JOURNAL OF COMPUTER AND SYSTEM SCIENCES 32, 105–135 (1986)

A Multiparameter Analysis of the Boundedness Problem for Vector Addition Systems

LOUIS E. ROSIER AND HSU-CHUN YEN

Department of Computer Sciences, University of Texas, Austin, Texas 78712

Received November 16, 1984; revised July 9, 1985

Let VASS(k, l, n) denote the class of k-dimensional n-state *Vector Addition Systems with States*, where the largest integer mentioned, in an instance, can be represented in l bits. Using a modification of the technique used by C. Rackoff (*Theoret. Comput. Sci.* **6** (1978), 223–231) we show that the *Boundedness Problem*[1] (BP), for VASS(k, l, n), can be solved in $O((l+\log n)*2^{c*k*\log k})$ nondeterministic space. By modifying R. Lipton's ("The Reachability Problem Requires Exponential Space," Report No. 62, Department of Computer Sciences, Yale University, Jan. 1976) result, a lower bound is then shown of $O((l+\log n)*2^{c*k})$ nondeterministic space. Thus, the upper bound is optimal with respect to parameters l and n, and is nearly optimal with respect to the parameter k. This yields an improvement over the result of Rackoff, especially when compared with the lower bound of Lipton. This is because the lower bound, of $O(2^{c*k})$ space, was essentially given for VASS(k, 1, 1). Now Rackoff's corresponding upper bound, just for the instances of VASS(k, 1, 1) constructed by Lipton, is no better than $O(2^{c*k^2*\log k})$ space. (In general, it can get much worse.) Our result, however, yields an upper bound of $O(2^{c*k*\log k})$, over the entire class. We also investigate the complexity of this problem for small, but fixed, values of k. We show that the BP is PSPACE-complete for four-dimensional VASSs, and NP-hard for two-dimensional VASSs. The above results can then be extended for the case without states. In particular, we are able to show that the BP is NP-hard for VASS(3, l, 1) and PSPACE-complete for VASS(4, l, 1). Extensions to related problems (e.g., covering and reachability) are also discussed.

1. INTRODUCTION

In this paper, we investigate the complexity of the boundedness problem (BP) for *Vector Addition Systems* (VASs), or equivalently *Vector Addition Systems with States* (VASSs). This problem was first considered in [14], where it was shown to be decidable. The algorithm presented there was, however, basically an unbounded search and consequently no complexity analysis was shown. Subsequently, in [17], a lower bound of $O(2^{c*m})$ space was shown, where m represents the dimension of the problem instance (and c is some constant). Finally, an upper bound of $O(2^{c*n*\log n})$ space was given in [23]. Here, however, n represents the size or number of bits in the problem instance. A close analysis of the result in [17] reveals, in

[1] More precisely, the problem of deciding whether a system is unbounded.

0022-0000/86 $3.00

terms of n, a lower bound of $O(2^{c^*\sqrt{n}})$ space, since the size of the systems constructed in [17] required $O(m^2)$ bits.

The VASS model was introduced in [10], where it was shown that a $k+3$-dimensional VAS could simulate a k-dimensional VASS. Although, the addition of states adds no expressive power to the notion of VAS, it does add to the functional ease in which these systems can be used to model processes. For example, these systems can be used to simulate or model networks of *Communicating Finite State Machines* (CFSMs) (cf. [4, 6, 7, 24, 27]), where each machine is constrained to be able to send only a single type of message to every other machine in the network. (A network of CFSMs consists of two or more finite state machines each pair of which communicate by sending and receiving messages via two one-directional, potentially unbounded, FIFO channels.) The BP for such networks of CSFMs was first considered in [4], where an unbounded search algorithm, similar to the one in [14], was presented. A faster algorithm was later shown for networks of 2 such CFSMs in [27]. As it turns out, it is reasonably easy to show that such a network of k n-state CFSMs can be simulated by an $O(k^2)$-dimensional $O(n^k)$-state VASS. (In fact, the maximum size of any integer mentioned in the definition of the simulating VASS need be no more than 1.) Likewise, a k-dimensional n-state VASS, where the maximum size of any integer mentioned in the VASS is 1, can be simulated by such a network of CFSMs. The simulating network requires no more than $O(\sqrt{k})$ machines where the number of states in each machine is bounded by a polynomial in n and 2^k.

In [8], the deadlock detection problem was considered for the class of networks consisting of k CFSMs where each machine is constrained to send only a single type of message (hereafter this class of networks is referred to as CFSM(k)). In a later paper enlarging upon these results, the BP for CFSM(k) was also considered [6]. It was noted there that the result of [23] could be used to show that the BP for CFSM(k) was solvable in PSPACE, providing k was considered to be a fixed known constant. Here we show that, in such cases, the result can be improved to PTIME.

In this paper, we consider the complexity of the BP for VASSs with respect to the following three natural numeric parameters of a given instance:

- the dimension,
- the maximum size of any integer mentioned in the description, and
- the number of states.

For ease of expression, we let VASS(k, l, n) denote the class of k-dimensional n-state VASSs where the largest integer mentioned, in an instance, can be represented in l bits. In Section 2, we use a modification of Rackoff's technique to show that the BP, for VASS(k, l, n), can be solved in $O((l+\log n)^*2^{c^*k^*\log k})$ nondeterministic space. (All of our results are expressed with respect to nondeterministic complexity classes unless otherwise stated. This could also be said for [17, 23], although in their case such algorithms can be determinized without altering the complexity

class, as only the constant c gets changed [26]. Our subsequent discussion, however, involves subexponential complexity classes where this is not necessarily the case. Hence, it is important in our analysis that we consider the problem of deciding whether systems are unbounded rather than bounded.) This offers an improvement over the result in [23], especially when compared with the best known lower bound. This is because the lower bound, of $O(2^{c*k})$ space, was essentially given for VASS(k, 1, 1). Now the corresponding upper bound from [23], just for the instances of VASS(k, 1, 1) considered in [17], is no better than $O(2^{c*k^{2*}\log k})$ space. (In general, it can get much worse.) Our result, however, yields an upper bound of $O(2^{c*k*\log k})$, over the entire class.

Whenever k is a fixed known constant, however, the algorithm requires at most nondeterministic linear space in terms of l. In Section 3, we show that the BP is PSPACE-complete for four-dimensional VASSs. Although we are unable to provide the corresponding result for either two- or three-dimensional VASSs, we are able to show that the problem is NP-hard for these classes. The above results can then be extended for the case without states. In particular, we are able to show that the BP is NP-hard for VASS(3, l, 1) and PSPACE-complete for VASS(4, l, 1). Consequently, it is unlikely that the problem can be solved in PTIME with respect to the parameter l. When $k = 1$, however, the BP can be solved in PTIME. When both k and l are considered to be fixed known constants the problem becomes NLOGSPACE-complete. Hence, the complexity of the algorithm, given in Section 2, is unlikely to be improved with respect to the parameter n either. Now recall that for each member of CFSM(k), the simulating VASS is such that the parameter l is 1. Hence, the BP for CFSM(k) can be solved in NLOGSPACE (and consequently PTIME [3]), providing k is considered to be a fixed known constant.

Now in the last section, we show how the proof of [17] can be augmented to obtain a lower bound of $O((l+\log n)*2^{c*k})$ nondeterministic space for VASS(k, l, n). In order to do this, we define a class of problems each of which has three natural numeric parameters, say k, l, and n, whose nondeterministic space complexity is bounded by $2^{k}*(l+\log n)$. We then show that the BP for VASS(k, l, n) is as "hard" as any problem in this class with respect to a certain type of reducibility. Our goal here is to provide a simultaneous lower bound over the three parameters, as opposed to examining the lower bound when one (or two) of the parameters is fixed (as was done in Section 3). (Thus, an algorithm for this problem with nondeterministic space complexity $O(2^{c*k}+l+\log n)$ cannot exist. Note, however, that the existence of such an algorithm was not precluded by the results given in the previous section.) Thus, the algorithm presented in Section 2 is optimal with respect to parameters l and n, and is nearly optimal with respect to the parameter k. We do not know at this time, however, whether the $\log k$ factor can be eliminated. This result can also be used to show that although the BP for CFSM(k) is solvable in PTIME (for fixed k), it is likely that the order of the polynomial depends on k.

The results of this paper thus provide improved upper and lower bounds for the VASS boundedness problem. The multiparameter analysis used here also sheds

some light on the gap between the previously best known upper and lower bounds of [17, 23]. As it turns out this gap is at least partially caused by the complexity being measured as a function of the input size. In terms of that measure it is questionable whether the gap can be improved.

Lastly, we briefly mention how our results can be extended to some other related problems concerning VASSs. For example, the lower bounds established in Section 4 hold for the covering and reachability problems. (Variations of those given in Section 3 hold as well.) Using a similar approach, as was used in Section 2, an upper bound of $O((l+\log n)*2^{c*k*\log k})$ can be shown for the covering problem. (Finding better upper/lower bounds for this problem was mentioned as an open problem in [18].) Previously, the best known bounds for this problem were essentially the same as those for the boundedness problem. See [17, 18, 23].

2. The Upper Bound

Let Z (N, N^+) denote the set of integers (nonnegative integers, positive integers, respectively). For a vector $v \in Z^k$, let $v(i)$, $1 \leqslant i \leqslant k$, denote the ith component of v. For a given value of k, let $\mathbf{0}$ in Z^k denote the vector of k zeroes (i.e., $\mathbf{0}(i)=0$ for $i=1,\ldots,k$). Now given vectors u, v, and $w \in Z^k$ we say:

- $v=w$ iff $v(i)=w(i)$ for $i=1,\ldots,k$,
- $v \geqslant w$ iff $v(i) \geqslant w(i)$ for $i=1,\ldots,k$,
- $v>w$ iff $v \geqslant w$ and $v \neq w$, and
- $u=v+w$ iff $u(i)=v(i)+w(i)$ for $i=1,\ldots,k$.

A k-dimensional *vector addition system* (VAS) is a pair (v_0, A) where v_0 in N^k is called the *start vector* and A, a finite subset of Z^k, is called the set of *addition rules.* The *reachability set* of the VAS (v_0, A), denoted by $R(v_0, A)$, is the set of all vectors z, such that $z=v_0+v_1+\cdots+v_j$ for some $j \geqslant 0$, where each v_i $(1 \leqslant i \leqslant j)$ is in A and for each $1 \leqslant i \leqslant j$, $v_0+v_1+\cdots+v_i \geqslant \mathbf{0}$. A k-dimensional *vector addition system with states* (VASS) is a 5-tuple (v_0, A, p_0, S, δ) where v_0 and A are the same as defined above, S is a finite set of states, δ ($\subseteq S \times S \times A$) is the transition relation, and p_0 is the initial state. Elements (p, q, x) of δ are called *transitions* and are usually written $p \to (q, x)$. A *configuration* of the VASS is a pair (p, x), where p is in S and x is a vector in N^k. (p_0, v_0) is the initial configuration. The transition $p \to (q, x)$ can be applied to the configuration (p, t) and yields the configuration $(q, t+x)$, provided that $t+x \geqslant \mathbf{0}$. In this case, $(q, t+x)$ is said to *follow* (p, t). Let s and s' be two configurations. Then s' is said to be reachable from s iff $s=s'$ or there exist configurations $s_1,\ldots,s_r$ such that $s=s_1$, $s'=s_r$, and s_{i+1} follows s_i for $i=1, 2,\ldots, r-1$. The *reachability set* of the VASS (v_0, A, p_0, S, δ), denoted by $R(v_0, A, p_0, S, \delta)$, is the subset of $S \times N^k$ containing all configurations reachable from (p_0, v_0). The *boundedness problem* (BP) for VASs (or VASSs) is to determine whether the reachability set, of a given instance, is *infinite.*

The boundedness problem for vector addition systems was first studied in [14], where a nonprimitive recursive decision procedure was proposed. This procedure is based on a search algorithm which generates the reachability set $R(v_0, A)$ and, at the same time, attempts to find a "pumpable loop" that can exploited to reach an arbitrary number of distinct vectors. It was shown that either the reachability set is finite or such a loop exists. Hence, the algorithm must eventually terminate. Unfortunately, the size of $R(v_0, A)$ is not bounded by a primitive recursive function of the size of (v_0, A). Rackoff, in [23], basically showed that if such a pumpable loop could be executed by some computation of the system, then it could be executed by a "short" computation. As a result, Rackoff presented an algorithm which required at most $2^{c*m^{*\log m}}$ space, for some constant c. Here, the parameter m represents the size (i.e., number of bits) of the instance. Actually, this parameter, upon careful examination, can be decomposed into three natural parameters k, l, and n, where k is the dimension of the VAS, l is the maximum length of the binary representation of each component in A (the set of addition rules), and n is the number of rules (i.e., the number of vectors in A). (The reader should note that these parameters are equally natural for Petri nets—a notational variant of VASs. There k would represent the number of places, l the maximum number of inputs or outputs of a transition, and n the number of transitions. See, e.g., [22].) However, it is always the case that $n \leqslant (2*2^l+1)^k$. Hence, for simplicity, we use VAS(k, l) to denote the class of VASs with parameters k and l. Later, when we discuss VASS, we introduce another parameter m to represent the number of states. It should be clear that the three parameters k, l, and m are mutually independent. Therefore, the class of VASSs with parameters k, l, and m will be denoted as VASS(k, l, m).

Our motivation for separating the parameter m into (k, l) (or (k, l, m)) comes from an observation that, in some cases when one or two parameters are fixed, the exponential space results can be improved. (It also serves to analyse what effect large numbers have with respect to the complexity of this problem.) For example, consider a network of two CFSMs in which each machine sends only one type of message to the other machine. It is not difficult to see, that this kind of network can be simulated by an instance of VASS$(2, 1, m)$, for some m. Furthermore, the BP for this kind of network is known to be NLOGSPACE-complete [24]. This seems to suggest that the parameters k, l, and m may contribute differently to the complexity of the problem. Therefore, by considering the complexity of the BP, concurrently, with respect to these three parameters, we hope to develop new insight into the problem. As a result, we are able to show for VAS(k, l), an upper bound of $O((l+\log n)*2^{c*k^{*\log k}})$ nondeterministic space, where c is a constant and n is the number of addition rules contained in the instance. Recall that in Rackoff's result, the BP can be solved in $O(2^{d*m^{*\log m}})$ space where m is the size of the VAS. In general, m can be greater than k^2, and hence, the use of distinct parameters does improve the complexity result to some degree.

In the remainder of this section, we show how to obtain an upper bound of $O((l+\log n)*2^{c*k^{*\log k}})$ nondeterministic space, where c is a constant and n is the number of rules, for the VAS(k, l) boundedness problem. What we actually show,

as did Rackoff, is a bound on the length of the shortest system computation, if one exists, which will execute a pumpable loop. In fact, our proof is a modification of the one in [23]. As a result, many of the details in the following proof are omitted especially when they are similar to those in [23].

Before going into detail, we require the following definitions. Most of them are the same as in [23]. The following notation is also taken, more or less, from [23]. For a VAS (v, A), a finite sequence of vectors $w_1, w_2, ..., w_m \in Z^k$ is said to be a *path* in (v, A), of length m, if $w_1 = v$ and $w_{i+1} - w_i \in A$ for all i, $1 \leqslant i < m$. Let $w \in Z^k$ and $0 \leqslant i \leqslant k$. The vector w is *i bounded* if $w(j) \geqslant 0$ for $1 \leqslant j \leqslant i$. If $r \in N^+$ is such that $0 \leqslant w(j) < r$ for $1 \leqslant j \leqslant i$, then w is called *i–r bounded.* Let $p = w_1, ..., w_m$ be a sequence of vectors, we say p is i bounded (i–r bounded) if every member in p is i bounded (i–r bounded). Moreover, if $w_j < w_m$, for some j, $1 \leqslant j < m$, then p is said to be *self-covering.* p is called an *i loop* if the first i places $v_1 = v_m$ and $v_{j_1} \neq v_{j_2}$ for all $1 \leqslant j_1 < j_2 \leqslant m$; i.e., p is a path such that the start and end vectors have their first i components identical and no other intermediate points have this property. The loop value of p is defined to be $v_m - v_1$. Let $0 \leqslant i \leqslant k$. For each $v \in Z^k$, define $m'(i, v)$ to be the length of the shortest i bounded, self-covering path in (v, A), if one exists; otherwise, define $m'(i, v) = 0$. Define $g(i) = \max\{m'(i, v) \mid v \in Z^k\}$. This function g (of A) then, will give us a bound on the length of the shortest computation which can execute a pumpable loop. Note that this upper bound does not depend on the start vector v.

To prove the upper bound result, we need the following lemma concerning the bounds of solutions of linear equations. The lemma is from [23]. (The proof is essentially from [2].)

LEMMA 2.1. *Let $d_1, d_2 \in N^+$, let B be a $d_1 \times d_2$ integer matrix and let b be a $d_1 \times 1$ integer matrix. Let $d \geqslant d_2$ be an upper bound on the absolute values of the integers in B and b. If there exists a vector $v \in N^{d_2}$ which is a solution to $Bv \geqslant b$, then for some constant c independent of d, d_1, d_2, there exists a vector $v \in N^{d_2}$ such that $Bv \geqslant b$ and $v(i) < d^{cd_1}$ for all i, $1 \leqslant i \leqslant d_2$.*

LEMMA 2.2. *If there exists an i–r bounded, self-covering path in (v, A), then there exists an i–r bounded, self-covering path with length $< (r*2^l)^{k^c}$, for some constant c independent of r, l, k, and n.*

Proof. The proof of this lemma is very similar to the corresponding one in [23]; hence only a sketch is given here. Let $v_1, ..., v_{m_0}, w_1, ..., w_{m_1}$ be an i–r bounded, self-covering path and $w_1 < w_{m_1}$. First, consider the path $v_1, ..., v_{m_0}$. Clearly, we can assume $m_0 \leqslant r^i \leqslant r^k$; since, otherwise, there exists some i loop and then we can make it short. Next, consider the path $w_1, ..., w_{m_1}$. This path, clearly, can be decomposed into a path s and some i loops, such that the length of $s \leqslant (r^k + 1)^2$. Let the loop values of those i loops be $l_1, ..., l_p$. Furthermore, it should be clear that a loop value is just the sum of at most r^k members of A, and therefore each place of l_i is of absolute value $\leqslant 2^l r^k$. Hence, there are at most $(2(2^l r^k) + 1)^k$ distinct loop values.

To find a shorter path, it suffices to find $n_1,..., n_p$ such that $n_1 l_1 + \cdots + n_p l_p + (s) > \mathbf{0}$ where (s) is the difference of the end and start points of the path s. By letting $d_1 = k$ and $d = (2^l r)^{c' k^2}$ and using Lemma 2.1, we are able to find $n_1,..., n_p$ such that $n_i \leqslant (2^l r)^{k^{c''}}$, and therefore, $m_1 \leqslant (2^l r)^{k^c}$, for some c. By combining the above results, the lemma is proved. ▮

LEMMA 2.3. *$g(0) < (2^l n)^{ck}$, for some constant c independent of l, k, and n.*

Proof. Recall that n is the number of rules in A. To evaluate $g(0)$, it suffices to solve $m_1 v_1 + \cdots + m_n v_n > \mathbf{0}$ where $A = \{v_1,..., v_n\}$. Choose $d_1 = k$ and $d = 2^l n$. Using Lemma 2.1 we can find a solution with $m_i < (2^l n)^{c' k}$. Clearly then the lemma is proved. ▮

LEMMA 2.4. *$g(i+1) < (2^{2^l} * g(i))^{k^c}$, for some constant c independent of l, k, and n.*

Proof. Case 1: If there is an $(i+1) - 2^l * g(i)$ bounded, self-covering path in (v, A), then from Lemma 2.2, there exists a short one with length $< (2^l * 2^l * g(i))^{k^c}$.

Case 2: Otherwise, let $v_1,..., v_{m_0}, v_{m_0+1},..., v_n$ be the path such that v_{m_0} is the first one not $2^l g(i)$ bounded. Without loss of generality, assume that $v_{m_0}(i+1) > (2^l g(i))$. Now no two of $v_1,..., v_{m_0}$ can agree on the first $i+1$ places, for then the sequence could be made even shorter. Hence, $m_0 < (2^l g(i))^{i+1}$. Let p be an i bounded, self-covering path in (v_{m_0}, A) of length $< g(i)$. Since $v_{m_0}(i+1) > 2^l g(i)$ and since each place in each vector in A is at most 2^l in absolute value, p must also be $(i+1)$ bounded. So the path $v_1,..., v_{m_0-1}, p$ is an $(i+1)$ bounded, self-covering path of length $(2^{2^l} g(i))^{i+1} + g(i) < (2^{2^l} g(i))^{k^c}$. ▮

THEOREM 2.1. *The* BP *can be decided in $O(2^{c * k^{* \log k}} * (l + \log n))$ nondeterministic space, for some constant c independent of l, k, and n.*

Proof. By recursively applying Lemmas 2.3 and 2.4, it is easy to derive the result that $g(k) \leqslant (2^l * n)^{2^{c * k^{* \log k}}}$. This means if a VAS is unbounded, there must exist a path, of length no more than $(2^l * n)^{2^{c * k^{* \log k}}}$, that leads to the "pumpable loop." Therefore, the BP can be solved in $O((l + \log n) * (2^{c * k^{* \log k}}))$ nondeterministic space. ▮

Now consider a VASS in the class VASS(k, l, m). Clearly the number of possible rules is no more than $m * (2 * 2^l + 1)^k$. Then according to Theorem 2.1, we have the following corollary:

COROLLARY 2.1. *The* BP *for the class* VASS(k, l, m) *can be solved in $O((l + \log m) * 2^{c * k^{* \log k}})$ nondeterministic space, for some constant c independent of l, k, and m.*

Note that the number of edges, between two nodes in such a graph, can be as great as $2^{l+1} + 1$. Thus, the actual size or number of bits in the description, of a VAS (or VASS), can be exponential in l. Note, however, that the size of the systems constructed in the previous theorem are only polynomial in l. What is, perhaps, somewhat surprising is that such an increase in the density of edges does not alter the complexity, of the problem, in terms of the parameters k, l, and m.

3. The Complexity of the BP When k Is a Fixed Known Constant

In this section, we focus our attention on the BP for VASS(k, l, m) when k is fixed. A summary of these results can be found in Fig. 3.1. We show that the boundedness problem is PSPACE-complete for $k \geqslant 4$. Consequently, it is unlikely that one can improve the complexity of the aforementional algorithm with respect to the parameter l. At this time, we do not know whether the PSPACE-complete result can be improved for the case when $k = 2$ or 3. However, we are able to show that it is NP-hard when $k = 2$. Later, we show that when $k = 1$, there exists a PTIME algorithm. If l is also fixed, then the problem becomes NLOGSPACE-complete. However, we do not know whether the previous PTIME result can be improved to NLOGSPACE. Finally, we consider the case where $k = 1$ such that zero detection is allowed. (When $k = 2$, this augmentation results in the BP becoming undecidable [7, 9, 22].) In this case, the BP becomes NP-hard. These results should be compared with earlier PSPACE-hard results concerning VASs (or VASSs) in the literature [12], where the parameter k instead of l is allowed to vary.

THEOREM 3.1. *The boundedness problem for k-dimensional VASSs, when $k \geqslant 4$, is PSPACE-complete.*

Proof. The result of being in PSPACE was shown in Section 2. It suffices, therefore, to show that it is PSPACE-hard.

Consider a deterministic linear bounded automaton (DLBA, for short) $W = \langle Q, \Sigma, \Gamma, \delta, q_0, \#, \$, F \rangle$, where

Q is the (finite) set of states,

Σ is the (finite) set of input symbols,

Γ is the finite set of allowable tape symbols,

δ is the transition or next move function,

q_0 is the initial state,

$\#$ and \$ are the left and right endmarkers, respectively, and

$F \subseteq Q$ is the set of accepting states.

	VASS (k, ℓ, m)	VAS (k, ℓ)
PSPACE-Complete	k ≥ 4	k ≥ 4
NP-hard	k ≥ 2	k ≥ 3
PTIME	k = 1	-----
trivial	-----	k = 1

FIG. 3.1. The complexity of the BP for VASSs (VASs) where k is fixed.

Let $l = a_1 \cdots a_n$ be an input to W. Without loss of generality, we assume that each a_i has value 0 or 1, that the endmarkers are implicitly defined within the string $a_1, ..., a_n$, and that each transition of δ causes W to move its tape head one position to the left or one position to the right. Then one can construct a four-dimensional VASS V to simulate the execution of W on l. V will allow a counter to become arbitrarily large iff the simulation results in l's acceptance. Since the membership problem for DLBAs is well known to be PSPACE-complete, the boundedness problem for k-dimensional VASSs ($k \geqslant 4$) is, therefore, PSPACE-hard.

For each $q \in Q$, let q' and q'' be distinct new state names. Let $\{[q, i, j], [q', i, j], [q'', i, j] \mid q \in Q, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n+1\}$ be the set of states of V. Intuitively, if V is in the state $[p, i, j]$ ($[p', i, j]$, $[p'', i, j]$, respectively) then W's current state (head position) is p (i). The purpose of j will be made clear later. The four-dimensional vector $\langle A, A', B, B' \rangle$ is used to encode the contents of the tape in such a way that, at some instant, the numbers B and A represent the tape contents to the left and to the right of the current head position, respectively, and B' and A' represent the *complement* of the tape in a similar way. More specifically, when the current head position is at i, then the encodings are (assume $a_0 = a'_0 = a_{n+1} = a'_{n+1} = 0$)

$$B = \Sigma_{j=0}^{i-1} a_j * 2^{n+1-j},$$

$$A = \Sigma_{j=i}^{n+1} a_j * 2^{n+1-j},$$

$$B' = \Sigma_{j=0}^{i-1} a'_j * 2^{n+1-j},$$

$$A' = \Sigma_{j=i}^{n+1} a'_j * 2^{n+1-j}.$$

where a'_j is the complement of a_j, for $1 \leqslant i \leqslant n$.

The crux of the simulation is the ability to read the ith symbol on the tape by doing some arithmetic computation on the numbers A, A', B, B'. This can be done by introducing the following transitions: Assuming that the head position is at i and the current state is p.

$$t_0\colon [p, i, i] \to ([q, i, i], \langle 0, -2^{n+1-i}, 0, 0 \rangle)$$

$$t_1\colon [p, i, i] \to ([q, i, i], \langle -2^{n+1-i}, 0, 0, 0 \rangle)$$

Clearly, the transition t_0 (t_1) is executable iff a_i equals 0 (1). Now, consider a transition t of W

$t\colon (p, a_t) \to (q, a_l, \Delta)$, where Δ ($= -1$, or $+1$) indicates the direction of the head movement.

In what follows, we show how the transition t will be simulated on the VASS. Without loss of generality, we assume that $a_t = 0$ and $a_l = 1$. The other three cases can be handled in a very similar way. Recall that Δ stands for the direction of the head movement. Therefore, we have the following two cases:

1. **move right**:

We introduce the following transition to the VASS:

$$t_0: [p, i, i] \to ([q, i+1, i+1], \langle 0, -2^{n+1-i}, +2^{n+1-i}, 0\rangle)$$

Note that the number 2^{n+1-i} is added to the position B. This corresponds to the fact that, a "1" has been written on the input tape and the input head has been moved to the right one position.

2. **move left**:

Now, the left $\_$ move transition is a bit more involved, and consists of the execution of $2n-1$ stages in V, as shown in Fig. 3.2. First we introduce the transition

$$t_0: [p, i, i] \to ([q, i-1, i+1], \langle 0, -2^{n+1-i}, +2^{n+1-i}, 0\rangle).$$

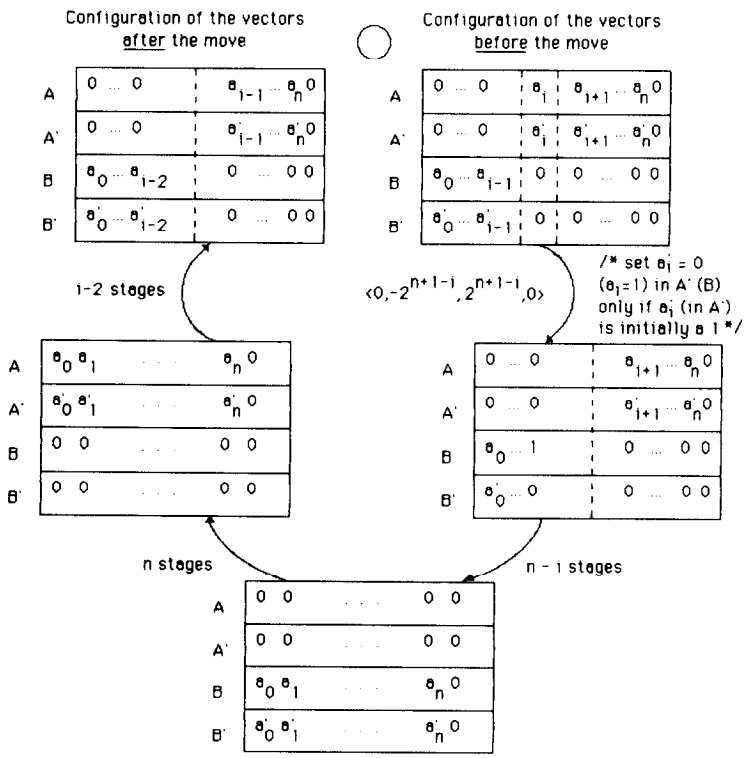


FIG. 3.2. The simulation of the left-move transition.

Here the updated ith bit is copied from the vector A' to the vector B. The remaining bits of A (A') are then copied unchanged to B (B'). For this purpose we introduce the following transitions for all j, $i<j\leqslant n$:

$$:[q,i-1,j]\to([q,i-1,j+1],\langle 0,-2^{n+1-j},0,+2^{n+1-j}\rangle)$$
$$:[q,i-1,j]\to([q,i-1,j+1],\langle -2^{n+1-j},0,+2^{n+1-j},0\rangle)$$
$$:[q,i-1,n+1]\to([q',i-1,1],\langle 0,0,0,0\rangle).$$

Note that at each step only one of these transitions can be applied. Afterwards, all the bits are copied (unchanged) from B (B') to A (A'). For this purpose we introduce the following transitions for all j, $1\leqslant j\leqslant n$:

$$:[q',i-1,j]\to([q',i-1,j+1],\langle +2^{n+1-j},0,-2^{n+1-j},0\rangle)$$
$$:[q',i-1,j]\to([q',i-1,j+1],\langle 0,+2^{n+1-j},0,-2^{n+1-j}\rangle)$$
$$:[q',i-1,n+1]\to([q'',i-1,1],\langle 0,0,0,0\rangle).$$

Again, note that at each step only one of the transitions can be applied. Finally, all bits left of the $(i-1)$-st bit are copied from A (A') to B (B'). For this purpose we introduce the following transitions for all j, $1\leqslant j<i-1$:

$$:[q'',i-1,j]\to([q'',i-1,j+1],\langle 0,-2^{n+1-j},0,+2^{n+1-j}\rangle)$$
$$:[q'',i-1,j]\to([q'',i-1,j+1],\langle -2^{n+1-j},0,+2^{n+1-j},0\rangle)$$
$$:[q'',i-1,i-1]\to([q,i-1,i-1],\langle 0,0,0,0\rangle).$$

Again, note that at each step only one of the transitions can be applied. Now the simulation is ready to continue since the vector is now positioned correctly with respect to the input head.

Finally, for each accepting state q_f in F, we introduce the transition

$$:[q_f,j,j]\to([q_f,j,j],\langle +1,+1,+1,+1\rangle),\qquad \forall 1\leqslant j\leqslant n.$$

Clearly then, W accepts the input $a_1\ldots a_n$ iff V is unbounded. In addition, the number of vectors we use is a polynomial with respect to n, and, therefore, the above construction can be done in deterministic log space. ∎

In [10], it is shown that an n-dimensional VASS can be simulated by an $(n+3)$-dimensional VAS. Using this fact, it follows immediately that for $k\geqslant 7$, the BP for k-dimensional VASs is PSPACE-complete. However, using a more careful encoding, we are able to utilize the positions of the VAS more efficiently during the simulation of the DLBA. As a result, we obtain the following improved corollary:

COROLLARY 3.1. *The* BP *for k-dimensional* VASs, *when* $k\geqslant 4$, *is* PSPACE-*complete.*

Proof. As shown in [10], each state can be simulated by a sequence of transitions as follows. For each state q_i, let $a_i = i$ and $b_i = (k+1)(k+1-i)$, where k is the number of states. Assume that the current configuration is (q_i, v). Then a transition $q_i \to (q_j, w)$ in the VASS can be simulated as (see [10] for details)

$$(a_i, b_i, 0, v) \to (0, a_{k-i+1}, b_{k-i+1}, v) \to (b_i, 0, a_i, v) \to (a_j, b_j, 0, v+w).$$

Note that three extra positions have been used to simulate a state. Upon a careful examination, of the previous proof, one can see that at most one position will be subtracted at a time. This suggests a more efficient way to encode the tape contents using the VASS. For example, let a_i and b_i be the same as before. Let n denote the input length of the DLBA. During the simulation of the DLBA, assume that the current configuration is $(q_i, \langle A, A', B, B' \rangle)$ and the transition $q_i \to (q_j, \langle -a, b, c, d \rangle)$, where a, b, c, and d are nonnegative integers, is applicable. Then our method is to use the vector $(A, A', a_i{*}2^{n+1} + B, b_i{*}2^{n+1} + B')$ to represent the configuration $(q_i, \langle A, A', B, B' \rangle)$. The basic idea here is that, we use positions B and B' to store both state and tape information in such a way that the high order bits represent the state and the low order bits represent the tape. Then since no subtraction of the low order bits of B or B' will be made for this transition, the arguments in [10] and in the proof of Theorem 3.1 still work. Similarly, for a transition $q_i \to (q_j, \langle a, b, -c, d \rangle)$, the vector $(a_i{*}2^{n+1} + A, b_i{*}2^{n+1} + A', B, B')$ will be used. In addition, it should be clear that some rule like $(a_i{*}2^{n+1}, b_i{*}2^{n+1}, -a_i{*}2^{n+1}, -b_i{*}2^{n+1})$ will be added if necessary. The rest of the simulation is then straightforward. ∎

At the present time, we do not know whether the PSPACE result can be extended for the case when $k = 2$ or 3. We surmise PSPACE is needed, however, since there exist instances in VASS$(2, l, m)$, where the shortest path that leads to a "pumpable loop" requires an exponential number of steps in l and m. To show this, consider the following instance in VASS$(2, l, m)$ (see also Fig. 3.3):

1. $(q_1, \langle 1, 0 \rangle)$ is the initial configuration.
2. The transitions are:
   a. for $1 \leqslant i \leqslant m-1$ and i is odd,
      $q_i \to (q'_i, \langle -1, 1 \rangle)$
      $q'_i \to (q_i, \langle 0, 1 \rangle)$
      $q_i \to (q_{i+1}, \langle 0, 0 \rangle)$;
   b. for $1 \leqslant i \leqslant m-1$ and i is even,
      $q_i \to (q'_i, \langle 1, -1 \rangle)$
      $q'_i \to (q_i, \langle 1, 0 \rangle)$
      $q_i \to (q_{i+1}, \langle 0, 0 \rangle)$.
   c. $q_m \to (q_{m+1}, \langle -2^m, 0 \rangle)$.
   d. $q_{m+1} \to (q_{m+1}, \langle 1, 1 \rangle)$.

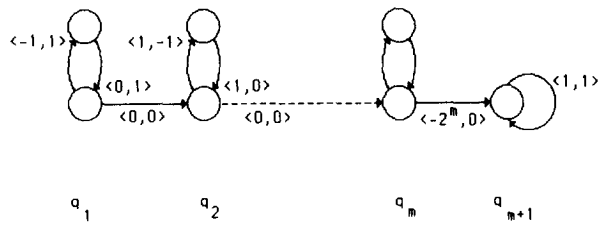


FIG. 3.3. An unbounded VASS that requires an exponential number of steps.

It is then clear that the VASS is unbounded iff the state q_{m+1} is reachable. Hence 2^m steps are required.

On the other hand, we are able to show that the problem is NP-hard. In order to show that the boundedness problem for k-dimensional VASSs, when $k \geq 2$, is NP-hard, we reduce the Knapsack problem to the boundedness problem. Since the Knapsack problem is known to be NP-complete, it follows immediately that the boundedness problem for VASSs is NP-hard. Recall that the Knapsack problem is the following:

Instance: Finite set U, for each $u \in U$, a size $s(u) \in N$ and a value $v(u) \in N$, and positive integers B and K.

Question: Is there a subset $V \subseteq U$, such that $\Sigma_{u \in V} s(u) \leq B$ and $\Sigma_{u \in V} v(u) \geq K$?

THEOREM 3.2. *The boundedness problem for k-dimensional* VASSs, *when* $k \geq 2$, *is* NP-*hard.*

Proof. Consider an instance of the Knapsack problem as stated above. Let $U = \{u_1, \ldots, u_n\}$. In what follows, we are going to construct a two-dimensional VASS in such a way that the Knapsack problem has a solution iff the VASS is unbounded. Consequently, the boundedness problem for two-dimensional VASSs is, at least, as "hard" as the Knapsack problem.

The VASS we construct has $n+3$ states, namely, $p_0, p_1, \ldots, p_{n+1}, p'_{n+1}$. Let p_0 and $\langle 0, 0 \rangle$ be the initial state and vector, respectively. The transitions of the VASS are the following:

1. $p_0 \to (p_1, \langle 0, B \rangle)$.
2. For $i = 1, \ldots, n-1$,
   $p_i \to (p_{i+1}, \langle +v(u_i), -s(u_i) \rangle)$
   $p_i \to (p_{i+1}, \langle 0, 0 \rangle)$.
3. $p_n \to (p_{n+1}, \langle -K, 0 \rangle)$
   $p_n \to (p'_{n+1}, \langle 0, 0 \rangle)$.
4. $p_{n+1} \to (p_{n+1}, \langle +1, +1 \rangle)$
   $p'_{n+1} \to (p'_{n+1}, \langle 0, 0 \rangle)$.

It should be clear that the VASS is unbounded iff the state p_{n+1} is reachable via a path from p_0. On the other hand, such a path must correspond to a solution of the Knapsack problem. Note that, since we only introduce $n+3$ states, the above construction can easily be done in deterministic log space. ∎

Using encoding techniques similar to those described in Corollary 3.1., we can obtain the following corollary:

COROLLARY 3.2. *The* BP *is* NP-*hard for k-dimensional* VASs *when* $k \geqslant 3$.

Finally, we have the following theorems concerning one-dimensional VASSs.

THEOREM 3.3. *The* BP *is* NLOGSPACE-*complete for one-dimensional* VASSs, *when the parameter l is also* 1.

Proof. The problem was shown to be decidable in NLOGSPACE, in the previous section, whenever k and l were considered to be fixed constants. The hardness follows from an easy reduction from the graph reachability problem [26]. See [12, 24] for similar reductions. ∎

THEOREM 3.4. *For one-dimensional* VASSs, *the* BP *can be decided in* PTIME.

Proof. Consider an instance of VASS$(1, l, n)$ with the initial configuration (q_0, v_0). It is well known that the VASS is unbounded iff there exists a reachable state q_i, such that from q_i a loop with a positive gain can be executed [14]. Based on this idea, the BP can be solved by detecting the existence of this kind of loop, $(q_0, v_0) \to (q_l, v) \to (q_l, v + \Delta)$, where each segment is no longer than n steps. Now the largest value obtainable in any state q within at most n steps can be calculated iteratively as

$$d_0^0 = v_0, \qquad d_i^0 = -\infty, \qquad i \neq 0.$$

$$d_i^{k+1} = \max\{d_i^k, \max_j\{d_j^k + t \mid \text{where } q_j \to (q_i, t) \text{ is a transition in the VASS and } d_j^k + t \geqslant 0\}\}.$$

Now, such a pumpable loop exists $\Leftrightarrow \exists\ 0 \leqslant j \leqslant n,\ 0 \leqslant s < t \leqslant n+1$, such that $d_j^t > d_j^s$. ∎

The last result should be compared with the related results concerning the shortest and longest path problem studied in [16]. (See also [5].) The difference here is, of course, that we do not require the path in question to visit each node at most once.

Examples illustrating the limitations of the modeling power of VASs [1, 15, 20–22] indicate that the limiting factor is the inability of the VAS to test a position for zero and take a particular action on the outcome of the test. (See also [25].) As a result, the literature contains many extensions to the basic model which allow such a zero test. (See, e.g., [22].) In most cases, when zero testing is allowed, two

potentially unbounded positions are sufficient to render the BP undecidable. (See [9, 22].) However, when only one position is allowed, we have the following:

THEOREM 3.5. *The* BP *is* NP*-hard for one-dimensional* VASSs *when zero-detection is allowed.*

Proof. This can be proved by reducing the PARTITION problem to the BP. Since the PARTITION problem is known to be NP-complete, it follows that the BP is NP-hard.

Recall that an instance of the PARTITION problem consists of a finite set A and a "size" $s(a) \in N^+$ for each $a \in A$. Let $T = \Sigma_{i=1}^{n} s(i)$. Let n denote the number of elements in A. We construct an instance of 0-detecting VASS$(1, l, n+3)$ as follows:

1. (q_1, T) is the initial configuration.
2. For $i = 1,..., n$,
   $q_i \to (q_{i+1}, -s(i))$
   $q_i \to (q_{i+1}, 0)$
3. $q_{n+1} \to (q, -T/2)$.
4. $q \to (q_f, 0)$ if the vector in q equals zero.
5. $q_f \to (q_f, +1)$.

It should be clear that the PARTITION problem has a solution iff the corresponding VASS is unbounded (i.e., the state q_f is reachable). Therefore, the BP is NP-hard. ∎

We surmise, but are unable to show, that the aforementioned problem is solvable in NP. First, note that a "pumpable loop" cannot contain a 0-move. Now, consider the shortest path, which executes a "pumpable loop." Let B denote the last conditional move executed on that path. Then the number of steps executed after B can be no more than $n+1$. However, as noted earlier, the number of steps proceeding B can be exponential. But, it has to be the case, that every loop proceeding B must either cause a net loss or contain a 0-move. As a result, we have that the computation before B is bounded by $O(n^*2^l)$. The best we can do, at this time, then, is to deduce that the problem is doable in PSPACE.

4. THE LOWER BOUND

In this section, we fix the alphabet over which problems can be specified. Without loss of generality then we let this alphabet be $\Sigma = \{0, 1\}$. A decision problem, over Σ, is said to be solvable in $\Psi(x)$ deterministic time ($\Phi(x)$ nondeterministic space) iff there exists a deterministic Turing machine (TM, for short) (nondeterministic TM) with tape alphabet Σ which decides the problem using at most $\Psi(x)$ time ($\Phi(x)$

space). (Note that Ψ (Φ) is not a function of the input length.) A function $g: \Sigma^* \to \Sigma^*$ is said to be computable in $S(m)$ space iff there exists a deterministic TM M such that, given an input $x \in \Sigma^*$, M will output $g(x)$ using at most $S(|x|)$ space. Now, consider two problems L and L' over Σ^*. L is said to be (S, Q)-reducible to L' via the function g iff g is computable in $S(|x|)$ space such that:

- $x \in L$ iff $g(x) \in L'$, and
- $\forall x \in \Sigma^*$, $|g(x)| \leqslant Q(|x|)$.

The following lemma is from [13].

LEMMA 4.1. *If a function $f: \Sigma^* \to \Sigma^*$ is computable by an $S(n)$ space bounded* TM *M with tape symbols $\{0, 1, \#\}$ such that at any time the work tape contains at most k #'s, then f is computable in $S(n) + (k+2)^* \log S(n)$ space by a* TM *M' with tape symbols $\{0, 1\}$.*

The following lemma is a slight modification of one given in [13].

LEMMA 4.2. *Suppose L is (S, Q)-reducible to L' via the function g.*

(1) *If L' is solvable in $\Psi(x)$ deterministic time, then there is a constant c such that L is solvable in $\Psi(g(x)) + c^*|x|^*S(|x|)^*2^{S(|x|)}$ deterministic time.*

(2) *If L' is solvable in $\Phi(x)$ nondeterministic space, then there is a constant c such that L is solvable in $\Phi''(x) + c^* \log \Phi''(x)$ nondeterministic space, where $\Phi''(x) = \Phi(g(x)) + 2^* \log(Q(|x|)) + S(|x|)$.*

In this section, we establish a lower bound for the VASS boundedness problem in terms of the three natural numeric parameters, k, l, and n. Our goal is to provide a simultaneous lower bound over the three parameters, as opposed to examining the lower bound when one (or two) of the parameters is fixed (as was done in Sect. 3). In order to do this, we define a general class of problems whose complexity can also be stated in terms of three natural numeric parameters. In particular, we focus on the following class which we denote as C:

C: A problem P belongs to C if P is a problem with three natural numeric parameters, say k, l, and n, can be solved in $2^{k*}(l + \log n)$ nondeterministic space, and, for any instance x, of P, the following three conditions are satisfied:

1. $|x| \geqslant \max\{k, l, n\}$, ($|x|$ represents the length of x),
2. k, l, and n can be computed from x and written down in deterministic $\log|x|$ space, *and*
3. $k \geqslant \log|x|$ or $l \geqslant \log|x|$ or $n \geqslant |x|/2$.

Let L and L' be two problems with natural parameters (k, l, n) and (k', l', n'), respectively. Let d be a constant. For such problems we say L is d-(S, Q)-reducible to L' iff there exists a function g such that L is (S, Q)-reducible to L' via g and

$k' \leqslant d^*k$, $l' \leqslant d^*l$, and $n' \leqslant d^*n^2$. (Here for an arbitrary $x \in \Sigma^*$, k, l, and n (k', l', and n', respectively) denote the natural parameters with respect to x and L ($g(x)$ and L', respectively).) Let $\mathscr{C}$ be a class of problems over Σ. L is said to be $\mathscr{C}$-hard with respect to $d\text{-}(S, Q)$-reducibility if for every L' in $\mathscr{C}$, there exists a constant c such that, L' is $d\text{-}(S(n), c^*Q(n))$ reducible to L. We will show that the BP for the class VAS(k, l, n) is hard for C, with respect to $d\text{-}((2+\varepsilon)\log(\log|x| + \log n), |x|^3)$-reducibility for some constant d.

To prove our result, we require the following lemma, which is similar to the one in [17], where Lipton constructed a VAS to simulate a multicounter machine.

LEMMA 4.3. *There exists a positive integer constant h, such that for any 3-tuple of integers k, l, and n one can construct a* VASS *in* VASS(h^*k, l, n), *that can manipulate a counter, whose value can range from* 0 *to* $(n^*2^l)^{2^k}$. *Furthermore, this counter can be incremented, decremented, and tested for zero. (I.e., the resulting* VASS *can simulate a counter machine with a finite state control.)*

Proof. Basically, the construction is a straightforward generalization of the one shown in [17]; and hence the proof sketched here (for the sake of completeness) corresponds very closely to the one in [17]. There, Lipton proved this theorem for the case when l and n were equal to 1. Lipton showed how to maintain and store a number, whose value ranged between 0 and 2^{2^k}, by such a system. (Lipton actually showed this for a class of parallel programs which correspond quite naturally to VASSs.) This number could then be incremented by 1 (as long as the current value was below the upper limit), decremented by 1 (as long as the current value exceeded 0), and tested for zero. The zero-test was the hard part.

Now the construction is shown in two parts. From these two parts it should be clear that the Lemma holds. Each part is shown via induction. It should be noted that the main difference in our construction (from the one in [17]) is in the proof of the base cases. We use flowcharts to describe the VASS being constructed. In our flowcharts, circles correspond to states of the VASS, while directed lines with accompanying boxes represent transitions. The contents of a box indicate the changes made to the system vector by the transition. Recall that a transition can only be executed if the resulting system vector is nonnegative. Now given a flowchart representation, one will easily be able to construct the corresponding VASS.

First some preliminaries. Let A_k be defined recursively as

$$A_1 = n^*2^l$$

$$A_{k+1} = A_k{}^2.$$

Clearly then, $A_i = (n^*2^l)^{2^i}$.

For $k \geqslant 0$, consider an $8^*k + 2$-dimensional vector v_0 of the form

$$\langle S_1, S'_1, x_1, x'_1, y_1, y'_1, B_1, C_1, \ldots, S_k, S'_k, x_k, x'_k, y_k, y'_k, B_k, C_k, r, r' \rangle$$

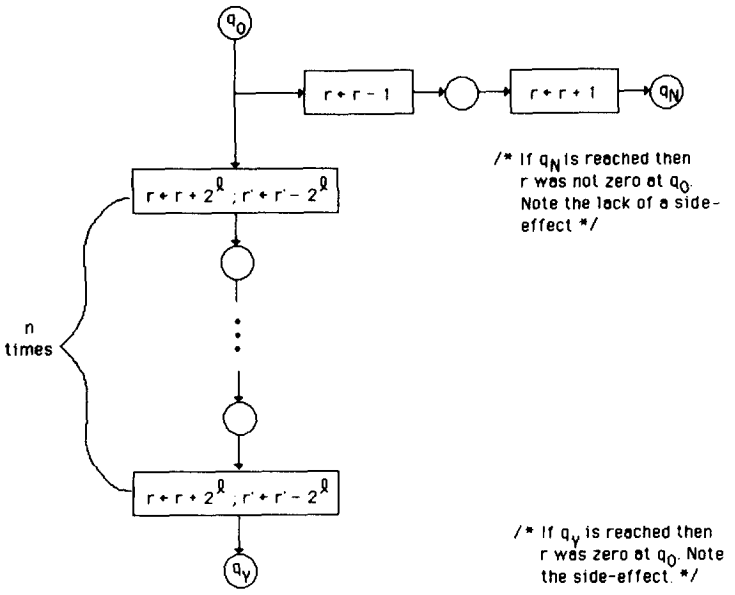


FIG. 4.1. The flowchart to test whether $r=0$ when $r+r'=A_0$.

which satisfies

(1) $r+r'=A_{k+1}$,

(2) $\forall i,\ 1\leqslant i\leqslant k,\ x_i=y_i=S_i=0$ and $x'_i=y'_i=S'_i=A_i$,

(3) $\forall i,\ 1\leqslant i\leqslant k,\ B_i=C_i=0$.

Let $v_N=v_0$. Let v_Y be identical to v_0 except that the values of r and r' are reversed.

In the first induction, we show how to construct a VASS $\mathscr{V}=(v_0, A, q_0, S\cup\{q_0, q_N, q_Y\}, \delta)$ in VASS(c^*k, l, n), for some constant c, in which the configuration (q_N, v_N) $((q_Y, v_Y))$ is reachable iff $r\neq 0$ $(r=0)$ in v_0. The reversal of the values of r and r' in v_Y is subsequently referred to as the side effect.

(*Base Step*) Figure 4.1 illustrates the VASS when $k=0$. Clearly if q_N can be reached we have that $r>0$. Also, if q_Y can be reached, it must be the case (at q_Y) that $r=n^*2^l$. Since at each state $r+r'=n^*2^l$, we have that r was originally 0. Hence, the base case is proved. Furthermore, only $n+2$ states are needed and the biggest number used in a vector can be represented in l bits.

(*Induction Hypothesis*) Assume that the assertion is true for $k-1\geqslant 0$.

(*Induction Step*) The construction here is the same as the one in [17], although the presentation has been somewhat altered. The VASS to be constructed is shown in Fig. 4.2(a); the missing portions labelled α and β are shown in Fig. 4.2(b). Consider the flowchart in Fig. 4.2(a). Again, q_N can be reached iff r was originally non-zero. We will later show that if the state α_0^k (β_0^k) is entered, the transition from α_N^k

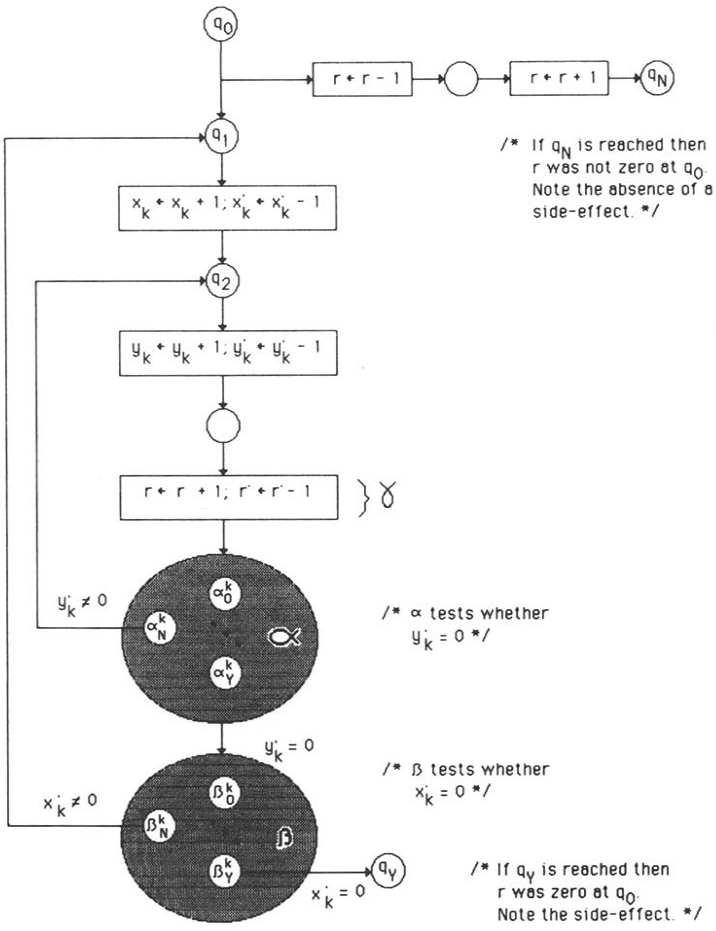


FIG. 4.2a. The flowchart to test whether $r = 0$ when $r + r' = A_k$.

(β_N^k) to q_2 (q_1) can be taken iff $y'_k \neq 0$ $(x'_k \neq 0)$ at state α_0^k (β_0^k). Likewise we will show that the transition from α_Y^k (β_Y^k) to β_0^k (q_Y) can be taken iff $y'_k = 0$ $(x'_k = 0)$ at state α_0^k (β_0^k). Furthermore, it will be the case that both α and β upon exiting via state α_N^k (β_N^k) leave the system vector unchanged. However, if they exit via state α_Y^k (β_Y^k) then the only side-effect will be that the values of y_k (x_k) and y'_k (x'_k) are reversed. Thus, if q_N is not reachable then the γ portion of the flowchart can be traversed A_k $(= x'_{k-1} * y'_{k-1})$ times (since both x'_{k-1} and y'_{k-1} equal A_{k-1} initially), provided that the initial value of r' was A_k. Therefore, if q_Y can be reached, it must be the case that the value of r' was originally A_k and hence r was originally 0 (since $r + r' = A_k$). (Note also that when q_Y is reached $r = A_k$ and $r' = 0$; i.e., the required side effect has occurred.) Now the flowchart of Fig. 4.2(a) requires one to test, at times, whether y'_{k-1} and x'_{k-1} are zero. These tests could be accomplished by inserting flowchart segments provided by the induction hypothesis.

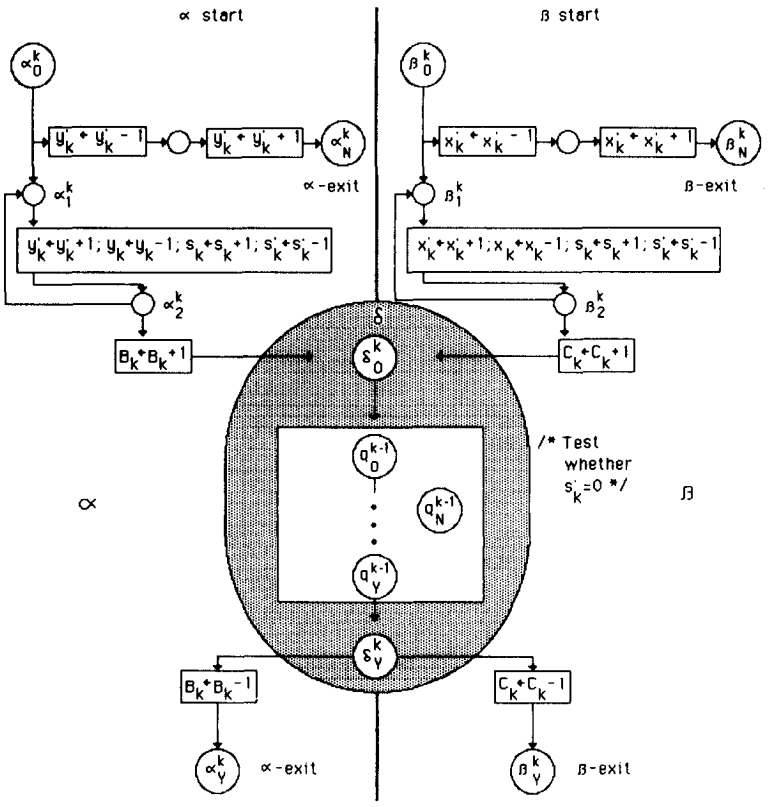


FIG. 4.2b. The flowchart for α and β.

However if this is done, the total number of states required would be at least twice what is required at the previous level of the induction, thus making the total number of states exponential in k. In order to avoid this problem, Lipton showed how to simulate the required tests using what he called a primitive type of parameter passing and subroutine calling mechanism. Following Lipton then, such tests can be implemented using the flowchart segments α and β shown in Fig. 4.2(b). The induction hypothesis is only used in the construction of the test to determine whether $S'_k = 0$ when state δ_0^k is reached. This ensures that the total number of states required is proportional to k. Consider the flowchart shown in Fig. 4.2(b). Clearly, α_N^k (β_N^k) is reachable iff it was the case, at α_0^k (β_0^k), that $y'_k \neq 0$ ($x'_k \neq 0$). Also, note that the system vector at α_N^k (β_N^k) must be identical to the one at α_0^k (β_0^k). Now the δ portion of the flowchart acts as a "subroutine" that will verify whether $y'_k = 0$ ($x'_k = 0$). S_k and S'_k are used to pass the parameters y'_k (x'_k) and y_k (x_k), respectively. The loop in α (β) is used to copy y'_k (x'_k) to S_k and y_k (x_k) to S'_k, respectively, so that δ can use the same input parameters S_k and S'_k for both α and β. The state δ_Y^k will be reachable (with the only side effect being the reversal of

values in S_k and S'_k) iff $S'_k = 0$ at δ_0^k. Now if the loop in α (β) fails to be executed until y_k (x_k) is copied to S'_k, then S'_k could not have been zero when state δ_0^k was reached and hence δ_Y^k will not be reachable. Note that B_k (C_k) is set to one before α (β) causes the system to reach δ_0^k; hence, state α_Y^k (β_Y^k) (and not state β_Y^k (α_Y^k)) can be reached iff at some later time δ_Y^k is reached. Let $\mathscr{L}_{k-1}$ be the flowchart segment with states q_0^{k-1} (the start state), q_Y^{k-1} (the "yes" state), q_N^{k-1} (the "no" state), provided by the induction hypothesis where S_k (S'_k) plays the role of r (r'). Then if we plug in $\mathscr{L}_{k-1}$ for the missing portion of δ the lemma will follow. This is because whenever q_Y^{k-1} is reached the following two things must be true:

- the value of S'_k (at δ_0^{k-1}) must have been zero, and
- the only side effect is that the values of S_k and S'_k are reversed.

But the values of S_k and S'_k were reversed earlier by α (β) providing that S'_k ends up being zero at δ_0^{k-1}. Hence by the time α_Y^k (or β_Y^k) are reached the side effect with respect to S_k and S'_k (and α (or β)) has been cancelled. The only other side effect to explain is the fact that α (β) has reversed the values of y_k (x_k) and y'_k (x'_k). But this side effect is precisely the one desired.

Now consider the size of $\mathscr{V}$. The base case requires $n+2$ states and l bits to represent a number in a vector. Inductively, one can easily see that $\mathscr{V}$ requires only $n + h'*k$ states, for some constant h'. (Note that h' is the number of states added in Figs. 4.2(a)–(b).) Therefore, $\mathscr{V}$ is in VASS($8*k+2, l, h'*k+n$), and hence in VASS($c*k, l, n$), for some constant c independent of k, l, and n (since the $h'*k$ states can be simulated by $h'*k$ additional positions).

In the second induction, we show for any $k \geqslant 1$ how to construct a VASS in VASS($d*k, l, n$) (for some constant d) that, when given an initial vector v_0 of the form

$$\langle S_1, S'_1, x_1, x'_1, y_1, y'_1, B_1, C_1, \ldots, S_k, S'_k, x_k, x'_k, y_k, y'_k, B_k, C_k \rangle$$

where each component equals 0, can reach state q_F with the resulting vector satisfying

(1) $\forall i,\ 1 \leqslant i \leqslant k,\ x_i = y_i = S_i = B_i = C_i = 0,$

(2) $\forall i,\ 1 \leqslant i \leqslant k,\ x'_i = y'_i = S'_i = A_i.$

The base case (for $k=1$) is shown in Fig. 4.3. The construction for the inductive step (where $k = m+1$) is shown in Fig. 4.4. The missing portions of the flowchart, labelled σ and ω (i.e., the tests for zero on y'_m and x'_m) are implemented via the induction hypothesis using the same programming trick that was employed in the first part. See [17] for further details. Hence, this VASS is in VASS($d*k, n, l$), for some constant d.

By combining the above results, we can construct a VASS that can initialize the vector

$$\langle S_1, S'_1, x_1, x'_1, y_1, y'_1, B_1, C_1, \ldots, S_k, S'_k, x_k, x'_k, y_k, y'_k, B_k, C_k, r, r' \rangle$$

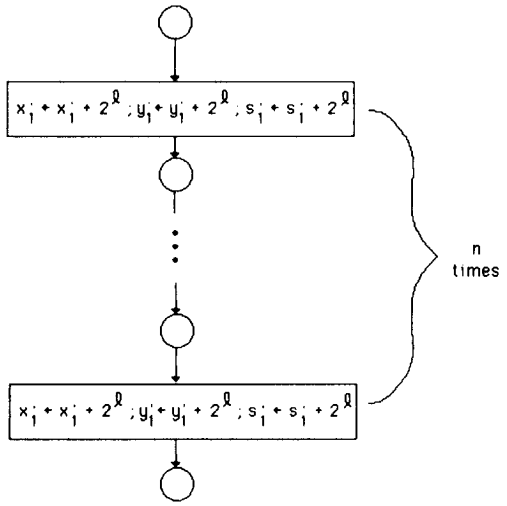


FIG. 4.3. The flowchart used to set x'_1, y'_1, and s'_1 to $n * 2^l$.

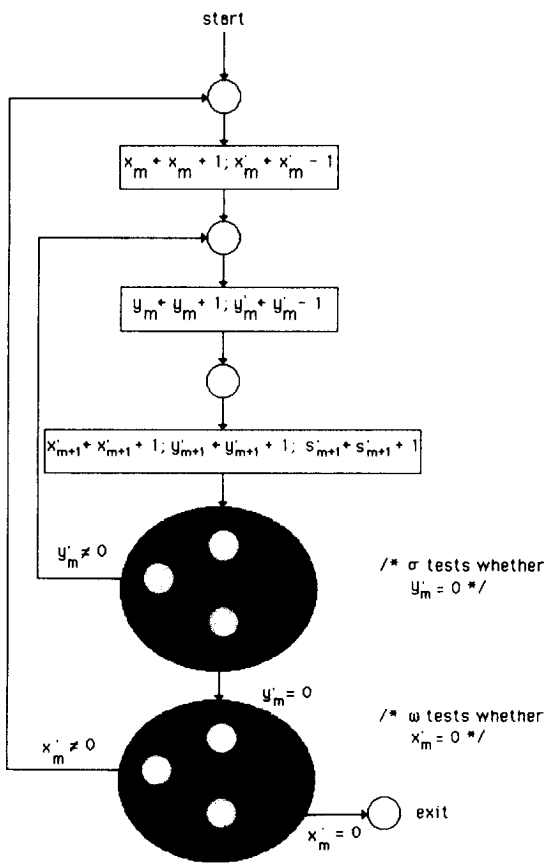


FIG. 4.4. The flowchart used to set x'_{m+1}, y'_{m+1}, and s'_{m+1} to A_{m+1}

so that $\forall i,\ 1 \leqslant i \leqslant k,\ x_i = y_i = S_i = B_i = C_i = 0,\ x'_i = y'_i = S'_i = A_i,\ r = 0$, and $r' = A_{k+1}$. Now, the position r can be used to simulate a counter. In other words, r can be

(1) decremented by 1,
(2) incremented by 1,
(3) tested for 0.

Subroutines for these three steps are trivial and are shown in Figs. 4.5(a)–(c). Note that in Fig. 4.5(c), the repeated test is to cancel the side effect. Further details can be found in [17]. ▮

Using the above result, one can construct a VASS in VASS($h*k, l, n$) (for some constant h) such that a pair of positions (r, r') can be used to simulate a counter. By using no more than three times the number of positions one can then construct such a VASS that can simulate, in some sense, a three-counter machine, ala Minsky [19]. Such a three-counter machine (3CM) has a read-only input tape, a finite state control, and three counters. During a single computation step, such a machine can, depending on the current state and input symbol, check each counter for zero, and

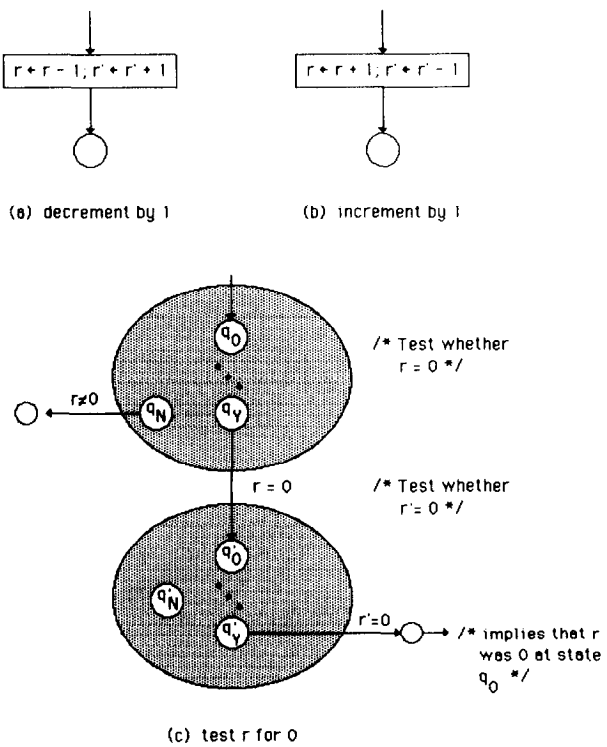


FIG. 4.5. Subroutines for simulating CM moves.

the move accordingly. It can move its input head one cell either to the left or to the right and increment or decrement each of its counters by 1 (as long as they remain nonzero). For more details, see [11, 19]. We are now ready to show the following:

THEOREM 4.1. *The* BP *for* VASS(k, l, n) *is hard for* C, *with respect to* $d\_((2+\varepsilon)(\log|x|+\log n), |x|^3)$*-reducibility.*

Proof. This is shown by reducing an arbitrary problem in C to the BP for VASS(k, l, n). Let P be an arbitrary problem in C. Then there exists a nondeterministic TM M that solves P in $2^k*(l+\log n)$ space. It is known that this TM can be simulated by a 3CM M' in such a way that each counter counts up to no more than $(n*2^l)^{2^k}$. (See, e.g., [11, 19].) Let p be the number of states in M'. In what follows, given an input x for M, we show how to construct a VASS $V_{M',x}$ to simulate the computation of M' on x such that M' accepts x iff $V_{M',x}$ is unbounded.

Informally, the state of $V_{M',x}$ is a pair (q, i), where q is a state in M' and i indicates the head position. The contents of the input tape will be implicitly embedded in the transitions of the VASS. Three pairs of positions will be used to simulate the three counters in M'. Moreover, from the previous lemma the operations of adding, subtracting, and testing for zero can be applied to each of these counters. The simulation is then straightforward (given Lemma 4.3). $V_{M',x}$ will allow a counter to become arbitrarily large iff the simulation results in the acceptance of x. At this point, it is important to realize that at most $O(p*|x|*n)$ states will be required in $V_{M',x}$. From Lemma 4.3 we have that the total number of transitions is bounded by $h'*k$ for some constant h'. Therefore, $V_{M',x}$ can be in VASS$(h'*k, l, h''*p*|x|*n)$ for some constant h''. Now one of the following must be true:

- $n \geqslant |x|/2$, and $V_{M',x}$ can be in VASS$(d*k, d*l, d*n^2)$ for $d=\max\{h', 2*h''*p\}$,
- $l \geqslant \log|x| \geqslant \log n$, and since a state can be simulated by three extra positions, $V_{M',x}$ can be in VASS$(h'*k+3, l+\log|x|+\log(n)+\log(p)+\log(h''), 1)$, which is certainly in VASS$(d*k, d*l, d*n^2)$ for $d=\max\{h'+3, 3+\log(p)+\log(h'')\}$, *or*
- $k \geqslant \log|x| \geqslant \log n$, and $O(\log|x|)$ extra positions can be used to simulate states, $V_{M',x}$ can be in VASS$(h'*k+c*\log|x|, l, 1)$ for some constant c, which is also in VASS$(d*k, d*l, d*n^2)$ for $d=\max\{h', c\}$.

Furthermore, each transition is of length $O(k*l+\log|x|+\log n)$. Hence, the length of $V_{M',x}$ is $O(|x|^3)$ (since $|x| \geqslant \max\{k, l, n\}$).

Now consider the complexity, given x, of constructing $V_{M',x}$. Clearly the construction can be carried out by a TM transducer whose work tape is of the form $\#i\#j\#$, where i and j are binary segments, of length $\log|x|$ and $(\log|x|+\log n)$, respectively, which are used to store a pointer to the input tape head and generate the states, respectively. Therefore, according to Lemma 4.1 one needs only $(2+\varepsilon)*(\log|x|+\log n)$ bits, for any $\varepsilon>0$, for the work space during the construction. This completes the proof. ∎

THEOREM 4.2. *There exist some constants c, c', and h independent of k, l, and n, such that the* BP *for* VASS($k\,l, n$) *requires* $2^{k/c-c'}*(l+\log n)$ *nondeterministic space for* $k \geqslant h$.

Proof. Assume that the BP can be solved in $(2^{k/c-c'}-\varepsilon)*(l+\log n)$ nondeterministic space for any constants c, c', and any $\varepsilon>0$. Let P be a problem in C. Now given any instance x (with parameters k, l, and n), we can construct a corresponding VASS in VASS($d*k$, $d*l$, $d*n^2$) with respect to $d\_((2+\varepsilon_1)*(\log|x|+\log n), |x|^3)$ reducibility, for some constant d (which depends only on P). Let $c=d$. From Lemma 4.2, we know that there are constants c_1 and c_2 such that P can be solved in $\Phi''(x)+c_2*\log\Phi''(x)$ nondeterministic space where $\Phi''(x) = (2^{k-c'}-\varepsilon)*(d*l+\log(d*n^2))+2*\log(c_1*|x|^3)+(2+\varepsilon_1)*(\log|x|+\log n)$. One can easily see that $\Phi''(x)+c_2*\log\Phi''(x)$ is within $2*d*(2^{k-c'}-\varepsilon)*(l+\log n)$ $+(9+c_2+\varepsilon_1+\varepsilon_2)(k+\log|x|+\log n)$ for any $\varepsilon_2>0$. Since $k\geqslant\log|x|$ or $l\geqslant\log|x|$ or $n\geqslant|x|/2$, the above amount is within $(2^k-\varepsilon+\varepsilon_1+\varepsilon_2)*(l+\log n)$ for arbitrary $\varepsilon_1, \varepsilon_2>0$ and $k\geqslant h$, where h is a constant independent of k, l, and n. Choosing $\varepsilon_1+\varepsilon_2<\varepsilon$, we have that P can be solved in $(2^k-\varepsilon_3)*(l+\log n)$ for $k\geqslant h$ and $\varepsilon_3>0$—a contradiction. ∎

Note that given a DLBA, we could construct an equivalent 3CM where two counters are used to simulate the tape and the third is used as a working counter. Moreover, each counter counts up to no more than $2^{c*|x|}$ for some constant c, where $|x|$ is the length of the DLBA's input string. Based on the discussion in this section, one can then construct a VASS in VASS($6, c*|x|, n$) to simulate the 3CM in a way that each counter is represented by a pair of positions. Consequently, for $k\geqslant 6$ the BP becomes PSPACE-hard. The reader should recall, however, that in Section 3, we were able to show that the BP was PSPACE-hard for $k\geqslant 4$.

In the remainder of this section, we examine the consequence of Theorem 4.1 with respect to networks of CFSMs. Before proceeding, the following definitions concerning CFSMs are required.

A CFSM M is a directed labelled graph with two types of edges, namely *sending* and *receiving* edges. A sending (receiving) edge, in M, is labelled *send*(M', g) (*receive*(M', g)), for some other machine M' and some message g in a finite set G of messages. If $|G|=1$, send(M', g) and receive(M', g) will be abbreviated as send(M') and receive(M'), respectively. A unique node in M is called the *initial node*. A network of CFSMs consists of two or more CFSMs each pair of which communicate by sending and receiving messages via two one-directional, potentially unbounded, FIFO channels. See [4, 6, 7, 24, 27] for more detailed definitions.

In what follows, we first establish the relationship between VASS($k, 1, n$) and CFSM(k), and then, as a consequence of Theorem 4.1, the BP for VASS($k, 1, n$) and CFSM(k), both can be solved in PTIME providing k is a known fixed constant. More precisely, we have the following two lemmas:

LEMMA 4.4. *For each network N in* CFSM(k) *with n states (i.e., the number of*

states in each machine is no greater than n), there exists a VASS V *in* VASS($O(k^2)$, 1, $O(n^k)$) *such that N is unbounded iff V is unbounded.*

Proof. The proof is quite easy and hence we only sketch the result. Each state of V will contain a state for each CFSM in N. Basically, each channel in N will be represented by a position of the vector of V, and each action of sending (receiving) a message to (from) a channel in N will be simulated by adding (subtracting) one to (from) the corresponding position of the vector. The corresponding state change of N is recorded in the state of V. It is also easy to see that this construction can be done in deterministic logspace (providing k is a fixed known constant). Details are left to the reader. ∎

LEMMA 4.5. *For each* VASS V *in* VASS(k, 1, n), *there exists a network of* CFSMs N *in* CFSM($O(\sqrt{k})$) *with* $O(p(2^k, n))$ *states, where p is a polynomial in* 2^k *and n, such that V is unbounded iff N is unbounded.*

Proof. Let V be in VASS(k, 1, n). We wish to assume that only one position is altered for each transition in V. If this is not the case, intermediate transitions can be introduced to make V behave in the desired fashion. Furthermore, no more than 2^k*n^2*k-1 additional states will be required. Let $m=2^k*n^2*k-1$. In what follows, we construct a network N in CFSM($O(\sqrt{k})$) with $p(m, n)$ states, where p is a polynomial in m and n, to simulate the computation of V in such a way that V is unbounded iff N is unbounded. Basically, N consists of the following machines and channels: (See Fig. 4.6.)

Machines:

- D
- $M_0(i)$ and $M_1(i)$ $(0 \leqslant i \leqslant \lceil \log k \rceil)$
- $N_i (0 \leqslant i \leqslant \lceil \sqrt{k} \rceil)$

Channels:

- For every i $(0 \leqslant i \leqslant \lceil \log k \rceil)$, there are channels between D and $M_0(i)$ $(M_1(i))$.
- For every i and j, $(0 \leqslant i \leqslant \lceil \log k \rceil, 0 \leqslant j \leqslant \lceil \sqrt{k} \rceil)$, there are channels between N_j and $M_0(i)$ $(M_1(i))$.
- For every i and j, $(0 \leqslant i, j \leqslant \lceil \sqrt{k} \rceil, i \neq j)$, there are channels between N_i and N_j.

Note that the total number of channels that connect N_i to N_j, over all i and j, is $(\lceil \sqrt{k} \rceil + 1)*\lceil \sqrt{k} \rceil$ $(>k)$. In the simulation, exactly k of them will be used to simulate the k positions in V. They can be chosen arbitrarily and are, in what follows, considered to be labelled 0,..., $k-1$. Let $T_{i,-1}$ be the machine N_j $(0 \leqslant j \leqslant \lceil \sqrt{k} \rceil)$ such that N_j's *input* channel is labelled i. Similarly, we define $T_{i,+1}$ to be the machine N_j such that N_j's *output* channel is labelled i. In this way, adding (subtracting) one to (from) position i in V will be simulated by letting $T_{i,+1}$ $(T_{i,-1})$

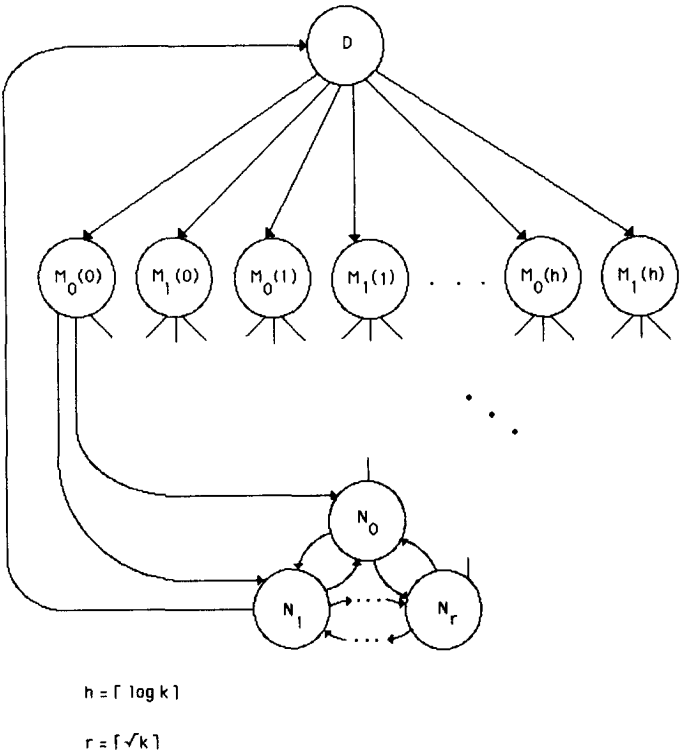


FIG. 4.6. The network N in CFSM($O(\sqrt{k})$).

send (receive) a message to (from) channel i. In other words, the number of messages in the channel will represent the value of the corresponding position in the vector.

Informally speaking, the machine D acts as a "driver" that simulates a state transaction of V and at the same time, issues commands that will cause the corresponding operations to occur on the vector. For every state i in V, there is a corresponding state, say v_i, in D. Now, consider the simulation of the transition in V

$$s \to (t, \langle 0,..., l,..., 0 \rangle)$$

where s and t are states in V, $l = 0$, $+1$, or -1. Furthermore, assume that l is in the qth $(0 \leqslant q \leqslant k-1)$ position. Let send(M) (receive(M)) denote the transition of sending (receiving) a message to (from) machine M. We have the following three cases:

Case 1. $l = 0$. In this case, since no change to the channel is needed, N will simply change from state v_s to state v_t, where v_s and v_t are states of D corresponding to states s and t of V, respectively.

Case 2. $l = +1$. In this case, a message should be added to channel q. Let $d_{h-1} \cdots d_1 d_0$ ($h = \lceil \log k \rceil$) be the binary representation of q. To simulate this move, D will execute the transitions

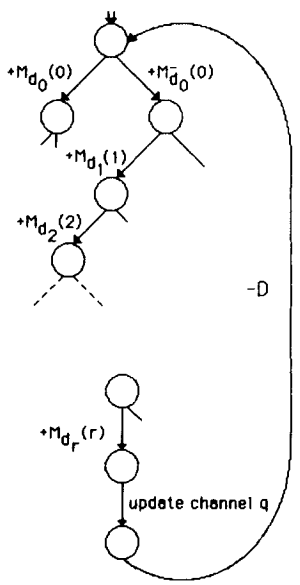
$$v_s - \text{send}(M_{d_0}(0)) \rightarrow v_0 - \text{send}(M_{d_1}(1)) \rightarrow \cdots v_{h-1} - \text{send}(M_1(h)) \rightarrow v_h - \text{receive}(T_{q,+1}) \rightarrow v_t.$$

The first $h-1$ sends are to indicate the address of the machine with channel q as its input or output channel. The last send is to represent an addition operation. The last receive is to wait for an acknowledgment from the machine that updated the channel q (i.e., $T_{q,+1}$).

Case 3. $l = -1$. Similar to Case 2 except that the last send is to the machine $M_0(h)$, and the last receive is from machine $T_{q,-1}$.

The structure of the machine $M_i(j)$ ($i = 0$, 1 and $0 \leqslant j \leqslant \lceil \log k \rceil$) is quite simple. Each time $M_i(j)$ receives a message from D, it simply sends a message to some N_r ($0 \leqslant r \leqslant \lceil \sqrt{k} \rceil$) chosen at random.

The structure of the machine N_r is shown in Fig. 4.7. Let $d_{h-1} \cdots d_0$ be the binary representation of q as defined above. If $N_r = T_{q,+1}$, then there is a branch with transitions receive($M_{d_0}(0)$),..., receive($M_{d_{h-1}}(h-1)$), receive($M_1(h)$), "send a message to the channel," send(D).



+M: receive a message from M
-M: send a message to M

FIG. 4.7. The machine N_r.

Similarly, if $N_r = T_{q,-1}$, then we have a sequence of transitions receive($M_{d_0}(0)$),..., receive($M_{d_{h-1}}(h-1)$), receive($M_0(h)$), "receive a message from the channel," send(D).

The simulation will then proceed in the following way. Each time a transition updating the position q of the vector in V is to be simulated, the driver D will send a message to each of the machines $M_{d_0}(0)$, $M_{d_1}(1)$,..., $M_{d_h}(h)$ $(0 \leqslant h \leqslant \lceil \log k \rceil)$, where $(d_h,..., d_0)$ is the binary representation of q. Intuitively, one may think of the M's as "registers" that store the address of the channel to be updated. The function of each N_j is first similar to that of a "decoder," after which N_j proceeds according to the messages (i.e., instructions) received from the M's. More precisely, the machine N_j with the channel q as its input (or output) channel will update that channel only when it receives one message from *each* $M_{d_i}(i)$ as stated above. So, if during this time all the M's do not choose to send their messages to the same (and correct) N_j, the simulation becomes blocked. Therefore proper simulations are guaranteed although each machine M sends messages to an N_j chosen at random. Finally, N_j sends an acknowledgment to D after finishing the change on channel q. The entire procedure then repeats.

Now consider the number of states in N. It should be fairly easy to see the following:

1. D has $O((m+n)^* \log k)$ states.
2. For every i, $M_0(i)$ (or $M_1(i)$) has $O(\sqrt{k})$ states.
3. For every i, N_i has $O(k)$ states.

Hence the total number of states is bounded by a polynomial in terms of 2^k and n. Moreover, there are $O(\sqrt{k})$ machines in N. Furthermore, the reader can check that the construction of N can be carried out in deterministic logspace (providing k is a fixed known constant). The lemma now follows. ∎

Now the BP for CFSM(k) was considered in [6], where it was noted that the problem was solvable in PSPACE, providing k was a fixed known constant. Here we show that, in such cases, the result can be improved to PTIME. The fact that the problem is solvable in PTIME follows from the result of Section 2. In what follows, we show that under a conjecture used in [13], the BP for CFSM(k) requires $O(|x|^{2^{k^2/c}})$ deterministic time, for some constant c. To show this, we require the following corollary which can be derived from a modified version of Theorem 4.2:

COROLLARY 4.1. *The* BP *for* VASS *of dimension* k *requires* $2^{k/c-c'}* \log|x|$ *nondeterministic space, for some constants* c *and* c'.

In [13], it was conjectured that for any $\varepsilon > 0$, NLOGSPACE($2^k* \log|x|$) $\not\subseteq$ DTIME($|x|^{2^k-\varepsilon}$). This indicates that for any k, NLOGSPACE($2^k* \log|x|$) contains problems which require at least $O(|x|^{2^k})$ deterministic time. Based on this conjecture and Lemma 4.2, we have the following corollary concerning the time complexity of the BP for VASS, of dimension k, and CFSM(k):

COROLLARY 4.2. *For some constants c, c', and h, the* BP *for* VASS *of dimension k (or* CFSM($\sqrt{k}$)*) requires* $O(|x|^{1/3*2^{k/c}-c'})$ *deterministic time for* $k \geq h$, *under the above conjecture.*

Proof. Assume that the BP can be solved in $O(n^{1/3*2^{k/c}-c'-\varepsilon})$ deterministic time, for any c and $\varepsilon > 0$. Then according to Lemma 4.2 and Theorem 4.1 (and choosing $c = d$), any problem in C (with $l = 1$) can be solved in $(c_1*|x|^3)^{1/3*2^k-c'-\varepsilon} + c_2*|x|*(2+\varepsilon_1)*(\log|x| + \log n)*2^{(2+\varepsilon_1)*(\log|x|+\log n)}$ deterministic time for some c_1, c_2, and any $\varepsilon_1 > 0$. This amount is within $O(|x|^{2^k-c'-\varepsilon+\varepsilon_2})$ deterministic time, for any $\varepsilon_2 > 0$. Since ε_2 can be arbitrary, any problem in C can be solved in $O(|x|^{2^k-c'-\varepsilon_3})$ deterministic time, for some $\varepsilon_3 > 0$. This contradicts the conjecture. ∎

ACKNOWLEDGMENT

We thank the referee for suggestions which improved the presentation of our results.

REFERENCES

1. T. AGERWALA AND M. FLYNN, Comments on capabilities, limitations, and correctness of Petri Nets, *in* "Proceedings of the First Annual Symposium on Computer Architecture," pp. 81–86, ACM, New York, 1973.
2. I. BOROSH AND L. TREYBIS, Bounds on positive integral solutions of linear Diophantine equations, *Proc. Amer. Math. Soc.* **55** (1976), 299–304.
3. S. COOK, Characterizations of pushdown machines in terms of time bounded computers, *J. Assoc. Comput. Mach.* **18** (1971), 4–18.
4. P. CUNHA AND T. MAIBAUM, A synchronous calculus for message oriented programming, *in* "Proceedings of the Second International Conference on Distributed Computing Systems, April 1981," pp. 443–445.
5. M. GAREY AND D. JOHNSON, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1979.
6. M. GOUDA, E. GURARI, T. LAI AND L. ROSIER, "Deadlock Detection in Systems of Communicating Finite State Machines," Tech. Rep. No. 84–11, University of Texas at Austin, Department of Computer Sciences, April 1984.
7. M. GOUDA AND L. ROSIER, Priority networks of communicating finite state machines, *SIAM J. Comput.* **14** (1985), 569–584.
8. E. GURARI AND T. LAI, Deadlock detection in communicating finite state machines, *ACM SIGACT News*, **15**, No. 4; **16**, No. 1 (Winter–Spring 1984), 63–64.
9. M. HACK, "Decidability Questions for Petri Nets," MIT, LCS, TR. 161, June 1976.
10. J. HOPCROFT AND J. PANSIOT, On the reachability problem for 5-dimensional vector addition systems, *Theoret. Comput. Sci.* **8** (1979), 135–159.
11. J. HOPCROFT AND J. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, Mass., 1979.
12. N. JONES, L. LANDWEBER AND Y. LIEN, Complexity of some problems in Petri nets, *Theoret. Comput. Sci.* **8** (1979), 277–299.
13. KASAI, T. AND S. IWATA, Gradually intractable problems and nondeterministic logspace lower bounds, *Math. Systems Theory* **18** (1985), 153–170.

14. R. KARP AND R. MILLER, Parallel program schemata, *J. Comput. System Sci.* **3** (1969), 147–195.
15. S. KOSARAJU, Limitations of Dijkstra's semaphore primitives and Petri nets, *Operating Systems Rev.* **7** (1973), 122–126.
16. E. LAWLER, "Combinatorial Optimization: Networks and Matroids," Holt, Rinehart & Winston, New York, 1976.
17. R. LIPTON, "The Reachability Problem Requires Exponential Space," Yale University, Department of Computer Sciences, Report No. 62, Jan. 1976.
18. E. MAYR AND A. MEYER, The complexity of the world problems for commutative semigroups and polynomial ideals, *Adv. in Math.* **46** (1982), 305–329.
19. M. MINSKY, "Computation: Finite and Infinite Machines," Prentice–Hall, Englewood Cliffs, N. J., 1967.
20. D. PARNAS., On a solution to the cigarette smoker's problem (without conditional statements), *Comm. ACM* **18** (1975), 181–183.
21. S. PATIL, "Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes," Group Memo 57, Project MAC, MIT, Feb. 1971.
22. J. PETERSON, "Petri Net Theory and the Modeling of Systems," Prentice–Hall, Englewood Cliffs, N. J., 1981.
23. C. RACKOFF, The covering and boundedness problems for vector addition systems, *Theoret. Comput. Sci.* **6** (1978), 223–231.
24. L. ROSIER AND M. GOUDA, On deciding progress for a class of communication protocols, *in* "Proceedings of the Eighteenth Annual Conference on Information Sciences and Systems, Princeton University, 1984," pp. 663–667.
25. L. ROSIER AND H. YEN, Boundedness, empty channel detection and synchronization for communicating finite state machines, *in* "Proceedings of the Second Annual Symposium on Theoretical Aspects of Computer Science, Saarbrucken, West Germany, 1985," pp. 287–298.
26. W. SAVITCH, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** (1970), 177–192.
27. Y. YU AND M. GOUDA, Unboundedness detection for a class of communicating finite state machines, *Inform. Process. Lett.* **17** (1983), 235–240.