On Two-Way Nondeterministic Finite Automata with One Reversal-Bounded Counter [1]

Zhe Dang

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

Oscar H. Ibarra [2]

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

Zhi-Wei Sun

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 $w_1^* \dots w_k^*$ for some nonnull words $w_1, \dots, w_k$) is decidable, resolving a problem left open in [4,7]. 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.

Key words: Counter machine, reversal-boundedness, emptiness problem, bounded language

[1] A preliminry version of this paper was presented at the *13th Annual International Symposium on Algorithms and Computation*, Vancouver, Canada.

[2] Corresponding author: ibarra@cs.ucsb.edu, Tel. +1-805-893-4171. The research of Oscar H. Ibarra was supported in part by NSF Grants IIS-0101134 and CCR02-08595.

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 could 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 [12] 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 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 [5]. 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) [5]. 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 $w_1^* \dots w_k^*$ for some fixed nonnull words $w_1, \dots, w_k$) [4]. This result was later shown to hold for the general case when the input is not over a bounded language [7], by a reduction to the bounded case. 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 [4,7] 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. The proof is an involved reduction to the solution of a special class of systems of quadratic Diophantine equations, which we show decidable. The systems are more general than the ones used in [4], where the emptiness problem for the deterministic case was reduced to a simpler system (that was shown decidable in [9].) Moreover, unlike the proof for the deterministic case [4], where the reduction from the machine

to the system of equations was carried out in one step, in our proof for the nondeterministic case, we use an intermediate machine model, called two-phase programs. 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 [5].

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. Section 6 compares the computing capabilities of nondeterministic and deterministic two-way one-counter machines with one reversal-bounded counter. Section 7 is a brief conclusion.

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 symbols $a_1, a_2, \ldots, a_k$ 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. For convenience we will simple use the term bounded language. There are other equivalent definitions of "boundedness" that we will use in the paper. We will need the following results.

Theorem 2.1 *The emptiness problem is decidable for the following classes: (a) 2DCM(1) [7], (b) 2NCM(c) over a unary alphabet (i.e., over a bounded language on one symbol) [7], and (c) finite-crossing 2NCM(c) for every c [3,5].*

Next, we recall the definitions of (semi)linear sets and their connection to counter machines. Let $\mathbb{N}$ be the set of nonnegative integers and k be a positive integer. A subset S of $\mathbb{N}^k$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in $\mathbb{N}^k$ such that $S = \{v \mid v = v_0 + b_1 v_1 + \cdots + b_t v_t,\ b_i \in \mathbb{N}\}$. The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set S. The set $S \subseteq \mathbb{N}^k$ is *semilinear* if it is a finite union of linear sets. Clearly, semilinear sets are closed under union. Let $\Sigma = \{a_1, a_2, \ldots, a_k\}$ be an alphabet. For each word w in Σ^*, define the *Parikh map* of w to be $\psi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_k})$, where $|w|_{a_i}$ denotes the number of symbol a_i's in word w, $1 \leqslant i \leqslant k$. For a language $L \subseteq \Sigma^*$, the *Parikh map* of L is $\psi(L) = \{\psi(w) \mid w \in L\}$. The language L is *semilinear* if $\psi(L)$ is a semilinear set.

Obviously, if the languages accepted by machines in a class C are effectively semilinear (that is, for each M in C, the semilinear set $S = \psi(L)$ can be effectively constructed), then the emptiness problem for C is decidable. We will need the following theorem from [5]:

Theorem 2.2 *Let M be a finite-crossing 2NCM(c). Then $\psi(L(M))$ is a semilinear set effectively computable from M.*

Note that the result above is not true for machines that are not finite-crossing. For example, a 2DCM(1,1) can recognize the language $\{0^i 1^j \mid i \text{ divides } j\}$, which is not semilinear.

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 $0 \leqslant c < d$. 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 $(j_1, \cdots, j_n)$ in $\mathbb{N}^n$. Q is *Presburger-definable* if there is a Presburger formula $p(x_1, \cdots, x_n)$ 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 [2]. 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 $a_1 B + a_2 x < b$ with $a_2 \in \{-1, 0, 1\}$. We say that 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 $a, b \in \mathbb{N}$. T is a *line* if T is $x = aB + b$, or T is $B = b$ (called a vertical line), for some $a, b \in \mathbb{N}$. T is a *sector* if T is $x \geqslant aB + b$, or T is $aB + b \leqslant x \leqslant a'B + b'$, for some $a < a', b \leqslant b' \in \mathbb{N}$. Observe that if T is mod-free and unitary, then for each sufficiently large d, $T(d + B, x)$ can be written into a (finite) disjunction of points, lines, and sectors.

An *atomic $\mathcal{D}$-formula* over nonnegative integer variables $x_1, \cdots, x_n$ is either $f(x_1, \cdots, x_n) =$

0 or a divisibility $f(x_1, \cdots, x_n) | g(x_1, \cdots, x_n)$, 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 $(j_1, \cdots, j_n)$ in $\mathbb{N}^n$ is *$\mathcal{D}$-definable* if there is a $\mathcal{D}$-formula $p(x_1, \cdots, x_n)$ such that the set of nonnegative integer solutions of p is exactly Q. As stated in the Lipshitz's Theorem [9], the satisfiability of $\mathcal{D}$-formulas is decidable. We will also need some basic results in number theory.

Theorem 2.3 *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) $\gcd(m_1, m_2)|r_1 - r_2$.*

Theorem 2.3 follows from the following result in [10]: for any positive integers m_1, m_2 and integers r_1, r_2, $\gcd(m_1, m_2) \mid r_1 - r_2$ if and only if there exists a nonnegative integer n such that $n \equiv_{m_1} r_1$ and $n \equiv_{m_2} r_2$. We will also need the following well-known theorem of Frobenius.

Theorem 2.4 *Let $a_1, \cdots, a_n$ be positive integers. Then there exists a positive integer b_0 such that, for each integer $b \geqslant b_0$ with $\gcd(a_1, \cdots, a_n)|b$, the linear equation $a_1x_1 + \cdots + a_nx_n = b$ has nonnegative integer solutions.*

Given n positive integers $a_1, \cdots, a_n$ with $\gcd(a_1, \cdots, a_n) = 1$, the Frobenius problem is to determine the greatest integer $g(a_1, \cdots, a_n)$ not in the set $\{a_1x_1 + \cdots + a_nx_n : x_1, \cdots, x_n \in \mathbb{N}\}$. It is well known that $g(a_1, a_2) = a_1a_2 - a_1 - a_2$. In 1942, A. Brauer [1] proved that $g(a_1, \cdots, a_n) \leqslant \sum_{k=1}^{n} a_k \left(\frac{d_{k-1}}{d_k} - 1\right)$ where $d_0 = 0$ and $d_k = \gcd(a_1, \cdots, a_k)$ for $k = 1, \cdots, n$. In 1992, R. Kannan [8] showed that for any fixed n there is an algorithm in polynomial time to compute $g(a_1, \cdots, a_n)$. In 1996, J. L. Ramirez-Alfonsin [13] proved that the Frobenius problem is NP-hard.

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 [11]. In this section, we exhibit a nontrivial decidable class of Diophantine systems of degree 2. We will use this result later.

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

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

and

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

Let Θ be a predicate on nonnegative integer k-tuples satisfying, for all nonnegative integers $B_1, \cdots, B_k$, $\Theta(B_1, \cdots, B_k)$ is true iff the conjunction of (1) and (2) has a nonnegative integer solution for $u, s_1, \cdots, s_m, t_1, \cdots, t_n$. 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 2.4 and Theorem 2.3.

Lemma 3.1 *The predicate $\Theta(B_1, \cdots, B_k)$ defined above is effectively $\mathcal{D}$-definable.*

Proof. We will construct the $\mathcal{D}$-formula that defines $\Theta(B_1, \cdots, B_k)$ as required. Inequalities (1) and (2) can be rewritten into the following equations by introducing new nonnegative integer variables x, y, z, w:

$$\sum_{1 \leqslant i \leqslant m} U_i s_i = u - x - U, \tag{3}$$

$$\sum_{1 \leqslant i \leqslant m} \Phi_i s_i = y + x - \Phi, \tag{4}$$

$$\sum_{1 \leqslant j \leqslant n} V_j t_j = u - z - V, \tag{5}$$

$$\sum_{1 \leqslant j \leqslant n} \Gamma_j t_j = w + z - \Gamma. \tag{6}$$

Observe that, for all nonnegative integers $B_1, \cdots, B_k$, $\Theta(B_1, \cdots, B_k)$ iff the equation system of (3,4,5,6) has a nonnegative integer solution for $u, s_1, \cdots, s_m, t_1, \cdots, t_n, x, y, z, w$. Let us fix a 4-tuple η of subsets $(\eta_1, \eta_2, \eta_3, \eta_4)$ where $\eta_1, \eta_2 \subseteq \{1, \cdots, m\}, \eta_3, \eta_4 \subseteq \{1, \cdots, n\}$. η_1 derives a *zero-condition* on $B_1, \cdots, B_k$ that makes each coefficient U_i in equation (3) with $i \in \eta_1$ not zero (all the others are zero). That is, the zero-condition of η_1 is

$$\bigwedge_{i \in \eta_1} U_i(B_1, \cdots, B_k) > 0 \wedge \bigwedge_{i \notin \eta_1} U_i(B_1, \cdots, B_k) = 0.$$

Similarly, we may define the zero-conditions of η_2, η_3, η_4 but for the other three equations (i.e., (4), (5), (6), respectively). The zero-condition $\mathcal{Z}_\eta$ of η is the conjunction of the four zero-conditions derived from $\eta_1, \eta_2, \eta_3, \eta_4$. Notice that $\mathcal{Z}_\eta$ is Presburger, hence $\mathcal{D}$-definable.

We use Θ_η to denote $\Theta \wedge \mathcal{Z}_\eta$. Clearly, since there are only finitely many choices for η and $\Theta = \vee_\eta \Theta_\eta$, it suffices for us to show that Θ_η is $\mathcal{D}$-definable. We have the following two cases to consider.

Case 1. None of $\eta_1, \eta_2, \eta_3, \eta_4$ is empty. We use $\gcd(U_i : i \in \eta_1)$ to denote the gcd of all the U_i's in (3) with $i \in \eta_1$ (all the other U_i's are zero, under the condition of $\mathcal{Z}_\eta$). Similarly, we may define $\gcd(\Phi_i : i \in \eta_2)$, $\gcd(V_j : j \in \eta_3)$, $\gcd(\Gamma_j : j \in \eta_4)$, for (4), (5), (6), respectively. Consider the following divisibilities:

$$\gcd(U_i : i \in \eta_1) | u - x - U, \tag{7}$$

$$\gcd(\Phi_i : i \in \eta_2) | y + x - \Phi, \tag{8}$$

$$\gcd(V_j : j \in \eta_3) | u - z - V, \tag{9}$$

$$\gcd(\Gamma_j : j \in \eta_4) | w + z - \Gamma. \tag{10}$$

We claim that,

Claim 1. For all nonnegative integers $B_1, \cdots, B_k$ satisfying $\mathcal{Z}_\eta$, $\Theta(B_1, \cdots, B_k)$ iff there is a nonnegative integer solution of u, x, y, z, w for the system of (7,8,9,10).

The only-if part is obvious. To show the if-part, let $B_1, \cdots, B_k$ be any given nonnegative integers satisfying $\mathcal{Z}_\eta$. Suppose that u^0, x^0, y^0, z^0, w^0 constitutes a nonnegative integer solution to (7,8,9,10) for the given $B_1, \cdots, B_k$. We are going to argue that the system of (3,4,5,6), for the given values of $B_1, \cdots, B_k$, has a nonnegative integer solution for $u, x, y, z, w, s_1, \cdots, s_n$, $t_1, \cdots, t_m$. Consider

$$b_1 = \gcd(U_i : i \in \eta_1) \gcd(V_j : j \in \eta_3) h + u^0 - x^0 - U$$

and

$$b_2 = \gcd(U_i : i \in \eta_1) \gcd(V_j : j \in \eta_3) h + u^0 - z^0 - V.$$

Observe that $\gcd(U_i : i \in \eta_1) | b_1$ and $\gcd(V_j : j \in \eta_3) | b_2$ for any value of h. From Theorem 2.4, there is a $h_0 > 0$ such that, for every $h \geqslant h_0$,

$$\sum_{1 \leqslant i \leqslant m} U_i s_i = b_1 \tag{11}$$

and

$$\sum_{1 \leqslant j \leqslant n} V_j t_j = b_2 \tag{12}$$

has a nonnegative solution for $s_1, \cdots, s_m, t_1, \cdots, t_n$. Now, let h be h_0 and therefore, b_1 and b_2 be also fixed. We use $s_1^0, \cdots, s_m^0, t_1^0, \cdots, t_n^0$ to denote a solution to (11) and (12). Let $d = \max(|y^0 + x^0 - \Phi|, |w^0 + z^0 - \Gamma|)$, $d_1 = \sum_{1 \leqslant i \leqslant m} U_i d$, $d_2 = \sum_{1 \leqslant j \leqslant n} V_j d$. It is left to the reader to verify that the following forms a solution to the system of (3,4,5,6):

$s_i = s_i^0 + dd_2, 1 \leqslant i \leqslant m$,
$t_j = t_j^0 + dd_1, 1 \leqslant j \leqslant n$,
$u = \gcd(U_i : i \in \eta_1)\ \gcd(V_j : j \in \eta_3)h_0 + d_1d_2 + u^0$,
$x = x^0$,
$y = \Phi - x^0 + \sum_{1 \leqslant i \leqslant m} \Phi_i \cdot (s_i^0 + dd_2)$, (notice that $y \geqslant 0$ by the choice of d)
$z = z^0$,
$w = \Gamma - z^0 + \sum_{1 \leqslant j \leqslant n} \Gamma_j \cdot (t_j^0 + dd_1)$, (notice that $w \geqslant 0$ by the choice of d).

This completes the proof of Claim 1.

Next, we argue that the system of (7,8,9,10) is $\mathcal{D}$-definable. We only look at (7): $\gcd(U_i : i \in \eta_1)|u - x - U$, the other formulas of (8),(9) and (10) are similar. Pick an element $i_0 \in \eta_1$. If i_0 is the only element in η_1, then we are done since (7) is now $U_{i_0}|u - x - U$ which is $\mathcal{D}$-definable. If otherwise, we write $\gcd(U_i : i \in \eta_1)$ as $\gcd(U_{i_0}, \gcd(U_i : i \in \eta_1 - \{i_0\}))$. A technique used in [9] can be applied as follows. By introducing a new nonnegative integer variable μ_0 and using Theorem 2.3, (7) can be transformed into $U_{i_0}|\mu_0 - u$ and $\gcd(U_i : i \in \eta_1 - \{i_0\})|\mu_0 - x - U$. This process can be continued until all the elements $i_0, \cdots, i_p$ in η_1 are enumerated. Eventually, (7) can be written into the following conjunction

$$U_{i_0}|\mu_0 - u \wedge U_{i_p}|\mu_{p-1} - x - U \bigwedge_{1 \leqslant j < p} U_{i_j}|\mu_j - \mu_{j-1} \tag{13}$$

which is $\mathcal{D}$-definable. Notice that, in above, we introduce new variables $\mu_0, \cdots, \mu_{p-1}$. Hence, the system of (7,8,9,10) can be transformed into a $\mathcal{D}$-definable formula Ψ over u, x, y, z, w as well as a number of newly introduced variables. Using Theorem 2.3, it can be concluded that

> Claim 2. for all nonnegative integers $B_1, \cdots, B_k$ that satisfy $\mathcal{Z}_\eta$, $\Theta(B_1, \cdots, B_k)$ iff there is a nonnegative integer solution of u, x, y, z, w and the newly introduced variables to Ψ.

Hence, Θ_η is $\mathcal{D}$-definable.

Case 2. At least one of $\eta_1, \eta_2, \eta_3, \eta_4$ is empty. If $\eta_1 = \emptyset$, then we replace (7) with $u - x - U = 0$ (since the left-hand side of (3) is now 0). Similarly, if $\eta_2 = \emptyset$ (resp. η_3, η_4), then we replace (8) (resp. (9), (10)) with $y + x - \Phi = 0$ (resp. $u - z - V = 0, w + z - \Gamma = 0$). Notice that the replacing formula (e.g., $u - x - U = 0$ for (7)) is simply a linear constraint over $B_1, \cdots, B_k$ and thus is $\mathcal{D}$-definable. After the replacement, Claim 1 in Case 1 is still true, by using a similar (and easier) proof. Completely analogous to the usage of Theorem 2.3 in showing Claim 2 of Case 1, we may conclude that Θ_η is $\mathcal{D}$-definable.

This completes the proof of Lemma 3.1. ∎

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 do not change values during computation; i.e., they are parameterized constants. Each increment made on the counter satisfies some Presburger constraint in two variables; e.g., $B_1 \leqslant \delta \leqslant 2B_1$ 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 $B_1, \cdots, B_k$. We are interested in the following question: is there an assignment for $B_1, \cdots, B_k$ 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 $\langle S, B_1, \cdots, B_k, x, \mathcal{T}, E\rangle$ where

- S is a finite set of control states, with two special states designated as the initial state and the final state.
- $B_1, \cdots, B_k$ are k input (nonnegative integer) variables,
- x is the nonnegative integer counter which is always nondecreasing,
- $\mathcal{T}$ is a finite set of Presburger formulas on two nonnegative integer variables,
- $E \subseteq S \times \{1, \cdots, k\} \times \mathcal{T} \times S$ 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 $(s, v_1, \cdots, v_k, u)$ in $S \times \mathbb{N}^k \times \mathbb{N}$ is a tuple of a control state s, values $v_1, \cdots, v_k$ for the input variables $B_1, \cdots, B_k$, and value u for the counter x. We use $(s, v_1, \cdots, v_k, u) \rightarrow^G (s', v'_1, \cdots, v'_k, u')$ to denote 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., $(v_1, \cdots, v_k) = (v'_1, \cdots, v'_k)$,
- The evolution pair (i, T) is satisfied; i.e., $T(u, u' - u)$ is true (hence, $u \leqslant u'$ since T is defined on nonnegative integers).

A *path* is a sequence $(s_0, v_1, \cdots, v_k, u_0) \cdots (s_i, v_1, \cdots, v_k, u_i) \cdots (s_m, v_1, \cdots, v_k, u_m)$, for some $m \geqslant 1$, such that $(s_i, v_1, \cdots, v_k, u_i) \rightarrow^G (s_{i+1}, v_1, \cdots, v_k, u_{i+1})$ for each $0 \leqslant i \leqslant m-1$. 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* $(u, \cdots, v_k, u_m)$.

A *two-phase program* G_{+-} consists of two simple programs G_+ and G_- that share the same S, input variables $B_1, \cdots, B_k$ and $\mathcal{T}$. 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 $v_1, \cdots, v_k$ is *accepted by the two-phase program* G_{+-} if there is a counter values u such that $(v_1, \cdots, v_k, u)$ 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 $T \in \mathcal{T}$ is a point, a line, or a sector. Then, the result follows by making use of Lemma 3.1.

Theorem 4.1 *The tuple language accepted by a two-phase program is $\mathcal{D}$-definable.*

Proof. The theorem states that the tuple language accepted by a two-phase program G_{+-} is $\mathcal{D}$-definable; i.e.,

$$L(G_{+-}) \text{ is } \mathcal{D}\text{-definable.} \tag{14}$$

The following four arguments will establish that we need only consider a special class of two-phase programs in showing (14).

(Argument 1) It suffices to show statement (14) by assuming that G_{+-} is 2-mod-free (i.e., each $T \in \mathcal{T}$ is 2-mod-free). If G_{+-} is not 2-mod-free, let D be the multiplier of all the constants d such that a mod-constraint in the form of $x \equiv_d j$ (with $0 \leqslant j < d$) appears in some $T \in \mathcal{T}$. Now, we build a new two-phase program G'_{+-}, consisting of a new G'_+ (with counter $\hat{x_+}$) and a new G'_- (with counter $\hat{x_-}$), which is 2-mod-free. The state set of G'_{+-} is $S' = \{(s, j) : 0 \leqslant j < D\}$. Edges in G'_{+-} are constructed from edges in G_{+-} as follows. For each $0 \leqslant j, j' < D$, we create an edge $((s, j), i, T_{jj'}, (s', j'))$ in G'_{+-} from an edge (s, i, T, s') in G_{+-}, where $T_{jj'}(B, \hat{x})$, a Presburger formula over two nonnegative integer variables B and $\hat{x}$, is defined as $T(B, D \cdot \hat{x} + j' - j)$. Notice that, when $j > j'$ (resp. $j < j'$), the requirement of $D \cdot \hat{x} + j' - j \geqslant 0$ in T naturally makes $\hat{x} > 0$ ($\hat{x} \geqslant 0$). It is also noticed that, counter x increments from u to u' on edge (s, i, T, s') iff counter $\hat{x}$ increments

from $\hat{u}$ to $\hat{u'}$ on edge $((s,j),i,T_{jj'},(s',j'))$, where $u = D \cdot \hat{u} + j$ and $u' = D \cdot \hat{u'} + j'$. The new G'_+ (resp. G'_-) can be obtained by replacing each edge in G_+ (resp. G'_-) with the D^2 new edges constructed above. Notice that both G'_+ and G'_- are 2-mod-free, since each $T_{jj'}$ is 2-mod-free. Obviously, a tuple of $v_1, \cdots, v_k$ is in $L(G_{+-})$ iff there is $\hat{u}$ such that, for some $0 \leqslant j < D$,

- G'_+ (resp. G'_-) starts from state $(s,0)$ and ends with state (s',j), where s and s' are the initial and the final states of G_+ (resp. G_-), and
- $(v_1, \cdots, v_k, \hat{u})$ is accepted by both G'_+ and G'_-.

The argument follows by noticing that there are only finitely many choices of j.

(Argument 2) It suffices to show statement (14) by assuming that G_{+-} is unitary (i.e., each $T \in \mathcal{T}$ is unitary) and 2-mod-free. According to (Argument 1), we suppose that G_{+-} is 2-mod-free. If G_{+-} is not unitary, let D be the absolute value of the multiplier of all the non-zero coefficients of x in atomic linear relations appearing in all $T \in \mathcal{T}$ (recall T is a Presburger formula over two variables B and x). Choose any $0 \leqslant j_1, \cdots, j_k < D$. For each $1 \leqslant i \leqslant k$, let $\hat{B}_i$ be a nonnegative integer variable. By assuming $B_i = D \cdot \hat{B}_i + j_i$, $T(B_i, x)$ is transformed into $T_i(\hat{B}_i, x) = T(D \cdot \hat{B}_i + j_i, x)$, for each $1 \leqslant i \leqslant k$. Notice that T_i is unitary and 2-mod-free, since a linear constraint $a_1 B_i + a_2 x < b$ (w.l.o.g., $a_2 > 0$) in $T(B_i, x)$ is now $a_1 D \hat{B}_i + a_2 x < b - a_1 j_i$, which is equivalent to one of the following two unitary constraints: $(\frac{a_1 D}{a_2})\hat{B}_i + x < \frac{b - a_1 j_i}{a_2}$ when $a_2 | (b - a_1 j_i)$, $(\frac{a_1 D}{a_2})\hat{B}_i + x < \lceil \frac{b - a_1 j_i}{a_2} \rceil$ when otherwise. Define $\mathcal{T}' = \{T_i : T \in \mathcal{T}, 1 \leqslant i \leqslant k\}$. The argument follows from the fact that there are only finitely many choices for $j_1, \cdots, j_k$ and for each such choice, G_{+-} (with input variables $B_1, \cdots, B_k$ and $\mathcal{T}$) corresponds to some 2-mod-free and unitary two-phase program (on input variables $\hat{B}_1, \cdots, \hat{B}_k$ and $\mathcal{T}'$).

(Argument 3). It suffices to show statement (14) by assuming that G_{+-} is mod-free (i.e., each $T \in \mathcal{T}$ is 1-mod-free and 2-mod-free) and unitary. According to (Argument 2), we assume that G_{+-} is 2-mod-free and unitary. If G_{+-} is not 1-mod-free, let D be the multiplier of all the constants d such that a mod-constraint in the form of $B \equiv_d j$ (with $0 \leqslant j < d$) appears in some $T(B, x)$ in $\mathcal{T}$. Choose any $0 \leqslant j_1, \cdots, j_k < D$. By making $B_i = D \cdot \hat{B}_i + j_i$, $T(B_i, x)$ is transformed into $T_i(\hat{B}_i, x) = T(D \cdot \hat{B}_i + j_i, x)$, for each $1 \leqslant i \leqslant k$. Notice that T_i is mod-free and unitary, since each mod-constraint (on B_i) in T now is either true or false. Similar to (Argument 2), we can establish this argument.

(Argument 4). It suffices to show statement (14) by assuming that G_{+-} is single (i.e., each $T \in \mathcal{T}$ is a point, a line, or a sector). According to (Argument 3), we suppose that G_{+-} is unitary and mod-free. Hence, we may find a large number d such that, for each $T \in \mathcal{T}$, $T(d + B, x)$ is a disjunction $T_1(B, x) \vee \cdots \vee T_h(B, x)$ (for some h) of points, lines, and sectors. We assume that each $B_i \geqslant d$. [3] A transform of $B_i = \hat{B}_i + d$ will bring G_{+-} to the

[3] If, for some i, $B_i < d$, we may explicitly assuming B_i to be a concrete value less than d, and an induction procedure can be used upon G_{+-} with a smaller k (since B_i is gone). As for the base of the induction, observe that, for $0 \leqslant B_1, \cdots, B_k \leqslant d$, the emptiness problem for the two-phase program is decidable, since, in this case, each B_i is bounded and the two-phase program can be

form that is desired (by replacing each edge (s, i, T, s') with edges (s, i, T_j, s'), $1 \leqslant j \leqslant h$).

(Argument 5). It suffices to show statement (14) by assuming that G_{+-} is single and each $T \in \mathcal{T}$ is not a point nor a vertical line. According to (Argument 4), we suppose that each $T \in \mathcal{T}$ is a point, a line, or a sector. Argument 5 can be established by assuming each $B_i \geqslant d$ for some large d. d can be chosen such that it is greater than the B-coordinate of each point and each vertical line in $\mathcal{T}$. A transform of $B_i = \hat{B}_i + d$ will bring G_{+-} to the form that is desired. (When, for some i, $B_i < d$, we use the footnote in Argument 4 discussed in above.)

From the above arguments, we assume that each $T \in \mathcal{T}$ is in the form of a (non-vertical) line $x = aB + b$ ($a \geqslant 0, b \geqslant 0$), a sector $aB + b \leqslant x$ ($a \geqslant 0, b \geqslant 0$), or a sector $aB+b \leqslant x \leqslant a'B+b'$ ($a < a', b \leqslant b'$). Notice that, $T(B, x)$ is always satisfiable for any fixed B. Now, we construct a counter machine M_+ with counters $K_1^+, \cdots, K_k^+, \Delta_1^+, \cdots, \Delta_k^+, \tau^+$ (all counters start from 0) that simulates G_+ as follows. Whenever G_+ executes an edge $\langle s, i, T, s' \rangle$, M_+ moves from state s to state s' and does the following to the counters:

- If T is a line in the form of $x = aB + b$ ($a \geqslant 0, b \geqslant 0$), then in M_+,
  $K_i^+ := K_i^+ + a$;
  $\tau^+ := \tau^+ + b$;
  and all the other counters in M_+ do not change.
- If T is a sector $aB + b \leqslant x \leqslant a'B + b'$ with $0 \leqslant a < a', 0 \leqslant b \leqslant b'$, then in M_+,
  $K_i^+ := K_i^+ + a$;
  $\Delta_i^+ := \Delta_i^+ + (a' - a)$,
  $\tau^+ := \tau^+ + c$, for some c that is nondeterministically chosen with $b \leqslant c \leqslant b'$;
  and all the other counters in M_+ do not change.
- If T is a sector $aB + b \leqslant x$ with $a \geqslant 0, b \geqslant 0$, then in M_+,
  $K_i^+ := K_i^+ + a$;
  $\tau^+ := \tau^+ + c$, for some c that is nondeterministically chosen with $c \geqslant b$;
  and all the other counters in M_+ do not change.

Notice that, all the counters in M_+ are nondecreasing. The relationship between G_+ and M_+ can be easily seen as follows. $(B_1, \cdots, B_k, u)$ is accepted by G_+ iff $\Sigma_{1 \leqslant i \leqslant k} K_i^+ \cdot B_i + \tau^+ \leqslant u \leqslant \Sigma_{1 \leqslant i \leqslant k}(K_i^+ + \Delta_i^+) \cdot B_i + \tau^+$ for some $(K_1^+, \cdots, K_k^+, \Delta_1^+, \cdots, \Delta_k^+, \tau^+)$ that is reachable in M_+ at the final state. We use a predicate

$$P(K_1^+, \cdots, K_k^+, \Delta_1^+, \cdots, \Delta_k^+, \tau^+) \tag{15}$$

to denote that the counter values $K_1^+, \cdots, K_k^+, \Delta_1^+, \cdots, \Delta_k^+, \tau^+$ are reachable in M_+ at the final state. Since the counters are nondecreasing, P is Presburger (Theorem 2.2). Similarly, we may construct a counter machine M_- from G_- and obtain another Presburger formula

$$Q(K_1^-, \cdots, K_k^-, \Delta_1^-, \cdots, \Delta_k^-, \tau^-). \tag{16}$$

reduced to a 1-reversal counter machine, whose emptiness is decidable (Theorem 2.1). Therefore, for each $0 \leqslant B_1, \cdots, B_k \leqslant d$, the truth value of $(B_1, \cdots, B_k) \in L(G_{+-})$ is computable.

Hence, $(B_1, \cdots, B_k)$ is in $L(G_{+-})$ iff the following statement, called $\Xi(B_1, \cdots, B_k)$, is true for $B_1, \cdots, B_k$:

There is u such that
$$\sum_{1 \leqslant i \leqslant k} K_i^+ \cdot B_i + \tau^+ \leqslant u \leqslant \sum_{1 \leqslant i \leqslant k} (K_i^+ + \Delta_i^+) \cdot B_i + \tau^+ \quad (17)$$
and
$$\sum_{1 \leqslant i \leqslant k} K_i^- \cdot B_i + \tau^- \leqslant u \leqslant \sum_{1 \leqslant i \leqslant k} (K_i^- + \Delta_i^-) \cdot B_i + \tau^- \quad (18)$$
hold for some nonnegative integers $K_1^+, \cdots, K_k^+, \Delta_1^+, \cdots, \Delta_k^+, \tau^+, K_1^-, \cdots, K_k^-$, $\Delta_1^-, \cdots, \Delta_k^-, \tau^-$ satisfying (15) and (16).

Therefore, in order to show that the tuple language $L(G_{+-})$ is $\mathcal{D}$-definable, we need only to prove so for $\Xi(B_1, \cdots, B_k)$. The two Presburger formulas in (15) and (16) define two semilinear sets P and Q. Hence, each of P and Q can be written into a finite union of linear sets. It suffices for us to argue that $\Xi(B_1, \cdots, B_k)$ is $\mathcal{D}$-definable assuming that P and Q are two linear sets, which are generated by nonnegative integer variables $s_1, \cdots, s_m$ and $t_1, \cdots, t_n$, respectively. That is, each of $K_1^+, \cdots, K_k^+, \Delta_1^+, \cdots, \Delta_k^+, \tau^+$ can be written into a positive linear polynomial (i.e., the generator) over $s_1, \cdots, s_m$; each of $K_1^-, \cdots, K_k^-$, $\Delta_1^-, \cdots, \Delta_k^-, \tau^-$ can be written into a positive linear polynomial over $t_1, \cdots, t_n$. Substituting these generators in (17) and (18) and re-organizing the terms into the form of (1) and (2). From Lemma 3.1, $\Xi(B_1, \cdots, B_k)$ is $\mathcal{D}$-definable. Thus, $L(G_{+-})$ is $\mathcal{D}$-definable. ∎

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

Theorem 4.2 *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
$$\text{¢}\underbrace{a \cdots a}_{B}\$$$
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 $B \geqslant 1$) 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 5.1 *$T_{LL}(B,x)$, $T_{LR}(B,x), T_{RR}(B,x)$, and $T_{RL}(B,x)$ are Presburger formulas for any M specified above.*

Proof. We only prove the lemma for $T_{LR}(B,x)$. The other cases are similar. Let S be the number of control states in M_{LR}. Let J be some fixed positive integer (the value will be made clear in a moment). Clearly, for each $B \leqslant J$, $T_{LR}(B,x)$ is Presburger. This is because, when the input size bounded by J is stored in the finite control, a finite automaton can be used to accept all the unary encodings $a^B 1^x$ of (B,x) with $T_{LR}(B,x)$. The encodings form a regular language which is semilinear. Consider an input of size $B > J$ and a number x with $T_{LR}(B,x)$. Let e be an accepting execution of M_{LR} that witnesses the fact of $T_{LR}(B,x)$. Now, we fix any state s. During e, the input head may be at the same state s and the same input cell c (an input cell is one containing input symbol a) twice – obviously, during which no endmarkers is read. We call this a *loop* at s. The *left-radius* (resp. *right-radius*) of the loop is the distance between c and the leftmost (resp. rightmost) input cell scanned during the loop. The *length* δ of the loop is the number of increments made on the loop. Since the input has endmarkers, the left-radius (resp. right-radius) cannot exceed the distance between cell c and the left endmarker (resp. right endmarker).

Now, we modify M_{LR} into M' by assuming that the input does not have endmarkers; i.e., the input is infinite (in both directions). Suppose M' starts from state s and cell c. What is the set Δ of the lengths of all the possible loops at (s,c)? When each length in Δ is represented as a unary string, Δ can be accepted by a pushdown automaton. The automaton, starting from state s, simulates M': it pushes (resp. pops) a symbol whenever M' moves to the right (resp. left). The automaton reads its own input (the unary encoding of a length) whenever M' makes a counter increment. The automaton accepts the length if the stack is empty (i.e., M' returns to cell c) with state s and the automaton is at the end of its own input. Hence, Δ, being context-free, is a semilinear set. This gives the fact that Δ is regular since $\Delta \subseteq \mathbb{N}$. Therefore,

Claim 1. There are non-negative integers α_i, β_i, $1 \leqslant i \leqslant n$ for some n, such that

$$\Delta = \bigvee_{1 \leqslant i \leqslant n} \{\alpha_i + k\beta_i |\ k \in \mathbb{N}\}.$$

Define $Q := max_{1 \leqslant i \leqslant n}\{\alpha_i + \alpha_i\beta_i, \alpha_i, \beta_i\}$. Any $\delta \in \Delta$ can be written in the form of

$$\delta = kq + p \tag{19}$$

where $k \in \mathbb{N}$, $q, p \in \Delta$, $p, q \leqslant Q$. This can be shown by considering cases for $\alpha_i = 0$, $\beta_i = 0$, and both of them positive (in this case, if $\delta = \alpha_i + k'\beta_i$ and $\delta > Q$ for some k, then $\delta = (\alpha_i + k''\beta_i) + (k'''\beta_i)\alpha_i$ where $k' = k'''\alpha_i + k''$ with $k'' < \alpha_i$. Take $p = \alpha_i + k''\beta_i$ and $q = \alpha_i$ and $k = k'''\beta_i$). Formula (19) essentially says that, in M', any long loop (i.e., $\delta > Q$) can be replaced by a number of short loops (i.e., loops of lengths p and q which are bounded by Q). Let R be the smallest number such that, for each $q \leqslant Q$ with $q \in \Delta$ (there are finitely many such q's), there is a loop of length q whose left-radius and right-radius are both less or equal to R. Formula (19) implies that each long loop in M' can be simulated by short loops ($\leqslant Q$) with small ($\leqslant R$) radius.

Though M_{LR} and M' are different, a loop in M_{LR} is also a loop in M'. In particular, if the cell c is at least R cells away from both of the end markers, then any loop in M_{LR} at (s, c) can be simulated by short loops with small radius in M_{LR}, according to (19). What if the cell c is $r < R$ cells away from the left end marker? In this case, we can construct a machine M'' that is similar to M'. The input of M'' is with left end marker but without the right end marker (i.e., the input is right-infinite): $\text{¢}\ aaa\cdots$. M'' starts from state s and the same cell c. M'' works exactly the same as M' during which M'' never reads the left end marker. By memorizing the position r of the cell c in the finite control of M'', we can similarly to conclude that, there are numbers Q_r^1 and R_r^1 such that every loop of M'' at (s, c) can be simulated by short loops ($\leqslant Q_r^1$) in M'' at (s, c) with small right-radius ($\leqslant R_r^1$). When the cell c is $r < R$ cells away from the right end marker, we can analogously define M''' and the numbers Q_r^2 and R_r^2 such that every loop of M''' at (s, c) can be simulated by short loops ($\leqslant Q_r^2$) in M''' at (s, c) with small left-radius ($\leqslant R_r^2$). Define $\mathcal{Q}_s = \max_{0 \leqslant r \leqslant R}\{Q, Q_r^1, Q_r^2\}$ and $\mathcal{R}_s = \max_{0 \leqslant r \leqslant R}\{R, R_r^1, R_r^2\}$. Take $J = 4 \cdot \max_s \mathcal{R}_s$ and $\mathcal{Q} = \max_s \mathcal{Q}_s$. It can be concluded that,

> Claim 2. Whenever $B > J$, any loop of M_{LR} at (s, c) (no matter what the position of the cell c is) can be simulated by short loops ($\leqslant \mathcal{Q}$) at (s, c) in M_{LR} with small (both left and right) radius ($\leqslant \frac{J}{4}$).

We divide the input (with $B > J$) into three blocks: $\mathcal{B}_1$ (the first $\frac{J}{4}$ number of a's), $\mathcal{B}$ (the middle $B - \frac{J}{2}$ number of a's), and $\mathcal{B}_2$ (the last $\frac{J}{4}$ number of a's). Based upon Claim 2, we build three finite tables, called Tb_1, Tb, and Tb_2, as follows. For each state s, $0 \leqslant r \leqslant \frac{J}{4}$, $0 \leqslant q \leqslant \mathcal{Q}$,

- $Tb_1(s, r, q)$ is true iff M_{LR} has a loop at (s, c) with length q, where c is the r-th cell in $\mathcal{B}_1$;
- $Tb(s, q)$ is true iff M_{LR} has a loop at (s, c) with length q, where c is a cell in $\mathcal{B}$;
- $Tb_2(s, r, q)$ is true iff M_{LR} has a loop at (s, c) with length q, where c is the $(\frac{J}{4} - r)$-th cell in $\mathcal{B}_2$.

Notice that the truth value of the tables are unique and independent of the value of B (as long as $B > J$). Now, we construct a new machine M^* with one counter to simulate M_{LR}. M^*

works on input

$$\underbrace{¢a\cdots a}_{B}\underbrace{\$1\cdots 1}_{x}$$

which is the original input of M_{LR} padded with a number of 1's. M^* faithfully simulates M_{LR} except that the input head of M^* will not cross any a-cell (input cell containing symbol a) for more than S times. Obviously, M^* does not have the full power of M_{LR}, since some loops of M_{LR} may not be executed in M^*. In order to make M^* as strong as M_{LR}, whenever M^* reads an a-cell c and at state s, it looks at the three tables. If c is the r-th cell in $\mathcal{B}_1$, M^* nondeterministically and repeatedly increments its counter as follows:

(*) Nondeterministically pick a q satisfying $Tb_1(s, r, q)$;
increment the counter by q;
goto (*) or exit.

The cases when c is in $\mathcal{B}$ and $\mathcal{B}_2$ are similar. In this way, M^* is able to simulate any loops of M_{LR}. When M_{LR} reads the right endmarker and accepts, M^* starts to compare its counter value with the length of the padded string by decrementing the counter while reading the string. M^* accepts the padded input if the comparison is successful. Clearly, for $B > J$, (B, x) is accepted by M^* iff $T_{LR}(B, x)$. Notice that M^* is a finite-crossing reversal-bounded counter machine (with one 1-reversal counter) that accepts a semilinear language (Theorem 2.2). Hence, combining the case for $B \leqslant J$ mentioned earlier, we have established $T_{LR}(B, x)$ is Presburger. ∎

The rest of this section focuses 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 $b_1a_1^*b_2a_2^*\cdots b_ka_k^*b_{k+1}$ where b_i, $1 \leqslant i \leqslant k+1$, 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. We may assume, w.l.o.g, that, on a k-bounded input word $b_1a_1^*b_2a_2^*\cdots b_ka_k^*b_{k+1}$, a 2NCM(1,$r$) M makes a counter reversal only when it is reading one of the delimiters $b_1, \cdots, b_{k+1}$. Otherwise, we may insert up to r many new delimiters $c_1, \cdots, c_r$ to an input word of M and construct a new 2NCM(1,r) M' working on the new $k+r$-bounded word. M' simulates M properly and makes sure that, whenever M makes the i-th reversal, M' is reading the delimiter c_i. It is not difficult to show that, if the the bounded language accepted by M' is $\mathcal{D}$-definable, then so is the bounded language accepted by M.

We first consider the case when $r = 1$. Let M be a 2NCM(1,1) working on a k-bounded language. Let $w = b_1 1^{B_1}\cdots b_k 1^{B_k} b_{k+1}$ 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 $(B_1, \cdots, B_k)$. 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 $\alpha, \beta, \cdots$ 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 $\geqslant 0$ (resp. $\leqslant 0$) 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'), i-th block 1^{B_i}, and $d \in \{\text{LL,LR,RR, RL}\}$. According to Lemma 5.1, 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 $d =$ LL, then the transition is $((s, i+1), i, T, (s', i+1))$.
- If $d =$ LR, then the transition is $((s, i), i, T, (s', i+1))$.
- If $d =$ RR, then the transition is $((s, i), i, T, (s', i))$.
- If $d =$ 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_+ is equal to the initial state of G_-. It is observed that $(B_1, \cdots, B_k)$ is accepted by M iff there are some state s and some $1 \leqslant i_0 \leqslant k+1$ such that, $(B_1, \cdots, B_k)$ is accepted by G_{+-}^{s,i_0}. Since there are only finitely many choices of s and i_0, from Theorem 4.1, we obtain that the bounded language accepted by 2NCM(1,1) is effectively $\mathcal{D}$-definable.

Lemma 5.2 *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 the emptiness of 2NCM(1,r) M_r can be effectively reduced into the emptiness of the "intersection" of finitely many 2NCM(1,1)'s.

The r-reversal-bounded counter x behaves like this: $\nearrow, \searrow, \cdots, \nearrow, \searrow$. 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 $w = (B_1, \cdots, B_k)$ is an input to M_r, $PAD(w)$ is a string $(B_1, \cdots, B_k, E_1, \cdots, E_{m-1})$, i.e., it is $(B_1, \cdots, B_k)$ padded with some $(m-1)$-bounded word $(E_1, \cdots, E_{m-1})$. 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

$$\alpha = \langle d_1, q_1, d_2, q_2, \cdots, d_{m-1}, q_{m-1} \rangle,$$

where at the end of round $i = 1, \cdots, m-1$, 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 $\hat{M}_1, \cdots, \hat{M}_m$ such that:

(*) a k-bounded word w is in $L(M_r)$ iff $\alpha PAD(w)$ is in $L(\hat{M}_1) \cap \cdots \cap L(\hat{M}_m)$ 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 $i = 2, \cdots, m-1$, $\hat{M}_i$ carries out the following two phases:

(1) Restores the value of the counter to E_{i-1}, then moves its 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 5.2 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 [4,7].

Theorem 5.3 *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.*

We note that in the construction above, if the original machine M_r is a DCM(1,r), i.e., deterministic, then the machines $\hat{M}_1, \cdots, \hat{M}_m$ are also deterministic. Moreover, the machine M_r need not operate on a bounded language, since the construction of M'_r (that uses delimiters $c_1, \cdots, c_r$) does not really need this assumption. Hence:

Corollary 5.4 If M_r is a DCM(1,r) (resp. NCM(1,r)), one can construct m DCM(1,1)'s (resp. NCM(1,1)'s) $\hat{M}_1, \cdots, \hat{M}_m$ such that a word w is in $L(M_r)$ iff $\alpha PAD(w)$ is in $L(\hat{M}_1) \cap \cdots \cap L(\hat{M}_m)$ for some α and $PAD(w)$. Hence, the emptiness for NCM(1,r), $r = 1, 2, \cdots$ is decidable iff one can decide whether $L(\hat{M}_1) \cap \cdots \cap L(\hat{M}_m)$ is empty for any number m of machines $\hat{M}_1, \cdots, \hat{M}_m$ in NCM(1,1).

6 Comparing 2NCM(1) and 2DCM(1)

Recall that 2NCM(1) represents the class of machines $\cup_r$2NCM(1,r) and 2DCM(1) represents its deterministic version. Here, we compare their accepting capabilities for two cases: (1) when the inputs are bounded, and (2) when the inputs are unrestricted.

6.1 Bounded Inputs

Currently, it is open whether the classes 2NCM(1) and 2DCM(1) are equivalent over bounded languages. We believe that this is unlikely, even though over bounded languages, a finite-crossing 2NCM(c) can be converted to a finite-crossing 2DCM(c) for any c [5]. We use 2NCM(1)$\subseteq$ $\exists$2DCM(1) to stand for the following statement: for any 2NCM(1) with k-bounded input, there exists a 2DCM(1) with $k+n$-bounded input (for some n) such that, for any $B_1, \cdots, B_k$, $(B_1, \cdots, B_k)$ is accepted by the 2NCM(1) iff there exist $E_1, \cdots, E_n$ with $(B_1, \cdots, B_k, E_1, \cdots, E_n)$ being accepted by the 2DCM(1).

Theorem 6.1 *2NCM(1)$\subseteq$ $\exists$2DCM(1).*

Proof. From Theorem 5.3, the bounded language accepted by a 2NCM(1) is $\mathcal{D}$-definable. Since a tuple language definable by a ground formula in $\mathcal{D}$ can be accepted by a 2DCM(1) (a 2DCM(1) can check divisibility), the result follows. ∎

We use 2NCM(1)=$\exists$2DCM(1) to stand for the following statement: for any 2DCM(1) with $k+n$-bounded input, there exists a 2NCM(1) such that, for any $B_1, \cdots, B_k$, $(B_1, \cdots, B_k)$ is accepted by the 2NCM(1) iff there exist $E_1, \cdots, E_n$ with $(B_1, \cdots, B_k, E_1, \cdots, E_n)$ accepted by the 2DCM(1). It is open whether 2NCM(1)=$\exists$2DCM(1) is true or not. However, if it is true, then on bounded languages, 2NCM(1) is as expressive as $\mathcal{D}$-formulas, which can be shown immediately:

Theorem 6.2 *If 2NCM(1)=∃2DCM(1), then a bounded language is $\mathcal{D}$-definable iff it is accepted by a 2NCM(1).*

Next consider the following decision question:

Given: An equation E of the form $L_0 + L_1x_1 + \cdots + L_nx_n = 0$, where $x_1, \cdots, x_n$ are nonnegative integer variables and $L_0, L_1, \cdots, L_n$ are linear polynomials with integer coeffi cients $(+, -, 0)$ over nonnegative integer variables $y_1, \cdots, y_m$.
Question: Does E have a nonnegative integer solution in $x_1, \cdots, x_n, y_1, \cdots, y_m$?

Associate the string $0^{i_1}\#0^{i_2}\#\cdots\#0^{i_k}$ with each k-tuple of nonnegative integers $(i_1, \cdots, i_k)$, where $\#$ and 0 are distinct symbols. (Note that $0^0 = \epsilon$, the null string.) If the k-tuple is over the integers, then we can use another symbol, say 1, to represent a negative number, e.g., if $i_j = -4$, then this is represented by $1^{|i_j|}$, i.e., 1^4. However, for notational convenience we use $(i_1, \cdots, i_k)$ to also denote the string representing it. Note that the set of strings representing a set of k-tuples of integers is a bounded language over the alphabet consisting of symbols $\#, 0, 1$.

Given equation E, we defi ne the set Q of $(2n+m+1)$-tuples of integers $\alpha = (L_0, L_1, \cdots, L_n, Z_1, \cdots, Z_n, y_1, \cdots, y_m)$, where:

(1) Each L_i is positive, negative, or zero.
(2) Each Z_i has the same sign as L_i.
(3) Each y_i is nonnegative.
(4) For each $i = 1, \cdots, n$, L_i divides Z_i, if L_i is not zero; if L_i is zero, Z_i is zero.
(5) $L_0 + Z_1 + \cdots + Z_n = 0$.
(6) L_i is the value of the linear polynomial $L_i(y_1, \cdots, y_m)$.

Clearly, E has a solution if and only if Q is nonempty. It is straightforward to construct a 2DCM(1,r) for some r accepting the bounded language corresponding to Q. Since the emptiness problem for 2DCM(1,r)'s is decidable (Theorem 2.1), it follows that we can decide whether E has a nonnegative integer solution in $x_1, \cdots, x_n, y_1, \cdots, y_m$.

Now, consider a simpler equation E' of the form $L_0 + L_1x_1 + \cdots + L_nx_n = z$, where $x_1, \cdots, x_n, z$ are nonnegative integer variables and $L_0, L_1, \cdots, L_n$ are linear polynomials with *nonnegative* integer coeffi cients over nonnegative integer variables $y_1, \cdots, y_m$. Defi ne the set Q' of $(n + m + 1)$-tuples of integers $\alpha = (Z_1, \cdots, Z_n, y_1, \cdots, y_m, z)$, where the components of the tuples are defi ned similarly as above (note that $L_0, L_1, \cdots, L_n$ are no longer components of the tuple). Again, E' has a nonnegative integer solution if and only if Q' is nonempty, and Q' can be accepted by a 2DCM(1,r) for some r.

If, instead of Q', we defi ne Q'' as the set of $(m + 1)$-tuples of nonnegative integers $(y_1, \cdots, y_m, z)$ such that for some $x_1, \cdots, x_n$, E' is satisfi ed. One can easily construct a 2NCM(1,1) to accept Q''. The nondeterministic machine "guesses" the $x_i's$. However, we believe no 2DCM(1,r) can accept Q'' for any r. In fact, even if E' is of the simple form $a_1y_1x_1 + \cdots + a_ny_nx_n = z$, Q'' does not seem to be recognizable by a 2DCM(1, r) for any r, when n

is at least 2.

6.2 Unrestricted Inputs

In the case of 2NCM(1) and 2DCM(1) over unrestricted inputs, it has recently been shown in [6] that there is a language accepted by a 2NCM(1) that cannot be accepted by a 2DCM(1). For completeness, we describe the language that separates the two classes.

Consider only single-tape TMs Z over the alphabet $\{s_1, s_2, s_3\}$ (one symbol represents blank). We assume that these symbols are different from 0 and 1. Let $q_1, q_2, \cdots$ be the states, where q_1 is the initial state, and q_2 is the unique halting state.

Let r be a transition rule of the form $(q_i, a) \rightarrow (q_j, b, d)$, where d = 0 (1) represents left (right) move. We encode this rule by the string $E(r) = 1^i * a * 1^j * b * d$. If R is a set of rules = $\{r_1, \cdots, r_k\}$, let $E(R) = E(r_1)\%E(r_2)\% \cdots \%E(r_k)$. Note that R need not necessarily constitute a deterministic set of rules.

We represent a configuration of the TM on the tape as a string $w = u1^iv$, where u and v are strings in $\{s_1, s_2, s_3\}^*$. This represents the configuration where the tape content is uv, the read/write head is on the first symbol of v, and the state is q.

Let Σ be the alphabet $\{s_1, s_2, s_3, 0, 1, *, \%, \#\}$. Define the following language L_h over Σ as follows: A string $x\#w_1\#x\#w_2 \cdots \#x\#w_k$ is in L_h if:

(1) $x = E(R)$ is an encoding of a set of rules of a TM.
(2) $w_1, w_2, \cdots, w_k$ is a halting sequence of configurations of the TM represented by $E(R)$; i.e., for each i, w_{i+1} is a configuration that results from configuration w_i using a rule in x.

The following result was shown in [6].

Theorem 6.3 *Let L'_h be the complement of L_h, i.e., $L'_h = \Sigma^* - L_h$.*
(1) We can effectively construct a one-way nondeterministic finite automaton with one reversal-bounded counter M'_h accepting L'_h. Hence, L'_h is in 2NCM(1).
(2) L'_h is not in 2DCM(1).

7 Conclusions

We showed that the emptiness problem for two-way nondeterministic finite automata augmented with one reversal-bounded counter operating on bounded languages is decidable, resolving a problem left open in [4,7]. 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.

References

[1] A. Brauer. On a problem of partitions. *Amer. J. Math.*, 64:299–312, 1942.

[2] S. Ginsburg and E. Spanier. Semigroups, Presburger formulas, and languages. *Pacifi c J. of Mathematics*, 16:285–296, 1966.

[3] 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.

[4] E. M. Gurari and O. H. Ibarra. Two-way counter machines and Diophantine equations. *Journal of the ACM*, 29(3):863–873, 1982.

[5] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.

[6] O. H. Ibarra and Z. Dang. On two-way FA with monotonic counters and quadratic Diophantine equations. *Theoretical Computer Science*, 312:359–378, 2004.

[7] 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.

[8] R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12:161–177, 1992.

[9] L. Lipshitz. The Diophantine problem for addition and divisibility. *Transactions of AMS*, 235:271–283, 1978.

[10] K. Mahler. On the Chinese remainder theorem. *Math. Nachr.*, 18:120–122, 1958.

[11] Y. V. Matiyasevich. *Hilbert's tenth problem*. MIT Press, 1993.

[12] 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.

[13] J. L. Ramirez-Alfonsin. Complexity of the Frobenius problem. *Combinatorica*, 16:143–147, 1996.