AN ANALYSIS OF THE NONEMPTINESS PROBLEM FOR CLASSES OF REVERSAL-BOUNDED MULTICOUNTER MACHINES

Rodney R. Howell and Louis E. Rosier

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

TR-85-16 September 1985

Abstract

In this paper, we present an efficient nondeterministic algorithm to decide nonemptiness for reversal-bounded multicounter machines. This algorithm executes in time polynomial in the size of the input and the number of reversals the counters are allowed. Previously the best known upper bound required space logarithmic in the size of the input and linear in the number of reversals. We argue that our algorithm is optimal for many classes of these machines. Furthermore, we show that in most cases, the complexity of the nonemptiness problem does not change significantly when the reversal bound is dropped for one of the counters. In addition, we explore the changes in the complexity of the nonemptiness problem for these classes as different parameters are fixed or are given in either unary or binary. Our results yield as corollaries the answers to several unanswered questions regarding the disjointness, equivalence, and containment problems for reversal-bounded multicounter machines. Finally, we use our results to solve several problems regarding deadlock detection and unboundedness detection for systems of communicating finite-state machines.

1. Introduction

A comprehensive analysis of the decidable and undecidable properties of several classes of reversal bounded multicounter machines was given in [13]. Problems considered there include emptiness, disjointness, containment and equivalence. Some of the respective complexity issues were subsequently considered in [11]. Let DCM(m,r) (NCM(m,r)) be defined as the class of deterministic (nondeterministic) m-counter machines whose counters are allowed no more than r reversals. In [11], the nonemptiness problem was shown to have a lower bound of PSPACE for $\cup_{m,r}$ DCM(m,r) and an upper bound of EXPSPACE for $\cup_{m,r}$ NCM(m,r), where r is expressed in binary. In addition, several upper and lower bounds were given for the classes derived by fixing m and/or r; however, the only completeness results given were for DCM(m,r) and NCM(m,r), where m and r are fixed constants. Many of the results were also extended for the disjointness, containment, and equivalence problems.

In this paper, we define the class of machines NCM+(m,r) (DCM+(m,r)) as the set of machines in NCM(m,r) (DCM(m,r)) augmented with a single unrestricted counter. (This class was also studied in [10].) We then give a new nondeterministic algorithm to solve the nonemptiness problem for NCM+(m,r). [10] showed that if a machine in NCM+(m,r) accepted some input then it had a "short" accepting computation. As a result upper bounds for the nonemptiness problem, similar to those mentioned above for NCM(m,r), followed. We further show the existence of a "short" accepting computation with additional properties that allow large portions of the computation to be guessed very efficiently. The result is an algorithm that can detect, in time polynomial in |M| and r, a computation whose length is potentially exponential in m and r. We are able to show, using standard techniques, that this algorithm is optimal for the classes $\cup_{m,r}$ DCM(m,r), $\cup_{m,r}$ NCM(m,r), $\cup_{m,r}$ DCM+(m,r), and $\cup_{m,r}$ NCM+(m,r) (i.e., we give completeness results for these classes). Furthermore, we are able to show that, in most cases, the algorithm remains optimal regardless of whether m and r are fixed or given as parameters, or whether r is given in unary or binary. Table 1-1 summarizes these results, most of which are improvements over those shown in [10, 11].

m	r	NCM (DCM) Lower[1]	NCM (DCM) Upper[1]	NCM+ (DCM+) Lower[1]	NCM+ (DCM+) Upper[1]
fixed	fixed	LOGSPACE	LOGSPACE[2]	LOGSPACE	LOGSPACE[3]
parameter	fixed	NP	NP	NP	NP
1	unary	LOGSPACE	LOGSPACE[3,4]	NP	NP
fixed>1	unary	NP	NP	NP	NP
parameter	unary	NP	NP	NP	NP
1	binary	LOGSPACE	LOGSPACE[3,4]	PSPACE	2^{cn} time
2	binary	PSPACE	2^{cn} time	2^{dn} time	2^{cn} time
fixed>2	binary	2^{dn} time	2^{cn} time	2^{dn} time	2^{cn} time
parameter	binary	2^{dn} time	2^{cn} time	2^{dn} time	2^{cn} time

[1] For some fixed constants c and d.
[2] Using an algorithm from [11].
[3] Using an algorithm from [10].
[4] Using an algorithm from [19].

Table 1-1: Results concerning the nonemptiness problem for reversal-bounded multicounter machines. (All bounds are nondeterministic.)

Extensions of these results to the disjointness, containment, and equivalence problems are also discussed. Using techniques from [11], our results are easily shown to extend to the disjointness problem for NCM(m,r), DCM(m,r), and two-way reversal-bounded multicounter machines, and to the equivalence and containment problems for DCM(m,r). Although the equivalence (and, hence, containment) problem is undecidable for NCM(1,1) [1, 9, 11], and the disjointness and containment problems are undecidable for DCM+(0,0) (i.e., deterministic one-counter automata) [7, 13], whether our results extend to the equivalence problem for DCM+(m,r) remains an open question. Finally, we indicate that our results easily extend to the question of whether the number of reversals can exceed a given r, but we show that by not specifying r, or by allowing for

arbitrarily succinct representations of r, we can make the nonemptiness problem as hard as any decidable problem.

We then apply our results to questions regarding deadlock detection and unboundedness detection for communicating finite-state machines (CFSM's). Since the nonemptiness problem for $\cup_m$ NCM+(m,1) provides an upper bound for the deadlock detection problem (DDP) for systems of two CFSM's in which one channel is bounded over some given language of the form $a_1^* \ldots a_k^*$ [8], we are able to improve to NP the upper bound of PSPACE given in [8]. In addition, we show a lower bound of NP for this problem. Using similar techniques, we give other improvements and extensions to the results in [8].

We assume the reader is familiar with the basic concepts of automata theory (see, e.g., [6, 12]). The model of computation we use is the *nondeterministic Turing machine* (NTM). Unless otherwise noted, all completeness results in this paper are with respect to PTIME many-one reductions. The logspace reductions used are DLOGSPACE many-one reductions.

The remainder of the paper is organized as follows. Section 2 contains definitions of some of the terminology used in this paper. In Section 3, we present our algorithm for solving the nonemptiness problem for the classes DCM(m,r), NCM(m,r), DCM+(m,r), and NCM+(m,r), and give a careful analysis of the problem complexity, relating our results to other problems. In Section 4, we give applications of our results to the area of CFSM's.

2. Preliminary Definitions

DCM(m,r) (NCM(m,r)) is defined in [11] as the class of deterministic (nondeterministic) m-counter machines whose counters may make at most r reversals. We define DCM+(m,r) (NCM+(m,r)) similarly with the addition to each machine of a pushdown over a one-letter alphabet. This pushdown then behaves as a counter with no reversal bound, but we refer to it as a pushdown to avoid confusion with the reversal-bounded counters. [11] extends the definition of DCM(m,r) (NCM(m,r)) to the

class DCM(m,r,k) (NCM(m,r,k)) of two-way m-counter r-reversal machines whose input head may cross any boundary between input symbols at most k times. We define DCM+(m,r,k) (NCM+(m,r,k)) analogously.

The *nonemptiness problem* is the problem of deciding if a given machine M accepts some input string. The *disjointness problem* is the problem of deciding if the languages accepted by two given machines M_1 and M_2 are disjoint. The *equivalence problem* is the problem of deciding if the languages accepted by two given machines M_1 and M_2 are equivalent. The *containment problem* is the problem of deciding if the language accepted by one given machine M_1 is a subset of the language accepted by a second given machine M_2.

The description of a machine M will consist of a start state, a set of final states, a transition function, and an input alphabet. The length of this description will be denoted by |M|. Besides machine descriptions, an algorithm to solve any of the above problems must be given a value for r and, if applicable, a value for k. Therefore, we have two natural ways of evaluating problem complexity: in terms of the total size of the input, or in terms of $|M_1|$, $|M_2|$, r, and k. Unless otherwise noted, our complexity measures will be in terms of the total size of the input.

Let M be in NCM+(m,r,k) for some m,r,k. A *configuration* of M is an (m+2)-tuple $(q,p,c_1, \ldots ,c_m)$, where q is a state, p is the value of the pushdown, c_i is the value of the ith counter, $1 \leq i \leq m$. The *initial configuration* of M is the configuration $(q_0,0, \ldots ,0)$ where q_0 is the start state. A *final configuration* of M is a configuration M can reach when it halts.

A *computation* $\sigma = w_1 \ldots w_n$ on M is a series of configurations w_i, $1 \leq i \leq n$, such that w_1 is the initial configuration, w_n is a final configuration, and w_j can be reached from w_{j-1} in one move, for $1 < j \leq m$. If $\sigma = w_1 \ldots w_n$ is a computation on M and $\sigma' = w_i \ldots w_j$, $1 \leq i \leq j \leq n$, then σ' is a *subcomputation* on M. We sometimes refer to σ' as a *segment* of σ.

For any state q, a *loop on q* is a subcomputation $w_i \ldots w_j$ such that $w_i=(q,p_i,c_{i,1}, \ldots ,c_{i,m})$ and $w_j=(q,p_j,c_{j,1}, \ldots ,c_{j,m})$ for some state q, pushdown values p_i and p_j, and counter values $c_{i,1}, \ldots ,c_{i,m},c_{j,1}, \ldots ,c_{j,m}$. A *q-based loop* is a loop σ on q such that for any segment σ' of σ, if σ' contains s or more configurations, where s is the number of states in M, then at least one configuration of σ' contains state q.

If σ is a subcomputation on M and t is a transiton rule, then $\pi(\sigma,t)$ is the number of times t is taken in σ.

In any computation σ on a multicounter machine M, each counter must be in one of the following *counter modes* [11] at any given time:

- *zero* -- if its value is zero;
- *increasing* -- if its last change was an increment;
- *decreasing* -- if it is nonzero and its last change was a decrement.

A *counter-mode vector* [11] is a vector containing the current counter modes of each of the counters.

Communicating finite state machines (CFSM's) are defined in detail in [8]. Briefly, in a system of CFSM's, machines communicate exclusively by exchanging messages via connecting channels. There are two one-directional FIFO channels between any two machines in the system. Each machine has a finite number of states and transition rules, and each transition rule is accompanied by either sending one message to one of the machine's output channels or by receiving one message from one of the machine's input channels.

A CFSM is *deadlocked* if it is in a state in which all of the outgoing transition rules require a message to be received, but none of these messages ever become available. A system of CFSM's is deadlocked if every machine in the system is simultaneously deadlocked. The *deadlock detection problem* (DDP) is the problem of deciding if a given system of CFSM's can reach a deadlock. A channel in a system of CFSM's is *unbounded* if for any integer c, there exists a computation on the system which at some

time yields more than c messages simultaneously residing on that channel. The *unboundedness detection problem* (UDP) is the problem of deciding if a given system of CFSM's has an unbounded channel.

3. The Nonemptiness Problem

3.1. The Algorithm

Our strategy for deciding nonemptiness of a machine in NCM+(m,r) will be to guess segments of a computation such that in each of these segments, each counter is moving in only one direction. Hence, within each segment, the behavior of each counter is "nice" enough that the exact order of the transitions in the computation is not necessary to know. This is important, because from [11], the shortest accepting computation of a machine in NCM(m,r) (and, hence, in NCM+(m,r)) may have exponential length. However, the pushdown causes problems in that for a given computation, there may be an exponential number of points at which the pushdown must be checked for zero. In order to minimize the number of these points, we give the following lemmas. We use one-counter machines (1CM's) with no input tape in the lemmas because they capture precisely the characteristics that cause the problems.

Lemma 1: Let M be a 1CM with s states and no input tape. Let σ be a subcomputation on M. Then there exists a subcomputation σ' with the following properties:

1. σ' has the same beginning and ending configurations as σ;

2. $\pi(\sigma,t) = \pi(\sigma',t)$ for all transitions t;

3. σ' can be divided into $2s^2+3$ or fewer segments that satisfy the following properties:

   a. all segments except possibly the first and last begin and end with a counter value of s;

   b. in each segment, the counter either never exceeds $2s^4$ or never goes below s.

Proof: We will generalize a technique found in [19]. We begin by dividing σ into as few segments as possible that satisfy the above conditions, except that the number of segments might exceed $2s^2+3$. We will call the segments in which the counter begins and ends with s and never exceeds $2s^4$ *low segments*, and the segments in which the counter begins and ends with s and never goes below s *high segments*. Note that by minimizing the number of segments, we cause the high segments to alternate with the low segments (see Figure 3-1). We will now show how the computation can be "rearranged" so that the number of high segments is no more than s^2.

First, we note that in any high segment, the counter must exceed $2s^4$ at some point b; otherwise, it could have been included in an adjacent low segment, thus reducing the number of segments chosen. (If there are no low segments, σ can be divided into 3 or fewer segments, thus satisfying the conclusion.) Since the segment begins and ends with a counter value of s, the counter must have a value of s^4 at some point before and at some point after b. Let point a be the most recent time before b in which the counter value was exactly s^4, and let c be the next time after b in which the counter value is again s^4 (see Figure 3-1).

Define intervals (a_i,b_i), (b'_i,c_i), $0 \leq i \leq s-1$, where a_i is the last time before b that the counter value is s^4+i*s, b_i is the first time after a_i that the counter value is $s^4+(i+1)s$, b'_i is the last time before c that the counter value is $s^4+(i+1)s$, and c_i is the first time after b'_i the the counter value is s^4+i*s. Now between times a_i and b_i (b'_i and c_i), σ must contain some loop on some state q in which there was a gain (loss) in the counter value of between 1 and s. This must be the case, since during the interval (a_i, b_i), the counter gained s in value and at least s steps were executed.

Now there are s^3 such loops between points a and b. Hence, there must exist a state q_1 such that there are at least s^2 loops on q_1. Furthermore, there must exist a k, $1 \leq k \leq s$, such that there are at least s loops on q_1 with a gain in the counter contents of exactly k. Likewise there must exist a state q_2 and a j, $1 \leq j \leq s$, such that there are at least s loops on q_2 with a loss in the counter of exactly j. Let u, v be such that $k*u$ = the lowest common multiple of k and $j = j*v$ ($1 \leq u,v \leq s$).

Now if there is another high segment that passes first through q_1, then through q_2, with counter value at least 2s at both states, we can "move" u of the above loops on q_1 and v of the above loops on q_2 to this new segment

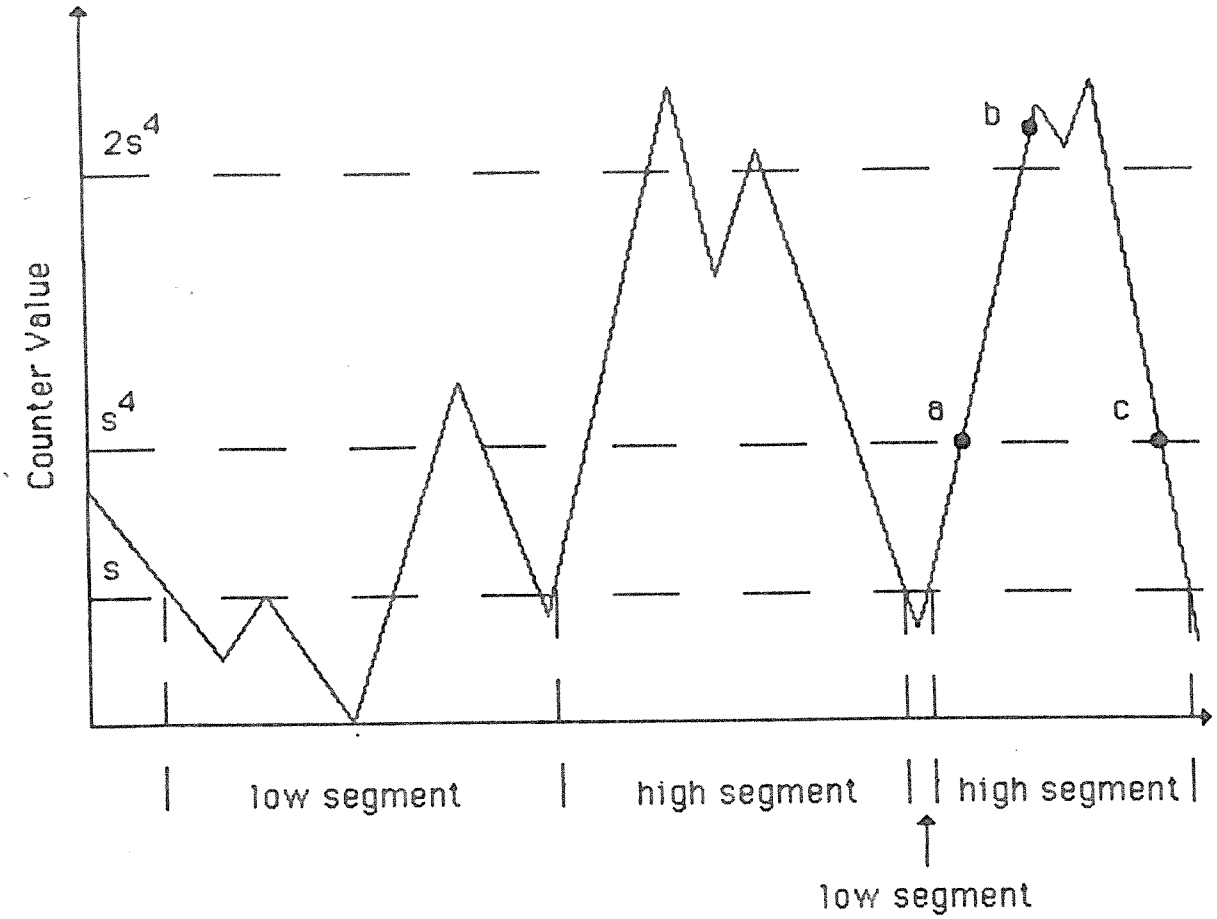


Figure 3-1 Division of a Subcomputation

without causing the counter to go below s in either segment if $s>1$. (If $s=1$, the computation can clearly be arranged arbitrarily.) Therefore, our strategy for rearranging the computation is as follows: starting with the second high segment, move loops in the manner described above to some prior high segment until it is no longer possible to do so; then go on to the next high segment. It will be impossible to move loops to a prior high segment only if either no prior high segment passes through the two necessary states in the proper order with the counter at least 2s, or the counter values for the current segment have been brought to $2s^4$ or less. In the former case, we have a new high segment passing through at least one pair of states which was not available in any previous high segment. In the latter case, we have a new low segment which can be merged with its neighboring low segment(s). Since there are s^2 distinct ordered pairs of states in M, we can thus generate a computation σ' that satisfies all of the given conditions. □

Our strategy of guessing a computation for a machine M in NCM+(m,r) will consist of guessing high and low segments, where the pushdown in M corresponds to the 1CM counter. Since the low segments have pushdown size bounded no greater than $2s^4$, we can translate them in PTIME into subcomputations on a machine in NCM(m,1). This allows us to guess the computation without worrying about the pushdown. However, more restrictions must be made to the high segments in order to allow us to guess them.

Lemma 2: Let M be a 1CM with s states and no input tape, and let σ be a subcomputation on M in which the counter does not go below s. Then there exists a subcomputation σ' on M with the following properties:

1. σ' has the same beginning and ending configurations as σ;

2. $\pi(\sigma,t) = \pi(\sigma',t)$ for all transitions t;

3. the counter stays strictly positive in σ';

4. σ' can be divided into 4s-1 or fewer segments with the following properties:

   a. all segments with more than s transitions are q-based for some state q;

   b. at any point in σ' that forms a boundary between segments, the counter has value at least s.

Proof: First, we mark in σ the first and last occurrences of each state. We now look for a segment of σ having at least s transitions but containing no marked points, except possibly at one endpoint. If no such segment exists, then we can divide σ at each of the marked points and have at most 2s-1 segments each containing no more than s transitions. Otherwise, we have a segment which must contain a q-based loop with no more than s transitions for some state q. If the loop causes a nonnegative change in the counter value, we can "move" it to the location immediately following the first marked occurrence of q (see Figure 3-2). This will not affect the counter value at any point following the original location of the loop or at any point preceeding the new location. All of the other points except those in the loop will have a nonnegative change in their counter values. Since the loop causes a nonnegative change in the counter value, and since the marked point has not changed value, both endpoints of the loop must retain a counter value of at least s. Since the loop has s or fewer transitions, the counter value must remain positive within the loop. Also note that no marked points have decreased in counter value; i.e., they all remain at least s. A similar argument may be given to allow a nonpositive q-based segment to be moved to the location immediately preceeding the last marked occurrence of q.

Since the above procedure does not reduce the counter value of any marked point, it may be repeated until no more loops can be moved. (To insure termination of this process, we must stipulate that once a loop has been moved, no transition within it may be moved again.) It should be clear that this procedure yields a computation σ' that satisfies the given conditions. □

We now present our algorithm for deciding the nonemptiness problem in NCM+(m,r).

Theorem 3: The nonemptiness problem for NCM+(m,r) is solvable in nondeterministic time polynomial in the size of the machine description and r. (I.e., there exists a positive constant c such that, for inputs of size n (r given in binary), the problem is solvable in 2^{cn} nondeterministic time.)

Proof: Let M be in NCM+(m,r) for some m and r. Suppose there is an accepting computation on M. From [1], there is a machine M_1 in NCM+(m$\lceil$(r+1)/2$\rceil$,1) whose size is polynomial in r and |M|, and which accepts the same language in the same amount of time as M. From [10], M_1 accepts some string if and only if it accepts a string in time $|M_1|^{cmr}$ for some fixed constant c. The computation of this string on M can be divided into at most m(2r+1)+1 subcomputations, each having a distinct counter-mode

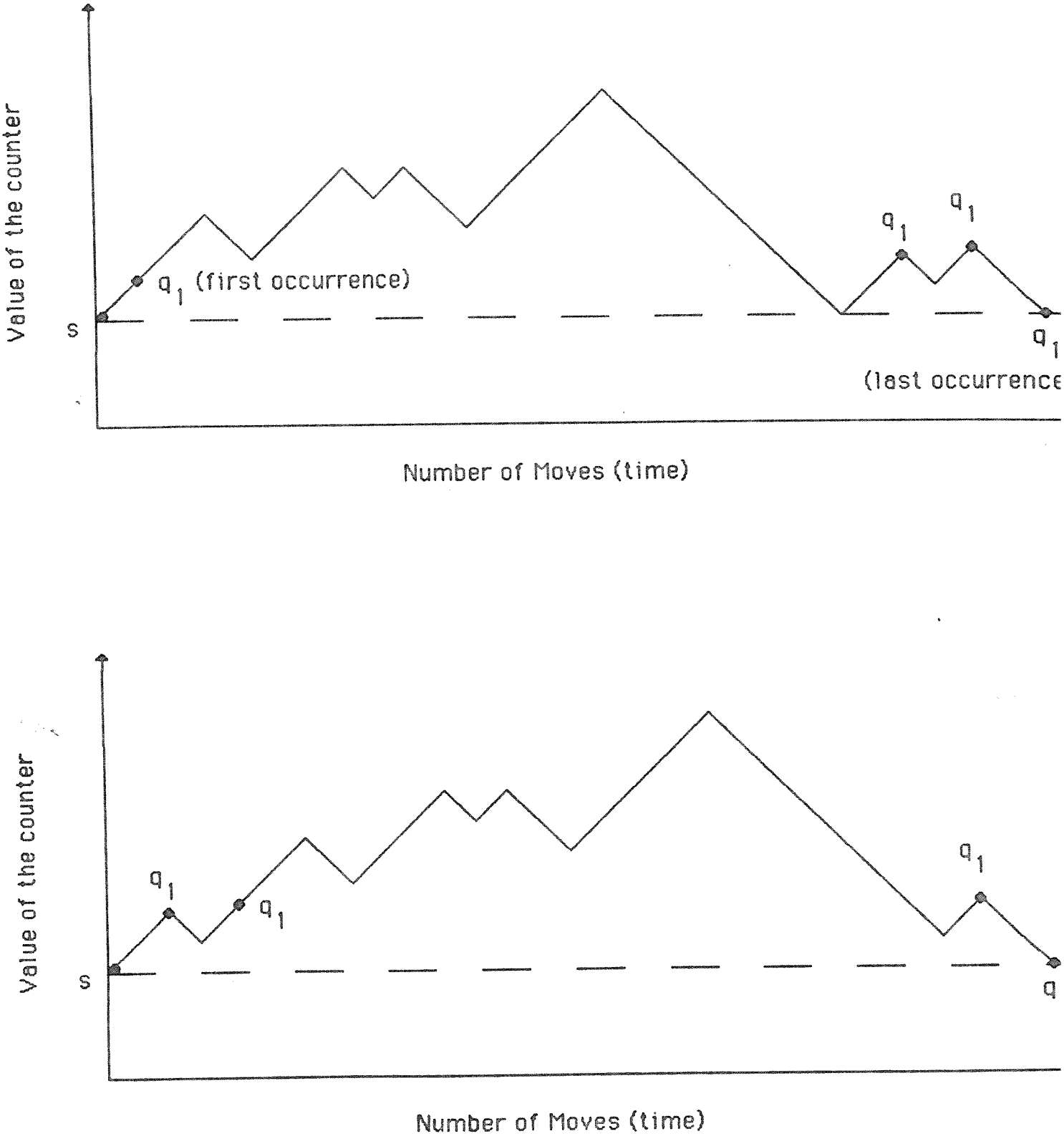


Figure 3-2 Effect of moving a q_1-based loop

vector. We can guess the configurations at the points at which the counter-mode vector changes: each configuration consists of a state and m+1 counter values, each of which can be stored in O(mr log(r|M|)) bits.

Consider one of the m(2r+1)+1 subcomputations. If the subcomputation is rearranged such that the same number of each transition rule is used, the n counters will yield the same result; we only need to be sure that the rearrangement does not cause the pushdown to become zero at the wrong time. To insure this, we will guess a computation that fits the conditions of Lemma 1.

We must therefore divide a subcomputation into at most $2s^2+3$ segments, where s is the number of states in M. We first consider the high segments. We will guess a subcomputation that fits the conditions of Lemma 2. Since we can clearly guess the segments that are no longer than s, we will explain only how we guess the q-based segments.

We wish to guess a q_1-based segment no longer than $O((r|M|)^{cmr})$ starting in configuration $(q_1,p,c_1,\ldots,c_m)$ where $p \geq s$. We create a vector $\alpha_0=(v_{0,1},\ldots,v_{0,s},p_0,c_{0,1},\ldots,c_{0,m})$ where $v_{0,1}$ is the number of "short" (i.e., s or fewer transitions) loops to be executed, $v_{0,2},\ldots,v_{0,s}=0$, $p_0=p$, and $c_{0,i}=c_i$, $1 \leq i \leq m$. Now, for each transition from q_1 that satisfies the current counter-mode vector, we guess the number of loops which will have that transition as their first move. We then create $\alpha_1=(v_{1,1},\ldots,v_{1,s},p_1,c_{1,1},\ldots,c_{1,m})$ where $v_{1,i}$ is the number of loops reaching state q_i after their first transition, $p_1=p_0+$ the sum of the effects of the guessed transitions on the pushdown, and $c_{1,j}=c_{0,j}+$ the sum of the effects of the guessed transitions on counter j (see Figure 3-3). We continue constructing vectors in this manner according to the following rules:

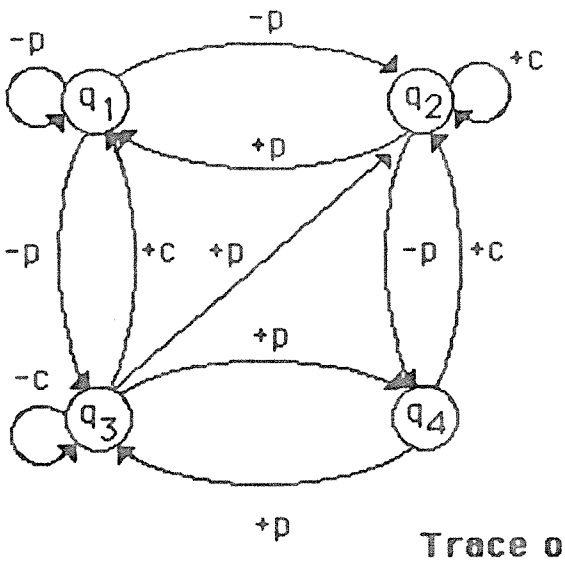
1. transitions from q_1 are taken only on the first move;

2. the total number of transitions taken on move i from state q_j should equal $v_{i-1,j}$, $1 \leq i \leq s$, $2 \leq j \leq s$.

We stop after creating α_s. If there exists $v_{s,i}$, $2 \leq i \leq s$, such that $v_{s,i} \neq 0$, or if $p_s<s$, the process fails.

We will now show that the above procedure can succeed if and only if there exists a q_1-based segment satisfying Lemma 2, and that if the procedure succeeds, there is such a segment yielding pushdown and counter values of $p_s,c_{s,1},\ldots,c_{s,m}$. Assume the above procedure succeeds. This clearly implies

We have the following machine in NCM+(1,2):



Where ± p indicates a change to the pushdown, and ± c indicates a change to the counter.

Beginning Configuration:

$(q_1, 4, 0)$

Ending Configuration:

$(q_1, 5, 11)$

Trace of the Algorithm

Number of Loops Ending

In State	After Move 0	1	2	3	4
q_1	15	1	6	7	15
q_2	0	8	4	6	0
q_3	0	6	0	2	0
q_4	0	0	5	0	0
Total Active	15	14	9	8	0

Number of Transitions Taken

	On Move 1	2	3	4
$q_1, q_1, -p$	1	0	0	0
$q_1, q_2, -p$	8	0	0	0
$q_1, q_3, -p$	6	0	0	0
$q_2, q_1, +p$	0	4	1	6
$q_2, q_2, +c$	0	2	3	0
$q_2, q_4, -p$	0	2	0	0
$q_3, q_1, +c$	0	1	0	2
$q_3, q_2, +p$	0	2	0	0
$q_3, q_3, -c$	----invalid---			
$q_3, q_4, +p$	0	3	0	0
$q_4, q_2, +c$	0	0	3	0
$q_3, q_3, +p$	0	0	2	0
Total	15	14	9	8

Value of

	After Move 0	1	2	3	4
Pushdown	4	-11	-4	-1	5
Counter	0	0	3	9	11

Figure 3-3 Trace of the Algorithm

the existence of a collection of loops on q_1, each with at most s transitions, and whose net effect on the pushdowns and counters is p_s-p_0, $c_{s,1}$-$c_{0,1}$, . . . , $c_{s,m}$-$c_{0,m}$. We can arrange these loops so that the ones causing a net gain in the pushdown are executed first, and the ones causing a net loss in the pushdown are executed last. Since both p_0 and p_s are at least s, the resulting subcomputation causes the pushdown to stay strictly positive. Hence, we have a subcomputation from $(q_1,p_0,c_0, \ldots ,c_m)$ to $(q_1,p_s,c_{s,1}, \ldots ,c_{s,m})$ satisfying the conditions of Lemma 2.

Now assume there exists a q_1-based segment satisfying the conditions of Lemma 2. This q_1-based segment can be broken into loops on q_1 with at most s transitions each. Our procedure can clearly guess these loops and succeed.

We now consider a subcomputation in which the pushdown does not exceed $2s^4$. We construct a new machine M′ in NCM+(m,1) in which each state is a pair (q,v), where q is a state of M and v is a value of the pushdown. Thus, there are $2s^5$ states in M′. M′ will simulate a subcomputation on M without using its pushdown; hence, we can assume the pushdown maintains a value of s throughout the subcomputation. Clearly, M′ can be constructed in time polynomial in |M|. We can now use the above procedure to guess the subcomputation. Therefore, we have a nondeterministic algorithm to determine whether M accepts some string, and the algorithm runs in time polynomial in |M| and r. □

The above algorithm runs in NP if r is a fixed constant or is given in unary. If r is given in binary, the algorithm is polynomial in |M| and $2^{|r|}$; hence, it runs in 2^{cn} time, where n is the total size of the input, and c is some fixed constant.

3.2. Lower Bounds

The following theorem and its corollary show that our algorithm is optimal for certain cases if EXPTIME $\neq$ NEXPTIME.

Theorem 4: There exists a positive constant d such that the nonemptiness problem for $\cup_r$ DCM(3,r) requires at least 2^{dn} nondeterministic time for infinitely many inputs, where n is the size of the input.

Proof: Let M be an NTM that operates in time 2^n on input of size n. For each input x we show how to construct in PTIME a machine M_1 in

$DCM(3,2^{c|x|})$ for some constant c such that M_1 accepts some input if and only if M can accept x. We first assign a fixed-length binary encoding to the symbols in the input and tape alphabets of M. We now construct M_1 to operate in two phases. In the first phase, it will push on one counter the unary equivalent of the binary encoding of x, where the least significant bit appears in the first character of x. The binary encoding of x will be a part of the finite-state control. A "0" will cause the number in the counter to be doubled (using one of the other counters as a work stack), and a "1" will double and add 1. Clearly, the size of this portion of M_1 is polynomial in the size of x.

In its second phase, M_1 will simulate M on x. M_1 will use one counter as the portion of the tape to the left of the input head, one counter as the portion of the tape to the right of the input head, and the other counter as a work stack. M_1 can always determine the least significant bit of one counter by dividing by two, using the work stack to retain the quotient. A tape symbol can be written to a counter in the manner outlined in phase 1. Thus, for each transition in M, M_1 makes at most some constant number of reversals. Since M is nondeterministic and M_1 is deterministic, M_1 will use its input tape to simulate the guesses of M. Hence, there is an input that M_1 accepts if and only if M can accept x.

Since M operates in time $2^{|x|}$, M_1 will have no more than $2^{c|x|}$ reversals for some constant c. Therefore, M_1 is in $DCM(3,2^{c|x|})$, and the reduction is in PTIME. Since M_1 can be written in space proportional to $|x|$, we can show that there is a positive constant d such that any NTM that solves the nonemptiness problem for $\cup_r DCM(3,r)$ requires at least 2^{dn} time for infinitely many inputs, where n is the size of the input. □

Corollary 5: The nonemptiness problems for the following classes are NEXPTIME-complete:

1. $\cup_{m,r} NCM(m,r)$ $(\cup_{m,r} DCM(m,r))$;
2. $\cup_{m,r} NCM+(m,r)$ $(\cup_{m,r} DCM+(m,r))$;
3. $\cup_r NCM(m,r)$ $(\cup_r DCM(m,r))$ where m is a fixed constant greater than 2;
4. $\cup_r NCM+(m,r)$ $(\cup_r DCM+(m,r))$ where m is a fixed constant greater than 1.

Corollary 5 is an improvement over [11], which shows the nonemptiness problem for $\cup_{m,r}$ DCM(m,r) is PSPACE-hard. [11] also shows that the nonemptiness problem for $\cup_r$ DCM(2,r) is NP-hard. [4] shows how a 2-counter machine can simulate a 4-counter machine. However, when the construction is applied to Theorem 4, the number of reversals becomes too large to write down in PTIME. The best we are able to show is the following theorem and its corollary.

Theorem 6: The nonemptiness problem for $\cup_r$ DCM(2,r) is PSPACE-hard.

Proof: Let M be a linear-bounded automaton (LBA), and let x be an input to M. We can construct in PTIME, as in Theorem 4, a machine M_1 in DCM(3,$2^{c|x|}$) for some constant c such that M_1 accepts some input if and only if M accepts x. Furthermore, the counter values in M_1 will not exceed $O(2^{|x|})$, so M_1 operates in time $O(2^{|x|})$. We now use a technique from [4] to construct a machine M_2 in DCM(2,$2^{c'|x|}$) for some constant c' such that M_2 will simulate M_1. M_2 will use one counter to record the contents of the 3 counters in M_1 and will use the other counter as a work stack. Suppose the counters of M_1 have values i, j, and k, respectively. M_2 represents these 3 numbers as $2^i3^j5^k$. To increment i, j, or k by 1, M_2 multiplies by 2, 3, or 5, respectively, using its work stack. Conversely, to decrement, we divide by 2, 3, or 5. Thus, every increment or decrement of M_1 requires a reversal in M_2. Therefore, M_2 is in DCM(2,$2^{c'|x|}$), and the construction is in PTIME. Since LBA acceptance is PSPACE-complete [16], the result follows. □

Corollary 7: The nonemptiness problems for $\cup_r$ NCM(2,r), $\cup_r$ DCM+(1,r), and $\cup_r$ NCM+(1,r) are PSPACE-hard.

We now consider the case in which r is fixed or given in unary. [11] shows that the nonemptiness problems for $\cup_m$ DCM(m,1) and $\cup_r$ DCM(2,r) are NP-hard. An examination of the proof reveals that the result for the latter problem holds even if r is expressed in unary. Therefore, from Theorem 3, we have the following corollary.

Corollary 8: The nonemptiness problems for the following classes are NP-complete:

1. $\cup_{m,r}$ NCM(m,r) ($\cup_{m,r}$ DCM(m,r)), where r is expressed in unary;

2. $\cup_{m,r}$ NCM+(m,r) ($\cup_{m,r}$ DCM+(m,r)), where r is expressed in unary;

3. $\cup_{r}$ NCM(m,r) ($\cup_{r}$ DCM(m,r)), where m is a fixed constant greater than 1 and r is expressed in unary;

4. $\cup_{r}$ NCM+(m,r) ($\cup_{r}$ DCM+(m,r)), where m is a fixed positive constant and r is expressed in unary;

5. $\cup_{m}$ NCM(m,r) ($\cup_{m}$ DCM(m,r)), where r is a fixed constant;

6. $\cup_{m}$ NCM+(m,r) ($\cup_{m}$ DCM+(m,r)), where r is a fixed constant;

Note that the nonemptiness problem for $\cup_{m}$ NCM+(m,1) is exactly the same as for r-CPM's and $\cup_{m}$ P(m,∞) (defined in [8] and [10], respectively), where, in each class, the pushdowns are over a one-letter alphabet. Hence, we have improved over results in [8, 10], which gave an upper bound of PSPACE for this class.

Finally, we have the following theorem.

Theorem 9: The nonemptiness problems for the following classes are log-complete for NLOGSPACE:

1. NCM(m,r) (DCM(m,r)), where m and r are fixed constants [11];

2. NCM+(m,r) (DCM+(m,r)), where m and r are fixed constants;

3. $\cup_{r}$ NCM(1,r) ($\cup_{r}$ DCM(1,r)), where r is expressed in unary;

4. $\cup_{r}$ NCM(1,r) ($\cup_{r}$ DCM(1,r)), where r is expressed in binary;

Proof: [15] shows the nonemptiness problem for DFA's to be NLOGSPACE-complete, thus giving us our lower bound. [11] shows NCM(m,r) to be in NLOGSPACE if m and r are fixed constants. It is a straightforward task to extend this result to NCM+(m,r). Finally, [19] shows the nonemptiness problem for deterministic 1CM's to be in NLOGSPACE. It is again easy to extend this result to nondeterministic 1CM's. □

3.3. Conclusions

We have now given extensive completeness results on the effects of fixing m, fixing r, and expressing r in unary as well as binary. From these results, we can draw several conclusions. First, nondeterminism does not change the complexity of these problems. This is because the question, "Does there exist an accepted string?" is no different from the question, "Does there exist an accepting computation?" Second, in all cases except possibly NCM(2,r), where r is in binary, the nonemptiness problem for NCM(m,r) has the same completeness result as does NCM+(m-1,r), $1 \leq m$, where NCM+(0,r) denotes 1CM's. In other words, allowing one counter in a machine in NCM(m,r) to have no reversal bound does not significantly change the complexity of the nonemptiness problem. This is interesting, because allowing two counters to have no reversal bound allows Turing machine power [17], thus rendering the nonemptiness problem undecidable [18].

We also note, as noted in [11], that given two machines, M_1 and M_2, in NCM(m,r), we can construct in PTIME a machine M in NCM(2m,r) that accepts the intersection of the languages accepted by M_1 and M_2; in fact, the construction can be done in DLOGSPACE. Furthermore, given any machine M in NCM(m,r), we can construct, in DLOGSPACE, machines M_1 and M_2 in NCM(m,r) such that M accepts some string if and only if there is a string accepted by both M_1 and M_2; we merely let M_1=M and M_2 accept all strings. Therefore, we can extend our results to sharpen known bounds on the complexity of the disjointness problem for NCM(m,r) (DCM(m,r)). Furthermore, since it is trivial to construct a machine that accepts the complement of the language accepted by a given machine M in DCM(m,r), we can extend these results to the equivalence and containment problems for DCM(m,r) [11]. Many of our results can also be extended to NCM(m,r,k) (DCM(m,r,k)) using techniques in [11].

Our results do not seem to extend to the containment or equivalence problems for NCM(m,r) or to the containment, equivalence, or disjointness problems for NCM+(m,r) (DCM+(m,r)) or NCM+(m,r,k) (DCM+(m,r,k)). As is noted in [1, 9, 11], the equivalence problem (and, hence, the containment problem) for NCM(1,1) is

undecidable. Furthermore, the containment and disjointness problems for DCM+(0,0) (deterministic one-counter automata) are undecidable [7, 13]. Also, since the above constructions double the counters, we end up with two pushdowns when attempting to extend our results to the equivalence problem for DCM+(m,r), and the equivalence problem is undecidable for 2CM's [17, 18]. Whether we can improve upon this construction is an open question. See [14], where related problems are considered.

Finally, we consider a couple of related questions. First, suppose we are given a machine M and a reversal bound r. We might ask the question "Is there any computation (accepting or not) that causes more than r reversals in a counter?" It should be clear that straightforward modifications to our proofs will extend our results to this question. Also, we might consider the nonemptiness problem for reversal-bounded multicounter machines in which the reversal bound is unknown. We can reduce, in a manner similar to Theorem 4, any halting NTM to this problem. Hence, even though this problem is decidable, it as hard as any decidable problem. Likewise, if we allow for representations of r that are more succinct than binary (e.g., a number n represents $2^{2^{\cdot^{\cdot^{2}}}}$, n 2's) we can make the problem arbitrarily hard.

4. Applications to CFSM's

We now turn to the problems of deadlock detection and unboundedness detection in CFSM's. [8] shows how the DDP for systems of two CFSM's in which one of the channels is over a given language of the form $a_1^* \ldots a_k^*$ can be reduced to the nonemptiness problem for r-CPM's whose pushdowns are limited to one-letter alphabets. It should be clear that a machine in NCM+(m,1) can be used instead of an r-CPM. Hence, from Theorem 3, this problem is in NP. We now show that it is NP-complete even if both channels are bounded over the given language.

Theorem 10: The DDP for a network of 2 CFSM's is NP-complete if one or both channels are bounded over a language $a_1^* \ldots a_k^*$ for some given $a_1, \ldots, a_k$.

Proof: We will use a technique found in [5, 11]. Let $F=C_1 \wedge \ldots \wedge C_m$ be an instance of 3-SAT (see [3]) over the variables $x_1, \ldots, x_n$. We will construct a network of 2 CFSM's (M_1,M_2) such that deadlock is possible in (M_1,M_2) if and only if F is satisfiable. M_1 will merely echo its input on its output channel. M_2 will guess a string 1^l0 and send it to M_1. M_2 can always have access to the integer l by simply sending to M_1 each bit it receives from M_1. Now let $p_1, \ldots, p_n$ be the first n prime numbers. M_2 checks whether the formula F is satisfied by the assignment

$$\begin{aligned} x_i &= 0 \quad \text{if } l \bmod p_i = 0 \\ &= 1 \quad \text{otherwise,} \end{aligned}$$

where $1 \leq i \leq n$. If the selection of l satisfies F, then M_2 enters an infinite read loop; otherwise, it enters an infinite loop writing 0's. Hence, (M_1,M_2) can enter a deadlocked configuration if and only if F is satisfiable. Since F contains 3m literals, both channels are bounded over the language $(1^*0^*)^{3m}$. By the Prime Number Theorem, $\sum_{i=1}^{n} p_i < n^3$. It follows that the size of M_2 is polynomial in the size of the formula F, and hence the transformation is in PTIME. Since 3-SAT is NP-complete [3], the result follows. □

We now consider the UDP for the same class of CFSM's. [8] shows this problem to be in PSPACE. We improve upon this result in the following theorem.

Theorem 11: The UDP for systems of two CFSM's is NP-complete if one or both of the channels are over a language $a_1^* \ldots a_k^*$ for some given messages $a_1, \ldots, a_k$.

Proof: We first show the problem to be NP-hard. First, observe that in the proof of Theorem 10, the system (M_1,M_2) will have an unbounded channel if and only if the selection of l does not satisfy F. We can therefore modify the construction so that (M_1,M_2) can have an unbounded channel if and only if F is satisfiable.

We now show the problem to be in NP. Let (M_1,M_2) be a system of two CFSM's such that the channel from M_1 to M_2 is over some given language $a_1^* \ldots a_k^*$. Note that the finite-state control for M_1 (M_2) can be divided into k phases depending on which message it is currently sending (respectively, receiving). By [8], we can assume the messages in the system are sent and

received in some order $(S_1^* R_2^* S_2 R_1)^* S_1^* (S_2 \cup R_2)^*$, where S_i and R_i denote the set of sending and receiving, respectively, transition rules of M_i, for $i=1,2$.

We can construct, as in [8], a machine M in NCM+(k+1,1) that simulates (M_1,M_2). The states of M will be pairs, (q_1,q_2), where q_1 is a state in M_1 and q_2 is a state in M_2. The message passing is simulated as follows:

1. Each receiving move of M_1 will be simulated simultaneously with a sending move of M_2; hence, the message need not be recorded.

2. M_1 sending a_i is simulated as follows:

   a. If M_2 is currently in phase i, increment the pushdown;

   b. otherwise, increment counter c_i.

3. M_2 receiving a_i is simulated as follows:

   a. M_2 can enter phase i from phase j, $j<i$, only if the pushdown and counters $c_j, \ldots, c_{i-1}$ are zero.

   b. If the pushdown is nonzero, decrement the pushdown.

   c. If the pushdown is zero, decrement counter c_i.

4. The number of messages sent after M_1 halts is recorded in counter c_{k+1}. (We allow the simulation to nondeterministically allow M_1 to halt at any time.)

The above construction yields a machine M in NCM+(k+1,1). Clearly, this construction is in PTIME, and (M_1,M_2) contains an unbounded channel if M contains an unbounded counter or pushdown. Furthermore, if the channel from M_1 to M_2 is unbounded, M will contain an unbounded counter or pushdown. It follows from [19] that if the channel from M_2 to M_1 is unbounded, then M_2 can enter a loop consisting entirely of sends; therefore, M_1 can halt, and the simulation will contain an unbounded counter. Thus, (M_1,M_2) contains an unbounded channel if and only if M contains an unbounded counter or pushdown.

Let s be the number of states in M, and suppose M has an unbounded counter or pushdown. Clearly, there must be an infinite computation σ on M. Since the counter mode vector can change at most some finite number of times, there must be some infinite segment σ' of σ in which the counter mode

vector remains unchanged. Since the number of states is finite, σ' must contain some loop l in which some counter or the pushdown has a net gain, no counter decreases, and the pushdown does not have a net loss. Without loss of generality, we can assume l has s or fewer transitions.

Our algorithm is now as follows. We guess in NP a loop l satisfying the above description. We then guess the status vector v for l and verify that it is consistent with l. Next, we determine in PTIME the minimum pushdown value p necessary for execution of l (note $p<s$). We must now determine whether the state q in which l begins can be reached with pushdown at least $p<s$ and status vector v. This problem can clearly be reduced in PTIME to the nonemptiness problem. If q is reachable with pushdown at least p and status vector v, then l can be pumped to yield an unbounded counter or pushdown. We have therefore shown the UDP to be in NP. □

Finally, we can extend two other results concerning the complexities of the DDP and the UDP. [8] shows both problems to be PSPACE-complete for systems of two CFSM's in which one channel is bounded by an integer h given in unary, and the DDP to be PSPACE-complete for systems of any number of CFSM's in which all channels are bounded by h given in unary. It is natural to ask how the complexity changes if h is given in binary. We give the following result.

Theorem 12: The following problems are EXPSPACE-complete:

1. the DDP for systems of two CFSM's in which one channel is bounded by an integer h given in binary;

2. the UDP for systems of two CFSM's in which one channel is bounded by an integer h given in binary;

3. the DDP for systems of CFSM's in which all channels are bounded by an integer h given in binary.

Proof: [2] gives a PTIME algorithm for constructing a system of 2 CFSM's to simulate an arbitrary NTM on a given input. In this construction, the channels are used solely to store the NTM tape contents and head position. Therefore, for any NTM M that operates in 2^{n^c} space for some fixed constant c on input of size n, and any input x, we can construct in PTIME a system of 2 CFSM's (M_1,M_2) whose channels are bounded by $O(2^{n^c})$, which can be

written in PTIME. Thus, we have our lower bound for 1 and 3. We can clearly modify this construction to yield a system with an unbounded channel if and only if M accepts x; therefore, we have our lower bound for 2. The algorithms given in [8] clearly run in EXPSPACE when h is expressed in binary. □

References

[1] Baker, B., and Book, R., Reversal-Bounded Multipushdown Machines, *Journal of Computer and System Sciences 8* (1974), 315-332.

[2] Brand, D., and Zafiropulo, P., On Communicating Finite-State Machines, *Journal of the ACM 30*, 2 (Apr 1983), 323-342.

[3] Cook, S., *The Complexity of Theorem-Proving Procedures*, pp. 151-158, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, (New York, 1971).

[4] Fischer, P., Turing Machines with Restricted Memory Access, *Information and Control 9*, 4 (1966), 364-379.

[5] Galil, Z., Hierarchies of Complete Problems, *Acta Informatica 6* (1976), 77-88.

[6] Garey, M., and Johnson, D., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* (W. H. Freeman and Company, San Francisco, 1979).

[7] Ginsburg, S., and Greibach, S., Deterministic Context-Free Languages, *Information and Control 9* (1966), 620-648.

[8] Gouda, M., Gurari, E., Lai, T., and Rosier, L., *On Deadlock Detection in Systems of Communicating Finite State Machines*, Rep. TR-84-11, (University of Texas at Austin, Austin, TX 78712, 1984). Revised Apr. 1985.

[9] Griebach, S., An Infinite Hierarchy of Context-Free Languages, *Journal of the ACM 16* (1969), 91-106.

[10] Gurari, E., *Transducers with Decidable Equivalence Problem*, Rep. TR-CS-79-4, (University of Wisconsin-Milwaukee, 1979). Revised 1982.

[11] Gurari, E., and Ibarra, O., The Complexity of Decision Problems for Finite-Turn Multicounter Machines, *Journal of Computer and System Sciences 22*, 2 (Apr 1981), 220-229.

[12] Hopcroft, J. and Ullman, J., *Introduction to Automata Theory, Languages, and Computation,* (Addison-Wesley, Reading, Mass., 1979).

[13] Ibarra, O., Reversal-Bounded Multicounter Machines and their Decision Problems, *Journal of the ACM 25* (1978), 116-133.

[14] Ibarra, O., and Rosier, L., On the Decidability of Equivalence for Deterministic Pushdown Transducers, *Information Processing Letters 13*, 3 (Dec 1981), 89-93.

[15] Jones, N., Space-Bounded Reducibility among Combinatorial Problems, *Journal of Computer and System Sciences 11* (1975), 68-85.

[16] Karp, R., Reducibility among Combinatorial Problems, in: Miller, R., and Thatcher, J., Ed., *Complexity of Computer Computations*, (Plenum Press, New York, 1972), pp. 85-103.

[17] Minsky, M., Recursive Unsolvability of Post's Problem of 'Tag' and Other Topics in the Theory of Turing Machines, *Annals of Mathematics 74*, 3 (1961), 437-455.

[18] Rice, H., Classes of Recursively Enumerable Sets and their Decision Problems, *Transactions of the AMS 89* (1953), 25-59.

[19] Rosier, L., and Gouda, M., *Deciding Progress for a Class of Communicating Finite State Machines*, pp. 663-667, Proceedings of the Eighteenth Annual Conference on Information Sciences and Systems, (Princeton, NJ, Mar 1984).