On the Emptiness Problem for Two-Way NFA with One Reversal-Bounded Counter *

Zhe Dang[1]**, Oscar H. Ibarra[2] and Zhi-Wei Sun[3]

[1] School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA

[2] Department of Computer Science
University of California
Santa Barbara, CA 93106, USA

[3] Department of Mathematics
Nanjing University
Nanjing 210093, China

Abstract. We show that the emptiness problem for two-way nondeterministic finite automata augmented with one reversal-bounded counter (i.e., the counter alternates between nondecreasing and nonincreasing modes a fixed number of times) operating on bounded languages (i.e., subsets of $w_1^* \cdots w_k^*$ for some non-null words $w_1, \ldots, w_k$) is decidable, settling an open problem in [11, 12]. The proof is a rather involved reduction to the solution of a special class of Diophantine systems of degree 2 via a class of programs called two-phase programs. The result has applications to verification of infinite state systems.

1 Introduction

Automata theory tries to answer questions concerning the relationship between formal languages and automata that recognize the languages. A fundamental decision question concerning any class of language recognizers is whether the emptiness problem (for the class) is decidable, i.e., whether there exists an algorithm to decide the following question: given an arbitrary machine M in the class, is the language accepted by M empty? Decidability of emptiness can lead to the decidability of other questions such as containment, equivalence, etc.

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 a decidable emptiness problem. When the two-way finite automaton is augmented with a storage device, such as a counter, a pushdown stack or a Turing machine tape, emptiness becomes undecidable (no algorithms exist). In fact, it follows from a result in [19] that the emptiness problem is undecidable for two-way finite automata augmented with one counter (even on a unary input alphabet). If one restricts

* The work by Oscar H. Ibarra has been supported in part by NSF Grant IIS-0101134.

** Corresponding author (zdang@eecs.wsu.edu).

the machines to make only a finite number of turns on the input tape, the emptiness problem is still undecidable, even for the case when the input head makes only one turn [11]. However, for such machines with one-way input, the emptiness problem is decidable, since they are simply pushdown automata with a unary stack alphabet.

Restricting the operation of the counter in a two-way one-counter machine makes the emptiness problem decidable for some classes. For example, it has been shown that emptiness is decidable for two-way counter machines whose input head is finite-crossing (i.e., for all inputs, the number of times the input head crosses the boundary between any two adjacent cells is bounded by a fixed number) and whose counter is reversal-bounded (i.e., the number of alternations between nondecreasing mode and nonincreasing mode is bounded by a fixed number, independent of the input) [11]. Interestingly, when the two-way input is unrestricted but the counter is reversal-bounded, emptiness is decidable when the machine is deterministic and accepts a bounded language (i.e., a subset of $w_1^* \ldots w_k^*$ for some nonnull words $w_1, \ldots, w_k$) [10]. This result was later shown to hold for the general case when the the input is not over a bounded language [12]. These machines are quite powerful. They can accept fairly complex languages. For example, such a machine can recognize the language consisting of strings of the form $0^i 1^j$ where i divides j. A question left open in [11, 12] is whether the aforementioned decidability of emptiness holds for *nondeterministic* machines (over bounded or unbounded languages). Our main result settles this question for the bounded case. More precisely, we show that the emptiness problem for two-way nondeterministic finite automata augmented with a reversal-bounded counter over bounded languages is decidable. At present, we are not able to generalize this result to the case when the input to the machine does not come from a bounded language. We note that when the machines are augmented with two reversal-bounded counters, emptiness is undecidable, even when the machines are deterministic and accept only bounded languages [11].

We believe that our main theorem will find applications in the area of verification. We note that the recent interest in counter machine models [4, 5, 7, 13, 6] is not motivated by investigations of their formal language properties but by their applications to model checking of infinite-state systems, motivated by the recent successes of model-checking techniques for finite-state systems [2, 3, 21, 15, 22]. The main result in this paper would be useful in establishing a number of new decidability results concerning various verification problems for infinite-state systems containing integer counters and parameterized constants.

The rest of this paper is organized as follows. In Section 2, we introduce some known results on reversal-bounded counters and number theory. These results are used in the proof of our main theorem. In Section 3, we show a decidable class of Diophantine systems of degree 2. The Diophantine systems are used in Section 4 to establish that a class of simple programs has a decidable emptiness problem. The main theorem follows in Section 5 by reducing it to the simple programs. We conclude in Section 6 with a verification example. Due to space limitation, most of the proofs are not included in the paper.

2 Preliminaries

Let c be a nonnegative integer. A c-counter machine is a two-way nondeterministic finite automaton with input endmarkers (two-way NFA) augmented with c counters, each of which can be incremented by 1, decremented by 1, and tested for zero. We assume, w.l.o.g., that each counter can only store a nonnegative integer, since the sign can be stored in the states. If r is a nonnegative integer, let 2NCM(c,r) denote the class of c-counter machines where each counter is r *reversal-bounded*; i.e., it makes at most r alternations between nondecreasing and nonincreasing modes in any computation; e.g., a counter whose values change according to the pattern $0\ 1\ 1\ 2\ 3\ 4\ \underline{4\ 3}\ 2\ 1\ \underline{0\ 1}\ \underline{1\ 0}$ is 3-reversal, where the reversals are underlined. For convenience, we sometimes refer to a machine in the class as a 2NCM(c,r). A 2NCM(c,r) is *finite-crossing* if there is a positive integer d such that in any computation, the input head crosses the boundary between any two adjacent cells of the input no more than d times. Note that a 1-crossing 2NCM(c,r) is a one-way nondeterministic finite automaton augmented with c r-reversal counters. 2NCM(c) will denote the class of c-counter machines whose counters are r-reversal bounded for some given r. For deterministic machines, we use 'D' in place of 'N'. If M is a machine, $L(M)$ denotes the language that M accepts.

A language is *strictly bounded* over k letters $a_1, a_2, \ldots, a_k$ if it is a subset of $a_1^* a_2^* \ldots a_k^*$. A language is *bounded* over k nonnull words $w_1, w_2, \ldots, w_k$ if it is a subset of $w_1^* w_2^* \ldots w_k^*$. 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 dealing with the emptiness question for machines over bounded languages, we need only handle the case when the machines accept strictly bounded languages. There are other equivalent definitions of "boundedness" that we will use in the paper.

We will also need the following results.

Theorem 1. *The emptiness problem is decidable for the following classes:*
(a) 2DCM(1) [12].
(b) 2NCM(c) over a unary alphabet (i.e., over a bounded language on 1 letter) [12].
(c) finite-crossing 2NCM(c) for every c [11, 9].

Let $\mathbf{N}$ be the set of nonnegative integers. Let X be a finite set of nonnegative integer variables. An *atomic Presburger relation* on X is either an atomic linear relation

$$\Sigma_{x \in X} a_x x < b,$$

or a mod constraint $x \equiv_d c$, where a_x, b, c and d are integers with $0 \leq c < d$. A Presburger formula can always be constructed from atomic Presburger relations using $\neg$ and $\wedge$. Presburger formulas are closed under quantification. Let Q be a set of n-tuples $(j_1, \ldots, j_n)$ in $\mathbf{N}^n$. Q is *Presburger-definable* if there is a Presburger formula $p(x_1, \ldots, x_n)$ such that the set of nonnegative integer solutions of p is exactly Q. It is known that Q is a semilinear set iff Q is Presburger-definable [8]. One may already notice that, for the purpose of this paper, we define a Presburger formula only over nonnegative integer variables (instead of integer variables).

Let $T(B, x)$ be a Presburger formula in two nonnegative integer variables B and x. T is *unitary* if T is a conjunction of atomic Presburger relations and each atomic linear relation in T is in the form of $a_1B + a_2x < b$ with $a_2 \in \{-1, 0, 1\}$. We say T is *1-mod-free* (resp. *2-mod-free*) if T does not contain any mod constraints in the form of $B \equiv_d b$ (resp. $x \equiv_d b$) for any b, d. We say T is *mod-free* if T is 1-mod-free and 2-mod-free. T is a *point* if T is $x = a \wedge B = b$ for some $a, b \in \mathbf{N}$. T is a *line* if T is $x = aB + b$, or T is $B = b$ (called a vertical line), for some $a, b \in \mathbf{N}$. T is a *sector* if T is $x \geq aB + b$, or T is $aB + b \leq x \leq a'B + b'$, for some $a < a', b, b' \in \mathbf{N}$. Observe that if T is mod-free and unitary, then T can be written into a (finite) disjunction of points, lines, and sectors. T is *single* if T is a point, a line, or a sector.

An *atomic $\mathcal{D}$-formula* over nonnegative integer variables $x_1, \ldots, x_n$ is either

$$f(x_1, \ldots, x_n) = 0$$

or a divisibility

$$f(x_1, \ldots, x_n) | g(x_1, \ldots, x_n)$$

where f and g are linear polynomials with integer coefficients. A *$\mathcal{D}$-formula* can be built from atomic $\mathcal{D}$-formulas using $\wedge$, $\vee$, and $\exists$. Notice that a Presburger formula is also a $\mathcal{D}$-formula. If a $\mathcal{D}$-formula does not contain $\exists$-quantifiers, the formula is called a ground formula. A set Q of n-tuples $(j_1, \ldots, j_n)$ in $\mathbf{N}^n$ is *$\mathcal{D}$-definable* if there is a $\mathcal{D}$-formula $p(x_1, \ldots, x_n)$ such that the set of nonnegative integer solutions of p is exactly Q. The following is Lipshitz's Theorem [16].

Theorem 2. *The satisfiability of $\mathcal{D}$-definable formulas is decidable.*

We will also need two basic results in number theory.

Theorem 3. *Let m_1, m_2 be positive integers and r_1, r_2 be nonnegative integers. The following two items are equivalent:*

(1) There is a nonnegative integer solution of n to $m_1|(n - r_1) \wedge m_2|(n - r_2)$,

(2) $\gcd(m_1, m_2)|r_1 - r_2$. [17]

The following is a well-known theorem of Frobenius (cf. [1, 14, 20]).

Theorem 4. *Let $a_1, \ldots, a_n$ be positive integers. Then there exists a positive integer b_0 such that, for each integer $b \geq b_0$ with $\gcd(a_1, \ldots, a_n)|b$, the linear equation*

$$a_1x_1 + \ldots + a_nx_n = b$$

has nonnegative integer solutions.

The main theorem of the paper is that the emptiness problem for 2NCM(1) over bounded languages is decidable. The next three sections constitute the entire proof. We first investigate a class of decidable Diophantine systems of degree 2 in Section 3. Then, we show that the emptiness problem for so-called "two-phase programs" is decidable in Section 4. The main theorem follows in Section 5 by reducing the emptiness problem for 2NCM(1) over bounded languages to the emptiness problem for two-phase programs.

3 A Decidable Class of Diophantine Systems of Degree 2

It is well-known that, in general, it is undecidable to determine if a Diophantine system of degree 2 (i.e., a finite set of Diophantine equations of degree 2) has a nonnegative integral solution, cf. [18]. In this section, we find a nontrivial decidable class of Diophantine systems of degree 2. This result will be used in our later proof.

Let $u, s_1, \ldots, s_m, t_1, \ldots, t_n, B_1, \ldots, B_k$ be nonnegative integer variables. A *positive linear polynomial* over $B_1, \ldots, B_k$ is in the form of $a_0 + a_1 B_1 + \ldots + a_k B_k$ where each a_i, $0 \leq i \leq k$, is a nonnegative integer. In this section, $U, U_i, \Phi, \Phi_i, V, V_j, \Gamma, \Gamma_j$ ($1 \leq i \leq m, 1 \leq j \leq n$) are positive linear polynomials over $B_1, \ldots, B_k$. Consider the following inequalities

$$\sum_{1 \leq i \leq m} U_i s_i + U \leq u \leq \sum_{1 \leq i \leq m} U_i s_i + U + \sum_{1 \leq i \leq m} \Phi_i s_i + \Phi \tag{1}$$

and

$$\sum_{1 \leq j \leq n} V_j t_j + V \leq u \leq \sum_{1 \leq j \leq n} V_j t_j + V + \sum_{1 \leq j \leq n} \Gamma_j t_j + \Gamma. \tag{2}$$

Θ is a predicate on nonnegative integer k-tuples such that, for all nonnegative integers $B_1, \ldots, B_k$, $\Theta(B_1, \ldots, B_k)$ is true iff the conjunction of (1) and (2) has a nonnegative integer solution for $u, s_1, \ldots, s_m, t_1, \ldots, t_n$. The following lemma states that Θ is effectively $\mathcal{D}$-definable; i.e., a $\mathcal{D}$-formula defining Θ can be computed from the description of (1) and (2). The proof uses Theorem 4 and Theorem 3.

Lemma 1. *The predicate $\Theta(B_1, \ldots, B_k)$ defined above is effectively $\mathcal{D}$-definable.*

4 Two-Phase Programs

In this section, we introduce an intermediate machine model called simple programs. A simple program is intended to model a class of nondeterministic programs with a single nondecreasing counter and a number of parameterized constants. For instance, consider the following simple program

Input (B_1, B_2, B_3);
1: $x := 0$;
2: Increment x by any amount (nondeterministically chosen) between B_1 and $2B_1$;
3: Nondeterministically goto 4, 5, or 7;
4: Increment x by B_2;
5: Increment x by B_3;
6: Goto 2;
7: Halt.

In the program, the input nonnegative integer variables remain unchanged during computation; i.e., they are parameterized constants. Each increment made on the counter satisfies some Presburger constraint in two variables; e.g., $B_1 \leq \delta \leq 2B_1$ holds for the increment δ made in step 2 above. A two-phase program is simply a pair of simple

programs G_1 and G_2 that share the same array of input variables $B_1, \ldots, B_k$. We are interested in the following question: is there an assignment for $B_1, \ldots, B_k$ such that the counter in G_1 and the counter in G_2 have the same value when both G_1 and G_2 halt? A decidable answer to this question will be given in this section. The reader might have noticed that there is some inherent relationship between two-phase programs and 2NCM(1) over bounded languages. Indeed, this intermediate result will be used in the next section to prove our main theorem. Before we proceed further, we need a formal definition.

A *simple program* G is a tuple

$$\langle S, B_1, \ldots, B_k, x, \mathbf{T}, E \rangle$$

where

- S is a finite set of control states, with two special states designated as the initial state and the final state.
- $B_1, \ldots, B_k$ are k input (nonnegative integer) variables,
- x is the nonnegative integer counter which is always nondecreasing,
- $\mathbf{T}$ is a finite set of Presburger formulas on two nonnegative integer variables,
- $E \subseteq S \times \{1, \ldots, k\} \times \mathbf{T} \times S$ is a finite set of *edges*. Each edge $\langle s, i, T, s' \rangle$ in E denotes a transition from state s to state s' while incrementing the counter x according to the *evolution pair* (i, T).

The semantics of G is defined as follows. A configuration $(s, v_1, \ldots, v_k, u)$ in $S \times \mathbf{N}^k \times \mathbf{N}$ is a tuple of a control state s, values $v_1, \ldots, v_k$ for the input variables $B_1, \ldots, B_k$, and value u for the counter x.

$$(s, v_1, \ldots, v_k, u) \rightarrow^G (s', v'_1, \ldots, v'_k, u')$$

denotes a *one-step transition* satisfying the following conditions:

- There is an edge $\langle s, i, T, s' \rangle$ in G connecting state s to state s',
- The value of each input variable does not change; i.e., $(v_1, \ldots, v_k) = (v'_1, \ldots, v'_k)$,
- The evolution pair (i, T) is satisfied; i.e., $T(v_i, u' - u)$ is true (hence, $u \leq u'$ since T is defined on nonnegative integers).

A *path* is a finite sequence

$$(s_0, v_1, \ldots, v_k, u_0) \ldots (s_i, v_1, \ldots, v_k, u_i) \ldots (s_m, v_1, \ldots, v_k, u_m)$$

for some $m \geq 1$ such that $(s_i, v_1, \ldots, v_k, u_i) \rightarrow^G (s_{i+1}, v_1, \ldots, v_k, u_{i+1})$ for each $0 \leq i \leq m-1$. In particular, if $u_0 = 0$ (the counter starts from 0), s_0 is the initial state and s_m is the final state, then G *accepts* $(v_1, \ldots, v_k, u_m)$.

A *two-phase program* G_{+-} consists of two simple programs G_+ and G_- that share the same S, input variables $B_1, \ldots, B_k$ and $\mathbf{T}$. We shall use x_+ (resp. x_-) to denote the counter in the *positive* (resp. *negative*) program G_+ (resp. G_-). A k-tuple of nonnegative integer values $v_1, \ldots, v_k$ is *accepted by the two-phase program* G_{+-} if there is a counter values u such that $(v_1, \ldots, v_k, u)$ is accepted by both G_+ and G_-. We shall use $L(G_{+-})$ to denote all the k-tuples accepted by G_{+-}. $L(G_{+-})$ is called the

tuple language accepted by G_{+-}. A two-phase program models some one counter system where the counter starts from 0 and, after a number of increments followed by a number of decrements, moves back to 0. In G_{+-}, the positive program models the increasing phase and the negative program models the decreasing phase (but the counter in the negative program is always increasing). Therefore, we need further argue whether the total increments made by the positive program equals the total increments made by the negative program. The main result of this section is that the tuple language accepted by a two-phase program G_{+-} is $\mathcal{D}$-definable. The proof first shows that it suffices to consider a special form of a two-phase program G_{+-}: each $T \in \mathbf{T}$ is a point, a line, or a sector. Then, the result follows by making use of Lemma 1.

Theorem 5. *The tuple language accepted by a two-phase program is $\mathcal{D}$-definable.*

Consider a finite set of two-phase programs $\mathcal{G}$, each of which has k-ary input B_1, ..., B_k. The Presburger emptiness problem for $\mathcal{G}$ is to decide, given a Presburger formula $R(B_1, \ldots, B_k)$, whether there is some input $B_1, \ldots, B_k$ accepted by each program in $\mathcal{G}$. Since $R(B_1, \ldots, B_k)$ is $\mathcal{D}$-definable and $\mathcal{D}$-definability is closed under intersection, we have

Theorem 6. *The Presburger emptiness problem for a finite set of two-phase programs is decidable.*

5 2NCM(1) over Bounded Languages

Before we discuss 2NCM(1,r), we first look at a property of a 2NCM(1,0) M over a unary input (i.e., a two-way NFA with a unary input tape augmented with a nondecreasing (i.e., monotonic) counter). The input is in the form of

$$\underbrace{¢a \ldots a}_{B}\$$$

of size B for some B, where ¢ and \$ are the left and right endmarkers. M works exactly as a two-way NFA except that, at some move (i.e., a left move, a right move, or a stationary move), M can increment the counter by 1. Suppose the counter initially starts from 0. When the input head is initially at the left endmarker, we use M_{LL} (resp. M_{LR}) to denote the restricted version of M that M returns to the left (resp. right) endmarker upon acceptance (during which M does not read the endmarkers). When the input head is initially at the right endmarker, M_{RR} and M_{RL} are defined similarly. We use $T_{LL}(B, x)$ (resp. $T_{LR}(B, x), T_{RR}(B, x), T_{RL}(B, x)$) to stand for the fact that M_{LL} (resp. M_{LR}, M_{RR}, M_{RL}) accepts the input of size B and upon acceptance, the counter has value x.

If we allow the input head to return to the endmarkers for multiple times, $T(B, x)$ can not be characterized by a Presburger formula. For instance, let M be such as machine. M keeps scanning the input (of size $B \geq 1$) from ¢ to \$ and back, while incrementing the counter. M nondeterministically accepts when \$ is reached. Obviously, $T_{LR}(B, x)$ now is exactly $\exists n(2nB + B|x)$ that is not Presburger. However, with the restrictions of M_{LR}, $T(B, x)$ is Presburger. The proof uses a complex loop analysis technique.

Lemma 2. *$T_{LL}(B,x)$, $T_{LR}(B,x)$, $T_{RR}(B,x)$, and $T_{RL}(B,x)$ are Presburger for any M specified above.*

Lemma 2 also works for a stronger version of M. We assume the counter in M is r-reversal-bounded for some r (instead of increasing only). Notice that the counter when decreasing can have negative values. We may similarly define restricted machines $M_{LL}, M_{LR}, M_{RL}, M_{RR}$. We shall use $T^+_{LL}(B,x)$ (resp. $T^-_{LL}(B,x)$) to denote that $x \geq 0$ and x (resp. $-x$) is the final value of the reversal-bounded counter in M_{LL} on input of size B. Similarly, we may define $T^+_{LR}(B,x), T^-_{LR}(B,x)$ etc.

Lemma 3. *$T^+_{LL}(B,x)$, $T^-_{LL}(B,x)$, $T^+_{LR}(B,x)$, $T^-_{LR}(B,x)$, $T^+_{RR}(B,x)$, $T^-_{RR}(B,x)$, $T^+_{RL}(B,x)$, and $T^-_{RL}(B,x)$ are Presburger for any M specified above and the counter in M is reversal-bounded.*

In the rest of this section, we focus on the emptiness problem for 2NCM(1,r) on bounded languages. A slightly different definition of boundedness, but equivalent to the one we gave in Section 2 with respect to decidability of emptiness is the following. A *k-bounded language* is a subset of

$$b_1 a_1^* b_2 a_2^* \dots b_k a_k^* b_{k+1}$$

where b_i, $1 \leq i \leq k+1$, is the i-th delimiter, and each block a_i^* between the two delimiters b_i and b_{i+1} is the i-th block. A bounded language is a k-bounded language for some k. Recall that a 2NCM(1,r) is a two-way NFA augmented with an r-reversal-bounded counter. When the input language of a 2NCM(1,r) M is restricted to a bounded language, M is called a 2NCM(1,r) over a bounded language.

Let M be a 2NCM(1,1) working on a k-bounded language. Let

$$w = b_1 1^{B_1} \dots b_k 1^{B_k} b_{k+1}$$

be an input word where 1^{B_i} is the i-th block of symbol 1's with length B_i. Sometimes, we simply call the input as $(B_1, \dots, B_k)$. Without loss of generality, we assume that the counter x in M, when M accepts the input, returns to 0 and the input head is on delimiter b_{k+1} with M being at the final state. An accepting computation C of M can be divided into a number of *segments*. Each segment is associated with a state pair (s, s') and a block 1^{B_i}. In the sequel, we shall use $\alpha, \beta, \dots$ to denote a segment. We have the following four cases:

(1). (a LL-segment) M, at state s, reads the $i+1$-th delimiter and M returns to the $i+1$-th delimiter with state s', during which M only reads symbols in 1^{B_i}.

(2). (a LR-segment) M, at state s, reads the i-th delimiter and M returns to the $i+1$-th delimiter with state s', during which M only reads symbols in 1^{B_i}.

(3). (a RR-segment) M, at state s, reads the i-th delimiter and M returns to the i-th delimiter with state s', during which M only reads symbols in 1^{B_i}.

(4). (a RL-segment) M, at state s, reads the $i+1$-th delimiter and M returns to the i-th delimiter with state s', during which M only reads symbols in 1^{B_i}.

A segment is *positive* (resp. *negative*) if the net counter change is ≥ 0 (resp. < 0) on the segment. Therefore, since the counter is one reversal-bounded, C can be treated as

a sequence C_+ of positive segments followed by a sequence C_- of negative segments. Obviously, since C is accepting, the total increments $\Delta(C_+)$ of the counter on C_+ equals the total decrements $\Delta(C_-)$ of the counter on C_-.

We use a *segment symbol* $+_{s,s',d,i}$ (resp. $-_{s,s',d,i}$) to abstract a positive (resp. negative) segment associated with state pair (s, s'), $d \in \{\text{LL,LR,RR,RL}\}$, and i-th block 1^{B_i}. According to Lemma 2 and Lemma 3, on a segment, the relationship between the absolute values of counter changes and the length of the block associated with the segment can be characterized by a Presburger formula (i.e., *the formula of the segment symbol*). Now, a two-phase program G_{+-} can be constructed such that each segment symbol $+_{s,s',d,i}$ corresponds to a transition in G_+ as follows (in below, T is the formula of the segment symbol):

- If $d = \text{LL}$, then the transition is $((s, i+1), i, T, (s', i+1))$.
- If $d = \text{LR}$, then the transition is $((s, i), i, T, (s', i+1))$.
- If $d = \text{RR}$, then the transition is $((s, i), i, T, (s', i))$.
- If $d = \text{RL}$, then the transition is $((s, i+1), i, T, (s', i+1))$.

Similarly, transitions in G_- can be constructed from symbols $-_{s,s',d,i}$. Let G_{+-}^{s,i_0} be a two-phase program consisting of G_+ and G_- such that

- the initial state of G_+ is $(s_0, i = 1)$ where s_0 is the initial state of M,
- the final state of G_-+ is (s, i_0),
- the initial state of G_- is (s, i_0),
- the final state of G_- is $(s_f, i = k+1)$ where s_f is the final state of M.

It is noticed that the final state of G_+ equals the initial state of G_-. It is observed that $(B_1, \ldots, B_k)$ is accepted by M iff there are some state s and some $1 \leq i_0 \leq k+1$ such that,

$(B_1, \ldots, B_k)$ is accepted by G_{+-}^{s,i_0}.

Since there are only finitely many choices of s and i_0, from Theorem 5, we obtain that the bounded language accepted by 2NCM(1,1) is effectively $\mathcal{D}$-definable.

Lemma 4. *The bounded language accepted by a 2NCM(1,1) is effectively $\mathcal{D}$-definable.*

Next, we show that the bounded language accepted by 2NCM(1,r) for any r is $\mathcal{D}$-definable. 2NCM(1,r), when $r > 1$, is more complex than 2NCM(1,1). However, we will show that we can effectively reduce the emptiness of 2NCM(1,r) M_r into the emptiness of the "intersection" of finitely many 2NCM(1,1)'s. We may assume w.l.o.g that, on a k-bounded input word

$$b_1 1^{B_1} \ldots b_k 1^{B_k} b_{k+1},$$

M_r makes a counter reversal only when it is reading one of the delimiters $b_1, \ldots, b_{k+1}$. (Otherwise, we may insert up to r many new delimiters $c_1, \ldots, c_r$ to an input word of M_r and construct a new 2NCM(1,r) M_r' working on the new $k+r$-bounded word. M_r' simulates M_r properly and makes sure that, whenever M_r makes the i-th reversal, M_r'

is reading the delimiter c_i. It is not difficult to show that, if the the bounded language accepted by M'_r is $\mathcal{D}$-definable, then so is the bounded language accepted by M_r.)

The r-reversal-bounded counter x behaves like this:

$$\nearrow, \searrow, \ldots, \nearrow, \searrow$$

(each $\nearrow$ stands for a nondecreasing phase; each $\searrow$ stands for a nonincreasing phase). Two consecutive phases of $\nearrow$ and $\searrow$ are called a *round*. Without loss of generality, we assume that r is odd and x makes exactly r reversals, so x has precisely $m = 1 + \frac{r-1}{2}$ rounds. We also assume that the machine starts with zero counter and accepts with zero counter. If $w = (B_1, \ldots, B_k)$ is an input to M_r, $PAD(w)$ is a string $(B_1, \ldots, B_k, E_1, \ldots, E_{m-1})$. That is, $PAD(w)$ is $(B_1, \ldots, B_k)$ padded with some $(m-1)$-bounded word $(E_1, \ldots, E_{m-1})$. Note that a given k-bounded word w has many $PAD(w)$'s.

A "trace" of the computation of M_r can be represented by a $2(m-1)$-tuple $\alpha = \langle d_1, q_1, d_2, q_2, ..., d_{m-1}, q_{m-1} \rangle$, where at the end of round $i = 1, ..., m-1$, M_r is at delimiter d_i (since M_r is about to reverse) in state q_i. Clearly, there are only a finite number of such α's. We will construct m 2NCM(1,1)'s $\hat{M}_1, ..., \hat{M}_m$ such that:

(*) a k-bounded word w is in $L(M_r)$ iff $\alpha PAD(w)$ is in $L(\hat{M}_1) \cap ... \cap L(\hat{M}_m)$ for some α and $PAD(w)$.

If w is an input to M_r, the input to each $\hat{M}_i$ is a string of the form $\alpha PAD(w)$. For $i = 2, ..., m-1$, $\hat{M}_i$ carries out the following two phases:

1. Restores the value of the counter to E_{i-1}, then moves it's input head to delimiter d_{i-1}, and then simulates M_r starting in state q_{i-1}. In the simulation, $\hat{M}_i$ ignores α and the paddings.
2. When M_r completes a round and starts to reverse (i.e., increments) the counter, $\hat{M}_i$ "remembers" the delimiter e_i and state s_i (when the reversal occurs), and goes to block E_i and verifies that the current value of the counter is E_i (note that if such is the case, the counter would be zero after checking). Then $\hat{M}_i$ moves its input head to the leftmost symbol and accepts if $d_i = e_i$ and $q_i = s_i$.

For $i = 1$, $\hat{M}_1$ does not need the restoration phase, but simulates M_r starting in state q_0 (the initial state of M_r). It also executes phase 2. For $i = m$, $\hat{M}_m$ executes the restoration phase only and accepts if M_r, after completing a round, accepts. Notice that, in the above construction, each E_i is used to denote the counter value of M_r at the end of each round. It is easy to verify that (*) above holds and each $\hat{M}_i$ is indeed a 2NCM(1,1). Hence, from Lemma 4 noticing that $\mathcal{D}$-definability is closed under intersection, union (over the α's) and $\exists$-quantification (for eliminating the padding E_i's), we have finally proved the main theorem of the paper that settles the open problem in [11, 12].

Theorem 7. *The bounded language accepted by 2NCM(1,r) is effectively $\mathcal{D}$-definable. Therefore, the emptiness problem for 2NCM(1,r) over bounded languages is decidable.*

6 Conclusion

We showed that the emptiness problem for two-way nondeterministic finite automata augmented with one reversal-bounded counter operating on bounded languages is de-

cidable, settling an open problem in [11, 12]. The proof was a rather involved reduction to the solution of a special class of Diophantine systems of degree 2 via a class of programs called two-phase programs. The result has applications to verification of infinite state systems.

For instance, consider a nondeterministic transition system M containing nonnegative integer parameterized constants $A, B_1, \ldots, B_k$ and a nonnegative integer counter x, which starts at 0. M's transition involves nondeterministically changing the state (from among a finite number of control states) and updating the counter by performing one of the instructions $x := x + B_i$, $1 \leq i \leq k$. M *terminates* if, M reaches some control state s with $x \leq A$, and any further execution of an updating instruction from s will make $x > A$. In practice, M can be used to model a buffer controller design that handles k types $b_1, \ldots, b_k$ of blocks. Every block of type b_i is with size B_i. The use of parameterized constants is common at the design stage; the constants are concretized in specific implementations. An instruction of the form $x := x + B_i$ on the edge from s to s' means that a block of type b_i is put into the buffer. Notice that, the graph of M makes the controller select blocks of various types according to some regular ordering. x and A in the controller represent the "used" and maximal capacity of the buffer, respectively. Hence, M terminates at the moment when the buffer does not overflow ($x < A$) and putting any additional block according to the ordering into the buffer will make the buffer overflow. Consider the following "efficiency problem" for M: for any $A, B_1, \ldots, B_k$ that satisfy a Presburger formula $P(A, B_1, \ldots, B_k)$ (e.g., a design constraint like $A > B_1 > \ldots > B_k$), when M terminates, the unused buffer size is less than each B_i (i.e., the buffer is maximally used). From the main result in this paper, the efficiency problem is decidable. To see this, we formulate the negation of the problem as follows: Are there values for $A, B_1, \ldots, B_k$ satisfying P such that there is a value x and M terminates with $A - x > B_i$ for some i? Let L be the bounded language representing the nonnegative integer tuples of $(A, B_1, \ldots, B_k)$ that satisfy the negation. It is not hard to construct a two-way nondeterministic finite automaton augmented with one reversal-bounded counter to accept bounded language L. We leave the details to the reader.

Thanks go to WSU PhD student Gaoyan Xie for discussions on the above example.

References

1. A. Brauer. On a problem of partitions. *Amer. J. Math.*, 64:299–312, 1942.
2. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop of Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer, 1981.
3. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
4. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1998.
5. H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 1999.

6. Zhe Dang, O. H. Ibarra, and R. A. Kemmerer. Decidable Approximations on Generalized and Parameterized Discrete Timed Automata. In *Proceedings of the 7th Annual International Computing and Combinatorics Conference (COCOON'01)*, volume 2108 of *Lecture Notes in Computer Science*, pages 529–539. Springer, 2001.
7. A. Finkel and G. Sutre. Decidability of reachability problems for classes of two counters automata. In *Proc. 17th Ann. Symp. Theoretical Aspects of Computer Science (STACS'2000), Lille, France, Feb. 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 346–357. Springer, 2000.
8. S. Ginsburg and E. Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. of Mathematics*, 16:285–296, 1966.
9. E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22:220–229, 1981.
10. E. M. Gurari and O. H. Ibarra. Two-way counter machines and Diophantine equations. *Journal of the ACM*, 29(3):863–873, 1982.
11. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.
12. O. H. Ibarra, T. Jiang, N. Tran, and H. Wang. New decidability results concerning two-way counter machines. *SIAM J. Comput.*, 24:123–137, 1995.
13. O. H. Ibarra, J. Su, Zhe Dang, T. Bultan, and R. A. Kemmerer. Counter machines: decidable properties and applications to verification problems. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS 2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 426–435. Springer-Verlag, 2000.
14. R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12:161–177, 1992.
15. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.
16. L. Lipshitz. The Diophantine problem for addition and divisibility. *Transactions of AMS*, 235:271–283, 1978.
17. K. Mahler. On the Chinese remainder theorem. *Math. Nachr.*, 18:120–122, 1958.
18. Y. V. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.
19. M. Minsky. Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. *Ann. of Math.*, 74:437–455, 1961.
20. J. L. Ramirez-Alfonsin. Complexity of the Frobenius problem. *Combinatorica*, 16:143–147, 1996.
21. A. P. Sistla and E. M. Clarke. Complexity of propositional temporal logics. *Journal of ACM*, 32(3):733–749, 1983.
22. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings 1st Annual IEEE Symp. on Logic in Computer Science, LICS'86, Cambridge, MA, USA, 16–18 June 1986*, pages 332–344, Washington, DC, 1986. IEEE Computer Society Press.