicitly, as a set of nodes and arcs, the semaphore invariant represents a system implicitly, as a set of simultaneous, linear equations. However, the semaphore invariant is shown in [2] to be incomplete in the sense that it defines a necessary, but not sufficient condition for deadlock. In some cases, the deadlock predicate is true when a system contains deadlocks that are unreachable as part of any execution sequence of the concurrent system. Additionally, it is shown in [1] that the deadlock predicate is not sufficiently powerful to express partial deadlock. These problems arise because the deadlock predicate defines what Keller calls a *state-intrinsic* property: a property of states, but not of transitions [7]. Reachability and partial deadlock are *state-extrinsic* properties; they are expressed in terms of sequences of state transitions.

#### 4. PROGRESS GRAPHS

The idea of using progress graphs to represent concurrent programs was first published in [3], though Coffman attributes the concept to Dijkstra. A *progress graph* is an  $N$  dimensional Cartesian graph in which the progress of each of  $N$  processes is measured against an independent time axis. The axis assigned to a process is labeled with the process's synchronizing events in the order in which they are executed. Some regions in the graph violate constraints on the relative progress of the processes imposed by the synchronization events. These regions are known as *forbidden regions*. The model developed here provides a way of characterizing the geometry of forbidden regions to determine whether the multiprogram that defines them can deadlock.

It should be noted that our model represents the first successful treatment of progress graphs of more than two dimensions. In particular, Lipski and

Papadimitriou remark that “generalizing our deadlock detection ideas to even the case of three transactions [processes] appears much more difficult . . . our two-transaction deadlock detection algorithm does not appear to generalize to many transactions” [8]. This is because certain properties of progress graphs that appear intuitive in two dimensions do not apply to more than two dimensions: the two-dimensional progress graph is, in a sense, a degenerate case. The reader should not be deceived by the apparent simplicity of our treatment of  $N$ -dimensional progress graphs; the model presented here has not been derived from apparently obvious (yet unsuccessful) extensions of the two-dimensional theory.

The forbidden regions of a progress graph are simply derived for certain problems. For instance, consider the following concurrent program in which two processes each access two critical sections:

```
cobegin
  A: cycle
     $P(a); P(b); V(b); V(a)$ 
  end
  //
  B: cycle
     $P(b); P(a); V(a); V(b)$ 
  end
end
```

The corresponding progress graph is shown in Figure 1. The forbidden regions represent relative degrees of progress of the two processes that are disallowed by the mutual exclusion properties of the semaphores. For instance, process  $B$  cannot perform the  $P(a)$  operation while process  $A$  is between its  $P(a)$  and  $V(a)$  operations.

Progress graphs have intrinsic properties that can be used to prove other, less obvious properties. First, each point in a progress graph represents a specific degree of progress for each process. Each *concurrency state*, or ordered set of process times of the system, is represented by a unique point; conversely, each point in a progress graph represents a unique concurrency state. Second, a state transition is a ray defined by an initial state (a point) and an ordered set of values (a vector) representing the amount of progress made by each process during the transition. Third, a forbidden region represents a collection of states that are inherently infeasible; thus, a point representing the state of a system cannot lie within a forbidden region. Fourth, the time between two synchronizing events within a process is assumed to be greater than zero. Lastly, no process can progress backward in time, implying that the vector portion of a state transition can have no negative component. These properties are formalized below as postulates:

- P1: The concurrency state of a system defines a unique point in a progress graph.
- P2: A transition from a state represented by a point  $p_1$  to a state represented by a point  $p_2$  is a ray rooted at  $p_1$  with direction  $\overrightarrow{p_1 p_2}$ .
- P3: A point is *feasible* if and only if it is not within a forbidden region.
- P4: The time between two synchronizing events within each process is greater than zero.

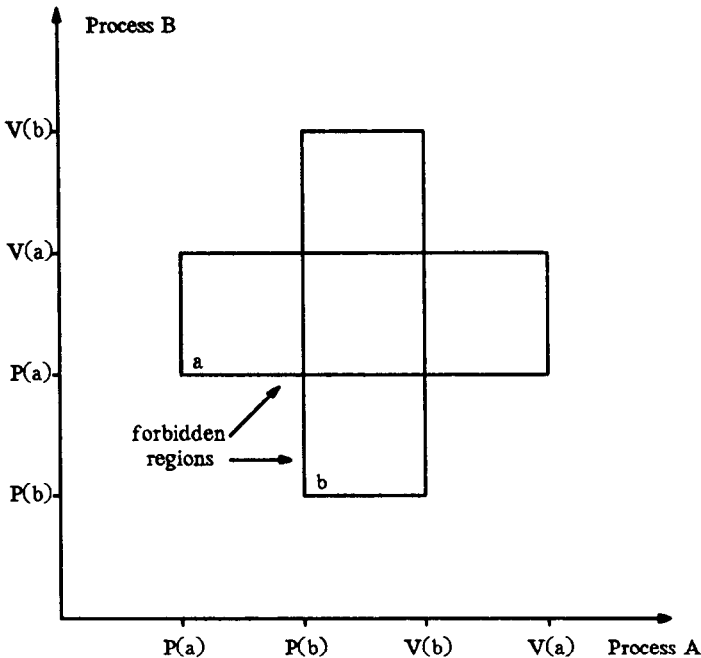


Fig. 1. A progress graph.

P5: Processes cannot progress backward in time; that is, no (scalar) component of a state transition vector can be negative.

P3 states that no feasible point is within a forbidden region; thus, a forbidden region is a collection of infeasible points. A *feasible* point (or equivalently, a feasible state) is one that represents a legitimate set of degrees of progress. In other words, a feasible state is one that does not violate any of the constraints imposed by the synchronization primitives used in a system. This is the property expressed most concisely by the semaphore invariant.

*Definition.* A point  $P(p_1, \dots, p_N)$  is *feasible* if and only if for all semaphores  $\sigma$ ,  $I_\sigma(P)$ .

Thus, the semaphore invariant defines the set of points that are not within forbidden regions.

P4 states that the synchronizing events within each process must be assigned a monotonically increasing sequence of times. One might wonder if a particular choice of times could influence the deadlock properties of the system. However, it is well known that only the order of events within each process determines whether or not deadlock can occur. In the semaphore invariant method, for example, the order of events within each process generates the pre- and post-conditions for each statement without making any assumption about the time between events (except that it is nonzero).

In the remainder of this paper, we choose unity as the time between synchronizing events. A discussion of other mappings is presented in [1]. Regardless of

the mapping we choose, we treat the progress graph as a continuous space of *concurrency* states, with an embedded, discrete space of *synchronization* states. A state transition is a continuous path between two states.

A *trajectory* in a progress graph is an ordered set of states that defines a particular execution sequence of the system of concurrent processes. Lemma 1 proves the intuitive result that the trajectory cannot pass through a forbidden region.

**LEMMA 1.** *A trajectory in a progress graph cannot cross the border of a forbidden region.*

**PROOF.** Directly from P3.  $\square$

Thus, a state transition defines a continuum of points between two named states, all of which must be feasible.

The vector portion of a state transition determines the amount of progress made by each process during the transition. Since forbidden regions restrict the set of possible transitions from a given state, the underlying synchronizing system (i.e., the scheduler) may at some time be required to inhibit the progress of certain processes to ensure that the system's trajectory avoids them. Accordingly, we might attempt to find potential deadlocks in the system by finding those reachable points on the surfaces of the forbidden regions at which no process can make further progress. The disadvantage with this method is that representing arbitrary surfaces in  $N$  dimensions requires exponential time and space. In the next section we study the properties of the intersections of forbidden regions and their ability to model deadlocks without representing arbitrary surfaces.

A *feasible state transition* is one that is allowed by the constraints imposed by the semaphore operations in a multiprogram. More formally, it is defined as follows:

*Definition.* A state transition in a system of concurrent processes is *feasible* if and only if, in the progress graph representing the system, the state transition ray representing the transition does not cross the border of a forbidden region.

The semaphore invariant method fails when the progress graph contains *trap states*, or unreachable deadlocks. In Figure 2, for example, state  $T$  is a deadlock state that is unreachable through any execution sequence of processes  $A$  and  $B$ . Clarke's strengthening procedure eliminates all unreachable feasible points from the progress graph. Although we use a different technique for eliminating unreachable points from consideration, the effect is the same as that of Clarke's algorithm.

We can define the reachability of a point (state) in terms of state transitions, starting from the initial state:

*Definition.* A point  $P$  in a progress graph is *reachable* if and only if there exists a sequence of state transition vectors,  $R_1, R_2, \dots, R_K$  such that, for all  $i$ ,  $\sum_1^i R_i$  is feasible, and  $\sum_1^K R_i = P$ .  $\square$

We will develop techniques for eliminating sets of unreachable points later in this paper.

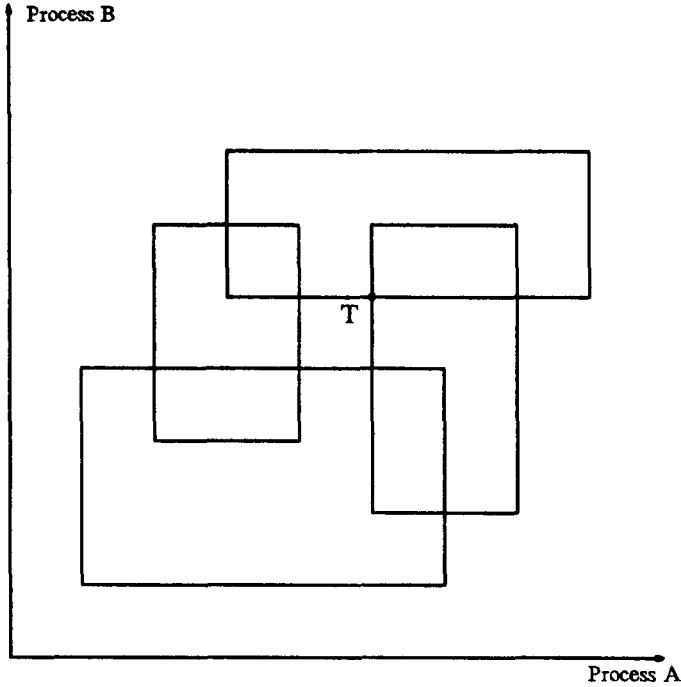


Fig. 2. A progress graph with trap state  $T$ .

## 5. FORBIDDEN REGIONS

Holt's graph model requires an explicit representation of a state graph, with its associated nodes and arcs. The geometric model, on the other hand, has smaller space requirements, since a set of infeasible points is represented as a single forbidden region. One possible method of representing the forbidden regions consists of constructing a graph in which only the nodes at the "corners" of the forbidden regions are stored.

The problem with this approach is that, even though the number of nodes represented becomes a small fraction of the total number of states as the number of processes (dimensions) grows, the number of nodes remains an exponential function. Although we will encounter the exponential space problem again later, it seems appropriate to seek better techniques for representing forbidden regions.

In this section we formally define a forbidden region, and find a representation that can be stored and manipulated in time proportional to the number of processes. We show how forbidden regions in a progress graph model blocking in a concurrent system by defining a property of states called *nearness*.

### 5.1 Representing Forbidden Regions

A forbidden region is an  $N$ -dimensional rectangular structure that is bounded by  $2N - 1$  dimensional bounded hyperplanes. The location and direction of each hyperplane are defined by the equation  $x_i = k$ , where  $x_i$  is the  $i$ th coordinate axis

and  $k$  is some constant. The bounds of each hyperplane are defined by its intersections with the other hyperplanes comprising the forbidden region. Each hyperplane is parallel to exactly one other hyperplane within a forbidden region and is orthogonal to all others. Further, each hyperplane is parallel to all coordinate axes except one (the one that defines the hyperplane). These properties allow us to represent an entire forbidden region without explicitly naming each "corner"; in fact, we can represent a region with only two points, which we will call the *vertex* and the *extent*.

The *vertex* of a forbidden region is the point on the region closest to the origin, whereas the *extent* of a forbidden region is the point on the region farthest from the origin. (The reader may envision diagonally opposed points on a square and on a cube to help understand this definition.) For instance, suppose we are given the points (1, 1, 1, 1) and (3, 4, 5, 6) as the vertex and extent of a forbidden region. These two points, plus the information that all sides are orthogonal, implicitly represent the 16 corners of this four-dimensional region:

(1, 1, 1, 1)	(1, 1, 1, 6)	(1, 1, 5, 1)	(1, 1, 5, 6)
(1, 4, 1, 1)	(1, 4, 1, 6)	(1, 4, 5, 1)	(1, 4, 5, 6)
(3, 1, 1, 1)	(3, 1, 1, 6)	(3, 1, 5, 1)	(3, 1, 5, 6)
(3, 4, 1, 1)	(3, 4, 1, 6)	(3, 4, 5, 1)	(3, 4, 5, 6)

Now we are prepared to formally define a forbidden region:

*Definition.* A *forbidden region* is a triple  $(\sigma, V, E)$ , where  $\sigma$  is the semaphore associated with the region,  $V$  is the vertex of the region, and  $E$  is the extent of the region.

The symmetric structure of a forbidden region permits a representation that grows linearly in size as the dimension of the space increases.

## 5.2 Blocking and Forbidden Regions

Forbidden regions model the blocking action that occurs when multiple processes compete for fewer resources than are available. A trajectory cannot penetrate a forbidden region. If a trajectory "collides" with a forbidden region, then the underlying synchronization system must inhibit the progress of certain processes, causing the trajectory to "circumvent" the forbidden region. In this section we prove that a forbidden region can block at most one process in a given trajectory. This result will be used in the next section to show how the intersections of multiple regions model deadlocks.

First, let us formalize the notion of blocking in the context of the geometric model.

*Definition:* A process,  $i$ , is *blocked* in a particular state if and only if the state transition  $(0, \dots, x_i = \delta, \dots, 0)$  is not feasible.

In this case  $\delta$  is a quantity large enough to cause the process  $x_i$  to enter a new (discrete) synchronization state, but small enough that  $x_i$  does not pass through any intermediate synchronization states. If the event times are mapped onto integers (as we assume they are), then unity is an appropriate choice for  $\delta$ . Intuitively, this means that a process is blocked if and only if it cannot proceed

to its next synchronization state independently. Notice that this definition is identical to the definition of a blocked process in the semaphore invariant method; it, too, lacks the power to state that a process is blocked for all time.

Another useful concept is that of containment.

*Definition.* A point  $P(p_1, \dots, p_N)$  is *contained* within forbidden region  $R(\sigma, V, E)$  if and only if  $\forall_i: v_i \leq p_i < e_i$ .

This definition says that the state of the system can legitimately lie on any of the hyperplanes defined by the extent of a region, but that it can only approach the hyperplanes defined by the vertex. We can interpret this to mean that a process can approach the point at which it is granted an unavailable resource, but that it cannot arrive at that point until the awaited resource becomes available.

We can say that a point is *near* a bounded hyperplane defined by  $x_i = k$  if its distance from the hyperplane is within  $\delta$  and it is within the bounds of the hyperplane in all dimensions but the  $i$ th. In other words, a point is near a bounded hyperplane if and only if the shortest distance between the point and the bounded hyperplane is a line segment perpendicular to the hyperplane of length less than or equal to  $\delta$ .

*Definition.* A point  $P(p_1, \dots, p_N)$  is *near* a hyperplane defined by  $x_i = \alpha$  and bounded by  $\forall_{j \neq i \in \{1 \dots N\}} : \beta_j^1 \leq x_j < \beta_j^2$  if and only if  $\forall_{j \neq i \in \{1 \dots N\}} : \beta_j^1 \leq p_j < \beta_j^2$  and  $\alpha - \delta \leq p_i < \alpha$ .

Note that each hyperplane in a forbidden region can be expressed in a form suitable for the above definition.

We can divide the hyperplanes of a forbidden region into two sets: those that intersect at the vertex, and those that intersect at the extent. These two sets are called the *vertex set* and the *extent set*, respectively.

**LEMMA 2.** *If a point is near a hyperplane in the extent set of a forbidden region  $R(\sigma, V, E)$ , then it is infeasible.*

**PROOF.** See [1].  $\square$

Informally, any point near an extent hyperplane of a forbidden region is contained within that forbidden region.

**LEMMA 3.** The  $\zeta$ th process is blocked at a feasible point  $P(p_1, \dots, p_N)$  if and only if  $P$  is near the  $\zeta$ th vertex hyperplane of some forbidden region  $R(\sigma, V, E)$ .

**PROOF.**

**IF.** From the definition of nearness, we have

$$v_\zeta - \delta \leq p_\zeta < v_\zeta \wedge \forall_{j \neq \zeta \in \{1 \dots N\}} : v_j \leq p_j < e_j.$$

So if the system attempts to change state from  $P$  via the transition  $(0, 0, \dots, x_\zeta = \delta, \dots, 0)$ , the  $\zeta$ th coordinate of the new state will be  $p'_\zeta$  such that  $p'_\zeta \geq v_\zeta$ . The new state satisfies the criteria for containment in  $R$ ; therefore the new state is infeasible. So, by definition, the  $\zeta$ th process is blocked at  $P$ .

ONLY IF. If the  $\zeta$ th process is blocked, then the transition  $(0, \dots, x_\zeta = \delta, \dots, 0)$  is by definition infeasible. So the point  $P'(p_1, \dots, p_\zeta + \delta, \dots, p_N)$  is contained in some forbidden region  $R(\sigma, V, E)$ . Thus

$$\forall_{i \neq \zeta \in \{1 \dots N\}} : v_i \leq p_i < e_i \quad (*)$$

and

$$v_\zeta \leq p_\zeta + \delta < e_\zeta. \quad (**)$$

Since  $P$  is feasible, it must be the case that  $p_\zeta < v_\zeta$ . This, combined with (\*\*) above, yields

$$v_\zeta - \delta \leq p_\zeta < v_\zeta. \quad (***)$$

From (\*) and (\*\*\*),  $P$  is near the hyperplane defined by the  $\zeta$ th coordinate of the vertex of  $R$ ,  $v_\zeta$ .  $\square$

Lemmas 2 and 3 prove that only the hyperplanes in the vertex set of a forbidden region can block processes. Since deadlock is a form of blocking, the lemmas suggest that only the vertex hyperplanes of forbidden regions are important to deadlock detection. The next two lemmas prove that a single forbidden region can only model the blocking of one process for a given trajectory. Later, we will use this result to show that a state in which  $k$  processes are blocked is near the intersection of  $k$  forbidden regions.

**LEMMA 4.** *If a point  $P(p_1, \dots, p_N)$  is near one hyperplane in the vertex set of a region  $R(\sigma, V, E)$ , then it is near no other hyperplane in the vertex set of  $R$ .*

**PROOF.** Let the hyperplane that  $P$  is near be the  $\zeta$ th. From the definition of nearness, we have

$$v_\zeta - \delta \leq p_\zeta < v_\zeta \wedge \forall_{j \neq \zeta \in \{1 \dots N\}} : v_j \leq p_j < e_j.$$

Now suppose that  $P$  is also near another vertex set hyperplane, say, the  $\eta$ th hyperplane. Then

$$v_\eta - \delta \leq p_\eta < v_\eta \wedge \forall_{j \neq \eta \in \{1 \dots N\}} : v_j \leq p_j < e_j,$$

which includes

$$v_\zeta \leq p_\zeta < e_\zeta.$$

This contradicts the hypothesis that  $p_\zeta < v_\zeta$ .  $\square$

**LEMMA 5.** *If a point  $P(p_1, \dots, p_N)$  is near a hyperplane in the vertex set of a forbidden region  $R(\sigma, V, E)$ , then there exists no transition to a point  $Q(q_1, \dots, q_N)$  such that  $Q$  is near a different hyperplane in the vertex set of  $R$ .*

**PROOF.** Again, let the hyperplane that  $P$  is near be the  $\zeta$ th. The familiar nearness relation holds at  $P$ :

$$v_\zeta - \delta \leq p_\zeta < v_\zeta \wedge \forall_{j \neq \zeta \in \{1 \dots N\}} : v_j \leq p_j < e_j.$$

Now, if the point  $Q$  is near a different hyperplane, say, the  $\eta$ th, then

$$v_\eta - \delta \leq q_\eta < v_\eta \wedge \forall_{j \neq \eta \in \{1 \dots N\}} : v_j \leq q_j < e_j.$$

So  $q_\eta$  is less than  $p_\eta$ , because  $p_\eta \geq v_\eta$  and  $q_\eta < v_\eta$ . To make a transition from  $P$  to  $Q$ , the  $\eta$ th process would have to make negative progress, contradicting postulate P5. Thus there is no transition from  $P$  to  $Q$ .  $\square$

We can interpret Lemma 5 to mean that, once a forbidden region has blocked one process in a particular trajectory, it can never block a different process. A single forbidden region cannot model a deadlock, because a deadlock is a state in which two or more processes are blocked forever. So a deadlock must occur when some spatial relationship among multiple forbidden regions is satisfied. Theorem 1 proves that a point at which  $\beta$  processes are blocked is near exactly  $\beta$  distinct forbidden regions.

**THEOREM 1.** *If  $\beta$  processes are blocked at a point  $P(p_1, \dots, p_N)$ , then  $P$  is near the vertex hyperplanes of  $\beta$  distinct forbidden regions.*

**PROOF.** From Lemma 3, if  $\beta$  processes are blocked, then  $P$  is near  $\beta$  distinct vertex hyperplanes. From Lemma 4, no two of the  $\beta$  hyperplanes can belong to the same forbidden region.  $\square$

## 6. NEARNESS REGIONS

Theorem 1 suggests that to find all of the states at which  $\beta$  processes are blocked it is necessary to find those points that are near exactly  $\beta$  forbidden regions. In this section we develop a method for generating sets of such points. We also show how the sets can be analyzed to find partial and total deadlocks.

So far, we have used the inherent properties of forbidden regions to prove our results. Now, however, we must make use of a result that restricts the way in which forbidden regions are generated from the source program.

**LEMMA 6.** *Given two points,  $P(p_1, \dots, p_N)$  and  $Q(q_1, \dots, q_N)$ , such that during the transition from  $P$  to  $Q$   $\beta$  processes progress to their next synchronizing events, and  $N - \beta$  processes make no progress: if the transition from  $P$  to  $Q$  is feasible, then there exists a trajectory from  $P$  to  $Q$  such that during each state transition exactly one of the  $\beta$  processes progresses to its next synchronizing event and all others make no progress.*

**PROOF.** See [1].  $\square$

The proof of Lemma 6 requires techniques that are beyond the scope of this paper. Intuitively, Lemma 6 means that, if we can go from  $P$  to  $Q$  by running multiple processes concurrently, we can go from  $P$  to  $Q$  running the process one at a time. This means, for example, that, if a diagonal path through a three-dimensional cube exists, then there also exists a path that follows the cube's edges. More important, Lemma 6 implies that, if, at a point  $P$ , no single-process, single-event transition is feasible, then there exists no feasible transition from  $P$ . It also implies that, if a given number of processes are blocked at a point, they remain blocked until another process allows them to progress.

Given Lemma 6, the converse of Theorem 1 is also true. Thus, all points near, say,  $k$  forbidden regions are those points at which  $k$  processes are blocked. The situation ruled out by Lemma 6 is that depicted in Figure 3. The point  $P$  is near

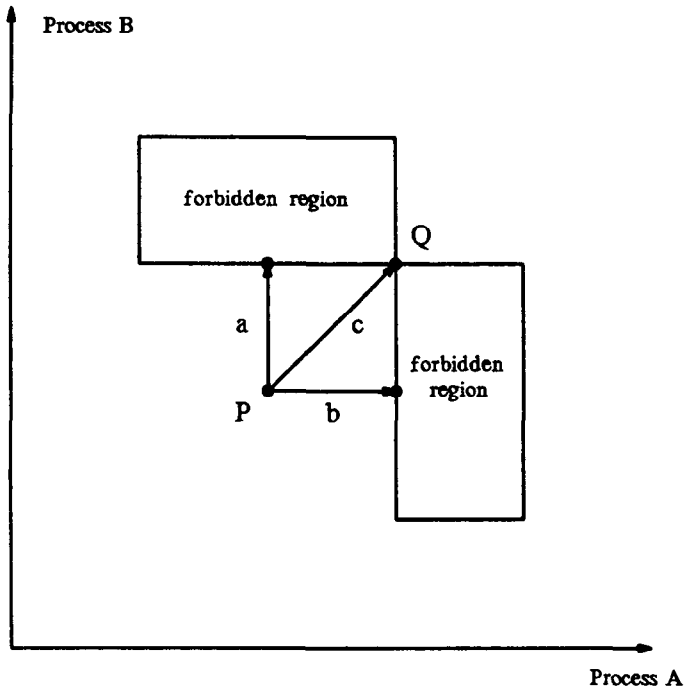


Fig. 3. Situation ruled out by Lemma 6. Transition  $c$  is feasible, but  $a$  and  $b$  are not.

the two forbidden regions shown in the figure, but since point  $Q$  is feasible, the system could progress via the trajectory  $PQ$ , even though neither process one nor process two can progress independently.

Defining the sets of points near the hyperplanes of a forbidden region is straightforward. Since we need only define those points near the vertex set hyperplanes, each forbidden region generates  $N$  (the number of processes) such sets. We call each set of points near a vertex hyperplane a *nearness region*. Since a nearness region has the same shape as a forbidden region, we can use a vertex and an extent to characterize its location and bounds. Additionally, we can associate a direction with each nearness region, which indicates which process is blocked at all points within.

*Definition.* 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 its extent.

It is easy to generate the vertices and extents of the nearness regions, especially if the event times in the progress graph are integers. Supposing that they are, we have only to form a region with the same shape as each vertex hyperplane, but with unit thickness. For instance, suppose that a forbidden region is defined (in two dimensions) as having a vertex  $(2, 2)$  and an extent  $(4, 4)$ . Each vertex is generated by subtracting unity from a particular vertex coordinate of the forbidden region, while each extent is generated by substituting the vertex coordinate

of the forbidden region into the dimension from which unity is subtracted. Thus, the vertices and extents of the nearness regions are

$$V_1 = (2, 1), \quad E_1 = (4, 2), \quad V_2 = (1, 2), \quad E_2 = (2, 4).$$

Figure 4 shows the forbidden region defined above, and the nearness regions it generates.

If we compute the intersection of two nearness regions with different associated directions, then we have a set of points at which two different processes are blocked. Likewise, we can compute intersections of more than two nearness regions, obtaining sets of points at which more than two processes are blocked. We call a nearness region that is the intersection of  $\zeta$  distinct nearness regions a *nearness region of degree  $\zeta$* . The nearness regions that are formed directly from forbidden regions are of degree 1. We will see how the coordinates of nearness regions of higher degree are computed in the next section. For now, let us consider the properties of nearness regions.

A nearness region of degree  $\zeta$  has associated with it a set of blocked processes, called the *blocked set*, which are the directions of its parent nearness regions of degree 1. If the set is full (contains elements 1 through  $N$ ), then all processes are blocked; no process can act to free them. This represents a total deadlock.

If fewer than all the processes are blocked, then we must check to determine whether they are blocked for all time before we can state that the nearness region represents a partial deadlock. We can do this by checking the extent coordinates in those dimensions that represent unblocked processes to see if they are infinite. Ways in which nearness regions of infinite extent are formed are discussed in [1].

Since a nearness region of degree  $\zeta$  is finite in the direction of each of its  $\zeta$  parents' hyperplanes, it can be infinite in at most  $N - \zeta$  dimensions. If we associate a set of infinite directions with each nearness region, called the *invariant set*, then the nearness region is a partial deadlock only if

$$\{\text{blocked}\} \cup \{\text{invariant}\} = \{1, 2, \dots, N\},$$

the universal set of all directions (processes). Note that we have

$$\{\text{blocked}\} \cap \{\text{invariant}\} = \emptyset$$

by definition. In other words, a partial deadlock region is one in which the blocking of a set of processes (the blocked set) is invariant over the progress of the remaining processes (the invariant set). Using this method, even a single process can be involved in a partial deadlock, provided it remains blocked indefinitely.

*Definition.* A deadlock region is a four-tuple  $(I, B, V, E)$ , where  $I$  is the invariant set,  $B$  is the blocked set,  $V$  is the vertex, and  $E$  is the extent, such that  $B \cup I = \{1 \dots N\}$ .

Once we have found all the nearness regions of various degrees, we must still determine whether the nearness regions are feasible and whether they are reachable. Checking feasibility can be straightforward. Since the forbidden regions represent those points that are infeasible, if they totally enclose a nearness

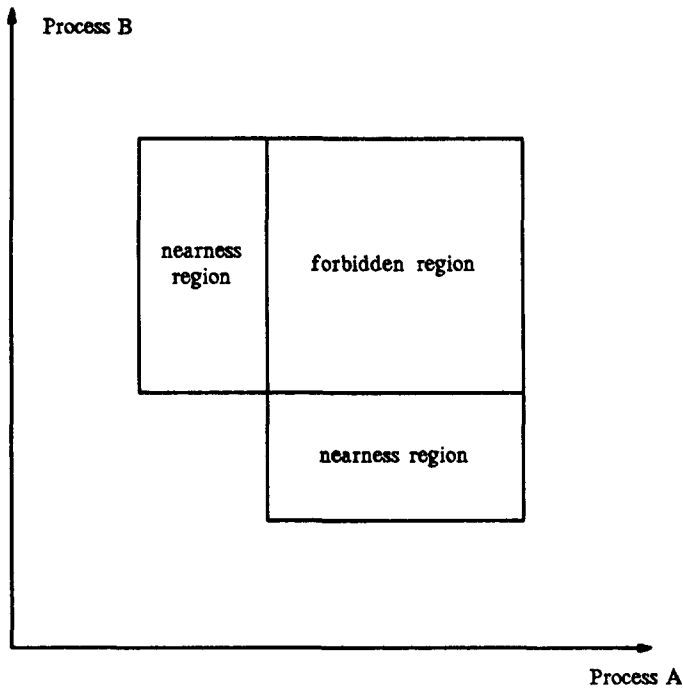


Fig. 4. Nearness regions.

region, then that nearness region is infeasible. If a nearness region is infeasible, then all of its children (nearness regions of higher degree that are the intersection of it and other nearness regions) are also infeasible.

Nearness regions possess several other useful properties. First, Lemma 4 indicates that no two nearness regions formed from the same forbidden region can intersect. Second, if we attempt to compute the intersection of two nearness regions and find that they do not intersect, then we know *a priori* that neither nearness region intersects the other's children. Third, we need only concern ourselves with nearness regions formed from a unique set of directions. There is nothing to be gained, for instance, from computing the intersection of a region with direction  $i$  with another region of direction  $i$ . The savings due to these properties is significant, as shown in [1].

## 7. INTERSECTION AND ENCLOSURE

The highly symmetrical structure of forbidden regions and nearness regions suggests a simple technique for calculating their intersections. We start with a "region of intersection" equal to the entire space. Then, considering a single dimension at a time, we eliminate those points that do not lie within the bounds of both regions in that dimension. If, in any dimension, there exist no points common to both regions, then the two regions do not intersect.

Since forbidden regions and nearness regions (and, indeed, most regions that we consider) are represented in the same fashion, it is convenient to work with *generic regions*. A generic region is specified by a vertex and an extent, but carries

none of the associated information (semaphore, blocked process) of forbidden regions, nearness regions, or deadlock regions. We will often use the term "region" to mean "generic region."

To find the set of points common to two generic regions in a given dimension, we use the appropriate vertex and extent coordinates. Suppose that the two regions are defined by  $(V^1, E^1)$  and  $(V^2, E^2)$ . Then a point,  $P(p_1, \dots, p_N)$ , satisfies the vertex and extent (containment) bounds for both regions in a direction  $i$  if and only if

$$v_i^1 \leq p_i \wedge v_i^2 \leq p_i \wedge p_i < e_i^1 \wedge p_i < e_i^2,$$

which can be more conveniently written as

$$\max(v_i^1, v_i^2) \leq p_i < \min(e_i^1, e_i^2). \quad (7.1)$$

In practice, we blindly compute the maximum vertex and minimum extent for each dimension, then compare them. If the maximum vertex is greater than or equal to the minimum extent in some dimension, then Eq. (7.1) is unsatisfiable in that dimension and the two regions do not intersect. If not, then the maximum vertex and the minimum extent become the corresponding vertex and extent coordinates (bounds) in the region of intersection. Figure 5 shows an example of intersecting two-dimensional regions.

**THEOREM 2.** *Two generic regions,  $(V^1, E^1)$  and  $(V^2, E^2)$ , intersect if and only if*

$$\forall i \in \{1 \dots N\} : \max(v_i^1, v_i^2) < \min(e_i^1, e_i^2). \quad (7.2)$$

**PROOF.**

**IF.** Suppose that (7.2) is satisfied. Then, for each  $i$ , there exists a number  $p_i$  such that  $\max(v_i^1, v_i^2) \leq p_i < \min(e_i^1, e_i^2)$ . It follows that  $v_i^1 \leq p_i < e_i^1 \wedge v_i^2 \leq p_i < e_i^2$ . So the point  $P(p_1, \dots, p_N)$  satisfies the conditions for containment in both regions, and the two regions intersect.

**ONLY IF.** If the regions intersect, then there exists a point  $P(p_1, \dots, p_N)$  that is contained in both regions. Now, let there be a subscript  $j$  such that  $\max(v_j^1, v_j^2) \geq \min(e_j^1, e_j^2)$ . From the definition of containment, we know that  $p_j \geq \max(v_j^1, v_j^2)$ . But this implies that  $p_j \geq \min(e_j^1, e_j^2)$ , so  $p_j$  is greater than or equal to the smaller extent coordinate, contradicting the hypothesis that  $P$  is contained in both regions.  $\square$

It is convenient to compute the blocked set and the invariant set when we compute the intersection of two nearness regions. The set of processes blocked at all points within a nearness region is formed from the union of its parents' blocked sets. A nearness region's invariant set, on the other hand, is the intersection of its parents' invariant sets.

Another property we need to define is called *enclosure*. A region is enclosed by another if the set of points it defines is a subset of that of the other. When one forbidden region encloses another, the enclosed forbidden region contributes nothing to the synchronization structure of the system and can be dropped from the analysis. Similarly, when a forbidden region encloses a nearness region the

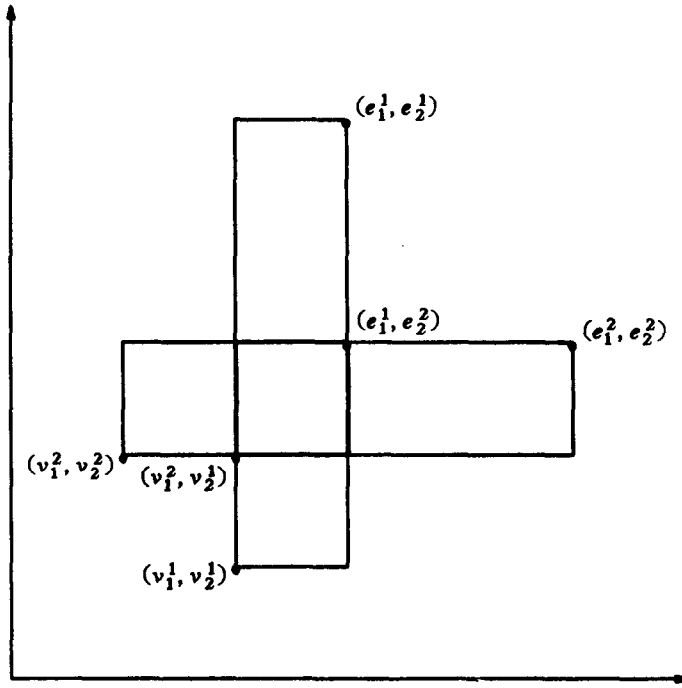


Fig. 5. Intersecting regions.

nearness region is infeasible and can be dropped. Although lack of enclosure by a single forbidden region is not sufficient to prove the feasibility of a nearness region, the enclosure test can be used in a negative fashion, to show infeasibility.

The conditions for enclosure are simple to formulate. Suppose we have two regions, defined by  $R^1(V^1, E^1)$  and  $R^2(V^2, E^2)$ . Then  $R^1$  encloses  $R^2$  if and only if every point contained within  $R^2$  is contained within  $R^1$ . In other words,

$$\forall_{i \in \{1 \dots N\}} : v_i^1 \leq v_i^2 \wedge e_i^1 \geq e_i^2. \quad (7.3)$$

Eq. (7.3) finds regions that enclose others. In general, this is all the information we need when determining whether or not a forbidden region contributes to the synchronization structure of the system. If we want to determine whether or not a nearness region is feasible, however, we may be faced with a sticky problem. Eq. (7.3) can be used to show infeasibility, but the failure of a single forbidden region to enclose a nearness region does not necessarily mean that the nearness region is feasible. A nearness region may be enclosed by the combination of more than one forbidden region.

When Eq. (7.3) fails to prove the infeasibility of a nearness region, we must attempt to prove its feasibility or infeasibility in another manner. There are two alternatives, neither of which is attractive. The first possibility is to simply examine every point (synchronization state) enclosed by the nearness region, checking for feasibility. If the nearness region is of degree  $N$ , then this involves checking a single point, the region's vertex. If, however, the nearness region is of degree less than  $N$ , we must check its entire volume. The possibility that the

nearness region is infinite in various directions is of no consequence, since it is shown in [1] that a (finite) number can be defined as "effective infinity." Nonetheless, it is easy to see that this problem grows exponentially as the difference of  $N$  and the degree of the nearness region.

The other alternative is to compute the volume of the nearness region, then compute the volume of the intersections of it and all the forbidden regions it intersects. If the volume of the nearness region equals the volume of its intersections with forbidden regions, then every point within the nearness region is infeasible. If the volume of the nearness region is greater, then there exist feasible states within.

Suppose that a nearness region,  $A$ , intersects two forbidden regions, and that we call the two resulting regions of intersection  $B$  and  $C$ . Then we must compare the volume of  $A$ ,  $V(A)$ , with the volume of  $B$  plus  $C$ . In general,

$$V(B + C) = V(B) + V(C) - V(B \cap C).$$

The above equation is analogous to the formula for the probability of the union of two possibly nondisjoint events.

Now suppose that an arbitrary number of forbidden regions,  $\zeta$ , intersect a nearness region. The formula for computing the total volume of the intersections of the forbidden regions with the nearness region is analogous to the formula for computing the probability of the union of  $\zeta$  possibly nondisjoint events:

$$\begin{aligned} & V(R_1 + R_2 + \dots + R_\zeta) \\ &= \sum_i V(R_i) - \sum_{i < j} V(R_i \cap R_j) \\ &\quad + \sum_{i < j < k} V(R_i \cap R_j \cap R_k) + \dots \\ &\quad + (-1)^{\zeta+1} V(R_1 \cap R_2 \cap \dots \cap R_\zeta). \end{aligned} \tag{7.4}$$

In the worst case, the number of terms in Eq. (7.4) is the sum of the  $\zeta$ -degree binomial coefficients,  $2^\zeta$ . This situation occurs whenever each  $R_i$  intersects all the others. The best case occurs when the regions of intersection are all disjoint. In this event the volume of their sum reduces to the sum of their volumes. In practice, we must weigh the likelihood of incurring worst-case performance with this method against the penalty of searching the entire volume of the region.

## 8. UNSAFE REGIONS

In the last section we showed how nearness regions in a progress graph model deadlocks in a system of concurrent programs. We briefly considered the problem of determining whether or not a deadlock region is feasible. The problem of determining whether or not a set of states is *reachable* still remains; we will consider that in the next section. First, though, let us examine another, related question—that of determining which states lead inevitably to deadlock.

A state that has the property that it always leads to deadlock is called an *unsafe state*. An *unsafe region* is one from which no feasible trajectory can exit. An unsafe region always encloses an associated deadlock region; the extent of the deadlock region (i.e., the state at which no further progress can be made) is the same as the extent of the unsafe region.

Consider Figure 6. In this two-dimensional progress graph there exist two forbidden regions,  $A$  and  $B$ , whose spatial relationship creates deadlock region  $C$ . Using the technique developed previously, we can find  $C$  by computing the intersections of the nearness regions of  $A$  and  $B$ . But notice that, while region  $C$  is a deadlock region, there are states outside  $C$  that must ultimately lead to deadlock. These are the states contained in region  $D$ . Region  $D$  is said to be an *unsafe region*.

*Definition.* An *unsafe region* is a triple  $(D, V, E)$ , where  $D$  is a deadlock region, and  $V$  and  $E$  are the vertex and extent, respectively, of the region containing all points from which all trajectories lead to a point in  $D$ .

To find the coordinates of an unsafe region, we first need to define a new relation, called *coincidence*, between two hyperplanes. A hyperplane is coincident with a second hyperplane if it has the same location and direction as the second, and if its bounds are within the second's bounds. More formally,

*Definition.* A hyperplane defined by location  $x_i = \alpha^1$  and bounded by  $\forall_{j \neq i \in \{1 \dots N\}} : L_j^1 \leq x_j < U_j^1$  is coincident with a hyperplane defined by  $x_k = \alpha^2$  and  $\forall_{j \neq k \in \{1 \dots N\}} : L_j^2 \leq x_j < U_j^2$  if and only if  $i = k \wedge \alpha^1 = \alpha^2 \wedge \forall_{j \neq i \in \{1 \dots N\}} : L_j^2 \leq L_j^1 \wedge \forall_{j \neq i \in \{1 \dots N\}} : U_j^1 \leq U_j^2$ .

Note that coincidence is not a reflexive relation, but that it is transitive.

Another new property is called *proper containment*. A point is properly contained in a region  $(V, E)$  if it is strictly inside the region's vertex and extent bounds. This definition differs from that of containment, since a point on a vertex hyperplane of the region is contained within the region.

*Definition.* A point  $P(p_1, \dots, p_N)$  is *properly contained* in a region  $(V, E)$  if and only if

$$\forall_{i \in \{1 \dots N\}} : v_i < p_i < e_i. \quad (8.1)$$

Now, suppose that a point is properly contained in a feasible region  $(V, E)$ . Lemma 7 proves that, to exit the region, a trajectory must cross one of the region's extent hyperplanes.

**LEMMA 7.** *Given two points,  $P(p_1, \dots, p_N)$  and  $Q(q_1, \dots, q_N)$ , such that there exists a transition from  $P$  to  $Q$  that does not violate P5, and a region  $R(V, E)$  such that  $P$  is properly contained in  $R$  and  $Q$  is not, any trajectory from  $P$  to  $Q$  must cross one of  $R$ 's extent hyperplanes.*

**PROOF.** Since every trajectory from  $P$  to  $Q$  leaves  $R$ , every trajectory must cross at least one of  $R$ 's hyperplanes. From the definition of proper containment,  $\forall_{i \in \{1 \dots N\}} : v_i < p_i$ . Thus the state transition vector from  $P$  to any point  $X(x_1, \dots, x_N)$  on one of  $R$ 's vertex hyperplanes, say, the  $i$ th, contains the negative component  $x_i - p_i = v_i - p_i < 0$ , violating P5. So, by elimination, the trajectory must cross one of  $R$ 's extent hyperplanes.  $\square$

Lemma 7 suggests a strategy for finding the coordinates of unsafe regions. Note that each of the  $N$  extent hyperplanes of a total deadlock region is coincident with a vertex hyperplane of one of  $N$  forbidden regions. In a partial deadlock

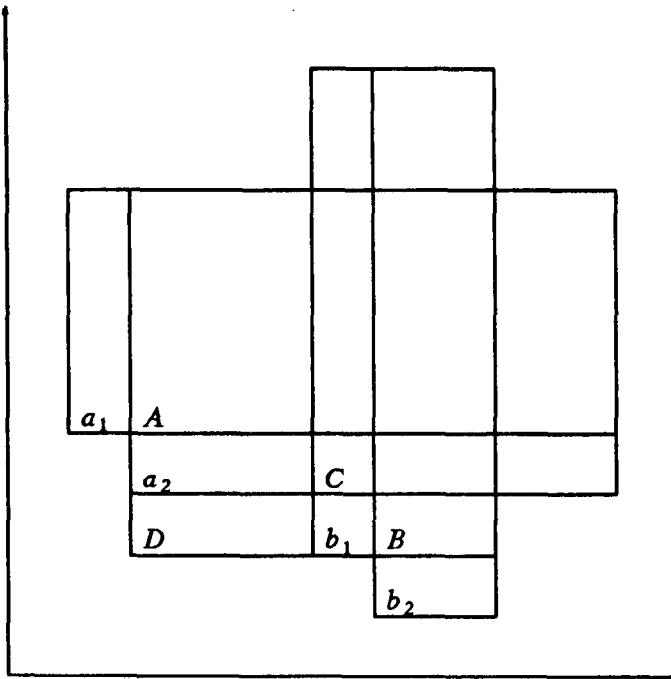


Fig. 6. Unsafe region. Here  $A$  and  $B$  are forbidden regions,  $a_1$ ,  $a_2$ ,  $b_1$ , and  $b_2$  are nearness regions,  $C$  is a deadlock region, and  $D$  is an unsafe region.

region, those extent hyperplanes whose associated directions are elements of the deadlock region's blocked set are coincident with the vertex hyperplanes of forbidden regions. Now, if the extent hyperplanes of a feasible region are coincident with vertex hyperplanes of forbidden regions, then no trajectory can exit the feasible region. To find the coordinates of an unsafe region, we find the coordinates of the largest region that has the same set of coincidence relations as a particular deadlock region.

Recall that a deadlock region is one in which some processes are blocked forever. We can think of an unsafe region as one in which some processes *will* block forever.

Suppose we have a deadlock region, with its associated blocked and invariant sets. The vertex of the associated unsafe region is formed from the vertex coordinates of the nearness regions of degree one that intersect to form the deadlock region.

Consider a single dimension, say, the  $i$ th, such that  $i$  is not a member of the invariant set of the deadlock region ( $i$  is thus a member of the deadlock region's blocked set). Recall that, in this case, there exists a forbidden region whose  $i$ th vertex hyperplane blocks the progress of the  $i$ th process at all points within the deadlock region. To find the  $i$ th vertex coordinate of the unsafe region, we must determine at what point in the  $i$ th process's progress it becomes inevitable that the hyperplane will block the  $i$ th process.

Keeping in mind that  $i$  is not a member of the deadlock region's invariant set, the point at which blocking due to the  $i$ th hyperplane becomes inevitable is the point at which all processes involved in the deadlock except the  $i$ th are blocked. This value is equal to the maximum  $i$ th vertex coordinate of the nearness regions of degree one, whose intersection is the deadlock region, not including the nearness region (of degree one) whose associated direction is  $i$ . If the deadlock region is of degree one, then deadlock is inevitable from the time the  $i$ th process is started, and the  $i$ th vertex coordinate of the unsafe region is zero.

The conditions for unsafeness are most easily envisioned if we consider a simple example. In Figure 6, nearness regions  $a_2$  and  $b_1$  intersect to form deadlock region  $C$ . Notice that  $C$ 's invariant set is  $\emptyset$ , and that  $C$ 's blocked set is  $\{1, 2\}$ ; both processes 1 and 2 are blocked within  $C$ . Now, let the  $i$ th vertex and extent coordinates of the nearness regions be  $v_i^{a_1}, e_i^{a_1}, v_i^{a_2}, e_i^{a_2}$ , and so on.

Using the above procedure, we first compute the first vertex coordinate of the unsafe region  $D$ ,  $v_1^D$ . The nearness region in which process 1 is blocked is  $b_1$ . Taking the maximum vertex coordinate of the remaining contributing nearness regions, namely,  $a_2$ ,  $v_1^D = v_1^{a_2}$ . Likewise, the nearness region in which process 2 is blocked is  $a_2$ . The (second) vertex coordinate of the remaining region,  $b_1$ , is the second vertex coordinate of  $D$ . Thus the vertex and extent of region  $D$  are  $(v_1^{a_2}, v_2^{b_1})$  and  $(e_1^b, e_2^a)$ , respectively.

The technique described applies if each direction is not a member of the deadlock region's invariant set. Now, suppose that we wish to find the  $i$ th vertex coordinate of an unsafe region, and that  $i$  is a member of the corresponding deadlock region's invariant set. In this case we must find the point in the  $i$ th process's progress at which the remaining processes become blocked. This is just the  $i$ th vertex coordinate of the deadlock region.

Now consider Figure 7. In this progress graph, the forbidden region  $A$  extends infinitely in the  $X_1$  direction. Thus the blocked and invariant sets for nearness region  $a_2$  are  $\{2\}$  and  $\{1\}$ , respectively. Notice that  $a_2$  is a deadlock region since the union of its blocked and invariant sets is the universal set.

To find the unsafe region corresponding to  $a_2$ , we first examine direction 1. Since 1 is a member of the invariant set, the first vertex coordinate of the unsafe region  $B$  is the first vertex coordinate of  $a_2$ , or  $v_1^{a_2}$ . Direction 2 is not a member of the invariant set, so we take the maximum vertex coordinate of all first degree nearness regions forming  $a_2$ , whose blocked set does not include direction 2. There are no such nearness regions, so  $B$ 's second vertex coordinate is zero. The extent of the unsafe region is the extent of the deadlock region,  $(e_1^{a_2}, e_2^{a_2})$ . It is easy to see that, if process 1 progresses to the event corresponding to  $v_1^{a_2}$  before process 2 progresses to the event corresponding to  $e_2^{a_2}$ , then it is inevitable that process 2 will deadlock.

Theorem 3 summarizes the above discussion, providing a single equation we can use to compute the coordinates of an unsafe region.

**THEOREM 3.** *Let  $D$  be a deadlock region, with invariant set  $I^D$ , blocked set  $B^D$ , and vertex and extent  $V^D$  and  $E^D$ . Further, let the nearness regions of degree one whose intersection is  $D$  be  $R^j(d^j, V^j, E^j)$ , for  $j \in B^D$ . If a point  $P(p_1, \dots, p_N)$  is*

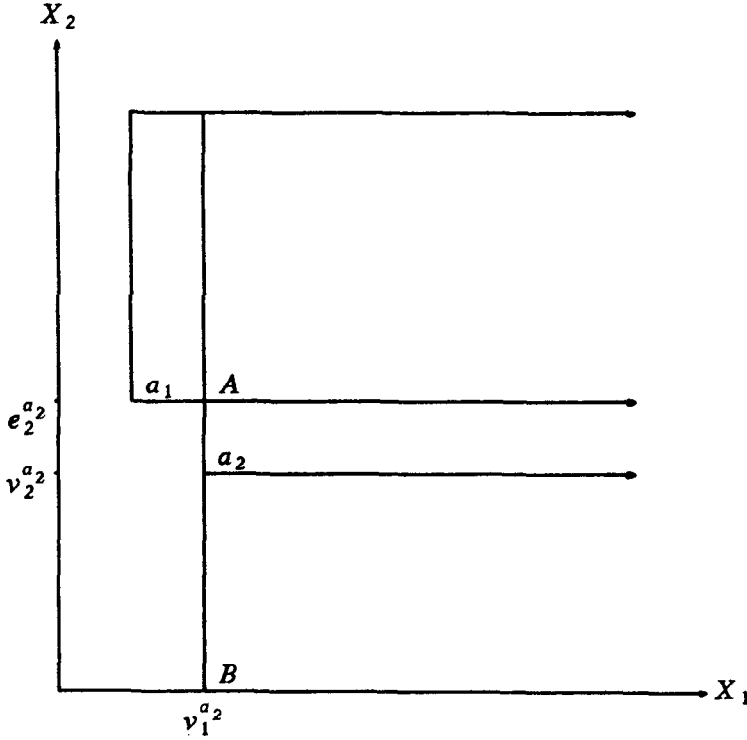


Fig. 7. Unsafe region with partial deadlock.

properly contained in a region defined by  $(V^U, E^U)$  such that

$$v_i^U = \begin{cases} 0 & B^D = \{i\} \\ \max(v_i^k, k \in B^D \wedge k \neq i) & B^D \neq \{i\} \end{cases} \quad (8.2)$$

and  $e_i^U = e_i^D$ , then  $P$  is an unsafe state. Therefore  $U(D, V^U, E^U)$  is an unsafe region.

**PROOF.** To prove this result we show that every extent hyperplane in  $U$  is either coincident with an extent hyperplane of a nearness region (and therefore coincident with a vertex hyperplane of a forbidden region), or infinitely distant from  $P$ .

The deadlock region  $D$  is composed of the intersection of nearness regions. If  $\zeta \in B^D$ , then there exists a nearness region whose  $\zeta$ th extent hyperplane is by definition coincident with the  $\zeta$ th vertex hyperplane of a forbidden region. The  $\zeta$ th extent hyperplane of the  $\zeta$ th nearness region is defined by

$$x_\zeta = e_\zeta^\zeta \wedge \forall_{i \neq \zeta \in \{1 \dots N\}} : v_i^\zeta \leq x_i < e_i^\zeta.$$

Now, the  $\zeta$ th extent hyperplane of  $U$  is

$$x_\zeta = e_\zeta^U \wedge \forall_{i \neq \zeta \in \{1 \dots N\}} : v_i^U \leq x_i < e_i^U.$$

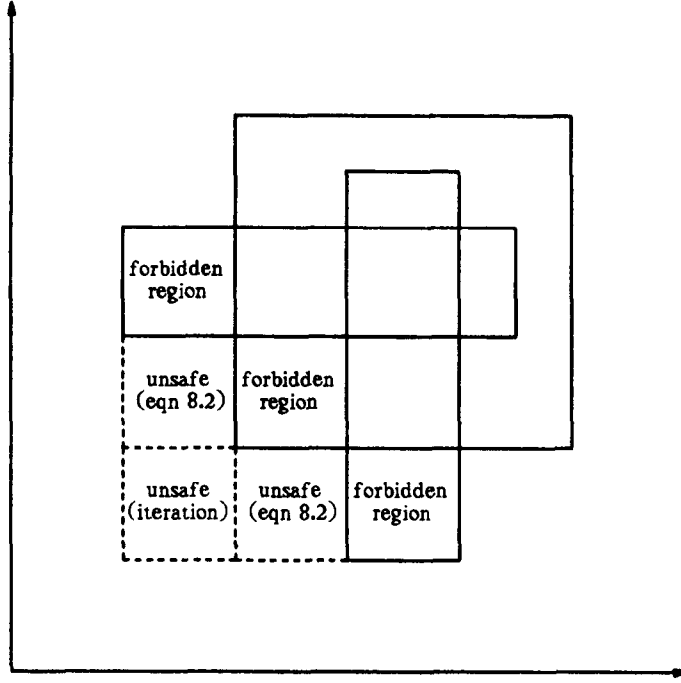


Fig. 8. Unsafe regions found by iteration.

But  $e_{\zeta}^U = e_{\zeta}^D = e_{\zeta}^{\xi}$ , and  $e_i^U = e_i^D$ , so we have

$$x_{\zeta} = e_{\zeta}^{\xi} \wedge \forall_{i \neq \zeta \in \{1 \dots N\}} : v_i^U \leq x_i < e_i^D.$$

We know that

$$\forall_{i \neq \zeta \in \{1 \dots N\}} : e_i^D \leq e_i^{\xi}$$

because the extent of the deadlock region is computed as the minimum  $e_i^{\xi}$  for all  $\zeta \in B^D$ . We also know that  $v_i^U \geq v_i^{\xi}$ , since  $v_i^U$  is computed in Eq. (8.2) as the maximum  $v_i^{\xi}$  for all  $\zeta \neq i$ . Therefore the  $\zeta$ th extent hyperplane of  $U$  is coincident with the  $\zeta$ th extent hyperplane of the  $\zeta$ th nearness region (of degree one) composing  $D$ , for all  $\zeta \in B^D$ .

The remaining extent hyperplanes of  $U$ , those directions  $\zeta$  such that  $\zeta \in I^D$ , are all infinitely far from  $P$ , thus no trajectory can cross them.

Thus every extent hyperplane of  $U$  is either coincident with the vertex hyperplane of a forbidden region or is infinitely distant, and no trajectory can exit  $U$ .  $\square$

Equation (8.2) describes the coordinates of regions in which all trajectories inevitably lead to a single deadlock. In the most general case, however, there may exist regions in which all trajectories lead to some deadlock, but not all trajectories lead to a single deadlock. In other words, at a point within such a region we can say that one of a number of deadlocks *will* occur. Eq. (8.2) is insufficiently powerful to characterize such regions.

To find these regions we use an iterative procedure. In the first iteration we find all unsafe regions using Eq. (8.2). We then find those regions all of whose extent hyperplanes are coincident with vertex hyperplanes of either unsafe regions or forbidden regions. To accomplish this we regard each unsafe region as a forbidden region and compute the new deadlock regions, starting anew. If we fail to find any deadlock regions during some iteration, then all unsafe regions have been discovered. Figure 8 shows unsafe regions found by iteration.

This procedure is clearly very expensive. However, two factors mitigate the cost. First, the conditions causing the need for multiple iterations are quite specialized and are thus rare. Second, this information is useful in that, once we have found all unsafe regions, we can make *repairs* to the concurrent system under analysis by inserting appropriate synchronization primitives. If we make every unsafe region infeasible (i.e., insert the primitives that create a forbidden region), then the concurrent system is deadlock free. Although we will not explore repair procedures any further in this paper, we will consider a similar problem in the next section.

## 9. REACHABILITY

Forbidden regions define sets of points that are infeasible because they violate the constraints imposed by the synchronizing primitives  $P$  and  $V$ . In general, though, there may exist points that satisfy the constraints of the synchronizing primitives, but that are not part of any valid execution sequence of the set of concurrent processes under analysis. These points, aptly called *unreachable* points, are unreachable because of certain spatial relationships among forbidden regions. As Clarke points out, the incompleteness of the semaphore invariant method is entirely due to the possibility that unreachable points might exist.

In this section we show how unreachability reduces to unsafeness if a simple linear transformation is applied to the progress graph. We then derive the transformation and its inverse.

Recall the fifth postulate:

P5: Processes cannot progress backward in time; that is, no (scalar) component of a state transition vector can be negative.

Now, suppose that there exists a trajectory from the origin to a point  $P(p_1, \dots, p_N)$ , such that no state transition in the trajectory violates P5. Then we can also construct a trajectory from  $P$  to the origin using the “inverse” of P5:

P5': Processes cannot progress forward in time; that is, no (scalar) component of a state transition vector can be positive.

Of course, P5' has no meaning in a real system. We use it here merely as a mathematical device.

The trajectory from  $P$  to the origin is composed of the additive inverses of the vectors comprising the trajectory from the origin to  $P$ . It is easy to see that, if it is possible to construct a trajectory from the origin to  $P$  using P5, then it is possible to construct a trajectory from  $P$  to the origin using P5'. More important, if it is not possible to construct a trajectory from  $P$  to the origin using P5', then it is not possible to construct a trajectory from the origin to  $P$  using P5. And, if

it is not possible to construct a trajectory from the origin to  $P$  using P5, then  $P$  is unreachable. In other words, we can legitimately attempt to verify reachability by working backward from  $P$ .

**LEMMA 8.** *Between the origin and a point  $P(p_1, \dots, p_N)$ , there exists a trajectory based on P5 if and only if there exists a trajectory from  $P(p_1, \dots, p_N)$  to the origin based on P5'.*

**PROOF.**

**IF.** Let  $T^1$  through  $T^k$  be a series of vectors that compose the trajectory between  $P$  and the origin. Laying the vectors end-to-end, we have  $\bar{0} = P + \sum_{i=1}^k T^i$ . Further, let the components of each  $T^j$  subscribe to P5'; that is,  $\forall_{i \in \{1 \dots N\}} : t_i^j \leq 0$ . Let  $S^1$  through  $S^k$  be a set of vectors such that  $s_i^j = -t_i^j$ . Then  $\forall_{i \in \{1 \dots N\}} : s_i^j \geq 0$ . Also, we have  $\bar{0} + \sum_{i=1}^k S^i = \bar{0} + P = P$ , so the trajectory composed of the  $S^i$  extends from the origin to  $P$ .

**ONLY IF.** Identically, let  $T^i$  be a series of vectors that subscribe to P5; that is,  $\forall_{i \in \{1 \dots N\}} : t_i^j \geq 0$ . We form a series of vectors  $S^j$  such that  $s_i^j = -t_i^j$ . Thus,  $\forall_{i \in \{1 \dots N\}} : s_i^j \leq 0$  and the trajectory composed of the  $S^j$  satisfies P5'.  $\square$

Unsafe regions are regions from which no trajectory subscribing to P5 can exit. We can think of an unreachable region as one from which no trajectory subscribing to P5' can exit. The simplicity of the relationship between trajectories constructed with P5 and those constructed with P5' suggests that we can use techniques previously derived from P5, suitably transformed, to find unreachable regions.

Suppose we have a collection of forbidden regions. If we reflect the regions through the origin, then the vertex and extent coordinates of each region are exchanged. Recalling that there exists a finite value that can be used as "effective infinity," we then translate the coordinate axes so that the (reflected) point  $(-\infty, -\infty, \dots, -\infty)$  becomes the origin. This new progress graph has the property that any trajectory in it that is constructed with P5 is the same as a trajectory in the original graph constructed with P5'. Likewise, any regions that are unsafe in the transformed progress graph are unreachable in the original progress graph. We can apply the techniques previously developed (based on P5) for finding unsafe regions, then reverse the transformation. The resulting regions are unreachable in the original progress graph.

For example, consider Figure 9. The first progress graph, (a), is one derived from a two-process PV program. The second progress graph, (b), is the graph obtained by reflecting and translating (a). Note that there exists an unsafe region,  $U$ , in this graph. We find its coordinates using the techniques described above. We then reverse the transformation, obtaining graph (c). Region  $U$  is unreachable in this graph.

The required transformation is simple. To reflect each forbidden region through the origin, we merely negate the coordinates of each vertex and extent. We then translate the forbidden regions back into the first "quadrant." Let there be a point  $\Phi(\phi_1, \phi_2, \dots, \phi_N)$  such that  $\phi_i$  is "effective infinity" for the  $i$ th coordinate axis. The combined reflection and translation is

$$x'_i = \phi_i - x_i. \quad (9.1)$$

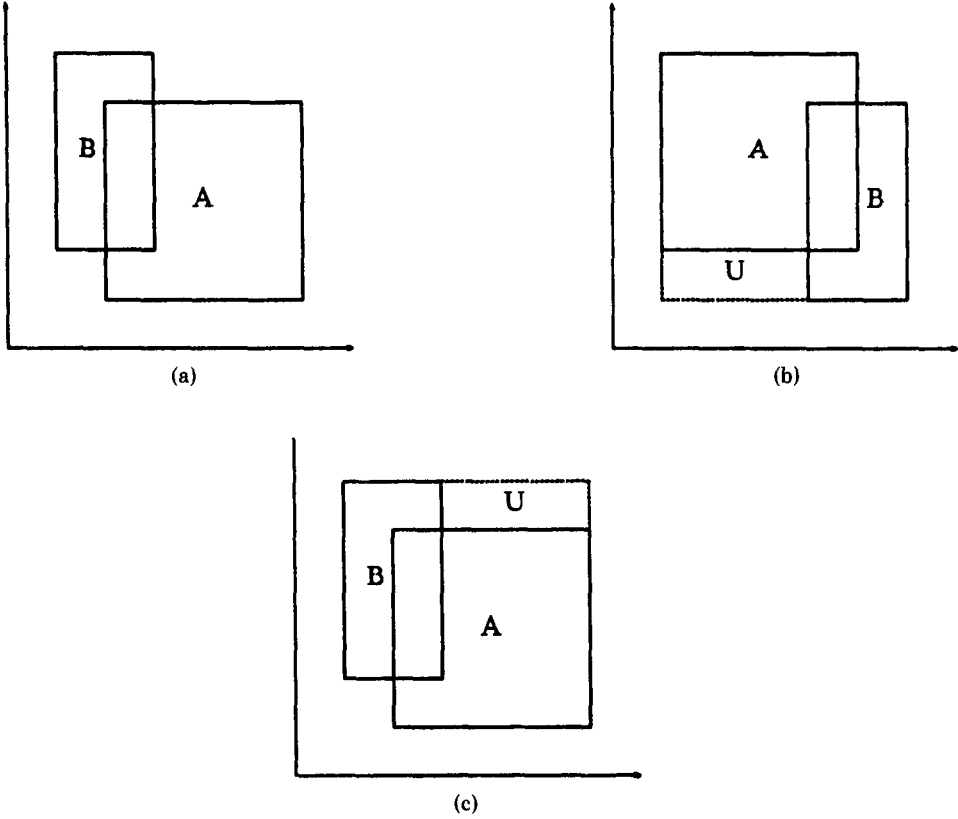


Fig. 9. Transforming to find unreachable regions. (a) Original progress graph. (b) Transformed progress graph.  $U$  is an unsafe region. (c) Original progress graph.  $U$  becomes an unreachable region.

Once this transformation has been applied, the vertex of each region is further from the origin than the extent. Thus, to complete the transformation we must swap the vertex and extent of each forbidden region.

This transformation is its own inverse. Thus,

$$\begin{aligned}
 x_i'' &= \phi_i - x_i' \\
 &= \phi_i - (\phi_i - x_i) \\
 &= x_i.
 \end{aligned} \tag{9.2}$$

Of course, we must again swap the vertex and extent of each forbidden region.

**THEOREM 4.** *Let  $P'$  be a point that is contained in an unsafe region in a transformed progress graph  $G'$ . Then the point  $P$ , transformed from  $P'$  by Equation (9.2), is unreachable in the original progress graph  $G$ .*

**PROOF.** By definition, “effective infinity” is unreachable from a point contained in an unsafe region. Thus, there exists no feasible trajectory between  $P'$  and  $\Phi$ . Now, applying the inverse transformation, the two points  $P'$  and  $\Phi$  in  $G'$

become  $P$  and  $\bar{0}$  in  $G$ . Since there is no feasible trajectory between  $P'$  and  $\Phi$ , there is no feasible trajectory between  $\bar{0}$ , the origin, and  $P$ , and by definition,  $P$  is unreachable in  $G$ .  $\square$

The above result proves that the geometric model correctly represents systems with unreachable feasible states.

Now, it is admittedly expensive to effectively perform deadlock detection twice, once to find unreachable regions and once to find deadlocks. However, if we find unreachable regions first, then most of the space required to represent the nearness regions can be reused during deadlock detection.

## 10. CONCLUSIONS

The geometric model presented in this paper formalizes the familiar notion of a progress graph, a multidimensional Cartesian model of synchronization between two or more processes. The model is sufficiently powerful to characterize state-intrinsic properties, such as blocking and total deadlock, as well as state-extrinsic properties, such as reachability and partial deadlock. Additionally, the model is one that is readily analyzed to find such properties. The techniques described in this paper have been implemented, requiring less than 200 lines of Pascal code.

The geometric model is exact in that it can be used to identify all *reachable* deadlock states. It is more powerful than the algebraic deadlock predicate, in that it has the ability to describe properties that hold over sequences of states. It has an advantage over graph-based approaches in that entire sets of states with like properties are represented as single entities, known as forbidden regions. The Cartesian interpretation provides the same information given by arcs in a directed graph, without requiring an explicit representation of each state transition or path.

We have described a compact representation for sets of states that share a spatial relationship: forbidden regions. This compact representation uses coordinates for defining such regions. The number of coordinates required is linearly related to the number of synchronization states (bounded by the number of processes being modeled) required to represent each set.

Although we have defined a formalism for the geometric model of concurrency, and for identifying regions and therefore synchronization properties of interest, we have not discussed methods for creating such geometric representations of concurrent programs. We have investigated this problem in [1] and have found that the geometric model lends itself to simplified analysis for certain classes of programs. For example, for *PV* programs in which semaphores are used only for mutual exclusion, the number of forbidden regions grows polynomially with the number of processes. This, in turn, simplifies the complexity of deadlock analysis.

The geometric model has the distinct advantage of capturing the spatial and thus temporal relationships among a set of cooperating processes. Using the analysis techniques we have described in this paper, it is possible to use these characteristics to identify, for example, points at which an appropriate mutual exclusion semaphore could be placed in order to avoid an otherwise inevitable

deadlock. The geometric model could be used, as well, to support program transformations and analysis of such transformations.

The geometric model is also easily extended to capture the behavior of broader classes of programs, such as joint  $P$  and  $V$  operations and Clarke's linear conditional critical regions. We are currently exploring these potential applications.

#### APPENDIX. An Example

The following example has been taken from [8]; the authors (Lipski and Papadimitriou) use it to illustrate the failure of their method when it is applied to three-process systems. In contrast, the methods we have described properly identify the three example processes as deadlock free.

In this example three processes,  $P_1$ ,  $P_2$ , and  $P_3$ , communicate via six semaphores,  $u$ ,  $v$ ,  $w$ ,  $x$ ,  $y$ , and  $z$ , as shown:

```
cobegin
  P1:  P(x); P(y); P(z); V(x); P(w); V(z); V(y); V(w)
      //
  P2:  P(u); P(v); P(x); V(u); P(z); V(v); V(x); V(z)
      //
  P3:  P(y); P(w); V(y); P(u); V(w); P(v); V(u); V(v)
end
```

A two-dimensional projection of the three-dimensional progress graph for  $P_1$ ,  $P_2$ , and  $P_3$  [8] appears to show that deadlock is possible. Our method proves otherwise. Following the techniques described in this paper and procedures described in [1], we first determine the coordinates of the forbidden regions (Table I).

Using this information we seek to identify one or more nearness regions of degree one, two, or three in which one or more cases of partial deadlock occur, or a nearness region of degree three in which total deadlock occurs. Therefore, our next step is to compute the coordinates of the associated nearness regions of degree one, as described in Section 6 (Table II).

Since no entry in the preceding table has  $\{1, 2, 3\}$  as the union of its blocked and invariant sets, we have no instances of partial deadlock at this stage of the analysis.

Next, we compute the (nonempty) intersections of these nearness regions, producing nearness regions of degree two. These represent regions in which two processes are blocked. The blocked and invariant sets are computed as in Section 7. For example, the first entry in the following table is formed from the second and third entries in the preceding table. The blocked set for this entry is the union of the blocked sets for the two regions whose intersection forms the new region of degree two. The remaining entries (Table III) are formed in a similar manner.

Once again we have no instances of partial deadlock: No entry in the preceding table has  $\{1, 2, 3\}$  as the union of its blocked and invariant sets.

Now we seek to form at least one nearness region of degree three such that its blocked set is  $\{1, 2, 3\}$  or such that the union of its blocked and invariant sets is

Table I.

Forbidden regions		
Vertex	Extent	Semaphore
(0, 1, 4)	( $\infty$ , 4, 7)	u
(0, 2, 6)	( $\infty$ , 6, 8)	v
(5, 0, 2)	(8, $\infty$ , 5)	w
(1, 3, 0)	(4, 7, $\infty$ )	x
(2, 0, 1)	(7, $\infty$ , 3)	y
(3, 5, 0)	(6, 8, $\infty$ )	z

Table II.

Nearness regions			
Vertex	Extent	Blocked set	Invariant set
(0, 3, 0)	(1, 7, $\infty$ )	{1}	{3}
(1, 2, 0)	(4, 3, $\infty$ )	{2}	{3}
(1, 0, 1)	(2, $\infty$ , 3)	{1}	{2}
(2, 0, 0)	(7, $\infty$ , 1)	{3}	{2}
(2, 5, 0)	(3, 8, $\infty$ )	{1}	{3}
(3, 4, 0)	(6, 5, $\infty$ )	{2}	{3}
(4, 0, 2)	(5, $\infty$ , 5)	{1}	{2}
(5, 0, 1)	(8, $\infty$ , 2)	{3}	{2}
(0, 0, 4)	( $\infty$ , 1, 7)	{2}	{1}
(0, 1, 3)	( $\infty$ , 4, 4)	{3}	{1}
(0, 1, 6)	( $\infty$ , 2, 8)	{2}	{1}
(0, 2, 5)	( $\infty$ , 6, 6)	{3}	{1}

Table III.

Nearness regions of degree 2			
Vertex	Extent	Blocked set	Invariant set
(1, 2, 1)	(2, 3, 3)	{1, 2}	$\emptyset$
(2, 2, 0)	(4, 3, 1)	{2, 3}	$\emptyset$
(2, 5, 0)	(3, 8, 1)	{1, 3}	$\emptyset$
(3, 4, 0)	(6, 5, 1)	{2, 3}	$\emptyset$
(4, 4, 2)	(5, 5, 5)	{1, 2}	$\emptyset$
(5, 4, 1)	(6, 5, 2)	{2, 3}	$\emptyset$
(4, 0, 4)	(5, 1, 5)	{1, 2}	$\emptyset$
(0, 3, 3)	(1, 4, 4)	{1, 3}	$\emptyset$
(1, 2, 3)	(4, 3, 4)	{2, 3}	$\emptyset$
(4, 1, 3)	(5, 4, 4)	{1, 3}	$\emptyset$
(0, 3, 5)	(1, 6, 6)	{1, 3}	$\emptyset$
(1, 2, 5)	(4, 3, 6)	{2, 3}	$\emptyset$
(2, 5, 5)	(3, 6, 6)	{1, 3}	$\emptyset$
(3, 4, 5)	(6, 5, 6)	{2, 3}	$\emptyset$

{1, 2, 3} (indicating partial deadlock). However, there is no pair of regions in the preceding table that forms a region of degree three that, in turn, satisfies these criteria. Therefore, the system is deadlock free.

We note that we have determined deadlock freedom in this case without having to perform feasibility or reachability analyses. However, other examples would not necessarily afford this luxury.

## REFERENCES

1. CARSON, S. D. Geometric models of concurrent programs. Ph.D. dissertation, Dept. of Computer Science, Univ. of Virginia, Charlottesville, 1984.
2. CLARKE, E. M. Synthesis of resource invariants for concurrent programs. *ACM Trans. Program. Lang. Syst.* 2, 3 (July 1980), 338-358.

3. COFFMAN, E. G., ELPHICK, M. J., AND SHOSHANI, A. System deadlocks. *ACM Comput. Surv.* 3, 2 (June 1971), 67-78.
4. DIJKSTRA, E. W. Co-operating sequential processes. In *Programming Languages*, F. Genuys, Ed. Academic Press, New York, 1968, 43-110.
5. HABERMANN, A. N. Path expressions. Tech. Rep., Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., June 1975.
6. HOLT, R. C. Some deadlock properties of computer systems. *ACM Comput. Surv.* 4, 3 (Sept. 1972), 179-196.
7. KELLER, R. M. Generalized Petri nets as models for system verification. Tech. Rep., Computer Science Dept., Univ. of Utah, Salt Lake City, 1977.
8. LIPSKI, W., AND PAPADIMITRIOU, C. H. A fast algorithm for testing for safety and detecting deadlocks in locked transaction systems. *J. Alg.* 2, 3 (Sept. 1981), 211-226.
9. OWICKI, S., AND GRIES, D. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM* 19, 5 (May 1976), 279-285.
10. PAPADIMITRIOU, C. H. Concurrency control by locking. *SIAM J. Comput.* 12, 2 (May 1983), 215-226.
11. YANNAKAKIS, M., PAPADIMITRIOU, C. H., AND KUNG, H. T. Locking policies: Safety and freedom from deadlock. In *Proceedings of the 20th FOCS* (San Juan, P.R., Oct. 29-31, 1979). IEEE, New York, 1979, 286-297.

Received April 1985; revised March 1986; accepted April 1986