New Decidability Results Concerning Two-way Counter Machines and Applications

Oscar H. Ibarra*, Tao Jiang**, Nicholas Tran*, Hui Wang***

Abstract. We look at some decision questions concerning two-way counter machines and obtain the strongest decidable results to date concerning these machines. In particular, we show that the emptiness, containment, and equivalence problems are decidable for two-way counter machines whose counter is reversal-bounded (i.e., the counter alternates between increasing and decreasing modes at most a fixed number of times). We use this result to give a simpler proof of a recent result that the emptiness, containment, and equivalence problems for two-way reversal-bounded pushdown automata accepting bounded languages (i.e., subsets of $w_1^* \dots w_k^*$ for some nonnull words $w_1, \dots, w_k$) are decidable. Other applications concern decision questions about simple programs. Finally, we show that nondeterministic two-way reversal-bounded multicounter machines are effectively equivalent to finite automata on unary languages, and hence their emptiness, containment, and equivalence problems are decidable also.

1 Introduction

A fundamental decision question concerning any class C of language recognizers is whether there exists an algorithm to decide the following question: given an arbitrary machine M in C, is the language accepted by M empty? This is known as the emptiness problem (for C). Decidability (existence of an algorithm) of emptiness leads to the decidability of other questions such as containment and equivalence (given arbitrary machines M_1 and M_2 in C, is the language accepted by M_1 contained (respectively, equal to) the language accepted by M_2) if the class of languages defined by C is effectively closed under union and complementation.

The simplest recognizers are the finite automata. It is well known that all the different varieties of finite automata (one-way, two-way, etc.) are effectively equivalent, and the class has decidable emptiness, containment, and equivalence (ECE, for short) problems.

When the two-way finite automaton is augmented with a storage device, such as a counter, a pushdown stack or a Turing machine tape, the ECE problems become undecidable (no algorithms exist). In fact, it follows from a result in [Min61] that the emptiness problem is undecidable for two-way counter machines even over a unary

* Department of Computer Science, University of California, Santa Barbara, CA 93106. Research supported in part by NSF Grant CCR89-18409.

** Department of Computer Science and Systems, McMaster University, Hamilton, Ontario L8S 4K1, Canada. Research supported in part by NSERC Operating Grant OGP 0046613.

*** Department of Computer Science, University of Alabama in Huntsville, AL 35899.

input alphabet. If one restricts the machines to make only a finite number of turns on the input tape, the ECE problems are still undecidable, even for the case when the input head makes only one turn [Iba78]. However, for one-way counter machines, it is known that the equivalence (hence also the emptiness) problem is decidable, but the containment problem is undecidable [VP75]. The situation is different when we restrict both the input and counter. It has been shown that the ECE problems are decidable for counter machines with a finite-turn input and a reversal-bounded counter (the number of alternation between increasing and decreasing modes is finite, independent of the input) [Iba78]. It is also known that the ECE problems are decidable for two-way counter machines with a reversal-bounded counter accepting bounded languages (subsets of $w_1^* \dots w_k^*$ for some nonnull words $w_1, \dots, w_k$) [GI82]. Note that these machines can accept fairly complex languages. For example, it can recognize the language consisting of strings of the form $0^i 1^j$ where i divides j. The decidability for the general case when the input is not over a bounded language was left open in [GI82]. We resolve this question here: we show that the ECE problems are decidable for two-way reversal-bounded counter machines. We believe this is the largest known class of machines, which are a natural generalization of the two-way finite automaton, for which the ECE problems are decidable. This result has some nice applications. We use it to give a simpler proof of a recent result that the emptiness, containment, and equivalence problems for two-way reversal-bounded pushdown automata accepting bounded languages (i.e., subsets of $w_1^* \dots w_k^*$ for some nonnull words $w_1, \dots, w_k$) are decidable. Other applications of this result concern decision questions about some special classes of simple programs.

Finally, we consider the nondeterministic version of reversal-bounded multi-counter machines. We show that when the input alphabet is unary, these machines accept only regular languages. Since the proof is constructive, we obtain as corollaries that the emptiness, containment, and equivalence problems for these machines are also decidable. This resolves an open question raised in [GI79], where a similar result was shown for deterministic such machines. This is the strongest result one can obtain since it is known that deterministic reversal-bounded 2-counter machines over $a_1^* \dots a_k^*$ (for distinct symbols $a_1, \dots, a_k$ for some k) can accept nonsemilinear sets and have an undecidable emptiness problem [Iba78].

The rest of this paper is organized as follows. In Section 2, we show the decidability of the ECE problems for two-way reversal-bounded counter machines. We give the applications in Section 3. Finally, we show the effective equivalence of two-way reversal-bounded nondeterministic multicounter machines and finite automata over unary languages, along with the decidability of their ECE problems in Section 4.

In the remainder of this section, we define the models of computation of interest and related concepts. A *language* L is a subset of $\{0,1\}^*$. A language is *strictly bounded* over k letters $a_1, a_2, \dots, a_k$ if it is a subset of $a_1^* a_2^* \dots a_k^*$. A language is *bounded* over k nonnull words $w_1, w_2, \dots, w_k$ if it is a subset of $w_1^* w_2^* \dots w_k^*$. A *counter machine* is a finite automaton augmented with a counter, whose value can be incremented, decremented, or tested for zero. An r-input reversal 2DCM is a two-way counter machine whose input head makes at most r reversals (but the counter is unrestricted). A two-way machine is *sweeping* if the input head reverses only on the endmarkers. *2DCM(c, r)* denotes the class of deterministic machines having a two-way input head and c counters, each of which makes at most r reversals

in any computation. *2NCM(c, r)* denotes the corresponding nondeterministic class. *2DPDA(r)* denotes the class of two-way deterministic pushdown automata whose pushdown stack makes at most r reversals. *2NPDA(r)* denotes the nondeterministic class.

In the following sections, we will study the emptiness problems for the above machines on bounded languages. A straightforward argument shows that a machine of any type studied in this paper accepts a nonempty bounded language if and only if there is another machine of the same type that accepts a nonempty *strictly* bounded language. So when we are dealing with the emptiness question for machines over bounded languages, we need only handle the case when the machines accept strictly bounded languages.

Note: Due to the space constraint, we omit or abbreviate some proofs in this paper. They will appear in the full version.

2 Reversal-Bounded Counter Machines

It was shown in [Iba78] that many decision problems such as emptiness, containment, and equivalence for the class $2DCM(c, r)$ for $c \geq 2$ and $r \geq 1$ are undecidable. The same paper raised the question of the decidability of these problems for $2DCM(1, r)$ for $r \geq 1$. A partial answer was given in [GI82], where it was shown that the emptiness problem for $2DCM(1, r)$ for $r \geq 1$ over bounded languages is decidable.

In this section we give a complete answer to this question; we show that the emptiness, equivalence, and containment problems for the class $2DCM(1, r)$, where $r \geq 1$, are decidable. First, we consider the emptiness problem and improve the result of Gurari and Ibarra by removing the requirement that the language be bounded.

Theorem 1. *The emptiness problem for 2DCM(1,r) is decidable for every* $r \geq 1$.

Proof. Fix an $r \geq 1$, and let M be a *2DCM(1,r)* with q states. We show that if $L(M) \neq \emptyset$, then M must accept an input in some bounded language over k words $w_1, w_2, \ldots, w_k$, where k and $w_1, w_2, \ldots, w_k$ are effectively computable from M. Suppose M accepts some input x. Without loss of generality, we assume there are exactly $r + 1$ phases in the computation of M on x, such that in each phase the counter is either increasing or decreasing.

Define the *phase crossing sequence* C_i at the boundary of two inputs squares to be the sequence (s, d) of M's states and input head directions at the times the input head crosses this boundary during phase i. During an increasing phase, the input head crosses any boundary at most $2q$ times, or else M, being deterministic, would get into an infinite loop. Hence, there are at most $(q + 1)^{2q}$ different crossing sequences for an increasing phase. During a decreasing phase, the input head may get into a loop which terminates only when the counter becomes empty, and in this case the input head may cross some boundary $O(|x|)$ times. Suppose the loop is repeated j times for some $j \geq 0$. Then the input head either crosses a boundary a constant number of times, or $O(j)$ times, depending on whether the boundary is involved in the loop. Thus any crossing sequence during this phase can be written in the form $C_i = uv^j w$ where $|u|/2, |v|, |w| \leq 2q$. Hence there are at most $(q + 1)^{8q}$ different crossing sequences during a decreasing phase.

Define the *crossing sequence* at the boundary of two input squares to be the string $C_1\#C_2\#\ldots\#C_{2r}$ of the phase crossing sequences at that boundary. From the above paragraph, we can see that the number of different crossing sequences is bounded by $c = (q+1)^{8q(r+1)}$, which is independent of x. Hence we can rewrite x as $u_0v_1u_1\ldots v_ku_k$, such that $0 \leq |u_j|, |v_j| \leq c$ for $j = 1,\ldots,k$ and the crossing sequences at the left and right boundaries of each v_j are identical. Call each v_j a *movable segment* and each u_j a *separator segment.* We say two movable segments v_m and v_n are of the same type if they are identical and have the same crossing sequences at the left and right boundaries. There are at most c^3 different types. From the definition, we can see that M does not make a counter reversal while the input head is on a movable segment.

We now transform x to another string x' by performing the following operation repeatedly on x. If $x = z_1v_1z_2v_2z_3$, where v_1 and v_2 are movable segments of the same type, then we rewrite x as $z_1v_1v_2z_2z_3$, i.e. we group movable segments of the same type together. It is easy to see that at the end of each phase, the state and counter value of M in the computation on x are the same as those in the computation on x', and therefore, M accepts x iff M accepts x'. Apply the same process of partitioning and transforming recursively on z_1 and z_2z_3 (at most $|x|$ times), so that at the end, x can be written as a bounded word $x' \in w_1^*w_2^*\ldots w_k^*$ where $|w_i| \leq c$, $k \leq c^3$, c is independent of x, and M accepts x iff M accepts x'. But since the emptiness problem for *2DCM(1,r)* on bounded languages is decidable [GI82], the theorem follows.

In contrast to Theorem 1 we state the following result from [Iba78].

Theorem 2. *[Iba78] The emptiness problem for 1-input reversal 2DCM is undecidable.*

Next, we show the decidability of the equivalence and containment problems. To do that, we need to ensure that each machine *2DCM(1, r)* halts on every input. Below we prove that this is in fact true for every *2DCM(c, r)*. We first prove a lemma which shows how to modify a 2DFA to make it halt on every input. This lemma is slightly stronger than a similar result by Sipser [Sip80], since our simulating 2DFA behaves exactly as the original machine until it detects that looping has occurred.

Lemma 3. *For every 2DFA N we can construct an equivalent halting 2DFA N' such that on each input N' moves exactly like N until N enters a loop.*

Proof. Let $x = a_1a_2\ldots a_n$ be an input. For simplicity, assume N moves every step. For each $i, 1 \leq i \leq n$, define a relation $R(i)$ as follows: for any two states p and q of N, the pair (p,q) is in $R(i)$ iff in state p, N will leave cell i to the right and return to cell i in state q the next time. Clearly N doesn't halt iff it moves to some cell i in state q and (q,q) is in the transitive closure of $R(i)$. We construct a 2DFA N' which simulates N and at the same time tries to construct the relation $R(i)$ for each cell being visited. To simplify the presentation, here we will allow N' to construct the relation $R(i)$ nondeterministically. The nondeterminism can be eliminated by the standard subset construction technique. Let δ_N be the transition function of N, and suppose N starts at cell 1.

N' initializes $R(1)$ to $\emptyset$. In general, suppose that N is at cell j. If N moves to the left, N' constructs deterministically $R(j-1)$ from the current $R(j)$ as follows. For each pair (p,q), N' puts (p,q) in $R(j-1)$ iff

1. $\exists s[\delta_N(p,a_{j-1}) = (s,right)\ \&\ \delta(s,a_j) = (q,left)]$, or
2. $\exists s_1,\ldots,s_k[\delta_N(p,a_{j-1}) = (s_1,right), \delta_N(s_k,a_j) = (q,left)\ \&\ (s_1,s_2),\ldots,(s_{k-1},s_k) \in R(j)]$.

If N moves to the right, N' constructs (nondeterministically) $R(j+1)$ from the current $R(j)$ as follows: For each pair (p,q) in $R(j)$, M' guesses a sequence of distinct states $s_1,\ldots,s_k$, such that $\delta_N(p,a_j) = (s_1,right)$ and $\delta_N(k,a_{j+1}) = (q,left)$, and puts all $(s_1,s_2),\ldots,(s_{k-1},s_k)$ in $R(j+1)$. In doing this, N' also makes sure that no two pairs in the resulting $R(j+1)$ share the same first state. Moreover, N' also checks the validity of the current $R(j)$ and abandons the computation if some pair (p,q) in $R(j)$ is non-realizable, i.e., $\delta_N(p,a_j) = (p',left)$ for some state p' or there exists no state q' such that $\delta_N(q',a_{j+1}) = (q,left)$.

N' halts if it detects that at some cell j, N is in state q and (q,q) is in current $R(j)$. Clearly, during the simulation, a relation $R(j)$ may contain some non-realizable pairs. But it is not hard to see that these non-realizable pairs will not cause a false loop detection. Thus we can establish the following claims. Suppose that N is at cell j.

1. If N is in state q and (q,q) is in the current $R(j)$, then N is in a loop.
2. Suppose that the rightmost cell that N has visited so far is cell k. Then for any pair (p,q) such that in state p, M will leave cell j to the right, stay in the cells $j+1$ through k, and return to cell j for the first time in state q, (p,q) is contained in $R(j)$.

Hence, N' can detect if N will enter a loop correctly.

Theorem 4. *For any c and $r \geq 1$, we can effective convert a $2DCM(c,r)$ to a $2DCM(c,r)$ that halts on every input.*

Proof. For simplicity, we prove the theorem for the case $c = 1$. The idea is the same for $c > 1$. Let M be a $2DCM(1,r)$ machine. We construct an equivalent halting $2DCM(1,r)$ machine M'. M' basically simulates M faithfully. We know that M can enter a loop only when (i) M is in an increasing phase or (ii) M is in an decreasing phase but the moves do not affect (i.e., really decrease) the counter (for otherwise M would not be reversal-bounded). Call the latter a zero-decrease period. So besides simulating M, M' also checks if M will enter a loop in each increasing phase and each zero-decrease period. The fact that M' is able to realize this follows from the observation that M behaves like a 2DFA in an increasing phase and a zero-decrease period, and the above lemma.

Corollary 5. *For each c, $\bigcup_r 2DCM(c,r)$ is effectivcly closed under complementation, intersection, and union. In particular, the class of languages recognized by two-way deterministic reversal-bounded multicounter machines is effectively closed under boolean operations.*

Corollary 6. *The containment and equivalence problems for* $\bigcup_r 2DCM(1,r)$ *are decidable.*

Open Question: Is 2DCM(c,r) closed under union or intersection ?

We state below a related interesting halting result, where $2DCM(1,\infty)$ denotes a two-way deterministic 1-counter machine whose counter can make unrestricted (unbounded) number of reversals. (Its proof appears in the full version of this paper.)

Theorem 7. *Each* $2DCM(1,\infty)$ *machine accepting a bounded language can be made halting.*

We obtain from the above theorem the next corollary.

Corollary 8. *The class of* $2DCM(1,\infty)$ *on bounded languages is effectively closed under complementation, intersection and union.*

Open Question: Is $2DCM(1,\infty)$ closed under complementation or union ?

3 Some Applications

We give three applications of Theorem 1 in this section. First we give a simpler proof of the decidability of the ECE problems concerning reversal-bounded deterministic pushdown automata on bounded languages. This result first appeared in [IJTW] as a corollary of a rather difficult theorem.

Theorem 9. *The emptiness problem for* 2DPDA(r) *for* $r \geq 1$ *on bounded languages is decidable.*

The proof follows from Theorem 1 and the following lemma.

Lemma 10. *Let* M *be a 2DPDA(r) accepting a strictly bounded language over* $a_1^* \ldots a_k^*$. *We can effectively construct a* $2DCM(1,r)$ *machine* M' *(not necessarily accepting the same language) such that* $L(M)$ *is empty iff* $L(M')$ *is empty.*

Proof. We may assume, without loss of generality, that on every step, M pushes exactly 1 symbol on top of the stack, does not change the top of the stack, or pops exactly 1 symbol, i.e., M is not allowed to rewrite the top of the stack. In the discussion that follows, we assume M is processing an input that is accepted, i.e., the computation is halting.

A writing phase is a sequence of steps which starts with a push and the stack is never popped (i.e., the stack height does not decrease) during the sequence. A writing phase is periodic if there are strings u, v, w with v nonnull such that for the entire writing phase, the string written on the stack is of the form uv^iw for some i (the multiplicity) , and the configuration of M (state, symbol, top of the stack) just before the first symbol of the first v is written is the same as the configuration just before the first symbol of the second v is written. Note that w is a prefix of v. Clearly, a writing phase can only end when the input head reaches an endmarker or a boundary between the a_i's. A writing phase can only be followed by a popping

of the stack (i.e. reversal) or another writing phase with possibly different triple (u, v, w).

By enlarging the state set, we can easily modify M so that all writing phases are periodic. One can easily verify that because M is reversal-bounded, there are at most a fixed number t of writing phases (in the computation), and t is effectively computable from the specification of M.

We now describe the construction of M. Let $x = a_1^{i_1} \dots a_k^{i_k}$ be the input to M. The input to M' is of the form: $y\#c_1\#c_2\# \dots \#c_t$, where the c_i's are unary strings; y is x but certain positions are marked with markers $m_1, m_2, \dots, m_t$. Note that a position of y can have $0, 1, \dots$ at most t markers.

M simulates M' on the segment y ignoring the markers. The c_i's are used to remember the counter values. Informally, every time the counter enters a new writing phase, the machine "records" the current value of the counter by checking the c_i's.

M' begins by simulating M on y (ignoring the marks). When a writing phase is entered, M' records the triple (u_1, v_1, w_1) in its finite control and the multiplicity in the counter. Suppose M enters another writing phase. Then M' checks that the input head is on a symbol marked by marker m_1; hence M' can "remember" the input head position. (If it's not marked m_1, M' rejects.) Then it "records" the current value of the counter on the input by checking that the current value is equal to c_1. (If it's not, M' rejects.) M' restores the input head to the position marked m_1 and resets the counter to 0. It can then proceed with the simulation. Next time M' has to record the value of the counter, M' use the input marker m_2 and checks c_2, while storing the triple (u_2, v_2, w_2) in its finite control, etc. Popping of the counter is easily simulated using the appropriate triple (u, v, w) and the counter value. Note that if in the simulation of a sequence of pops, the counter becomes 0, M' must first retrieve the appropriate c_i (corresponding to the pushdown segment directly below the one that was just consumed) and restore it in the counter before it can continue with the simulation; retrieving and restoring the count in the counter requires the input head to leave the input position, but M' can remember the "new" position with a new "marker". If after a sequence of pops M enters a new writing phase before the counter becomes 0, the "residual" counter value is recorded as a new c_i like before. We leave the details to the reader. It is clear that M' in in 2DCM(1,r) for some r.

Since the proof of Theorem 4 can be trivially modified to show that reversal-bounded 2DPDA's can be made halting (even for those accepting unbounded languages), the class of languages they define is effectively closed under complementation. In particular, we have

Corollary 11. *The containment and equivalence problems for* 2DPDA(r) *for* $r \geq 1$ *on bounded languages are decidable.*

As a second application, we use Theorem 1 to show the decidability of the emptiness problem for some classes of simple programs. The motivation for the following definition comes from the study of real-time verification [AHV].

Definition 12. A *simple program* P is a triple (V, X, I), where $V = \{A_1, A_2, \dots, A_n\}$ is a finite set of *input variables*, $X \notin V$ is the *accumulator*, and $I = (i_1; i_2; \dots; i_l)$ is a finite list of *instructions* of the form

- label s: $X \leftarrow X + A_i$
- label s: $X \leftarrow X - A_i$
- label s: if $X = 0$ goto label t
- label s: if $X > 0$ goto label t
- label s: if $X < 0$ goto label t
- label s: goto label t
- label s: halt

Note that the program is not allowed to change the value of an input variable.

Definition 13. For a program P, EMPTY(P) = yes if there are integers (positive, negative, or zero) $a_1, a_2, \ldots, a_n$ such that P on this input halts; otherwise EMPTY(P) = no. The emptiness problem for simple programs is deciding given P if EMPTY(P) is yes.

At present, we do not know if the emptiness problem for simple programs is decidable. However, for some special cases, we are able to show that the problem is decidable.

One such special case is a program whose accumulator crosses the 0 axis (alternates between positive values and negative values) at most k times for some positive integer k independent of the input. Call such a program *k-crossing*, and in general, a program *finite-crossing* if it is k-crossing for some $k \geq 1$.

For example, a program with three inputs A, B, C that checks the relation $A = (B - C)i$ for some i can operate as follows: add A to the accumulator, and iterate adding B and subtracting C and checking if the accumulator X is zero after every iteration (the program halts if X is zero and goes into an infinite loop if X is negative). Although the accumulator alternates between increasing and decreasing modes arbitrarily many times (which depends on the input), the accumulator crosses the 0 axis at most once. So such a program is 1-crossing.

Again, we apply Theorem 1 to show the emptiness problem for finite-crossing simple programs is decidable.

Theorem 14. *The emptiness problem for finite-crossing simple programs is decidable, even when instructions of the form $X \leftarrow X + 1$ and $X \leftarrow X - 1$, are allowed.*

Proof. We give an algorithm to decide whether a finite-crossing simple program P halts on any input. Suppose P has l instructions and n input variables $A_1, A_2, \ldots, A_n$. Define the instantaneous description (abbreviated as ID) $(c_t, sign(X_t))$ of P at time t to be the label of the instruction being executed and the sign of the accumulator, which is either negative or nonnegative.

Consider a halting computation of P on some set of input values $a_1, a_2, \ldots, a_n$ as given by a sequence of ID's of P from start to finish. Since the accumulator of P alternates between positive and negative values at most k times for some $k \geq 1$, this computation can be divided into at most $k + 1$ phases such that every ID in a phase has the same second component. Because P is deterministic, some ID must be repeated after the first l steps in each phase, and hence the sequence of ID's in each phase can be written in the form of uv^iw, where each u, v, and w is a concatenation of at most l ID's.

Using this observation about finite-crossing simple programs, we construct a reversal-bounded counter machine M such that M accepts some input if and only if P halts on some input. Since M can make only a finite number of reversals on its counter, it cannot simulate P faithfully. Rather, M uses the "padding" technique described in Lemma 10. For each phase in the computation of P, M precomputes the effect made on the counter by each possible loop v of instructions and "stores" it in the input, i.e. M verifies that its input is padded with this extra information. From then on, to simulate a loop M needs only add its corresponding net effect to the counter, and hence to simulate a phase, M needs at most a constant number of counter reversals. Also, to simulate the increment and decrement instructions, M introduces two new variables a_+ and a_- which holds $+1$ and -1 respectively. It is easy to see that M accepts some input iff P halts on some input. By Theorem 1, it is decidable whether $L(M) = \emptyset$.

Remark Allowing the use of "constant" instructions of the form $X \leftarrow X+1$ and $X \leftarrow X-1$ makes simple programs computationally more powerful. For example, the relation $R = \{A : 2|A\}$ can easily be verified by a finite-crossing program with constant instructions, but not by any program without constant instructions, even if it is not finite-crossing.

Although finite-crossing programs can accept fairly complicated relations, they cannot verify multiplication and squaring.

Define MULT $= \{(A, B, C)|C = A * B\}$ and SQUARE $= \{(A, B)|B = A^2\}$.

Claim 15. *MULT and SQUARE cannot be verified by finite-crossing programs.*

Although finite-crossing simple programs may seem equivalent at first glance to counter machines whose counters can become zero only a finite number of times regardless of its input (we call those *finite-reset* counter machines), the latter are in fact much more powerful computational devices. For example, there is a finite-reset counter machine M that accepts a set S similar to MULT, namely, $S = \{0^x\#(0^y\#)^x : x, y \geq 1\}$. Hence the emptiness problem for finite-reset counter machines is undecidable, because they can multiply and hence compute the value of any polynomial $p(y, x_1, x_2, \ldots, x_n)$ (see [Cha87] for the proof of an analogous result). This contrasts with Theorem 14. We can use the same technique in Theorem 14 to show that the emptiness problem is decidable for finite-reset counter machines on *strictly bounded languages* (and hence bounded languages also; see the remark at the bottom of Section 1). From this and Corollary 8, we have

Theorem 16. *The emptiness, containment, and equivalence problems for finite-reset counter machines over strictly bounded languages are decidable.*

It is interesting to note that there are strictly bounded languages that can be recognized by a finite-reset counter machine but not by any finite-reversal counter machine. Using an easy cut-and-paste argument, one can show that the language $L = \{0^a1^b2^c : a = n(b - c)$ for some $n \geq 0\}$ is such an example.

There is a restricted class of simple programs allowing "nondeterminism" that arises in the theory of real-time verification [AHV]:

Definition 17. A *restricted nondeterministic simple program* is a simple program that allows more than one choice of instruction for each label and has the property that the accumulator X is nonnegative (nonpositive) after each instruction of the form $X \leftarrow X + A_i$ ($X \leftarrow X - A_i$).

It is an open question whether the emptiness problem for this class of simple programs is decidable. However, we can show that the emptiness problem is decidable for restricted *deterministic* simple programs.

Theorem 18. *The emptiness problem for restricted deterministic simple programs is decidable.*

We can allow comparisons between input variables and the emptiness problem still remains decidable.

Corollary 19. *The emptiness problem is decidable for finite-crossing simple programs and restricted deterministic simple programs which use additional instructions of the form:*

$$\textbf{if } p(A_i, A_j) \textbf{ goto } label$$

where A_i and A_j are input variables, and the predicate p is $|$ (for divides), $>, <$, or $=$.

4 Nondeterministic Reversal-bounded Multicounter Machines

We do not know if the emptiness problem for $\bigcup_r 2NCM(1, r)$ is decidable. In fact, the question is open even for $\bigcup_r 2NCM(1, r)$ machines accepting only bounded languages. However, we can show that the emptiness, equivalence, and containment problems are decidable for $\bigcup_c \bigcup_r 2NCM(c, r)$ on unary alphabet. More precisely, we prove that unary languages accepted by $\bigcup_c \bigcup_r 2NCM(c, r)$ machines are effectively regular. This settles a conjecture of Gurari and Ibarra [GI79]. In [GI79] it was only shown that unary languages accepted by $\bigcup_c \bigcup_r 2DCM(c, r)$ machines are effectively regular. Our technique is totally different. We first state a lemma that characterizes regular unary languages.

Lemma 20. *A unary language L is regular iff there is a constant c such that for every $x \in L$, $|x| > c$, there is some j, $1 \leq j \leq c$, such that $x' = 0^{jn}0^{|x|-j} \in L$ for all $n \geq 1$.*

Theorem 21. *Every unary language accepted by a 2NCM(c, r) is regular for $c, r \geq 1$.*

Proof. It suffices to show the theorem for 2NCM(c, 1), since we can reduce the number of counter reversals by any 2NCM(c, r) to 1 by adding more counters. We first show the theorem for $c = 1$, using the characterization given in Lemma 20, and then extend the proof to the general case.

Suppose L is accepted by some 2NCM(1, 1) M with q states. To avoid stating unnecessary constants, we will use the word "bounded" in the following to mean "bounded by some constant depending only on q".

Let 0^x be in L, and let C be an accepting computation of M on 0^x. C has a *simple loop* if there are some $t_1 < t_2$ such that at times t_1 and t_2 M is in the same state and on the same input square, and furthermore M does not visit an endmarker during $[t_1, t_2]$. A simple loop is called *positive* or *negative* depending on the net change it makes to the counter value. C has a *sweep* if there are some $t_1 < t_2$ such that at times t_1 and t_2 M is on an endmarker, and M does not visit an endmarker during (t_1, t_2).

So given a computation C, we decompose C into sweeps of length $O(|x|)$ after first removing all simple loops. Since each input square is visited at most q times in any sweep, the number of different crossing sequences is bounded by $t = (q+1)^{2q} + 1$, and so we can divide each sweep into a bounded number of clusters of identical movable segments (segments which have the same crossing sequences at both ends) separated by separator segments as explained in the proof of Theorem 1. Let b be the least common multiple of the lengths of all types of movable segments. After consolidating, we may assume that every movable segment has length b, and each sweep has exactly m movable segments for some $m = \Omega(|x|)$. (We consider sweeps that start and end on the same marker to have "empty" movable segments whose net change to the counter is zero.) We use $v(F)$ to denote the net change to the counter value by a computation fragment F (such as a loop or a sweep) of C. For example, $v(C) = 0$.

Suppose we add a segment of length b to the input 0^x. We can obtain from C an "almost" valid computation C' by duplicating one arbitrary movable segment in each sweep. (The simple loops are not affected since the input head never visits the endmarkers during a simple loop.) We call a set of movable segments obtained by taking one movable segment from each sweep a *cross-section* E of C. A cross-section E is called *positive* or *negative* depending on the sign of $v(E)$. Our strategy is to manipulate the sweeps, loops, and cross-sections to obtain from C an accepting computation of M on 0^{x+kbn} for some bounded k and all $n \geq 1$.

We have four cases:

1. C has a negative cross-section and a positive cross-section
2. C has only positive cross-sections, but also negative simple loops
3. C has only positive cross-sections, but also positive simple loops
4. C has only positive cross-sections, and no simple loops

In all cases, it can be shown (proof omitted) that some bounded k exists such that 0^{x+kbn} is in L for all $n \geq 1$. Hence by Lemma 20, L is regular. This concludes the proof of the theorem for $c = 1$.

Now we show how to extend the above proof to the general case. Suppose L is accepted by some 2NCM(c, 1) M. Let 0^x be in L, and let C be an accepting computation of M on 0^x. We partition C into simple loops, movable segments, and sweeps as for the case of 2NCM(1, 1), except that now these units are defined relative to a single counter. So there are c different partitions of C, each concerning with net changes to only one counter and ignoring the other counters. Proceed as in the

construction above to find the size of the segment to be added for each case, say $k_1 b$, $k_2 b$, ..., $k_c b$. Then the final input segment to be added is $k_1 k_2 \ldots k_c b^c$.

The proof of Theorem 21 also works even if we shorten instead of lengthen the input. Thus, to decide whether a 2NCM(c, r) accept some input, we only need to check all inputs of length at most a bounded constant. Since the membership problem for 2NCM(c, r) is decidable [Cha81], we have

Corollary 22. *The emptiness, containment, and equivalence problems for* $\bigcup_c \bigcup_r 2NCM(c,r)$ *on unary alphabet are decidable.*

This is the strongest result one can obtain since it is known that deterministic reversal-bounded 2-counter machines over strictly bounded languages can accept nonsemilinear sets and have an undecidable emptiness problem [Iba78].

References

[AHV] R. ALUR, T. HENZINGER, AND M. VARDI, *Parametric delays in real-time reasoning.* In preparation.

[Cha81] T.-H. CHAN, *Reversal complexity of counter machines*, in Proc. 13^{th} Symp. on Theory of Computing, ACM, 1981, pp. 146–157.

[Cha87] T.-H. CHAN, *On two-way weak counter machines*, Math. System Theory, 20 (1987), pp. 31–41.

[GI79] E. M. GURARI AND O. H. IBARRA, *Simple counter machines and number-theoretic problems*, J. Comput. System Sci., 19 (1979), pp. 145–162.

[GI82] ———, *Two-way counter machines and diophantine equations*, J. Assoc. Comput. Mach., 29 (1982), pp. 863–873.

[Iba78] O. H. IBARRA, *Reversal-bounded multicounter machines and their decision problems*, J. Assoc. Comput. Mach., 25 (1978), pp. 116–133.

[IJTW] O. IBARRA, T. JIANG, N. TRAN, AND H. WANG, *On the equivalence of two-way pushdown automata and counter machines over bounded languages.* Accepted for STACS 93.

[Min61] M. MINSKY, *Recursive unsolvability of Post's problem of tag and other topics in the theory of Turing machines*, Ann. of Math., 74 (1961), pp. 437–455.

[Sip80] M. SIPSER, *Halting space-bounded computations*, Theoretical Computer Science, 10 (1980), pp. 335–338.

[VP75] L. G. VALIANT AND M. S. PATERSON, *Deterministic one-counter automata*, J. Comput. System Sci., 10 (1975), pp. 340–350.