Stability Analysis of Concurrent Systems with an Indefinite Number of Processes

Mahesh Girkar *Robert Moll*
Department Of Computer Science
University of Massachussetts
Amherst MA 01003

COINS Technical Report 92-72
November 13, 1992

Abstract

In a resource-oriented model for concurrent systems ([5, 22]) processes request resources which are controlled by finite state devices called synchronizers. The edges of a synchronizer $\mathcal{S}$ are labeled with symbols from an alphabet of operations on resources (denoted by $\sum$). When a process executes an operation, it causes a state change in $\mathcal{S}$. A program P for a process specifies the order in which it executes operations. Thus programs are strings over $\sum$. Concurrency among processes is modeled as an interleaving of their corresponding program strings. We denote by $(P^k, \mathcal{S})$ a concurrent system with k identical processes with program P and a synchronizer $\mathcal{S}$, and by $(P, \mathcal{S})$ a concurrent system with an indefinite number of identical processes. $(P^k, \mathcal{S})$ is *weak safe* if all interleavings of requests for resources made by the processes can be satisfied by $\mathcal{S}$; it is *strong safe* if for all partial executions of the processes, $\mathcal{S}$ can satisfy some outstanding request for a resource by some process. $(P, \mathcal{S})$ is *weak stable* (*strong stable*) *iff* there exists $N > 0$ such that $(P^k, \mathcal{S})$ is weak safe (strong safe) for all $k \geq N$. In this report, we formulate questions regarding weak and strong stability and prove their decidability by showing an effective procedure to determine if there exists $N > 0$ such that $(P^k, \mathcal{S})$ is weak safe (strong safe) for all $k \geq N$. As a corollary of this result, we prove the following *full weak (strong) safety* theorem: It is decidable if $(P^k, \mathcal{S})$ is weak safe (strong safe) for all $k > 0$. Finally we show that our machinery is powerful enough to model the standard gas-station customer problem.

1 Introduction

Most formal models for concurrent systems provide a means for describing, and ultimately, detecting such phenomenon as deadlock, starvation and liveness. In a resource oriented model, for example, such as that presented in [5, 22], processes are entities executing operations which request and release resources. These resources are controlled by one or many finite state devices called synchronizers. In such a model, properties such as deadlock and liveness can be expressed naturally using temporal logic. In general, a proof that establishes that the model possesses desirable properties depends on the model having a fixed number of processes. Even when the number of processes increase in a "uniform" way, the task of proving that the model still possesses desirable properties has to be performed all over again. In this report, we present a model of concurrency that will allow us to establish that proofs "scale" upwards. More formally, we present a simple model of concurrency, and we formulate the concepts of weak and strong stability for this model. Intuitively, if a model instance exhibits strong stability, then proofs regarding it are "scalable". Thus, we show that the following question is decidable: Does there exist a bound k such that models of size k or greater never deadlock. If such a k exists, we also show how to obtain it.

In Section 2, we discuss previous research related to our problem. Section 3 describes our notation. In Section 4 we define weak stability and give an effective way to decide it for concurrent systems, based on bounds on chain length for sequences of vectors in non-negative lattice points in n-space (see [7]). Section 5 discusses strong stability. In that section, we formulate for our model the problem of strong stability, and show its relationship with infinite-R-reachability of vector classes in a vector addition system with states (VASS). Section 6 sketches the proof for the decidability of reachability for a VASS (see Kosaraju [18]). Subsequently, in Section 7, we show how to modify the proof in [18] to decide infinite-R-reachability for vector classes in a VASS. Using this modification, in Section 7.2, we give an effective procedure for deciding strong stability. In Section 8, we show that our machinery is powerful enough to model one concurrent problem — The Gas-Station Customers Problem. Finally, in Section 9, we summarize our results.

2 Literature Survey

2.1 Modeling Concurrent Systems

Much research in concurrent systems has focussed on formal specification methods for analyzing important properties of such systems. In general, a concurrent system consists of several processes which execute operations. These operations result in inter-process communication (IPC). Several models for concurrent systems have been proposed in literature. These models differ from each other with respect to such basic features as whether IPC is obtained by message passing or by shared memory, whether resource control is centralized or distributed, and whether IPC is synchronous or asynchronous. In most models, the notion of "interleaving" of the operations executed, is used to describe concurrency.

One such model, and an approach for automatic synthesis of IPC code for the model, is presented by Ramamritham [5].In his model, processes contained in a concurrent program interact by accessing and modifying the state of shared resources. Resources are controlled by a finite state device called a synchronizer. The interleavings that can result when operations are executed by the processes may lead to such phenomenon as deadlock or failure of liveness for some operation. A desirable property that needs to be satisfied for the model is specified by a temporal logic formula. Using the specification of the processes, the synchronizer, and the property that needs to be satisfied, Ramamritham discusses automatic synthesis of IPC code for the model. Buy [21] and Buy and Moll [22] extend the above idea to involve multiple synchronizers. Processes can obtain resources from synchronizers or release them. Synchronizers too can cause invocation of operations in other synchronizers if they need a resource which is not controlled by them. In the current technical report we formulate stability questions regarding a subset of this model.

Concurrent systems have also been modeled using Petri Nets. For a general review of Petri Nets and its applications, see the paper by Tadoa Murata [20] and the book by Peterson [12].

2.2 Model Checking and Use Of Partial Orders

Analysis of desirable properties of concurrent systems normally consists of the following steps: Given a formal specification of a concurrent system, a finite state representation of the specification is constructed. Then, the property to be analyzed is expressed as a temporal logic formula. Finally, the temporal logic formula is verified.

Several algorithms have been proposed for checking if a finite-state concurrent system possesses a desirable property specified in temporal logic. Clarke, Emerson, and Sistla [2] devised an algorithm linear in the size of the global state graph determined by the concurrent system to verify any temporal logic formula regarding the system. Valmari [1] introduced a method which facilitates the generation of reduced state spaces, such that the truth values of a collection of linear temporal logic formulas are the same in the ordinary and reduced state spaces.

More recently, in [10], Godefroid shows that most of the state explosion due to the modeling of concurrency by interleaving can be avoided. The author builds a state graph where only one (rather than all) interleaving of execution of concurrent events is represented. In [11], the authors improve on Godefroid's algorithm for the problem of deadlock detection. They are able to generate a global representation of the concurrent program that chooses among independent enabled transitions in a completely arbitrary way. This simplification leads to an algorithm for deadlock detection that is easy to implement and more efficient than in [10]. They also show how to verify general safety properties regarding the model.

2.3 Vector Addition System With States

Vector Addition Systems with States (VASS) is a model closely related to Petri Nets. A VASS is essentially a finite state automaton. Its arcs are labeled with n-dimensional ($n \geq 1$) vectors whose coordinates have integer values. A configuration of a VASS is a pair (s, v), where s is a state of the VASS and v is an n-dimensional vector. Starting with an initial configuration (s_0, v_0), where s_0 is any state and v_0 is any initial vector, a path in the VASS induces a sequence of configurations where the vector in a configuration is obtained by componentwise vector addition of the vector in the previous configuration and the vector along the arc traveled. An R-path is a path for which all vectors along the path (including the initial vector) lie in the positive orthant. The reachability problem for a VASS is as follows: Given an initial configuration (s_0, v_0) and a final configuration (s_f, v_f) does there exist an R-path from (s_0, v_0) to (s_f, v_f).

Kosaraju [18], based on work in Sacerdote and Tenney [19] and Mayr [8], proves that the reachability problem for a VASS is decidable. Some previous related results are presented by Ginzburg and Yoeli [4]. In that paper, the authors prove that reachability is decidable for a VASS which defines a regular language. A similar result is proved for Petri Nets by Rudiger Valk and Guy Vidal-Naquet in [15]. Roshier and Yen [6] show a bound for the problem of determining the size of the reachability set from a certain initial configuration. In Section 5, we establish our result on strong stability by generalizing Kosaraju's result to a VASS reachability problem for certain special classes of vectors.

2.4 Reasoning With Infinitely Many Processes

So far we have discussed research that considers concurrent systems of fixed size, *i.e.* systems with fixed number of processes. We now discuss work that considers concurrent systems of increasing size.

Kurshan [13] shows that regularity of systems with identical processes may facilitate the construction of a homomorphic reduction in such systems to a constant size state space analysis which is independent of the number of processes. The construction of the above reduction is *ad hoc*. Clarke, Browne and Grumberg [3] introduce a method which uses an indexed form of branching time temporal logic for specifications. In their method, one needs to establish a bisimulation equivalence between global state graphs of systems of different sizes. This procedure is again *ad hoc*. In [14], Krushan and McMillan use induction for proving correctness of these systems. The authors construct an invariant process I from the given user processes. I has the following characteristic: The correctness property holds for I, and it also holds for any user process composed with I. The authors then conclude that the correctness property holds for any number of user processes. Constructing I is again a *ad hoc* procedure.

German and Sistla ([16, 17]) consider a model which consists of a control process and several identical user processes. The processes communicate through synchronous actions in the style of Milner's Calculus of Communicating Systems [9]. Using the machinery introduced by Milner [9], the authors describe two kinds of operations: those which cause inter-process communication and those which cause internal transitions within a user process and which can be executed independently of other user processes. The authors use linear time temporal logic for specifying desirable properties of the concurrent system. The execution of the user processes and the control processes is simulated by constructing a VASS. The following basic question is answered: Does there exist an n, such that for a system (C, U^n) (C denotes the control process and U^n denotes n user processes), there exists an infinite fair execution of the processes for which the temporal specification is not satisfied. Deadlock being a global property of the processes cannot be expressed in linear-time temporal logic. The authors address the question of detecting a potential deadlock in the system and show that the following question is decidable: Does there exist an n such that the system (C, U^n) deadlocks. However, they do not address the case of detecting whether there are infinitely many instances of deadlock.

Strong stability is a property which essentially deals with the problem of detecting infinite instances of deadlock – a completely different problem from detecting a single instance of deadlock. In general, if a model is strong stable, then it has only finitely many deadlock situations. Of course, the concept of strong stability need not be associated with deadlock. It can be associated with any global property of the concurrency model. In this report, we concern ourselves with answering the following question for a model similar to the model presented in [5, 21, 22]: Do there exists infinitely many instances of n such that for all these instances, deadlock occurs in the synchronizer process. We show the decidability of this question directly by showing the decidability of infinite-R-reachability of vector classes in a VASS (see Section 5). The proof is based on an extension of the proof presented in [18].

3 Preliminary Definition and Results

We rely on traditional regular language notation. For a string w over an alphabet $\sum$, $|w|$ denotes its length and $prefix(w)$ denotes its set of prefixes. If w is a non-empty string, its i^{th} symbol ($1 \leq i \leq |w|$) is denoted by $w[i]$ and its i^{th} prefix (denoted by w^i) is the prefix string $w[1]w[2]\ldots w[i]$. String w_1 is a "substring" of w_2 *iff* $|w_1| \leq |w_2|$ and there exists positive integers $p_1 < p_2 <$

$\ldots < p_{|w_1|}$ such that $w_1 = w_2[p_1]w_2[p_2]\ldots w_2[p_{|w_1|}]$. Let $S = \{p_1, p_2, \ldots, p_i\}$ be a subset of $\{1, 2, \ldots, |w|\}$ with $p_1 < p_2 < \ldots < p_i$. We write $w[S]$ for the string $w[p_1]w[p_2]\ldots w[p_i]$. (Note: If S is empty, then $w[S]$ is the empty string.) Let $w_1, w_2, \ldots, w_k$ be k strings. The string w is an "interleaving" of these k strings *iff*

1. $|w| = |w_1| + |w_2| + |w_3| + \ldots + |w_k|$; and
2. $S_1, S_2, \ldots, S_k$ partition $\{1, 2, \ldots, |w|\}$, such that $w_i = w[S_i]$, $1 \leq i \leq k$. We allow a partition to be empty.

We now describe our initial model for concurrency: Alphabet $\sum$ specifies the operations that can be performed on resources in the system. A Synchronizer $\mathcal{S}$ is an FSA $G = (V, E)$. It has a designated start state q_0, a labeling function T which assigns to each edge in $\mathcal{S}$ a symbol from $\sum$, and a transition function $\delta\,(V, \sum^*) \rightarrow V$. All explicit states of $\mathcal{S}$ are accepting states. $L(\mathcal{S})$ represents the language accepted by $\mathcal{S}$. In general, however, "dead" states are present implicitly. For example, when we discuss weak stability (see Section 4), any string over $\sum$ which represents an interleaving of operations and for which there does not exist a path inside $\mathcal{S}$ (or in other words, the string does not belong to $L(\mathcal{S})$) would lead to the dead state. (Note: The term "path" used above represents any sequence of states starting with q_0 and derived using δ.)

A "process" is an entity which executes operations. These operations request and release resources controlled by the synchronizer. The order in which a process executes its operations is specified by its "program", which is a string over $\sum$. We initially restrict all processes to have the same program. Let P be a program with string $a_1a_2\ldots a_l$, $a_i \in \sum$ (l is the length of the program for P). We also assume that the a_i's are distinct. (Thus $l \leq |\sum|$.)

Definition 1 *Let I_k be the set of interleavings of k processes, each of whose program is P. Thus, $w \in I_k$ if and only if w is an interleaving formed from k strings each of which is $a_1a_2\ldots a_l$. Let $I = \bigcup_{i=1}^{\infty} I_i$. We refer to I as the "Interleaving Language" for P.*

Below we define a many-to-one mapping from a string in prefix(I) to a vector in $l-1$ dimensions. Then, we define a one-to-one mapping from a string in prefix(I) to a sequence of vectors. Definition 2 is fundamental because it shows how to represent prefixes of interleavings as tuples in non-negative n-space. In Section 4, these vector sequences are analyzed in order to decide weak stability.

Definition 2 *Let program $P = a_1a_2\ldots a_l$. For any string $w \in prefix(I)$, let $vec(w) = (n_{a_1}, n_{a_2}, \ldots, n_{a_{l-1}})$ represent the vector in $l-1$ dimensions where n_{a_i} is the number of occurrences of a_i minus the number of occurrences of a_{i+1} in w. Let vec_seq(w) be the sequence of $(l-1)$-dimensional vectors*

$$\overline{0}, vec(w^1), vec(w^2), \ldots vec(w^{|w|}(= w))$$

where $\overline{0}$ is the vector with all coordinates zero.

Since each symbol in program $P = a_1a_2\ldots a_l$ is distinct, and, since for any string $w \in \text{prefix}(I)$, the number of occurrences of a_{i+1} are not more than the number of occurrences of a_i $(1 \leq i \leq l-1)$, vec(w) has non-negative integer coordinates. Further, since w^j, the jth prefix of w, also belongs to prefix(I), every vector in vec_seq(w) has non-negative integer coordinates.

As an example, if $\sum = \{(,)\}$, and the program P for each process is (), then $I_1 = \{()\}$, $I_2 = \{()(),(())\}$, etc. In this case, I is the language of matched parentheses. If $\sum = \{a,b,c\}$, the program P for all the processes is abc, and $w = aabcb$ ($w \in \text{prefix}(I)$), then w^2 (the second prefix of w) is aa, $w^4 = aabc$ (the fourth prefix of w), vec_seq(w) $= (0,0)(1,0)(2,0)(1,1)(1,0)$ and vec(w) $= (1,0)$. In this example, the poset is two-dimensional because P has three symbols.

With $|P| = 1$, I is regular. If $|P| = 2$, I becomes context-free. For $|P| > 2$, I is non-context-free.

4 Weak Stability

We begin with a few definitions.

Definition 3 *We denote by (P^k, S) a finite concurrent system with synchronizer S and k identical processes each of whose program is $P = a_1a_2\ldots a_l$. We denote by (P, S) a concurrent system with an indefinite number of processes, each with program P.*

Definition 4 (P^k, S) *is* weak safe iff $I_k \subseteq L(S)$.

Definition 5 (P, S) *is* weak stable iff $\exists k > 0$, *such that* $\forall j > k$, (P^j, S) *is weak safe.*

Definition 6 (P, S) *is* full weak safe iff $\forall j > 0$, (P^j, S) *is weak safe.*

Proposition 1 (P, S) *is weak stable* iff $I \subseteq L(S)$.

Proof: If $I \subseteq L(S)$, then obviously (P, S) is weak stable. To prove the result in the other direction, we proceed by contradiction. Assume (P, S) weak stable but $I \not\subseteq L(S)$. Then there exists $k > 0$ such that for all $j > k$, $I_j \subseteq L(S)$. Also, there exists $w \in I$ such that $w \notin L(S)$. Let w be an interleaving formed from m instances of program P. Consider the following set of interleavings: $\{wP, wPP, wPPP, \ldots\}$. Each of these belongs to I but does not belong to $L(S)$. Since they are formed from $m+1, m+2, \ldots, m+i, m+i+1, \ldots$ $(i > 0)$ processes respectively, this contradicts weak stability. $\Box$.

Several examples are presented in Figure 1. The process with program string "12" is weak stable with respect to the synchronizer shown in Figure 1(i) but it is not weak stable with respect to the synchronizers shown in Figures 1(ii), 1(iii) and 1(iv).

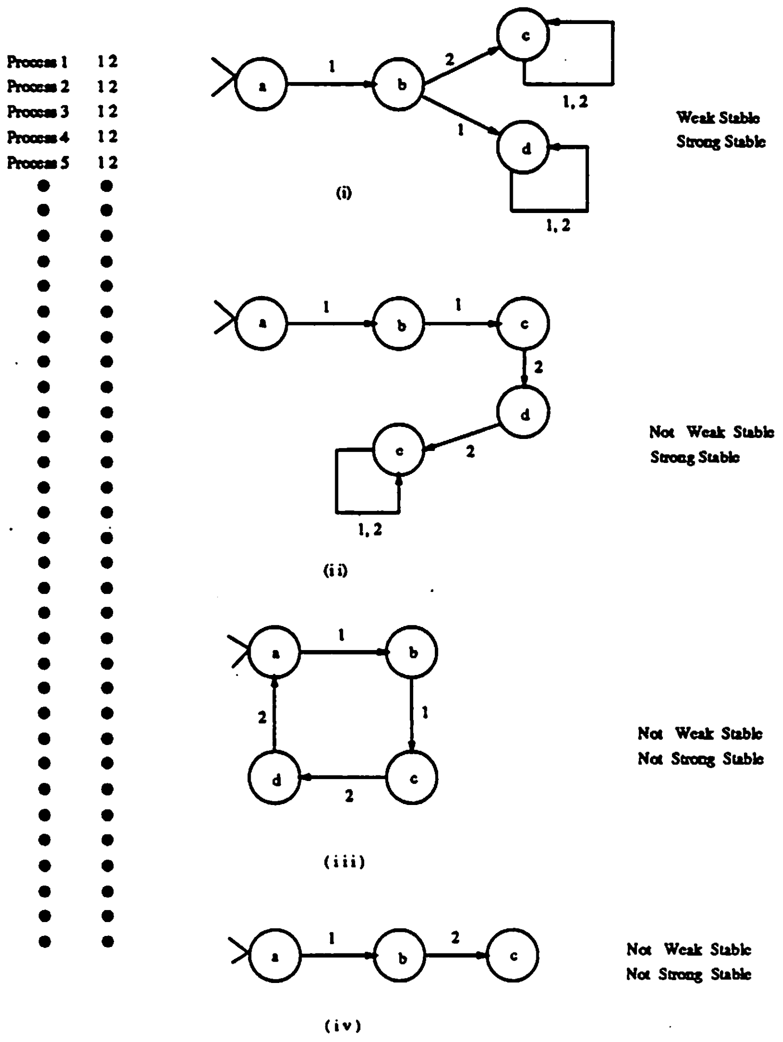


Figure 1: Weak and Strong Stability Examples

We augment the synchronizer $\mathcal{S}$ by adding a new "dead" state D. For each existing state s of $\mathcal{S}$, we add an arc with label $\sigma \in \sum$ from s to D if and only if there does not exist an arc with label σ out of state s. We also add arcs for every possible symbol in the alphabet from D back to D. Following the arguments presented in the proof for Proposition 1, we have

Corollary 1 *$(P, \mathcal{S})$ is not weak stable if and only if there exists $w \in prefix(I)$ such that $\delta(q_0, w) = D$.*

Theorem 1 *Weak stability is decidable.*

We give an effective procedure which takes as input an arbitrary concurrent system $(P, \mathcal{S})$ and determines a bound k such that if $(P^j, \mathcal{S})$ is weak safe for all $j \leq k$, then $(P, \mathcal{S})$ is weak stable. (Note: For a fixed j, checking if $(P^j, \mathcal{S})$ is weak safe is clearly algorithmic. One possible brute force method would be to check if any interleaving of j processes leads to the dead state D).

Before presenting our procedure, we recast our language theory formulation of concurrency as a combinatorial problem on the poset of non-negative lattice points in Euclidean n-space. We order two lattice points as follows:

$$a = (a_1, a_2, \ldots, a_n) \leq b = (b_1, b_2, \ldots, b_n)$$

iff $\forall i, a_i \leq b_i$. An *$N-$chain* is a set of N points which are pairwise comparable. A point belongs to the kth plane if the sum of its coordinates equals k. Now consider selecting points from successive planes, beginning with the kth plane. Given $N > 0$, Girkar and Moll [7] give an effective upper bound on the number of points that must be selected in order to guarantee the existence of an N-chain among the selected points. This bound is denoted by $L(n, N, k)$, where n stands for the dimension, N stands for the chain length one seeks, and k stands for the first plane from which a point is chosen.

Now consider a slightly different problem. Starting with some kth plane, select any first point. The second point can be selected from the kth, $(k+1)$st, or $(k-1)$st plane. In general, if the ith selected point is chosen from the mth plane, then one can choose the $(i+1)$th point from the mth, $(m+1)$th, or $(m-1)$th plane. (We may choose points that have been selected before). Also, suppose that the last point selected is such that all its coordinates are less than some positive integer B (Note: Intermediate selected points need not obey this constraint). For such a sequence of selections, we derive an upper bound $R(n, N, B, k)$ on the number of selection steps required to guarantee the existence of a non-increasing chain of N points from among the selected points.

Proposition 2 $R(n, N, B, k) \leq N(nB + L(n, N, nB))^n$

Proof: Let us denote the last point by b. The proof is by a case analysis:

Case 1: Suppose there is a point a in this sequence such that one of its coordinates exceeds $L(n, N, nB) + nB - 1$. Since b has all its coordinates less than B, and because of the nature of the selection process, these exists a reverse subsequence $c_{nB}, c_{nB+1}, \ldots, c_{nB+2}, \ldots, c_{nB+L(n,N,nB)-1}$ in between b and a where $c_{nB+i}, 0 \leq i \leq (L(n, N, nB) - 1)$ is a point in the $(i + nB)$th plane. The length of this subsequence is $L(n, N, nB)$. Using the result by Girkar and Moll [7], we conclude that there exists a strict increasing N chain in the reverse subsequence which implies a strict decreasing N chain in the original sequence.

Case 2: Suppose all the intermediate points are such that no coordinate exceeds $L(n, N, nB)+nB-1$. There are only $(nB+L(n, N, nB))^n$ such distinct points. Therefore, there exists a point which is repeated at least N times in the sequence of $R(n, N, B, k)$ points. This completes the proof. $\Box$.

Note that the above bound is independent of the plane from which the first point is selected. Hence, we write $R(n, N, B, k)$ as $R(n, N, B)$.

Note: For a string $w \in$ prefix(I), the vectors in the sequence vec_seq(w), is one possible sequence of selecting vectors as discussed in Proposition 2. The only difference is that the coordinates of the end vector vec(w) are not constrained.

Before presenting our procedure to decide weak stability (proof for Theorem 4), we make an important observation. As stated in Corollary 1, $(P, \mathcal{S})$ is not weak stable if and only if there exists a $w \in$ prefix(I) which leads to state D. If D is "reachable" via a string $w \in$ prefix(I), then there must exist a shortest such string. Let $\mathcal{S}$ have N states including the dead state D. Suppose there exists a string $w \in$ prefix(I), such that $\delta(q_0, w) = D$. Further, suppose there exists a length $N + 1$ non-increasing chain in the sequence vec_seq(w). (Let us denote such a property of of vec_seq(w) as property $\mathcal{X}$.) Then, by the pigeon hole property, some state of $\mathcal{S}$ must appear twice among these $N + 1$ points. Let us denote the first and second position where some state repeats by p_1 and p_2. Obviously, vec(w^{p_1}) $\geq$ vec(w^{p_2}). Let z be the string obtained from w by deleting the substring between positions p_1 and p_2 (*i.e.* symbols from position $p_1 + 1$ up to and including symbol at position p_2). Consider the sequence of points vec_seq(z). All points up to position p_1 in the sequence have non-negative coordinates. Further vec(z^{p_1}) = vec(w^{p_1}). Since vec(w^{p_1}) $\geq$ vec(w^{p_2}), points beyond p_1 in the sequence vec_seq(z) also have non-negative coordinates. Thus $z \in$ prefix(I). Also $\delta(q_0, z) = D$ since the same state appeared at end of p_1 and p_2 in w.

Now suppose vec_seq(w) is such that there exists a length $N + 1$ non-increasing chain only in a subset of coordinates. Suppose the first point on this chain occurs at position i in w. Suppose vec(w) (the final point in the sequence vec_seq(w)) is such that all the coordinates which do not participate in the chain are $> 2(|w| - i + 1)$. (Let us denote such a property of vec_seq(w) by property $\mathcal{Y}$.) It is easy to see that once again, we can delete the substring between two positions p_1 and p_2 (*i.e.* symbols from position $p_1 + 1$ up to and including the symbol at position p_2) in the non-increasing chain at which a state repeats to obtain a shorter string z such that $\delta(q_0, z) = D$ and $z \in$ prefix(I). (Essentially, the coordinates which do not participate are greater than the length of the the string from position p_2 to the end of w. Hence, pasting the string from position $p_2 + 1$ onwards after position p_1 leaves all these coordinates positive.)

We now present our procedure for proving the main result of this section, Theorem 1. Our procedure will obtain a bound k, such that, any string $w \in \text{prefix}(I)$ of length greater than k, and for which $\delta(q_0, w) = D$, can be shortened to obtain another string $w' \in \text{prefix}(I)$, such that, $\delta(q_0, w') = D$. Thus, if D is reachable via a string in prefix(I), then, it is reachable via a string in prefix(I) of length less than k. The bound k will be obtained using the bound $R(n, N, B)$ discussed in Proposition 2.

Define $U(0) = R(0, N+1, 0) = (N+1)$. (Note: The above definition assumes dummy values for the B part of the definition for $R(n, N, B)$ since we are in a 0-dimensional poset).

Let

$$\begin{array}{lcl} U(1) & = & R(1, N+1, 2U(0)), \\ U(2) & = & R(2, N+1, 2U(1)), \\ \ldots & \ldots & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots & \ldots & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ U(l) & = & R(l, N+1, 2U(l-1)) \end{array}$$

Theorem 1 follows directly from the following proposition.

Proposition 3 *A program of length l is weak stable if and only if state D is not reachable via any interleaving of length $U(l-1)$ or less.*

Proof: We use induction to prove the above proposition. Consider the base case when P contains one symbol a_0 ($|P| = 1$). For this case, $I = \text{prefix}(I) = a_0^*$. It is easy to see that if we consider all strings in prefix(I) of length $U(0) = N+1$, then D is not "reachable" for any of these interleavings if and only if S is weak stable. (Note that for programs of length one, the poset has zero dimensions).

To prove the induction step, assume by contradiction that for a program P of length l, the shortest string $w \in \text{prefix}(I)$ such that $\delta(q_0, w) = D$ is of length greater than $U(l-1)$. If the end point $b = \text{vec}(w)$ of w is such that all its coordinates are less than $2U(l-2)$, then by Proposition 2, we conclude that there exists a sequence of $N+1$ non-increasing vectors in vec_seq(w) (property $\mathcal{X}$), which implies that we can obtain a shorter string in prefix(I) which reaches D. This contradicts the fact that w is the shortest string such that $w \in \text{prefix}(I)$ and $\delta(q_0, w) = D$. Hence suppose that at least one of the coordinates of b is greater than $2U(l-2)$. Without loss of generality, assume that the coordinates of every vector in vec_seq(w) are rearranged according to the non-decreasing order of the components of b. Thus, $b_{l-1} > 2U(l-2)$. Consider b_1. If $b_1 \geq 2(N+1) = 2R(0, N+1, 0)$, then all the coordinates are $\geq 2(N+1)$, which implies that $N+1$ symbols back from b, all the coordinates are positive for all the points. Also, some state s in S will repeat for these penultimate $N+1$ selections. Let us denote the two positions where s repeats for these penultimate $N+1$ selections by p_1 and p_2. Then deleting the string between these two positions (symbols from position $p_1 + 1$ up to and including symbol at position p_2) will produce a shorter string in prefix(I) which reaches D. So suppose $b_1 < 2(N+1) = 2U(0)$. It follows that there exists an $i, 1 \leq i \leq (l-1)$ such that $b_i > 2U(i-1)$ and $b_j < 2U(i-2), 1 \leq j \leq (i-1)$ (property $\mathcal{Y}$). If we now look at the penultimate $U(i-1)$ steps in the processing of w, then there exists a non-increasing chain of $N+1$ vectors among coordinates 1 thru $(i-1)$. Further, all of the remaining higher coordinates from position i onwards remain positive for the penultimate $U(i-1)$ steps, since they were $> 2U(i-1)$ at the end. Once again, from the arguments presented following Corollary 1, we can obtain a shorter string in prefix(I) which reaches D. □

Thus, one needs to check if $U(l-1)$ processes are weak safe with respect to the given synchronizer in order to decide weak stability. This proves Theorem 1.

We now prove the decidability of full weak safety of a concurrent system using Proposition 3.

Corollary 2 *Full weak safety is decidable.*

Proof: We present a procedure to decide full weak safety of any concurrent system $(P, \mathcal{S})$. Given such a system, (P is a program of length l.), we systematically test if $(P^j, \mathcal{S})$ is weak safe for all j from 1 through $U(l-1)$. (For a fixed j, one brute force method to decide if $(P^j, \mathcal{S})$ is weak safe would be to check if all possible interleavings generated by j copies of program P are in $L(\mathcal{S})$.) Using Proposition 3, and, from the the definition of weak stability (see Definition 5), it follows that $(P, \mathcal{S})$ is full weak safe if and only if $(P^j, \mathcal{S})$ is weak safe for all j from 1 through $U(l-1)$. □.

In the proof of Proposition 3, we restricted each symbol in the program P to be distinct. The corollary below shows that both weak stability and full weak safety are decidable in the case of repeated symbols.

Corollary 3 *For a program $P = a_1 a_2 \ldots a_l$ where the a_i's are not necessarily distinct, and a given synchronizer $\mathcal{S}$, weak stability (full weak safety) is decidable.*

Proof: We present the proof only for deciding weak stability. The proof for deciding full weak safety is similar.

We transform P and $\mathcal{S}$ into a another program P' and synchronizer $\mathcal{S}'$ with transition function δ' over a new alphabet $\sum'$ such that each symbol in P' is distinct and $(P, \mathcal{S})$ is weak stable if and only if $(P', \mathcal{S}')$ is weak stable.

$\sum'$ has l symbols denoted by$\{b_1, b_2, \ldots, b_l\}$ and P' is the string $b_1 b_2 \ldots b_l$. Let I' be the Interleaving Language for program P'. Define function t from $\sum' \rightarrow \sum$ such that $t(b_i) = a_i$. For each a_i, $t^{-1}(a_i)$ is the collection of elements of $\sum'$ that are mapped to a_i by t. $\mathcal{S}'$ has the same states as $\mathcal{S}$. However, each arc in $\mathcal{S}$ with label a_i is replaced with $|t^{-1}(a_i)|$ arcs. Each of these $|t^{-1}(a_i)|$ arcs is labeled with a corresponding symbol from $t^{-1}(a_i)$.

Suppose there exists a string $w \in \text{prefix}(I)$ such that $\delta(q_0, w) = D$ (*i.e.* w leads to the dead state in $\mathcal{S}$). Consider the string $w' \in \text{prefix}(I')$ obtained from w by replacing every symbol a in w by any symbol from $t^{-1}(a)$. It is easy to see that $\delta'(q_0, w') = D$ for $\mathcal{S}'$. Similarly, if there exists a string $w' \in \text{prefix}(I')$ such that $\delta'(q_0, w') = D$, then the string w obtained from w' by replacing every symbol b in w' by $t(b)$ is such that $\delta(q_0, w) = D$. This completes the proof. □

It should be clear from the proof of Proposition 1 that if $(P^j, \mathcal{S})$ is not weak safe for some $j \geq 1$, then for all $k > j$ $(P^k, \mathcal{S})$ is also not weak safe. Thus, if weak stability fails, the set of positive integers j such that $(P^j, \mathcal{S})$ is not weak safe is always infinite. Essentially, we cannot have a bounded number of failures. This is not the case for strong stability, which we turn to next.

5 Strong Stability

For weak stability, we dealt with a synchronizer that did not do any "look-ahead". That is, if presented with an interleaving representing a sequence of operations, it tried to execute these operations on behalf of the processes in the same order. We now consider a slightly more powerful synchronizer which at every state chooses to service any process whose next operation it can execute. If there is more than one process whose operation can be executed, it arbitrarily chooses one among them. For such a model, deadlock occurs when the synchronizer reaches a state where it cannot execute any outstanding operation for any of the processes. Such a model is strong stable if there exists a bound k, such that k or more processes never deadlock for the synchronizer.

As an example, process "12" is strong stable for the synchronizers shown in Figures 1(i) and 1(ii), but it is not strong stable for the synchronizers shown in Figures 1(iii) and 1(iv). In Figure 1(i), the synchronizer accepts all strings, hence process "12" is strong stable for it. The synchronizer in Figure 1(ii) will deadlock if there is only one instance of process "12". However, for more than one instance, it will never deadlock. The synchronizer in Figure 1(iii) will deadlock at state b for odd number of processes, but will not deadlock for an even number of processes. The synchronizer in Figure 1(iv) deadlocks once it finishes requests for one instance of process "12", and then remains deadlocked in state c.

The examples in Figures 1 distinguish our work from the research presented in [16] and [17]. In Figure 1(ii), a potential deadlock exists with one process, however the model is strong stable since two or more processes are always strong safe. For Figure 1(iii), deadlock situation exists with odd number of processes, hence it is not strong stable. Thus, for both examples deadlock exists (and the procedure outlined in [16] will detect that); however, the first example is strong stable where as the second one is not.

We now formalize the above ideas. Let $SLA(v)$ represent the set of symbols on edges directed out of a node $v \in V$. (SLA can be thought of as "synchronizer look ahead".) For all strings $\sigma_1, \sigma_2 \in \sum^*$, let

$$LA_{\sigma_1}^{\sigma_2} = \begin{cases} a \in \sum & \text{if } \sigma_2 = \sigma_1 a x,\ x \in \sum^* \\ \emptyset & \text{otherwise} \end{cases}$$

Thus, LA is either the singleton set containing the "look ahead symbol" in σ_2 provided σ_1 is a proper prefix of σ_2, or else it is empty.

A *Partial Execution* of a process is any prefix (possibly empty) of its program string. Let $p_1, p_2, \ldots, p_k$ represent partial executions of k identical processes, each with program $P = a_1 a_2 \ldots a_l$. A partial execution of k identical processes with individual partial executions $p_1, p_2, \ldots, p_k$ is any interleaving of $p_1, p_2, \ldots, p_k$. Let $PE(p_1, p_2, \ldots, p_k)$ denote the set of all partial executions of k processes with individual partial executions $p_1, p_2, \ldots, p_k$ respectively. Since we are interested in the case when $p_i \neq P$ for some i, we define $PE(p_1, p_2, \ldots, p_k) = \emptyset$ if $\forall i, p_i = P$.

Definition 7 *A concurrent system (P^k,S) is* strong safe *iff for all partial executions $p_1, p_2, \ldots, p_k$ (p_i is a partial execution of process i) and*
$\forall \sigma \in (L(S) \bigcap PE(p_1, p_2, \ldots, p_k))$,

$$(\bigcup_{i=1}^{i=k} LA_{p_i}^{P}) \bigcap SLA(\delta(q_0, \sigma)) \neq \emptyset$$

Intuitively, if a concurrent system $(P^k$,S) is strong safe, then, for all possible partial executions, the synchronizer and the k processes never deadlock.

Definition 8 *(P,S) is* strong stable *iff $\exists m > 0$ such that $\forall k \geq m$, (P^k,S) is strong safe.*

Definition 9 *(P,S) is* full strong safe *iff $\forall k > 0$, (P^k,S) is strong safe.*

Theorem 2 *Strong stability is decidable.*

The proof of strong stability is based on the proof of the decidability of reachability for vector addition systems with states (VASS) (see Kosaraju [18]). Whenever possible, we have adopted Kosaraju's notation. A VASS is a finite state automaton in which the label on each arc is an n-tuple of integers. A configuration of the VASS is an ordered pair (q, v) where q in a state and v is a point in Z^n. For $v \in Z^n$, let $\Pi_j(v)$ be the jth coordinate of v. Starting with an initial configuration (q, x), a path from q in the automaton induces a sequence of configurations. A new vector is obtained from the previous configuration by a component-wise vector addition of the label on the arc "traveled" and the vector in the previous configuration. A configuration (q', y) is r-reachable from (q, x) *iff* there is a path from the initial configuration (q, x) to (q', y) (denoted by

$(q', y) \in r(q, x)$). Such a path is an r-path from (q, x) to (q', y). A configuration (q', y) is R-reachable from (q, x), *iff* there is a path from (q, x) to (q', y) such that all vectors along the path (including x and y) belong to the positive orthant in n-space (denote by $(q', y) \in R(q, x)$). Such a path is an R-path from (q, x) to (q', y). The reachability problem for vector addition systems is the problem of deciding, for an arbitrary q, q', x and y, if $(q', y) \in R(q, x)$.

Let $A \subseteq \{1, 2, \ldots, n\}$. A configuration (q', y) is semi-R-reachable (or SR-reachable) with respect to A from (q, x) *iff* there is a path from (q, x) to (q', y) such that all vectors v along the path (including x and y) are positive in the coordinates specified by A; (*i.e.* for all $j \in A$, $\Pi_j(v) \geq 0$). Such a path is an SR-path from (q, x) to (q', y).

5.1 Mapping Between Strong Stability and VASS

Consider k processes, each with program $P = a_1 \ldots a_l$, where all a_i's are distinct. We describe below a transformation of the synchronizer $\mathcal{S}$ into a VASS. The configurations of the VASS, which are non-negative integer tuples in n-space, will track partial executions of the processes.

The states and arcs of the VASS that captures $\mathcal{S}$ are identical to those in $\mathcal{S}$. We label the arcs of this VASS as follows: For each symbol in P, we associate a unique l-tuple of integers. For $a_i, 2 \leq i \leq (l-1)$, the tuple is such that its $(i-1)$th coordinate is -1, ith coordinate is 1, and, all of its other coordinates are zero. The tuple for a_1 is such that its first and last coordinates are both 1, and, all of its other coordinates are zero. The tuple of a_l has a -1 in the $(l-1)$th coordinate, and it is zero in all its other coordinates. Every arc of the VASS is annotated with the l-tuple of integers associated with the symbol on the corresponding arc in $\mathcal{S}$. The initial configuration for the VASS is $(q_0, \overline{0})$, where q_0 is the initial state of $\mathcal{S}$ and $\overline{0}$ is the l-dimensional zero vector.

As an example, consider Figure 2 in which $\sum = \{a, b, c\}$, $P = abc$, and $w = aabcab$ is one possible partial execution of processes 1, 2, and 3, with individual partial executions abc, ab, and a respectively (as shown by the staircase in the figure). To transform the synchronizer into a VASS, we associate a tuple as described above for each symbol in $\sum$. The initial state of the synchronizer is x. The sequence of configurations produced in the VASS after processing each symbol in w is as follows:
$(x, (0,0,0)), (y, (1,0,1)), (v, (2,0,2)), (u, (1,1,2)),$
$(x, (1,0,2)), (y, (2,0,3)), (z, (1,1,3))$

Thus, after processing a symbol in w, the VASS changes its state and reaches a new vector. After processing the ith symbol in w (*i.e.*, after processing w^i, the ith prefix w^i), the vector reached is such that its first (second) coordinate equals the difference between the number of occurrences of a's and b's (b's and c's) in w^i, and, the third coordinate equals the number of occurrences of a's in w^i.

In general, the sequence of configurations induced by the processing of a

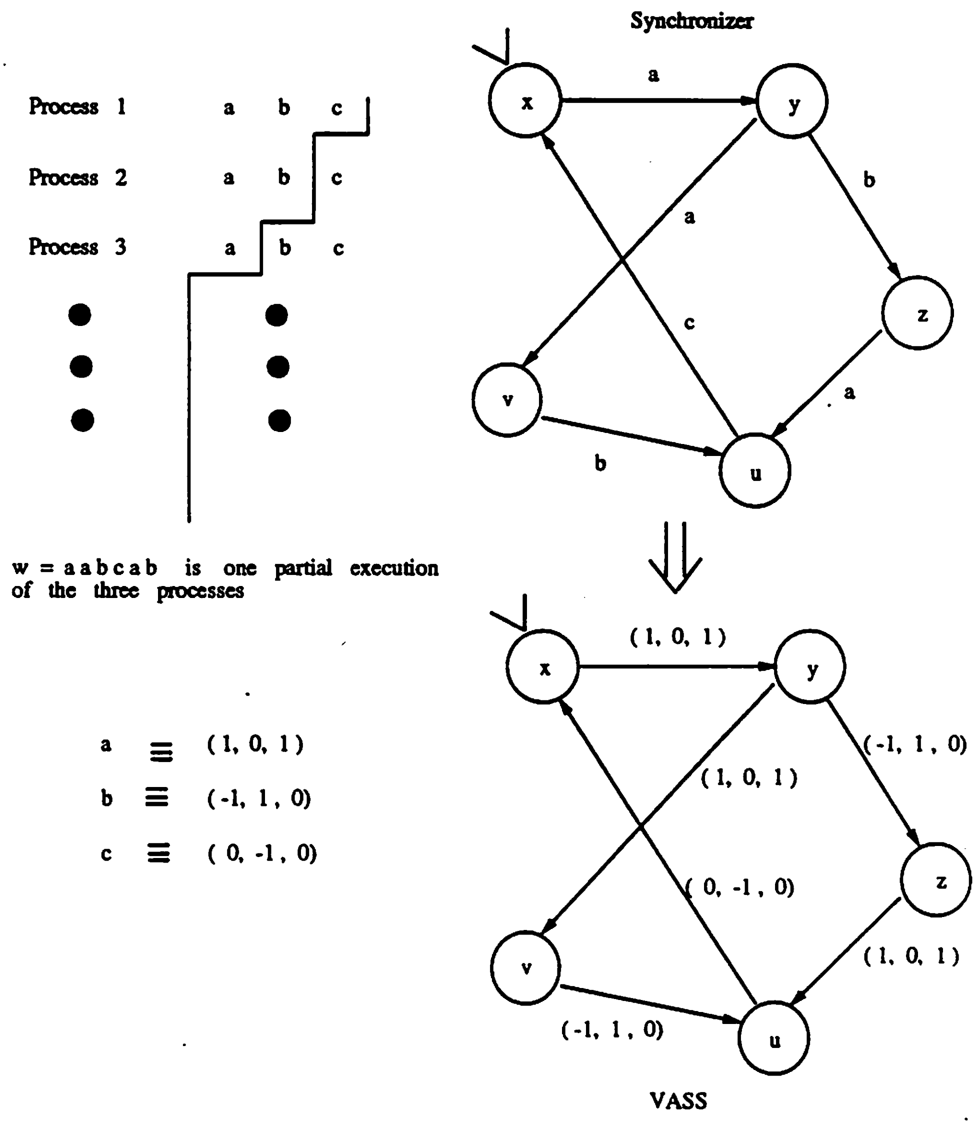


Figure 2: Synchronizer and its VASS

partial execution σ (and, which also belongs to $L(\mathcal{S})$) by the VASS describes an R-path. Suppose the $(m+1)$th configuration along this R-path is (s, v). Then, s is the state reached in $\mathcal{S}$ after processing the first m symbols of σ (equivalently, after processing the mth prefix of σ, namely σ^m) and v is the vector whose first $l-1$ coordinates count the differences between the number of occurrences of a_{i-1} and a_i, $(2 \leq i \leq l)$ in σ^m, and, its lth coordinate counts the number of occurrences of a_1 in σ^m.

Vector v has the following important properties:

1. Since each symbol of P is distinct, the last coordinate of v equals the number of processes involved in the partial execution σ^m. (Note: This coordinate will be used in determining the strong stability constant, provided the model is strong stable.)

2. The last coordinate of v is at least as large as its other coordinates.

3. The first $l-1$ coordinates of v are precisely the coordinates of vector vec(σ^m) (see Definition 2).

4. If the ith $(1 \leq i \leq l-1)$ coordinate of v is greater than zero, then, there exists a process whose next operation to be executed is a_{i+1}.

5. Since any partial execution of processes has the property that the difference between the number of occurrences of a_{i-1}'s and a_i's $(2 \leq i \leq l)$ is non-negative, and, since the last coordinate of v cannot be negative, v has only non-negative coordinates.

In what follows, we use $\mathcal{S}$ interchangeably to denote either the original synchronizer or the VASS into which it is transformed. The meaning should be clear from context.

5.2 Vector Classes, Infinite-R-reachability, and Bad States

In order to decide weak stability, we obtained bounds on length of chains of vectors in the sequence vec_seq(w) for a partial execution w. However, in order to decide strong stability, we need to consider grouping vectors according to certain properties satisfied by their coordinates. These properties are aimed at describing conditions under which a partial execution of processes deadlocks. Intuitively, a vector class is a set of vectors whose coordinates satisfy these properties.

For example, in Figure 3, consider the partial executions *ab*, *aabbccab*, and *aabbccaabbccab* constructed from one, three, and five processes respectively. These three partial executions deadlock at state U. The vectors reached at state U for these partial executions are $(0,1,1)$, $(0,1,3)$, and, $(0,1,5)$ respectively. (The first (second) coordinate counts the differences between number of occurrences of symbol a and symbol b (the differences between number of

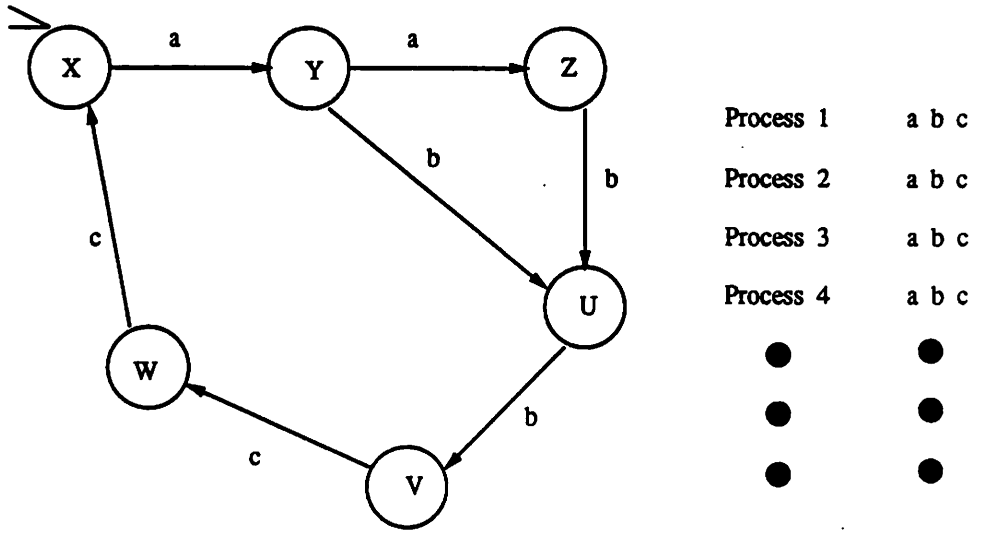


Figure 3: Vector Classes

occurrences of symbol b and c), and the third coordinate counts the number of occurrences of symbol a.) It is easy to see that any partial execution that reaches state U with an end vector whose first coordinate is zero, and second coordinate is positive, will deadlock at U. Thus, for the purposes of detecting deadlock at state U, the vector class would contain vectors with the property that their first coordinate is zero, and their second coordinate is positive.

We also describe when a pair consisting of a state of the VASS and a vector class is infinite-R-reachable from an initial configuration. Informally, a pair consisting of a state and a vector class is infinite-R-reachable from the initial configuration in a VASS, if and only if there are infinitely many R-paths from the initial configuration which reach the state with any vector that belongs to the vector class. These definitions are then used to precisely describe "bad" states (see Definition 11) of $\mathcal{S}$. Finally, Proposition 4 associates strong stability with detecting bad states.

Definition 10 *Consider vectors in Z^n. Let W, W' be subsets of $\{1, 2, \ldots n\}$ such that $W \bigcap W' = \phi$ and W, W' are not both empty. A Vector Class for W, W' (denoted by $\mathcal{V}^{W,W'}$)is the set of vectors v in the positive orthant with the property that $\Pi_i(v) = 0$ if $i \in W$, and for some $j \in W', \Pi_j(v) > 0$. Given W, W', we say $(q', \mathcal{V}^{W'W'})$ is R-reachable from (q, x) if and only if there exists $v \in \mathcal{V}^{W,W'}$ such that (q', v) is R-reachable from (q, x). Let $P^{W,W'}$ denote the set of R-paths from (q, x) to any $v \in \mathcal{V}^{W,W'}$. If $|P^{W,W'}|$ is infinite, we say that $(q', \mathcal{V}^{W,W'})$ is infinitely-R-reachable from (q, x). If $|P^{W,W'}|$ is a finite non-zero value, then we say that $(q', \mathcal{V}^{W,W'})$ is finitely-R-reachable from (q, x).*

In order to motivate the definition of bad states, we consider a scenario in which a partial execution of k processes deadlocks. More precisely, let $p_1, p_2, \ldots, p_k$ be partial executions of k processes and suppose there exists a σ such that

$$\sigma \in (L(S) \bigcap PE(p_1, p_2, \ldots, p_k))$$

and for which

$$(\bigcup_{i=1}^{i=k} LA^P_{p_i}) \bigcap SLA(\delta(q_0, \sigma)) = \emptyset$$

As noted earlier, the sequence of configurations induced by the processing of σ by S describes an R-path. The final vector in the sequence satisfies the following two conditions:

(*) For $1 \leq i \leq (l-1)$, if $a_{i+1} \in SLA(\delta(q_0, \sigma))$, then the ith coordinate of the final vector equals zero.

(**) If $a_1 \in SLA(\delta(q_0, \sigma))$, then all the p_j's are non-empty strings.

Condition (*) holds because if for some i between 1 and $(l-1)$, $a_{i+1} \in SLA(\delta(q_0, \sigma))$, and, if the ith coordinate is greater than 0, then, the difference between the number of occurrences of a_i and a_{i+1} in σ is greater than 0. This implies that there exists $j, 1 \leq j \leq k$, such that, $p_j = a_1 a_2 \ldots a_i$. But that contradicts

$$(\bigcup_{i=1}^{i=k} LA^P_{p_i}) \bigcap SLA(\delta(q_0, \sigma)) = \emptyset$$

Similarly, if there exists a partial execution p_j that is empty, for that p_j, $LA^P_{p_j} = \{a_1\}$, which again contradicts

$$(\bigcup_{i=1}^{i=k} LA^P_{p_i}) \bigcap SLA(\delta(q_0, \sigma)) = \emptyset$$

Hence, Condition (**) holds.

We now use the definition of vector classes and infinite-R-reachability (Definition 10) to formulate when a state of S is bad. Informally, a state s is bad when there exist arbitrarily large sets of processes such that a partial execution of these processes cause a deadlock to occur at s. As should be intuitively clear from Condition (*) and (**) above, only the first $l-1$ coordinates are crucial in determining the existence of deadlock. For each state, we first define an "unsuitable" vector class. Roughly speaking, the unsuitable vector class for a state contains all vectors representing partial executions, such that none of the pending operations in the partial executions can be executed by the synchronizer in that state. Using the definition of the unsuitable vector class for a state, we describe conditions when a particular state is bad.

Definition 11 *For a state s, denote by $\mathcal{V}^{U^s,\overline{U^s}}$ its "unsuitable" vector class where both U^s and $\overline{U^s}$ are subsets of $\{1,2,\ldots,l-1\}$, $U^s = \{i-1|a_i \in \{SLA(s)-a_1\}\}$, and, $\overline{U^s}$ is the complement of U^s with respect to $\{1,2,\ldots,l-1\}$. A state s is "bad"* iff

1. *$U^s \subset \{1,2,\ldots l-1\}$ and $(s,\mathcal{V}^{U^s,\overline{U^s}})$ is infinitely-R-reachable from $(q_0,\bar{0})$ (Condition B_1); OR*
2. *$U^s = \{1,2,\ldots l-1\}$ and $(s,\mathcal{V}^{U^s,\overline{U^s}})$ is infinitely-R-reachable from $(q_0,\bar{0})$, and $a_1 \notin SLA(s)$ (Condition B_2); OR*
3. *$(s,\mathcal{V}^{U^s,\overline{U^s}})$ is finitely-R-reachable from $(q_0,\bar{0})$, and $a_1 \notin SLA(s)$ (Condition B_3);*

If none of B_1, B_2 or B_3 holds, then s is "good".

Notice that condition B_1 and B_2 are mutually exclusive because, for B_2 to hold, $a_1 \notin SLA(s)$, an additional restriction.

We first prove a small fact regarding the cardinality of the set of vectors that belong to $\mathcal{V}^{U^s,\overline{U^s}}$ and which are reached from some initial configuration. This fact is used in the proof of Proposition 4.

Fact 1 *$(q',\mathcal{V}^{U^s,\overline{U^s}})$ is infinitely-R-reachable (finitely-R-reachable) from (q,x)* iff *$|\{v|v \in \mathcal{V}^{U^s,\overline{U^s}},(q',v) \in R(q,x)\}|$ is infinite (finite and non-zero).*

Proof: If $(q',\mathcal{V}^{U^s,\overline{U^s}})$ is infinitely-R-reachable from (q,x), then, by Definition 10, $P^{U^s,\overline{U^s}}$ is infinite. Each R-path in $P^{U^s,\overline{U^s}}$ corresponds to a partial execution of some $k>0$ processes. Since $P^{U^s,\overline{U^s}}$ is infinite, there must be an infinite subset $\mathcal{M}$ of $P^{U^s,\overline{U^s}}$, such that the number of processes involved in each R-path of $\mathcal{M}$ are distinct. Also, since the last coordinate of the final vector along any R-path counts the number of processes involved in the partial execution corresponding to the R-path, the final vectors for each of the R-paths in $\mathcal{M}$ are distinct. Hence, $|\{v|v \in \mathcal{V}^{U^s,\overline{U^s}},(q',v) \in R(q,x)\}|$ is infinite. In the other direction the proof is trivial. Similarly, one can prove that $(q',\mathcal{V}^{U^s,\overline{U^s}})$ is finitely-R-reachable from (q,x) *iff* $|\{v|v \in \mathcal{V}^{U^s,\overline{U^s}},(q',v) \in R(q,x)\}|$ is some finite non-zero value. □

The proposition below associates strong stability with good states.

Proposition 4 *$(P,\mathcal{S})$ is strong stable if and only if all states of $\mathcal{S}$ are good.*

Proof: Suppose $(P,\mathcal{S})$ is strong stable. Then there exists an $m>0$, such that $\forall k \geq m$, k identical processes with program P are strong safe for $\mathcal{S}$. Assume the contradiction: Suppose there is a state s which is bad. We make a case analysis.

Case 1: If B_1 holds for s, then, since $|\{v | v \in \mathcal{V}^{U^s,\overline{U^s}}, (s,v) \in R(q_0,\bar{0})\}|$ is infinite (Fact 1), there exists a vector $v \in \mathcal{V}^{U^s,\overline{U^s}}$ such that $(s,v) \in R(q_0,\bar{0})$ and for some $i, 1 \leq i \leq l$, the ith coordinate of v, $\Pi_i(v)$, is greater than m. Recall that the last coordinate of v ($\Pi_l(v)$) keeps track of the number of occurrences of the first symbol of program P in the partial execution corresponding to the R-path from $(q_0,\bar{0})$ to (s,v). $\Pi_l(v)$ is also the number of processes that "participate" in forming the R-path, and it is at least as large as any of the other coordinates. Thus, $\Pi_l(v) > m$. Further, since $\overline{U^s}$ is not empty, there exists $i \in \overline{U^s}$ such that $\Pi_i(v) > 0$. Thus, there exist partial executions $p_1, p_2, \ldots p_{\Pi_l(v)}$ of $\Pi_l(v)$ processes such that

$$(\bigcup_{i=1}^{i=\Pi_l(v)} LA^P_{p_i}) \bigcap SLA(\delta(q_0,\sigma)) = \emptyset$$

and for some $i, (1 \leq i \leq \Pi_l(v))$ p_i is not the entire program string P. This contradicts strong stability.

Case 2: Suppose B_2 holds for s. Since $(s, \mathcal{V}^{U^s,\overline{U^s}})$ is infinitely-R-reachable from $(q_0,\bar{0})$ and since $U^s = \{1,2,\ldots,l-1\}$ there exists an infinite set of positive integers j with the property that some interleaving σ of j processes each of whose partial execution is the entire program string P leads to state s from the initial configuration $(q_0,\bar{0})$. For each such j, consider $j+1$ processes with partial executions

$$p_1, p_2, \ldots, p_j, w$$

where all the p_i's are the entire program string P and w is the empty string. For these $j+1$ processes, since w is empty, not all partial executions are P. Further, state s does not contain arc with label a_1 to bail out the synchronizer. Since there are infinitely many j's, strong stability does not hold.

Case 3: If B_3 holds for s, consider the interleaving σ which leads to a configuration (s,v), where $v \in \mathcal{V}^{U^s,\overline{U^s}}$. Suppose, the number of processes needed to form σ is k'. Let the partial executions needed to form σ for these k' processes be $p_1, p_2, \ldots, p_{k'}$. Consider $k'+m$ processes with partial executions

$$p_1, p_2, \ldots, p_{k'}, w_1, w_2, \ldots, w_m$$

where $w_i, 1 \leq i \leq m$ is the empty string. Obviously, not all of the $k'+m$ partial executions are the entire program string P. Also, since $a_1 \notin SLA(s)$ the additional m processes cannot "bail" out S out of state s. Once again, we contradict strong stability.

To prove the converse of the theorem, suppose all states of S are "good" and assume that P is not strong stable for S. Hence there exists infinitely many $m > 0$ for which m processes are not strong safe. Let $Unsafe$ denote the set of positive integers such that if $m \in Unsafe$, then m processes are not strong safe. Obviously, $|Unsafe|$ is infinite. For each $m \in Unsafe$, let T_m denote the set of interleavings formed from partial executions of m processes which lead to an "unsafe" state. Formally, $\sigma \in T_m$ *iff* σ is a partial execution of m processes and

$$(\bigcup_{i=1}^{i=m} LA^P_{p_i}) \bigcap SLA(\delta(q_0, \sigma)) = \emptyset$$

Let

$$\eta = \bigcup_{m \in Unsafe} T_m$$

We make a case analysis:
Case 1: $|\eta|$ is finite. For every $\sigma \in \eta$, define

$$\sigma^{Unsafe} = \{m | \sigma \in T_m\}$$

Then there exists a $\sigma' \in \eta$ such that σ'^{Unsafe} is infinite, and it follows that for the state s reached from the initial state q_0 via σ', condition B_3 holds.

Case 2: $|\eta|$ is infinite. There are finitely many (s, U^s) ordered pairs. For every such ordered pair, we define the set $\eta^{(s,U^s)}$ as follows: $\sigma \in \eta^{(s,U^s)}$ *iff* $\sigma \in \eta$, $\delta(q_0, \sigma) = s$, and the final vector $v \in \mathcal{V}^{U^s, \overline{U^s}}$.

Obviously, there exists one ordered pair $(s', U^{s'})$ such that $|\eta^{(s,U^{s'})}|$ is infinite. If $U^{s'} \subset \{1, 2, \ldots, l-1\}$, then condition B_1 holds for state s'.

So suppose $U^{s'} = \{1, 2, \ldots, l-1\}$. Any vector $v \in \mathcal{V}^{U^{s'}, \overline{U^{s'}}}$ will have 0 for coordinates $1, 2, \ldots, l-1$. Any interleaving with the property that the final vector belongs to $\mathcal{V}^{U^{s'}, \overline{U^{s'}}}$ therefore has to be formed from strings which are individually equal to either the entire program string P or which are empty. (Note: The restriction that all symbols in P are distinct is necessary to conclude this fact). Now consider any $\sigma \in \eta^{(s,U^{s'})}$. Suppose it is formed from k processes with partial executions $p_1, p_2, \ldots, p_k$. We make the following observations:

1. Since σ is a partial execution, for some $i, 1 \leq i \leq k, p_i \neq P$.

2. Since the final vector for σ belongs to $U^{s'}$ and since all symbols in program P are distinct, either $p_i = P$ or p_i is the empty string.

3. Since σ leads the synchronizer to an unsafe state, we must have

$$(\bigcup_{i=1}^{i=k} LA^P_{p_i}) \bigcap SLA(\delta(q_0, \sigma)) = \emptyset$$

From the first and second observation, we conclude that $\bigcup_{i=1}^{i=k} LA^P_{p_i} = \{a_1\}$. Finally, using the third observation and $\bigcup_{i=1}^{i=k} LA^P_{p_i} = \{a_1\}$ we conclude that $a_1 \notin SLA(s)$. Thus condition B_2 holds for state s. □

Therefore in order to decide if $(P, \mathcal{S})$ is strong stable, it suffices to show that each state of $\mathcal{S}$ is "good". Equivalently, if any state is "bad", $(P, \mathcal{S})$ is not strong stable. A state is bad if one of the conditions B_1, B_2 or B_3 (see Definition 11) holds for it.

In Section 6, we sketch the proof presented by Kosaraju [18] for the decidability of reachability in a VASS. Subsequently, in Section 7, we show how to modify the proof in [18] to decide infinite-R-reachability for vector classes in a VASS. Using this modification, in Section 7.2, we show an effective procedure to decide strong stability. Provided $(P, \mathcal{S})$ is strong stable, the procedure will also yield a positive integer $m > 0$, such that for all $k \geq m$, $(P^k, \mathcal{S})$ is strong safe.

6 Deciding Reachability In a VASS

Following [18], we introduce the notion of a Generalized Vector Addition System (GVASS). An n-dimensional GVASS is a finite chain $G_1, G_2, \ldots G_s$ of VASS's, each of which is n-dimensional. There is one and only one arc from G_i to G_{i+1} for $1 \leq i \leq s-1$. Let us denote the head and tail states on the arc that connects G_i to G_{i+1} by q_i' and q_{i+1}. An r-path from (q_1, x) (q_1 is a state of G_1) to (q_s, y) (q_s is a state of G_s) goes through each of the connecting arcs exactly once. When the r-path reaches q_i for the first time, we call the corresponding n-dimensional arrival vector the *input vector* of G_i. Similarly, when the path reaches q_i' for the last time, we call the corresponding n-dimensional vector the *output vector* of G_i. For G_1, the input vector is x, and for G_s the output vector is y. For a given r-path from (q_1, x) to (q_s, y), we write g_i^a, g_i^b for the input and output vector for G_i. For every G_i, we define an *Input Constraint* V_i and an *Output Constraint* V_i' where V_i, $V_i' \subseteq (\{\omega\} \cup N)^n$. An r-path from (q_1, x) to (q_s, y) is said to satisfy the input constraints if and only if for every i, j, $1 \leq i \leq s$ and $1 \leq j \leq n$

1. $\Pi_j(g_i^a) = \Pi_j(V_i)$ if $\Pi_j(V_i) \in N$ and
2. $\Pi_j(g_i^a) \geq 0$ if $\Pi_j(V_i) = \omega$.

We define v_i and v_i' to be the vectors obtained from V_i and V_i' respectively by replacing every ω in V_i and V_i' by zero.

Similarly, we define an r-path satisfying an output constraint. An r-path that satisfies both its input and output constraints is a cr-path. Note that when a cr-path is inside a G_i, the vectors can have negative components. A CR-path from (q_1, x) to (q_s, y) is a cr-path from (q_1, x) to (q_s, y) with the additional restriction that all points along this path lie in the positive orthant. Thus a CR-path is an R-path. When a GVASS contains just one VASS, the existence of a CR-path from (q_1, x) to (q_s, y) implies the existence of an R-path from (q_1, x) to (q_s, y). For G_i, let $S_i = \{j | \Pi_j(V_i) \neq \omega\}$ and $S_i' = \{j | \Pi_j(V_i') \neq \omega\}$.

We call the coordinates in S_i and S_i' as the *constrained input* and *constrained output* coordinates of G_i. A subset R_i of $S_i \cap S_i'$ is called the set of *rigid coordinates* and is defined as follows: For every $j \in R_i$, $\Pi_j(e) = 0$ for every arc label e in G_i.

A GVASS G with initial configuration (q_1, x) and final configuration (q_s, y) is said to satisfy property θ *iff* the following conditions are satisfied:

Property θ:

θ_1: For every $m \geq 1$, there exists a cr-path from (q_1, x) to (q_s, y) such that

$\theta_1(a)$: every arc in every G_i is used at least m times, and

$\theta_1(b)$: for every i and j, if $j \notin S_i$ then $\Pi_j(g_i^a) \geq m$, and if $j \notin S_i'$ then $\Pi_j(g_i^b) \geq m$; and

θ_2: For every i, there exist Δ_i, Δ_i' such that for every $j \in S_i - R_i$, $\Pi_j(\Delta_i) \geq 1$ and for every $j \in S_i' - R_i$, $\Pi_j(\Delta_i') \geq 1$, and

$\theta_2(a)$: $(q_i, v_i + \Delta_i) \in SR(q_i, v_i)$ with respect to $S_i - R_i$ in G_i, and

$\theta_2(b)$: $(q_i, v_i' + \Delta_i') \in SR(q_i', v_i')$ with respect to $S_i' - R_i$ in G_i.

Informally, condition $\theta_1(b)$ states that every unconstrained input or output coordinate of every G_i can be made as large as possible. Condition $\theta_2(a)$ states that for every G_i, in the subspace of its constrained, but nonrigid, input coordinates, all components of its input constraint vector can be simultaneously increased by an R-path. Condition $\theta_2(b)$ states a similar condition for constrained, but nonrigid, output coordinates for the output constraint vector.

Define the *size* of a VASS G_i by a triple (n_{i1}, n_{i2}, n_{i3}) where n_{i1} = the number of non-rigid coordinates $(n - |R_i|)$, n_{i2} = the number of arcs of G_i (say k_i), and n_{i3} = the number of unconstrained input and output coordinates of G_i $(2n - |S_i| - |S_i'|)$. We order these triples lexicographically. The size of GVASS G, $SS(G)$, is the multiset of sizes of its G_i's. We do not impose any order on this multiset.

Let $c, p_1, p_2, \ldots, p_k \in Z^n$. A *Linear Set* with constant c and periods $p_1, p_2, \ldots, p_k$ (denoted by $L(c; p_1, p_2, \ldots, p_k)$) is the set of all vectors v such that $v = c + e_1p_1 + e_2p_2 + \ldots + e_kp_k$ for non-negative integers $e_1, e_2, \ldots, e_k$. A *Semilinear Set* is a finite union of linear sets. Let $1 \leq i_1 \leq i_2 \leq \ldots \leq i_k \leq n$ and let $S = \{i_1, i_2, \ldots, i_k\}$. For any $x \in Z^n$, $x(S) = (\Pi_{i_1}(x), \Pi_{i_2}(x), \ldots, \Pi_{i_k}(x))$. For any $L \subseteq Z^n$, let $L(S) = \{x(S) | x \in L\}$. For a VASS G, its *reverse*, denoted by G_{rev}, is obtained by reversing the arcs in G and by replacing every label x by $-x$. The r, R, and SR reachabilities of G_{rev} are denoted by r_{rev}, R_{rev}, and SR_{rev}. We now state without proof several lemmas and theorems from Kosaraju [18].

Lemma 1 *Let L be a semilinear set, and let A be a subset of $\{1, 2, \ldots, n\}$. If L does not satisfy the property that for every $m \geq 1$ there exists an $x \in L$ such* *that for every $j \in A, \Pi_j(x) \geq m$, then, when L is expressed as a union of linear sets $\cup_{i=1}^{k} L_i$, for every L_i there exists a $j \in A$ such that the jth component of the sum of the periods of L_i has zero value.*

Lemma 2 *For any semilinear set L and any set of coordinates A,*

1. *$L(A)$ is a semilinear set.*
2. *If L is a linear set, then $L(A)$ is also a linear set.*
3. *If L is a linear set with one period, then $L(A)$ can be expressed with one period also. If L is $L(c;p)$, then $L(A)$ is given by $L(c(A);p(A))$.*

Lemma 3 *Let L be a semilinear set, and A a subset of its coordinates. If L has the property that*

1. *For every $x \in L$ and every $j \notin A$, $\Pi_j(x)$ has a fixed value; and*
2. *for every $m \geq 1$ there exists an $x \in L$ such that for every $j \in A$, $\Pi_j(x) \geq m$,*

then L has a linear subset $L(c;p)$ where the jth component of p is nonzero iff *$j \in A$.*

Theorem 3 *In a VASS G, for every q_1, x, q_2, y and $\Delta x, \Delta y \geq \bar{0}$, if there exists $\Delta_1, \Delta_2 \in Z^n$, $A \subseteq \{1, 2, \ldots, n\}$ and $m_1, m_2 \geq 0$ such that*

1. *for every $j \in A$, the jth component of the label of every arc of G has value 0, and for every $j \notin A$,*
   (a) $\Pi_j(\Delta x) = 0 \Rightarrow \Pi_j(\Delta_1) \geq 1$, and
   (b) $\Pi_j(\Delta y) = 0 \Rightarrow \Pi_j(\Delta_2) \geq 1$, and
2. *$(q_2, y) \in r(q_1, x)$*
3. *$(q_1, x + m_1 \triangle x + \Delta_1) \in R(q_1, x + m_1 \triangle x)$, and*
4. *$(q_2, y + m_1 \triangle y + \Delta_2) \in R_{rev}(q_2, y + m_2 \triangle y)$, and*
5. *$(q_1, \Delta y)$ is r-reachable from $(q_1, \Delta x)$ by a path p such that every arc in G is traveled at least once,*

then $\exists j_0 \forall j \geq j_0((q_2, y + j \triangle y) \in R(q_1, x + j \triangle x))$

Theorem 4 *If a GVASS G satisfies property θ, and, if there exists a VASS G_i of G whose size is different from $(0,0,0)$, then there exists infinite CR-paths from (q_1, x) to (q_s, y). If a GVASS G satisfies property θ, and, if all VASS's of G have size $(0,0,0)$, then, there exists a single CR-path from (q_1, x) to (q_s, y).*

Theorem 5 *If a GVASS G does not satisfy property θ, and if $SS(G)$ contains a triple different from* $(0,0,0)$*, then G can be replaced by a finite number of GVASS's $G^1, G^2, \ldots, G^t$, such that*

1. *For every i, $SS(G^i)$ can be obtained from $SS(G)$ by replacing a triple by a finite number of triples, each of which is less than the triple being replaced; and*

2. *there exists a CR-path from (q_1, x) to (q_s, y)* iff *there exists a CR-path from (q_1, x) to (q_s, y) for some G^i.*

Essentially, Theorem 5 enables us to partition the CR-paths in the original GVASS G into CR-paths in the new GVASS's $G^1, G^2, \ldots G^t$. The two conditions stated in Theorem 5 are useful in deciding reachability in a GVASS [18]; however, they are inadequate in deciding infinite-R-reachability of a vector class in a GVASS. In order to decide infinite-R-reachability, when G is reduced to GVASS's $G^1, G^2, \ldots G^t$, we must ensure that all the CR-paths in the original GVASS G have equivalent CR-paths in the new GVASS's. (This is different from checking the second condition in Theorem 5 above, namely, there exists a CR-path from (q_1, x) to (q_s, y) *iff* there exists a CR-path from (q_1, x) to (q_s, y) in some G^i.) If each CR-path in the original GVASS is faithfully replicated in some new GVASS, then, the infinite-R-reachability property will be preserved in one of the new GVASS's. In subsequent sections, we show this replication using mappings from the states in GVASS G to the states of the new GVASS's.

Theorem 6 *In a GVASS G, if every member of its size set is* $(0,0,0)$ *and if G does not satisfy property θ, then there is no CR-path from (q_1, x) to (q_s, y).*

Theorem 7 *It is effectively decidable whether a GVASS, G, satisfies property θ.*

From Theorems 4, 5, 6, and 7 it follows that the existence of an R-path in a VASS from an initial configuration to a final configuration is decidable.

(Note: Following [18], we use the subscript notation G_i to refer to the ith component VASS of GVASS G, and the superscript notation G^i to refer to the new ith GVASS obtained.)

7 Modifications To The Proof Of Reachability Of VASS

Given a vector class $\mathcal{V}^{W,W'}$ (Recall that $W, W' \subseteq \{1, 2, \ldots, n\}$, $W \bigcap W' = \phi$, and W, W' are not both empty) consider a new property θ' that is the same as θ except for condition θ_1, which is replaced by

$\theta_1{}'$: For every $m \geq 1$, there exists a $y \in \mathcal{V}^{W,W'}$ and a *cr*-path from (q_1, x) to (q_s, y) such that

$\theta_1(a)'$: every arc in every G_i is used at least m times, and

$\theta_1(b)'$: for every i and j, if $j \notin S_i$ then $\Pi_j(g_i^a) \geq m$, and if $j \notin S_i{}'$ then $\Pi_j(g_i^b) \geq m$.

Note the difference between condition θ_1 in property θ and condition $\theta_1{}'$ in in property θ'. Condition θ_1 is for a fixed output vector y, whereas condition $\theta_1{}'$ applies to any output vector y that belongs to the vector class $\mathcal{V}^{W,W'}$. Note also that the output constraint vector for the final VASS in a GVASS $(S_s{}')$ need not be related to the the vector class $\mathcal{V}^{W,W'}$. This is discussed in Section 7.1.1.

The following theorem helps us in deciding whether $(q_s, \mathcal{V}^{W,W'})$ is infinitely-R-reachable from some initial configuration (q_1, x).

Theorem 8 *For a GVASS G, and a vector class $\mathcal{V}^{W,W'}$, if G satisfies property θ', and, if there exists some VASS G_i of G such that size of $G_i \neq (0,0,0)$, then, $(q_s, \mathcal{V}^{W,W'})$ is infinitely-R-reachable from some initial configuration (q_1, x). If G satisfies property θ', and, if all VASS's of G have size $(0,0,0)$, then, $(q_s, \mathcal{V}^{W,W'})$ is finitely-R-reachable from (q_1, x).*

Proof: The proof is similar to the proof of Theorem 4. Let k_i be the number of arcs in G_i and let $k = \sum_{i=1}^{s} k_i$. Any *cr*-path from (q_1, x) to (q_s', y) can be mapped into a $(2sn + k)$-tuple

$$(g_1^a, g_1^b, g_2^a, g_2^b, \ldots g_s^a, g_s^b, e_1, e_2, \ldots e_k)$$

where e_i is the number of times the corresponding arc is traveled in the *cr*-path from g_1^a to g_s^b and g_i^a, g_i^b's are the intermediate input and output vectors. For a *cr*-path p, we denote the above tuple by $\Pi^e(p)$ and call it the the "extended folding" of p. Let

$$L(G) = \{\Pi^e(p) | p \text{ is a cr-path from}(q_1, x) \text{ to } (q_s, \mathcal{V}^{W,W'})\}$$

$L(G)$ is a semilinear set since it has a Presburger formulation. Since G satisfies property $\theta_1{}'$, by Lemma 3, $L(G)$ contains a linear subset of the form $L(c;p)$, denoted by $\hat{L}_G$, where p has nonzero components corresponding to every unconstrained coordinate and every arc, and zero components corresponding to every constrained coordinate. Project $\hat{L}_G$ into G_i, *i.e.* consider $\hat{L}_G(i)$. Every element of $\hat{L}_G(i)$ is in N^{2n+k_i} and is an extended folding of a subpath of a *cr*-path from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$. By Lemma 2, $\hat{L}_G(i)$ can be written as $L((x^i, y^i, z^i); (\Delta x^i, \Delta y^i, \Delta z^i))$ where $x^i, y^i, \Delta x^i$ and Δy^i are n-tuples and z^i and Δz^i are k_i-tuples. By Lemma 2, we can infer that for every j,

$$\Pi_j(\Delta x^i) \geq 1 \quad \text{if and only if} \quad j \notin S_i,$$

and

$$\Pi_j(\Delta y^i) \geq 1 \quad \text{if and only if} \quad j \notin S_i{}',$$

and $\Delta x^i \geq \bar{1}$. As in proof of Theorem 4, we conclude that conditions 1, 2, 3, and 4 of Theorem 3 are satisfied with

$$q_1 = q_i, x = x^i, q_2 = q_i{}', y = y^i, \Delta x = \Delta x^i, \Delta y = \Delta y^i, \Delta_1 = \Delta_i, \Delta_1 = \Delta_i{}'$$

and $A = R_i$. Thus, we can conclude that there exists J such that

$$(\forall i)(\forall j \geq J)((q_i{}', y^i + j \Delta y^i) \in R(q_i, x^i + j \Delta x^i))$$

If for some G_i, $SS(G_i) \neq (0,0,0)$, then either $\overline{S_i}$ or $\overline{S_i}'$ is non-empty or there is at least one arc in G_i. If either $\overline{S_i}$ or $\overline{S_i}'$ is non-empty, then we can increase the coordinates specified by them by choosing $j = J, J+1, \ldots$, obtaining infinitely many CR-paths from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$. Also, if both $\overline{S_i}$ and $\overline{S_i}'$ are empty but the number of arcs of G_i is greater than zero, then because of condition $\theta_1(b)'$, for $j = J, J+1, \ldots$, we have CR-paths from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ such that the number of times the arcs in G_i are traveled increases. This yields infinite CR-paths from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$. If all G_i's have size $(0,0,0)$, then, trivially, there is a single path from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ (in this case, only one vector in $\mathcal{V}^{W,W'}$ is CR-reachable from (q_1, x)). Hence, in this case, $(q_s, \mathcal{V}^{W,W'})$ is finitely-R-reachable from (q_1, x). $\Box$

The theorem below asserts that it is decidable whether a GVASS satisfies property θ'. The proof is the same as the proof presented for Theorem 7 in [18].

Theorem 9 *It is effectively decidable whether a GVASS, G, satisfies property θ'.*

7.1 Reducing The Size Of A GVASS

We now show that if a GVASS G does not satisfy property θ' and not all of its triples are $(0,0,0)$, then G can be replaced by finitely many GVASS's $G^1, G^2, \ldots, G^t$. Each G^i will be obtained from G by a transformation which replaces some VASS G_j in G (whose size is different from $(0,0,0)$) by finitely many VASS's each of size less than G_j. The transformation will depend on the constituent property of θ' (*i.e.*, one of $\theta_1(a)', \theta_1(b)', \theta_2(a)'$, or, $\theta_2(b)'$) that is not satisfied by G.

Figure 4 shows the general scenario when GVASS G is transformed into a new GVASS G^i.

Intuitively, a reduction will partition the CR-paths in G into CR-paths in G^i's, $1 \leq i \leq t$, such that a CR-path in G will be a CR-path in some G^i and CR-paths in all G^i's will also be CR-path's in G. Of course, since VASS G_j is being replaced by finitely many VASS's, each of size less than G_j, there needs to be a notion of equivalence between CR-paths in G and G^i's.

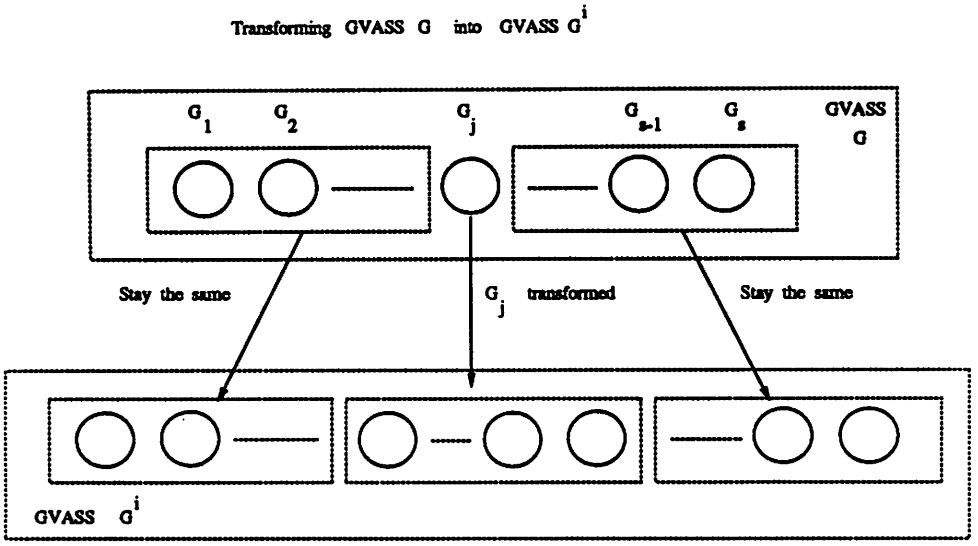


Figure 4: Reducing a GVASS

With this in mind, for every transformation, we define a one-to-many mapping M_i

$$M_i : \mathrm{St}(G) \rightarrow \mathrm{St}(G^i)$$

from the states of G onto states of G^i. The M_i's will be designed such that the states of the VASS's in G which are not changed by the transformation are mapped by M_i to their corresponding states in the transformed GVASS G^i; however, the states of the VASS G_j (which is replaced by finitely many VASS's in G^i (refer Figure 4)) will be mapped onto the states of the VASS's replacing G_j.

Under certain specific transformations (see Section 7.1.4), for the purposes of establishing path equivalence in G and G^i, a state of G^i can be ignored. To identify this state (if it exists), we define an *Ignore Set* Y_i, which is a set containing either one state of G^i or is empty. The new GVASS constructed under these transformations will be such that any CR-path will have at most one occurrence of the state in Y_i (provided Y_i is not empty).

We also define a one-to-one mapping I_i from the initial configuration (q_1, x) of G to an initial configuration of G^i and a one-to-one mapping F_i from the final configuration $(q_s, \mathcal{V}^{W,W'})$ to a final configuration of G^i.

For any path p in a GVASS, we let $S(p)$ denote the sequence of states along p. $S^i(p)$ denote the ith state in this sequence. For a CR-path p in G^i, we let $[S\backslash Y_i](p)$ denote the sequence of states which is the same as $S(p)$ but in which the state in Y_i is removed (Note: The number of states in $[S\backslash Y_i](p)$ will be less by one than the number of states in $S(p)$ if and only if Y_i is not empty and if the state in Y_i appears in $S(p)$). Let us denote the sequence of vectors in CR-path p (including the initial vector and the final vector) by $V(p)$. If Y_i is empty, we define $[V \backslash Y_i](p) = V(p)$; otherwise, $[V \backslash Y_i](p)$ is the sequence of vectors after removing the vector in $V(p)$ preceding the vector along state Y_i in p. (Note: We will design transformations that have the following property: When Y_i is not empty, the state in Y_i will appear exactly once on all CR-paths from the initial configuration of G^i to its final configuration.)

For a given M_i and Y_i, we say a CR-path $p \in G$ *corresponds* to a CR-path $p' \in G^i$ *iff* for all j, $[S \backslash Y_i]^j(p') \in M_i(S^j(p))$. (Note:If a path p in G corresponds to a path p' in G^i, then the number of states in p equals (is less by one than) the number of states in p' if and only if Y_i is empty (not empty)). Further, we say a CR-path $p \in G$ is *equivalent* to a CR-path $p' \in G^i$ *iff* p corresponds to p' and $[V \backslash Y_i](p') = V(p)$.

Intuitively, the M_i's will be designed to preserve path equivalence in G and G^i's. Since every transformation from G to G^i replaces some VASS G_j in G, and does not change any of the other VASS's in G, the mapping M_i $(1 \leq i \leq t)$ will map the states of all unchanged VASS's in G onto their corresponding states in G^i. The states of G_j (the VASS of G being replaced), however, will be mapped by M_i onto the states of the newly created VASS's in G^i such that every CR-path in G will have an equivalent CR-path in some G^i and vice-versa.

In subsequent sections, we consider different types of reductions for each of the cases when the conditions specified in property θ' fail. For each such reduction, we will prove the following theorem:

Theorem 10 *Let a reduction reduce GVASS G into t GVASS's $G^i, 1 \leq i \leq t$. There exists M_i's, Y_i's, I_i's and F_i's $(1 \leq i \leq t)$ such that the following property holds:*

Property 1 (Mapping Proposition) *Let p be any CR-path from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ in G. There exists a CR-path p' in some $G^i, 1 \leq i \leq t$, such that p and p' are equivalent. Also, for every CR-path p' from $(I_i(q_1, x))$ to $(F_i(q_s, \mathcal{V}^{W,W'}))$ in some $G^i, 1 \leq i \leq t$, there exists a CR-path $p \in G$ from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ such that p and p' are equivalent.*

7.1.1 Relationship Between W' and S_s'

Before proceeding to show different reductions, we make an important observation. So far we have made no assumptions regarding the relationship between $\mathcal{V}^{W,W'}$ and the output constraint vector for the final VASS in a GVASS G (S_s'). However, it is clear that if W' is not empty and S_s' is zero for all coordinates specified by W', then $(q_s, \mathcal{V}^{W,W'})$ is not R-reachable (and hence not infinite-R-reachable) from (q_1, x), since, a vector in $\mathcal{V}^{W,W'}$ cannot satisfy output constraint vector S_s'. (In fact, the set L_G, the set of foldings of cr-paths from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ is empty).

When G is "reduced" to GVASS's $G^1, G^2, \ldots, G^t$, it is possible that the output constraint vector of the final VASS in one of the G^i's may be different from the output constraint vector of the final VASS of G (coordinates which had an ω could be replaced by a fixed non-negative integer). If the output constraint vector of the final VASS of one of the G^i's becomes zero for all coordinates specified by W', then there does not exist a CR-path in that G^i from $(I_i(q_1, x))$ to $(F_i(q_s, \mathcal{V}^{W,W'}))$. However, this does not imply anything regarding existence of a CR-paths from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ in G. Only if all G^i's are such that there does not exist a CR-path in them from $(I_i(q_1, x))$ to $(F_i(q_s, \mathcal{V}^{W,W'}))$ will there be no CR-path in G from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$.

7.1.2 Reduction When $\theta 1(a)'$ Fails

We now consider different types of reduction when each of the sub-conditions specified in θ' fails to hold.

Suppose $\theta 1(a)'$ does not hold for some GVASS G. Let G contain s VASS's denoted by $G_1, G_2, \ldots, G_s$. Following the arguments in [18], we can write L_G (the set of foldings of cr-paths from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ as

$$L_G = L_1 \cup L_2 \cup \ldots \cup L_\beta$$

where $L_i = L(c_i, p_{i1}, p_{i2}, \ldots, p_{i\gamma_i})$. For each $i, 1 \leq i \leq \beta$, there exists an i_0 such that the i_0th component of $p_{i1} + p_{i2} + \ldots + p_{i\gamma_i} = 0$, and this i_0th component corresponds to some arc. Let this arc be $e = (q, q')$, and let it be in some VASS G_α of G. Note that $\Pi_{i_0}(c_i)$ specifies the number of times e in G_α gets used on each cr-path whose extended folding is in L_i. Let $G_{\alpha-e}$ be the VASS obtained by removing arc e from G_α. Consider the new GVASS G^i in which G_α is replaced by $\Pi_{i_0}(c_i) + 1$ copies of $G_{\alpha-e}$. Edge e connects state q in a previous copy to state q' of the next copy. Let e' be the connecting arc from state q'' in $G_{\alpha-1}$ to state q''' in G_α. There is a connecting arc e' from state q'' in $G_{\alpha-1}$ to state q''' of the first copy of $G_{\alpha-e}$ in G^i (this step can be ignored if G_α is the first VASS in G). If G_α is not the final GVASS, then we similarly connect the arc in G from G_α to $G_{\alpha+1}$ now from the last copy of $G_{\alpha-e}$ to $G_{\alpha+1}$ in G^i. The input constraint of the first copy is V_α, same as the input constraint of G_α and the output constraint of the last copy is $V_\alpha{}'$, same as the output constraint of G_α. For each of the copies, the remaining input and output constraints are identical (say B) and are given below:

$$\Pi_j(B) = \begin{cases} \omega & \text{if } j \notin R_\alpha \\ \Pi_j(V_\alpha) & \text{otherwise} \end{cases}$$

Essentially, VASS's 1 thru $\alpha - 1$ and $\alpha + 1$ thru s are not affected. Also $G_{\alpha-e}$ has the same number of states as G_α. If s is a state in G_α let us denote by s^i the corresponding state in the ith copy of $G_{\alpha-e}$. Let us denote the above reduction as $\mathcal{R}_1$. Note also that the size each of the $G_{\alpha-e}$ is less than the size of G_α (the second component, the number of arcs in a VASS is one less for each $G_{\alpha-e}$ than that of G_α.)

We now show that for the following mappings, Theorem 10 holds for $\mathcal{R}_1$:

$$M_i(s) = \begin{cases} s & \text{if } s \text{ is a state in some VASS } G_i,\ i \neq \alpha \\ \{s^i \mid 1 \leq i \leq \Pi_{i_0}(c_i)+1\} & \text{otherwise} \end{cases}$$

Y_i is empty, I_i maps the initial configuration (q_1, x) to $(M_i(q_1), x)$, and F_i maps the final configuration $(q_s, \mathcal{V}^{W,W'})$ to $(M_i(q_s), \mathcal{V}^{W,W'})$.

Proposition 5 *Under Reduction $\mathcal{R}_1$, and for the mappings defined above, Property 1 holds.*

Proof: Let p be a CR-path in G from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$. Let $p \in L_i$. As explained earlier, there exists i_0, such that the i_0th component of the sum of periods of L_i is 0, and i_0 corresponds to some arc e. Let e connect states q and q' of G_α in G. In path p, arc e is used exactly $\Pi_{i_0}(c_i)$ times. Let p be denoted by the following sequence of states

$$s_1 \to s_2 \to \dots q \xrightarrow[1]{e} q' \dots q \xrightarrow[2]{e} q' \dots q \xrightarrow[\Pi_{i_0}(c_i)]{e} q' \dots q_s$$

Consider the following path p' in G^i:

$$s_1 \to s_2 \to \dots q^1 \xrightarrow[1]{e} q'^2 \dots q^2 \xrightarrow[2]{e} q'^3 \dots q^{\Pi_{i_0}(c_i)} \xrightarrow[\Pi_{i_0}(c_i)]{e} q'^{\Pi_{i_0}(c_i)+1} \dots q_s$$

Clearly, p' is equivalent to p. The proof in the other direction is trivial. □

7.1.3 Reduction When $\theta 1(b)'$ Fails

We now consider the case when $\theta 1(b)'$ fails (Reduction $\mathcal{R}_2$). Following the arguments in [18], L_G can be written as

$$L_G = L_1 \cup L_2 \cup \dots \cup L_\beta$$

where $L_i = L(c_i, p_{i1}, p_{i2}, \dots, p_{i\gamma_i})$. Now, for each i, $1 \leq i \leq \beta$, there exists an i_0 such that the i_0th component of $p_{i1} + p_{i2} + \dots + p_{i\gamma_i} = 0$, and this i_0th component corresponds to some unconstrained input or output coordinate in some G_α. $\Pi_{i_0}(c_i)$ specifies the required fixed value of that coordinate which had previously the value ω. Replace ω in the corresponding constraint vector (input or output) by $\Pi_{i_0}(c_i)$. Let this modified GVASS be G^i. The size of G^i is less than the size of G because of the last component of the size vector of G_α in G^i is one less than that of G_α in G. The state structure of G^i is the same as that of G. Let M_i map states of G to corresponding states in G^i. Let Y_i be empty, I_i map the initial configuration (q_1, x) to $(M_i(q_1), x)$ and F_i map the final configuration $(q_s, \mathcal{V}^{W,W'})$ to $(M_i(q_s), \mathcal{V}^{W,W'})$.

Proposition 6 *Under Reduction $\mathcal{R}_2$, and for the mappings defined above, Property 1 holds.*

Proof: The proof follows on the same lines as the proof of the earlier proposition. □.

7.1.4 Reduction When Either $\theta 2(a)'$ Or $\theta 2(b)'$ Fails

The cases when $\theta 2(a)'$ or $\theta 2(b)'$ fails can be handled similarly and we show the reduction when $\theta 2(a)'$ fails (Reduction $\mathcal{R}_3$). Let $\theta 2(a)'$ fail for some G_α. Recall that S_α, R_α represent the set of constrained input coordinates and set of rigid coordinates respectively and v_α represents the input constraint vector V_α with ω's replaced by zero. Following the arguments in [18], we can conclude that a constant c can be effectively computed such that every point SR-reachable with respect to $S_\alpha - R_\alpha$ from (q_α, v_α) has its ith component value $\leq c$ for some $i \in S_\alpha - R_\alpha$. We shall treat the reduction as $|S_\alpha - R_\alpha|$ cases, one corresponding to each element in $S_\alpha - R_\alpha$. For each case, we shall modify G_α into at most $c+1$ new G_α's. In total, we generate at most $|S_\alpha - R_\alpha|(c+1)$ GVASS's.

Select any i in $S_\alpha - R_\alpha$. We make two cases:

Case 1: Suppose $\Pi_i(V_\alpha') = \omega$. Replace ω by $0, 1, 2, \ldots, c$ each giving a new G_α. This results in $(c+1)$ new GVASS's $G^1, G^2, \ldots, G^{c+1}$. As in Reduction 2 ($\mathcal{R}_2$), the state structure of each of the new GVASS's is the same. Let the mappings M_i's, Y_i's, I_i's and F_i's be the same as in case of $\mathcal{R}_2$. Let us denote this reduction by $\mathcal{R}_3$.

Proposition 7 *Under Reduction $\mathcal{R}_3$, and for the mappings defined above, all CR-paths in G from (q_1, x) to $(q_s, V^{W,W'})$ which satisfy the property that when the path is inside G_α its ith coordinate is $\leq c$ have equivalent CR-paths in some G^i and every CR-path in any G^i has an equivalent CR-path in G.*

Proof: The same as proof of Proposition 6 and 5. □.

Case 2: Suppose $\Pi_i(V_\alpha') = b$. Let $\Pi_i(V_\alpha) = a$. Let G_α have g states. We replace G_α by two new VASS's denoted by G_α' and H respectively. G_α' has a total of $g(c+1)$ states and VASS H has one state (which is also denoted by H since there is no ambiguity). A state of G_α' has as its label a pair (q, η) where q is a state of G_α and $0 \leq \eta \leq c$. There exists an arc labeled u from (q, η) to (q', η') *iff* in G_α there is an arc labeled u from q to q' and $\eta' = \eta + \Pi_i(u)$. The input state of G_α' is (q_α, a).The input constraint of G_α' is V_α, the set of rigid coordinates $R_\alpha \cup \{i\}$. The output constraint of G_α' is the V_α' with the ith coordinate being a instead of b. Let us denote this by V_α''. Thus,

$$\Pi_j(V_\alpha'') = \begin{cases} a & \text{if } j = i \\ \Pi_j(V_\alpha') & \text{otherwise} \end{cases}$$

The n-tuple label on z is given by

$$\Pi_j(z) = \begin{cases} b-a & \text{if } j=i \\ 0 & \text{otherwise} \end{cases}$$

The input and output constraint of VASS H is $V_\alpha{}'$ and the set of rigid coordinates of H is $R_\alpha \cup \{i\}$. The first component of the sizes of $G_\alpha{}'$ and H is one less than that of G_α. Let us denote this new GVASS by G'. Call this reduction $\mathcal{R}_4$.

Consider the following mapping M for the above reduction

$$M_i(s) = \begin{cases} s & \text{if } s \text{ is a state in some VASS } G_i,\ i \neq \alpha \\ \{(s,t) | 0 \leq t \leq c\} & \text{if } i = \alpha \text{ and } s \neq q_\alpha{}' \\ \{(s,t) | 0 \leq t \leq c\} \cup H & \text{if } i = \alpha \text{ and } s = q_\alpha{}' \end{cases}$$

$Y = \{H\}$, I_i maps the initial configuration (q_1, x) to $(M_i(q_1), x)$. If $\alpha = s$ (*i.e.* G_α is the final VASS of G, then F_i maps $(q_s, \mathcal{V}^{W,W'})$ to $(H, \mathcal{V}^{W,W'})$, else it maps $(q_s, \mathcal{V}^{W,W'})$ to $(M_i(q_s), \mathcal{V}^{W,W'})$.

Proposition 8 *Under Reduction $\mathcal{R}_4$, and for the mappings defined above, all CR-paths in G from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ which satisfy the property that when the path is inside G_α its ith coordinate is $\leq c$ have equivalent CR-paths in G' and every CR-path in any G' has an equivalent CR-path in G.*

Proof: Suppose p is a CR-path from (q_1, x) to $(q_s, \mathcal{V}^{W,W'})$ in G such that when the path is inside G_α its ith coordinate is $\leq c$. Consider the portion of this path inside G_α and let us denote it by the state sequence below

$$q_\alpha \rightarrow l_1 \rightarrow l_2 \rightarrow \ldots \rightarrow l_m \rightarrow q_\alpha{}'$$

Let e^i denote the edge in G_α between l_i and l_{i+1} with e^0 being the edge between q_α and l_1 and e^{m+1} being the edge between l_m and $q_\alpha{}'$. Let r_i denote the sum of the values in the ith coordinate for vectors that appear from states q_α up to l_i, with $r_0 = a$ and $r_{m+1} = b$ (this has to be the case since the path in question is CR-path and by definition a CR-path satisfies its input and output constraint. Since the ith coordinate of every vector on this CR-path is $\leq c$, all r_i's are $\leq c$. We can construct an equivalent portion of the above path in $G_\alpha{}'$ as shown below

$$(q_\alpha, a) \rightarrow (l_1, r_1) \ldots \rightarrow (l_i, r_i) \rightarrow \ldots (q_\alpha{}', b)$$

The other way round, *i.e.* if there is a CR-path in G' then it is also a CR-path in G is trivially true from the construction of G'. □.

Using Propositions 7 and 8 we conclude that under Reduction $\mathcal{R}_3$ Property 1 holds. Thus Theorem 10 is proved (Propositions 5, 6, 7 and 8). □.

Using Theorem 10, we can prove the following proposition:

Proposition 9 *$(q_s, \mathcal{V}^{W,W'})$ is infinitely-R-reachable from (q_1, x) in a GVASS G* iff *$(M_i(q_s), \mathcal{V}^{W,W'})$ is infinitely-R-reachable from $(M_i(q_1), x)$ for some G^i.*

Proof: Since a reduction transforms a GVASS G into finitely many GVASS's, if $(q_s, \mathcal{V}^{W,W'})$ is infinitely-R-reachable from (q_1, x) then for every such CR-path, there is an equivalent path from $(M_i(q_1), x)$ to $(M_i(q_s), \mathcal{V}^{W,W'})$ in some G^i (Theorem 10). Since there are only finitely many G^i's, one of the G^i's must be such that $(M_i(q_s), \mathcal{V}^{W,W'})$ is infinitely-R-reachable from $(M_i(q_1), x)$. The other direction is trivial. □

7.2 Decidability Of Strong stability

We are now in a position to prove Theorem 2. By Proposition 4, if there exists any state s that is bad (as defined in Definition 11), then program P is not strong stable for the given synchronizer.

We sketch below an algorithm to check if a given state of the synchronizer is bad. A state s is obviously not bad if $SLA(s) = \sum$. So we consider only those states for which $SLA(s) \neq \sum$, *i.e.* $SLA(s) \subset \sum$. We convert the synchronizer into a GVASS by replacing the label on each arc by a l-tuple as explained in Section 5.1. This GVASS contains a single VASS; the input constraint vector for this VASS is zero in all coordinates. The final vector class of interest for this GVASS is $\mathcal{V}^{U^s,\overline{U^s}}$ (Recall that U^s is the unsuitable vector class for state s). The output constraint vector has zeroes in the coordinates specified by U^s and ω's in all other coordinates.

We now construct a tree of GVASS's using the reductions described earlier. The root node of the tree is the above GVASS. When a GVASS at a node v does not satisfy property θ' (refer to Section 7), we construct GVASS's $G^1, G^2, \ldots G^t$ by using the reductions described earlier. We create t children for node v and attach GVASS G^i to the ith child.

We continue this process of expansion until all the leaf nodes contain a GVASS for which one of the three conditions below hold:

1. All the component VASS's have size $(0,0,0)$. In this case, following arguments in [18], we can decide if there exists a CR-path from $(q_0, \overline{0})$ to $(s, \mathcal{V}^{U^s,\overline{U^s}})$. We collect all such leaf nodes for which the above CR-path exists into set Z_1.

2. Property θ' is satisfied and conditions for infinite-R-reachability specified in Theorem 8 hold. We collect all such leaf nodes into set Z_2.

We prove the following proposition.

Proposition 10 *State s is bad (refer Definition 11)* iff

1. $a_1 \in SLA(s)$ and set Z_2 is not empty (Condition 1); OR

2. $a_1 \notin SLA(s)$ and any one of set Z_1 or Z_2 is not empty (Condition 2).

Proof: We prove that conditions B_1, B_2 and B_3 in Definition 11 imply conditions 1 or 2 above and vice-versa. Firstly, as explained earlier, a state s is good if $SLA(s) = \sum$. Therefore it suffices to consider only those states for which $SLA(s) \subset \sum$. For such a state s, if B_1 holds, then our algorithm will terminate with set Z_2 non-empty (follows from Theorem 8 and 10). Thus Condition 1 holds. If B_2 holds, then by Definition 11, a_1 does not belong to $SLA(s)$. Also, from Theorem 8 and 10 set Z_2 will be non-empty. Thus Condition 2 holds. If B_3 holds, then obviously set Z_1 will be non-empty.

To prove in the other direction, consider a state s for which $SLA(s) \subset \sum$. We make a case analysis.

Case 1: Suppose $a_1 \in SLA(s)$. If Condition 1 holds when the algorithm terminates, then from Theorem 8, B_1 holds for state s. (Note: Since a_1 belongs to $SLA(s)$ Condition 2 cannot arise.)

Case 2: $a_1 \notin SLA(s)$. If set Z_1 is non-empty then B_3 holds for s. If set Z_2 is non-empty and $U(s) \subset \{1, 2, \ldots, l-1\}$ B_1 holds. If set Z_2 is non-empty and $U(s) = \{1, 2, \ldots, l-1\}$ B_2 holds. □.

This proves Theorem 2. Now assuming all states are good, we show how to determine the strong stability constant k. (k has the following significance: k or more processes are strong safe for the given synchronizer.)

According to Definition 11, the negation of the condition $(B_1 \vee B_2 \vee B_3)$ gives the condition when a state is good. Expanding, we get the following condition for a good state:

1. $U^s = \{1, 2, \ldots l-1\}$ or $(s, \mathcal{V}^{U^s})$ is not infinitely-R-reachable from $(q_0, \bar{0})$ (Condition $\overline{B_1}$); AND

2. $U^s \subset \{1, 2, \ldots l-1\}$ or $(s, \mathcal{V}^{U^s})$ is not infinitely-R-reachable from $(q_0, \bar{0})$, or $a_1 \in SLA(s)$ (Condition $\overline{B_2}$); AND

3. $(s, \mathcal{V}^{U^s})$ is not finitely-R-reachable from $(q_0, \bar{0})$, or $a_1 \in SLA(s)$ (Condition $\overline{B_3}$);

Now, if $(s, \mathcal{V}^{U^s})$ is infinitely-R-reachable from $(q_0, \bar{0})$, then by condition $\overline{B_1}$ we have $U^s = \{1, 2, \ldots l-1\}$. From this condition it is clear that processes have finished executing their last request and there is no deadlock. (recall that if all coordinates $1, 2, \ldots l-1$ are zero implies that all the processes have finished executing.) Thus, for all the good states, we can ignore leaf nodes that belong to set Z_2. If for all the states, Z_1 is empty, then the strong stability constant is 1. To prove this, observe that if there exists a partial execution of more than one process which leads to a deadlock, then it contradicts Theorem 8 and 10 which ascertain that an equivalent path for the above path would have been found in one of the leaf GVASS's belonging to set Z_1.

So suppose that for some state, Z_1 is not empty. Let M be the maximum of the last coordinate in all the leaf GVASS's belonging to set Z_1 for each of the states. Recall that the last coordinate kept track of the number of occurrences of the first symbol (or number of processes needed to form that particular partial execution). Once again it is easy to observe that the strong stability constant is M.

We now prove decidability of full strong safety, which follows immediately from the decidability of strong stability.

Corollary 4 *Full strong safety is decidable.*

Proof: We present a procedure for deciding full strong safety of any concurrent system $(P, \mathcal{S})$. Given such a system, we first test if it is strong stable. Clearly, if $(P, \mathcal{S})$ is not strong stable, then it is not full strong safe. So suppose $(P, \mathcal{S})$ is strong stable. Using the decidability procedure for strong stability, we compute the strong stability constant k, such that for all $j > k$, $(P^j, \mathcal{S})$ is strong safe. Finally, we systematically test if concurrent system $(P^m, \mathcal{S})$ is strong safe for $1 \leq m \leq k$. Obviously, $(P, \mathcal{S})$ is full strong safe if and only if $(P^m, \mathcal{S})$ is strong safe for $1 \leq m \leq k$. □

So far we have assumed that the program for the processes should contain distinct symbols. The following corollary proves that both strong stability and full strong safety is decidable even if there are repeated symbols in the program. The proof is similar to the proof of Corollary 3.

Corollary 5 *For a program $P = a_1 a_2 \ldots a_l$ where the a_i's are not necessarily distinct, and a given synchronizer $\mathcal{S}$, strong stability (full strong safety) is decidable.*

7.3 How To Handle Different Processes

So far we have considered the case when the program for all the processes was the same. We now consider the case when processes can have different programs. We begin with the possibility of two distinct programs P_1 and P_2. We initially restrict all symbols in P_1 and in P_2 to be distinct, and we require that there be no symbol common to both P_1 and P_2. The synchronizer description is as before.

We first formulate questions concerning strong safety for this model. The definition below is similar to Definition 7 and describes when an ordered pair (k_1, k_2) is strong safe for a given synchronizer.

Definition 12 *For a given synchronizer $\mathcal{S}$, k_1 identical processes with program P_1 and k_2 identical processes with program P_2 are* strong safe iff *for all partial executions $p_1, p_2, \ldots, p_{k_1}$ of the first k_1 processes, for all partial executions*

$p_1', p_2', \ldots, p_{k_2}'$ of the next k_2 processes, and
$\forall \sigma \in (L(S) \bigcap PE(p_1, p_2, \ldots, p_{k_1}, p_1', p_2', \ldots, p_{k_2}'))$

$$\left(\left(\bigcup_{i=1}^{i=k_1} LA_{p_i}^{P_1}\right) \bigcup \left(\bigcup_{i=1}^{i=k_2} LA_{p_i'}^{P_2}\right)\right) \bigcap SLA(\delta(q_0, \sigma)) \neq \emptyset$$

We now define strong stability for this model. The definition is similar to Definition 8.

Definition 13 *Programs P_1 and P_2 are* strong stable *for a given synchronizer S iff there exists an ordered pair (k_1, k_2), such that, for all ordered pairs (k, k') with the property that either $k \geq k_1$ or $k \geq k_2$, k identical processes with program P_1 and k' identical processes with program P_2 are strong safe for S.*

We now show how to map the synchronizer into a VASS which simulates the partial executions of processes with these two programs. Let $P_1 = a_1 a_2 \ldots a_l$ and let $P_2 = b_1 b_2 \ldots b_t$. With each symbol of the synchronizer we associate a $((l-1)+(t-1)+1+1)$-tuple of integers. If the symbol belongs to the first program, then we set the first $(l-1)$ coordinates and the second last coordinate as explained in Section 5.1 and fill zeroes in the rest of the coordinates. Similarly, if the symbol belongs to the second program, then we set the $(t-1)$ coordinates after the first $l-1$ coordinates, and the last coordinate, as explained in Section 5.1 and fill zeroes in rest of the coordinates.

Let $g = l - 1$. For a state s, the unsuitable vector class U^s is now given by

$$U^s = \{i - 1 \,| a_i \in \{SLA(s) - \{a_1\}\}\} \cup \{(g + j - 1 | b_j \in \{SLA(s) - b_1\}\}$$

A state s is "bad" *iff*

1. $U^s \subset \{1, 2, \ldots l-1, g+1, g+2, \ldots g+t-1\}$ and $(s, \mathcal{V}^{U^s})$ is infinitely-R-reachable from $(q_0, \bar{0})$ (Condition D_1); OR
2. $U^s \supseteq \{1, 2, \ldots l-1\}$ and $(s, \mathcal{V}^{U^s})$ is infinitely-R-reachable from $(q_0, \bar{0})$, and $a_1 \notin SLA(s)$ (Condition D_2); OR
3. $U^s \supseteq \{g+1, g+2, \ldots g+t-1\}$ and $(s, \mathcal{V}^{U^s})$ is infinitely-R-reachable from $(q_0, \bar{0})$, and $b_1 \notin SLA(s)$ (Condition D_3); OR
4. $(s, \mathcal{V}^{U^s})$ is finitely-R-reachable from $(q_0, \bar{0})$, and $a_1 \notin SLA(s)$ and $b_1 \notin SLA(s)$ (Condition D_4)

A state is "good" if none of D_1, D_2, D_3, and D_4 are true.

Conditions D_1, D_2, D_3, and D_4 are an extension to the conditions in Definition 11. The proof of the proposition below is based on similar arguments to proof of Proposition 4.

Proposition 11 *Programs P_1 and P_2 are strong stable for a synchronizer S if and only if all states of S are "good".*

From Proposition 11 and from arguments presented for the proofs of Theorem 2, we get the following theorem.

Theorem 11 *Given programs P_1 and P_2 and a synchronizer S, strong stability is decidable.*

The assumption that all symbols in each of the programs be distinct and that there be no symbols common to both programs can also be removed as follows: First, for each common symbol b, replace it by a new distinct symbol b_1 in P_1 and by a new distinct symbol b_2 in P_2. For each arc labeled with resource b in S, replace it with two arcs b_1 and b_2. Now no symbol occurs in both the programs. If there are any repeated symbols in any of the programs, make them distinct by following the procedure outlined in the proof of Corollary 3. Let the transformed programs be P_1' and P_2' and the synchronizer be S'. It is easy to observe that P_1' and P_2' are strong safe for S' *iff* P_1 and P_2 are strong safe for S. The proof is similar to the argument presented in the proof of Corollary 3.

In an obvious way, we have the following extension to Theorem 11.

Corollary 6 *Given m programs $P_1, P_2, \ldots P_m$ (m fixed), and a synchronizer S, strong stability is decidable.*

8 The Gas-Station Customers Problem

So far, the model that we have considered involves one synchronizer and several user processes which have possibly distinct programs. Before we present our analysis of the Gas-Station Problem (GSP), we consider two types of enhancements to our model.

Enhancement $E1$: We allow the synchronizer to have a fixed number of internal variables which can take finitely many values. In addition to an operation symbol on each of the arcs, we allow a label of the form

$$\mathcal{P} \parallel \mathcal{A}$$

where

1. $\mathcal{P}$ is a predicate formed from equality assertions about the variables plus boolean connectives.
2. $\mathcal{A}$ is an action that specifies a sequence of assignment statements which change values of any of the variables.

The basic interpretation of such a label is as follows: The synchronizer can perform operation a provided it is in a state s such that $a \in SLA(s)$ and predicate $\mathcal{P}$ on the arc which is labeled with a is true. If it makes this transition, then the action specified by $\mathcal{A}$ is executed. An empty predicate or action is allowed. An empty predicate is always true and an empty action does not change the value of any variable.

It is easy to see that we can build an equivalent synchronizer in which the arcs are labeled with just the operator symbols by enumerating all possible states for each of the combination of possible values that the variables can take. Further, a program is strong stable for the original synchronizer if and only if it is strong stable for this new larger synchronizer.

Enhancement $E2$: Now suppose we have several synchronizers such that an execution of an operation in one of them invokes an execution of another operation in the other (Such a situation could arise if a synchronizer needs a resource controlled by another in order to execute one of its operations).

In order to model such an interaction, we label arcs as follows:

$$\mathcal{P} \,||\, \mathcal{R} \,||\, \mathcal{A}$$

where the $\mathcal{P}$ and $\mathcal{A}$ are as explained in Enhancement $E1$ and $\mathcal{R}$ is an operation (executed by some synchronizer). The interpretation now is that the synchronizer can perform operation a provided it is in a state s such that $a \in SLA(s)$, predicate $\mathcal{P}$ on the arc labeled with operation a is true, and another synchronizer executes the operation specified by $\mathcal{R}$. If $\mathcal{R}$ is not specified, it has the same interpretation as in Enhancement $E1$.

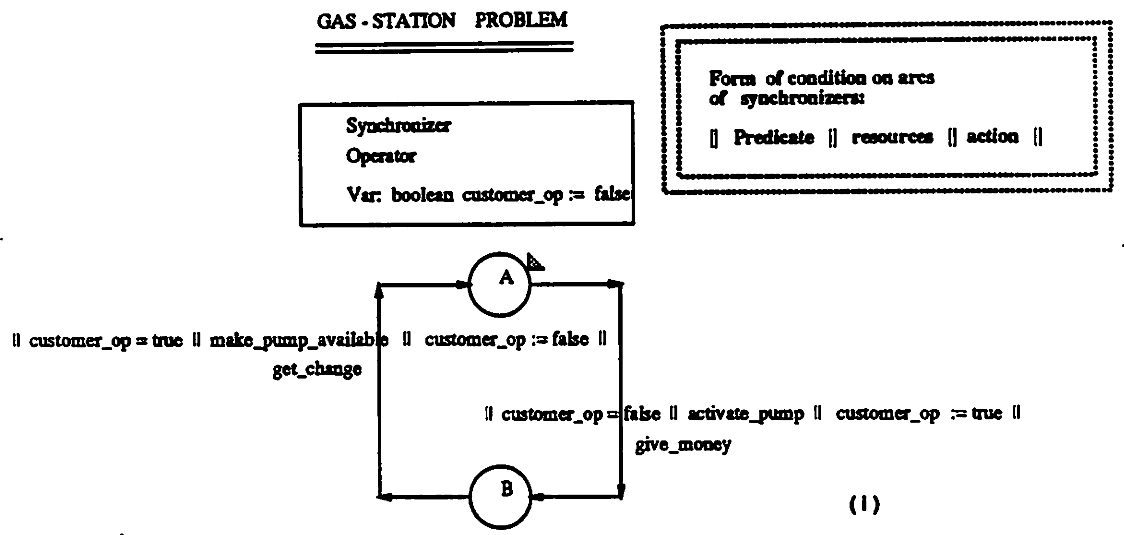
Provided there are no cyclic patterns of invocation of operations on the arcs (e.g. $\mathcal{R} = \{a\}$ for an arc with operation symbol b in synchronizer S_1, and, $\mathcal{R} = \{b\}$ for an arc with operation symbol a in synchronizer S_2) it is easy to construct an equivalent synchronizer whose arcs are of the form specified in Enhancement $E2$. A formal transformation is outlined in [22].

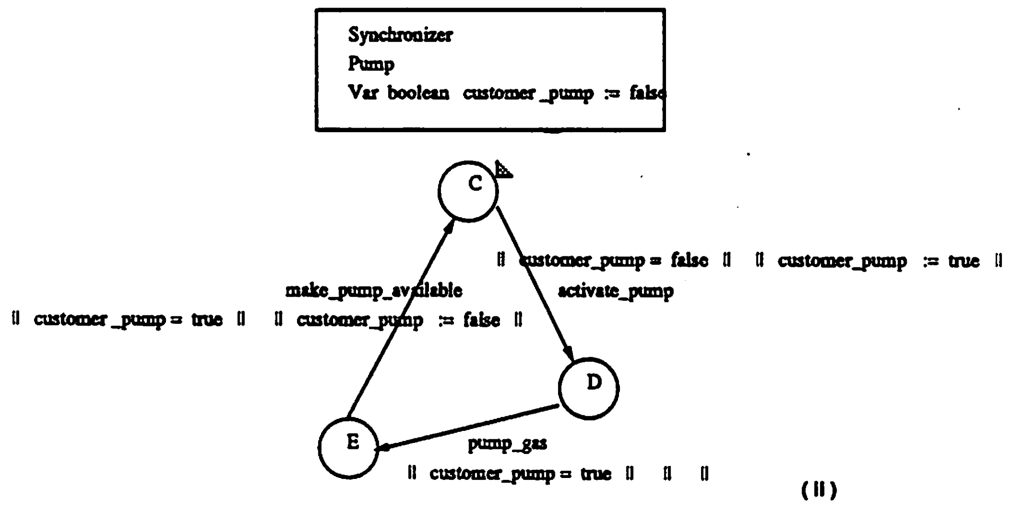
8.1 The Gas-Station Customer Problem

In GSP, customers line up to pump gas. They prepay money to an "operator". The operator activates a "pump". The customers then pump gas and leave after obtaining cash equal to the difference between what they prepaid and the amount they pumped.

In the spirit of presentation in [22], we use two synchronizers – "Operator" and "Pump" as shown in Figure 5 (i) and (ii) respectively to model this problem. $\sum$ is the set containing operations *give_money*, *get_change*, *activate_pump*, *pump_gas* and *make_pump_available*. The program for each of the customers (refer Figure 5 (iii)) is the string

$$\textit{give\_money} \quad \textit{pump\_gas} \quad \textit{get\_change}$$


GAS - STATION PROBLEM
Form of condition on arcs of synchronizers:
|| Predicate || resources || action ||
Synchronizer
Operator
Var: boolean customer_op := false
A
B
|| customer_op = true || make_pump_available || customer_op := false ||
get_change
|| customer_op = false || activate_pump || customer_op := true ||
give_money
(I)


Synchronizer
Pump
Var boolean customer_pump := false
C
D
E
|| customer_pump = false || || customer_pump := true ||
activate_pump
make_pump_available
|| customer_pump = true || || customer_pump := false ||
pump_gas
|| customer_pump = true || || ||
(II)

Program for Customers

Customer 1:	give_money	pump_gas	get_change
Customer 2:	give_money	pump_gas	get_change
Customer 3:	give_money	pump_gas	get_change
●	●	●	●
●	●	●	●
●	●	●	●

(III)

Figure 5: Gas-Station Customer Problem

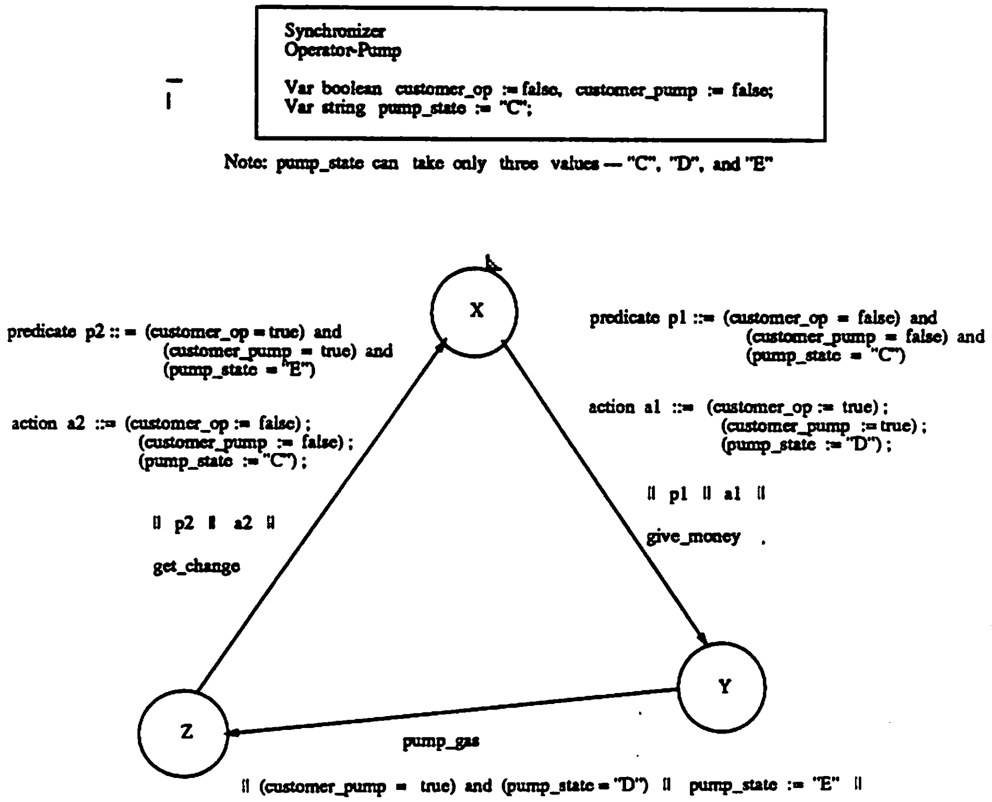

Synchronizer
Operator-Pump
Var boolean customer_op := false, customer_pump := false;
Var string pump_state := "C";
Note: pump_state can take only three values — "C", "D", and "E"
X
predicate p2 :: = (customer_op = true) and (customer_pump = true) and (pump_state = "E")
action a2 ::= (customer_op := false) ; (customer_pump := false) ; (pump_state := "C") ;
|| p2 || a2 ||
get_change
predicate p1 ::= (customer_op = false) and (customer_pump = false) and (pump_state = "C")
action a1 ::= (customer_op := true) ; (customer_pump := true) ; (pump_state := "D") ;
|| p1 || a1 ||
give_money
Y
Z
pump_gas
|| (customer_pump = true) and (pump_state = "D") || pump_state := "E" ||

Figure 6: Gas-Station Customer Problem for enhancement E1

The boolean variable *customer_op* in the synchronizer Operator keeps track of whether a customer is being serviced by the operator. The boolean variable *customer_pump* serves a similar purpose in synchronizer Pump.

This is a model which fits the description in Enhancement $E2$. We observe certain properties of this model before transforming it into a model that fits description $E1$. Firstly, the initial request to execute *give_money* from the first customer invokes the transition from state A to B in Operator. This in turn invokes the transition from state C to state D inside Pump, since $\mathcal{R} = \{activate\_pump\}$ for operation *get_change*. The first customer can then execute *pump_gas* (transition D to E in Pump). At this stage, no request for *give_money* can be satisfied. Hence the second customer is effectively blocked. The first customer then invokes *get_change* in Operator which in turn invokes the operation *make_pump_available* inside Pump. Thus, we are back at the two initial states in the two synchronizers.

We transform this model into an equivalent model which fits Enhancement

$E1$ as shown in Figure 6. (Note: A more formal approach to transform from the previous model to the model in Figure 6 is outlined in [22].) Synchronizer Operator-Pump replaces synchronizers Operator and Pump. In this synchronizer, we keep track of the state of synchronizer Pump in the variable *pump_state*. The resource *give_money* and *get_change* are now slightly "overloaded" resources. Requests to these resources internally take care of "activating" and releasing change via *make_pump_available*. Since these transitions are transparent to the customer process, they can be ignored. The predicates on this new synchronizer is a conjunction of the predicates of the two synchronizers in Figure 5 along with the predicate involving variable *pump_state*.

Thus our model of concurrency is powerful enough to model the GSP problem. Therefore, strong stability is decidable for the GSP problem (To decide strong stability, we need to check if all the states of Operator-Pump are "good" by applying the algorithm outlined in Section 7.2). Note that we have not modeled input and output parameters on requests for resources inside the synchronizer as in [22]. For the GSP problem, these parameters are used in [22] to obtain the exact amount of money that a customer gets from the pump operator for the amount prepaid for gas but not pumped. Such information is not relevant to the concurrency characteristics of the model and hence were ignored.

9 Conclusion

We have formulated questions regarding weak and strong stability for a very basic concurrency model in which a synchronizer controls resources and processes request resources from the synchronizers. We have shown for this model that weak and strong stability are decidable. The proof of weak stability is based on bounds on chain lengths on a particular sequence of vectors in the the positive orthant. The proof for strong stability is based on the the decidability of infinite-R-reachability of vector classes in a Vector Addition System with states. We also showed how to enhance the synchronizer and still answer decidability questions regarding the enhanced model by transforming the given synchronizer(s) into a single synchronizer of the proper form.

This work suggests number of directions for future research, including

- enhancing the model so that processes are finite state devices rather than strings.
- an efficient algorithm for judging if a state is good.
- expanding the work to consider systems in which the synchronizer grows in a regular way as the number of processes increase, e.g. as in the dining philosophers problem.

References

[1] A. Valmari. A Stubborn Attack On State Explosion. *Computer Aided Verification*, 1990.

[2] E. M. Clarke, Emerson, and Sistla. Automatic Verification Of Finite-state Concurrent Systems Using Temporal Logic Specifications. *10th ACM Symposium on Principles of Programming Languages*, 1983.

[3] E. M. Clarke, O. Grumberg, and M. C. Browne. Reasoning About Networks With Many Identical Finite State Processes. *Proc. 5th ACM Symposium on Principles of Distributed Computing*, August 1986.

[4] Ginzburg and Yoeli. Vector Addition Systems And Regular Languages. *Journal of Computer and System Sciences*, 1980.

[5] K. Ramamritham. Synthesizing Code For Resource Controllers. *IEEE Transactions on Software Engg*, August 1985.

[6] Louis Rosier and Hsu-chun yen. A Multiparameter Analysis Of The Boundedness Problem For Vector Addition Systems. *Journal of Computer and System Sciences*, 1986.

[7] Mahesh Girkar and Robert Moll. Bounds On Chain Lengths For Collections Of Non-negative Points In n-space. *Technical Report, Dept. of Computer Science, Univ. of Massachussetts*, April 1992.

[8] Mayr E. W. An Algorithm For The General Petri Net Reachability Problem. *Proceedings of the 13th ACM Symposium on Theory of Computing*, 1981.

[9] Milner R. A Calculus Of Communication Systems. *Lecture Notes in Computer Science*, 1980.

[10] Partice Godefroid. Using Partial Orders To Improve Automatic Verification Methods. *Proc. Workshop on Computer Aided Verification*, june 1990.

[11] Partice Godefroid and Pierre Wolper. Using Partial Orders For The Efficient Verification Of Deadlock Freedom And Safety Properties. Not Sure whether published or not.

[12] James L. Peterson. *Petri Net Theory And The Modeling Of Systems*. Pertice Hall, Inc. Englewood Cliffs, New Jersey, 1981.

[13] R. P. Kurshan. Modelling Concurrent Programs. *Symp. on Applied Mathematics*, 1985.

[14] R.P. Kurshan and K. McMillan. A Structural Induction Theorem For Processes. *Proc. Symp. on Eigth ACM Symposium on Principles of Distributed Computing*, August 1989.

[15] Rudiger Valk and Guy Vidal-Naquet. Petri Nets And Regular Languages. *Journal of Computer and System Sciences*, 1981.

[16] S. German and A. P. Sistla. Reasoning With Many Processes. *Proc. Symp. on Logic in Computer Science*, June 1987.

[17] S. German and A. P. Sistla. Reasoning About Systems With Many Processes. *JACM*, July 1992.

[18] S. Rao. Kosaraju. Decidability Of Reachability In Vector Addition Systems. *Proceedings of the 16th ACM Symposium on Theory of Computing*, May 1982.

[19] Sacerdote and Tenney. The Decidability Of Reachability In Vector Addition Systems. *Proceedings of the 9th ACM Symposium on Theory of Computing*, 1981.

[20] Tadoa Murata. Petri Nets, Properties, Analysis, And Applications. *Proceedings of IEEE*, 1988.

[21] Ugo Buy. *Automatic Synthesis Of Resource Sharing Concurrent Programs*. Phd. Thesis, University of Massachussetts, Amherst, 1990.

[22] Ugo Buy and Robert Moll. Analysis And Synthesis Of Inter-process Communication Code. To appear.