Restrictions on Multicounter and Partially-Blind Multicounter Languages

Oscar H. Ibarra Ian McQuillan

Abstract

We introduce a new way of defining languages accepted by multicounter machines. Given a multicounter machine M and $m \geq 0$, the m-crossing language accepted by M is the set of all words where there is an accepting computation of M on w such that, for each counter j, each value c that can appear in the counter is crossed at most m times. We study this concept with multicounter machines and also with the partially-blind restriction where a counter cannot detect whether it contains zero or not. We show that both multicounter and partially-blind multicounter languages with one cross define exactly the reversal-bounded multicounter languages, while two crosses can accept strictly more including non-semilinear languages. We find that surprisingly, the family of m-crossing languages accepted by multicounter machines, for some m, and the family of m-crossing languages accepted by partially-blind multicounter machines, for some m, coincide. Decidability properties regarding m-crossing languages are also analyzed.[1]

1 Introduction

There have been many different types of automata that are built with counters as data store. The most well known is one-way nondeterministic k-counter machines, which have k non-negative integers as data stores, and based on whether a counter is zero or non-zero, it can either add one, subtract one, or keep the value the same. It is well known that

doi:10.56415/csjm.v32.20

[1] Supported, in part, by Natural Sciences and Engineering Research Council of Canada Grant 2022-05092 (Ian McQuillan)

even 2-counter machines have the same power as Turing machines [1]. However, with restrictions, it can limit the power and make some decision problems decidable. For example, a machine is r-reversal-bounded (respectively, reversal-bounded) if, in each accepting computation, for each counter, the number of changes between sections of the counter increasing and decreasing is at most r (some number). Let NCM be the class of one-way reversal-bounded multicounter machines. These machines have some decision problems that are decidable, for example, the emptiness problem ("Given M, is $L(M) = \emptyset$?").

Another model is the class of *one-way nondeterministic partially-blind multicounter machines* (denoted by PBLIND), introduced by Greibach [2]. With this model, unlike traditional multicounter machines, the transition cannot detect whether each counter is zero or positive at each step. Hence, if a counter contains a positive number and there is a transition to subtract one from it, the computation can continue executing this transition. However, if the counter contains zero and there is a transition to subtract one, the computation simply cannot continue (or, it "crashes") with this transition. A computation is *accepting* if it reads the input in a final state with all counters being zero. It is known that the emptiness problem is decidable for PBLIND. This was determined since Greibach [2] showed that the following are equivalent: deciding emptiness for partially-blind multicounter machines, deciding the emptiness problem for Petri nets, and deciding reachability of vector addition systems. Later, reachability for Petri nets was shown to be decidable [3], [4] and, therefore, non-emptiness for PBLIND is decidable as well. It is also known that PBLIND is strictly more powerful than NCM [2].

Recently, a generalization of partially-blind multicounter machines was defined and studied [5]. This model makes testing whether a counter is zero or positive optional within each transition. Such a machine is *t-testable* (respectively, *finite-testable*) if, in every accepting computation, each counter has at most t (respectively, some finite number of) sections that decrease that counter where its status is tested. Then it is clear that 0-testable machines are the same as PBLIND, and furthermore it was proved that one-way nondeterministic finite-testable multicounter machines are in fact equivalent to PBLIND [2]. However,

for deterministic machines, there is a difference in capacity, and one-way deterministic partially-blind multicounter languages (denoted by DPBLIND) are strictly contained in the family of one-way deterministic finite-testable multicounter languages. Also interestingly, the containment problem ("Given two machines M_1, M_2, is $L(M_1) \subseteq L(M_2)$?") is decidable for one-way deterministic finite-testable multicounter machines. This problem is therefore also decidable for DPBLIND. These are the most general known models with a decidable containment problem as even the very simple class of nondeterministic machines with a single counter that cannot decrease after increasing, has an undecidable containment problem.

In this paper, we study a new model that is a variation of both multicounter and partially-blind multicounter languages. Given a k-counter machine, the *m-crossing* language accepted by M is the set of words with accepting computations where, for each counter, it switches from counter value c to $c+1$ at most m times. This is not simply a restriction of counter machines or partially-blind machines however, because they are defined such that a computation cannot continue (it crashes, akin to if a partially-blind counter attempts to go below zero) if it crosses between two consecutive counter values more than m times. Thus, the machines themselves are not m-crossing, and a machine itself has no direct way to remember when a counter value is crossed. We simply look at the subset of accepting computations that cross each value at most m times and use this to define a language. This is unlike the finite-crossing definition on Turing machines that has previously been studied (where the boundary between each two worktape cells is crossed at most a bounded number of times) [6] — with this definition, the machine itself can keep track of the crosses using the worktape. We make comparisons of these machines with NCM. We also surprisingly prove that finite-crossing multicounter languages are equal to finite-crossing partially-blind multicounter languages; therefore, the ability of multicounter machines to test within each transition whether a counter contains zero or is positive is not needed for finite-crossing languages.

2 Preliminaries

We denote by $\mathbb{Z}$ (respectively $\mathbb{N}, \mathbb{N}_0$) the set of all integers (respectively positive integers, non-negative integers). Given $n \in \mathbb{N}_0$, let

$$\text{sign}(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{otherwise.} \end{cases}$$

Given a set X and $n \in \mathbb{N}$, let X^n be the set of all n-tuples of X.

We assume a working knowledge of introductory automata and formal language theory [1]. This includes models such as finite automata and Turing machines. We denote by REG the family of regular languages (those accepted by finite automata).

Let Σ be a finite alphabet. By Σ^*, we denote the set of all words over Σ. Any $L \subseteq \Sigma^*$ is called a *language* over Σ. The empty word is denoted by λ. Given a word $w \in \Sigma^*$ and $a \in \Sigma$, let $|w|_a$ be the number of a's in w, and for a fixed ordering of letters, $\Sigma = \{a_1, \ldots, a_n\}$, the *Parikh image* of w is $\psi(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$, extended to languages $L \subseteq \Sigma^*$, $\psi(L)$ in the natural way. We will not define a language being semilinear here, but describe an equivalent criteria. A language L is *semilinear* if and only if there is a regular language R with $\psi(L) = \psi(R)$ [7].

Previously in the literature, multicounter machines and partially-blind multicounter machines have been studied [2] as defined next.

A *one-way nondeterministic k-counter machine* is a tuple $M = (Q, \Sigma, \delta, q_0, F)$ with a finite set of states Q, initial state q_0, the final state set $F \subseteq Q$, an input alphabet Σ, and a transition function δ from $Q \times (\Sigma \cup \{\lambda\}) \times \{0,1\}^k$ into subsets of $Q \times \{0, +1, -1\}^k$.

An *instantaneous description (ID)* of M is a member of $Q \times \Sigma^* \times \mathbb{N}_0^k$. Instantaneous descriptions change via the relation $\vdash_M$ (or just $\vdash$ if M is understood) with $(q, aw, y_1, \ldots, y_k) \vdash (q', w, y_1 + v_1, \ldots, y_k + v_k)$, if $(q', v_1, \ldots, v_k) \in \delta(q, a, u_1, \ldots, u_k)$ and $\text{sign}(y_j) = u_j$, (and $y_j + v_j \geq 0$) for $1 \leq j \leq k$. Then $\vdash^*$ is the reflexive, transitive closure of $\vdash$. A *computation* on $w \in \Sigma^*$ is a sequence of IDs,

$$(q_0, w_0, y_{0,1}, \ldots y_{0,k}) \vdash \cdots \vdash (q_n, w_n, y_{n,1}, \ldots, y_{n,k}), \tag{1}$$

where $w_0 = w$, and a computation is an *accepting computation of w* if $w_0 = w, y_{0,j} = y_{n,j} = 0, 1 \leq j \leq k, w_n = \lambda, q_n \in F$.

Such a k-counter machine M is *r-reversal-bounded*, if, in every accepting computation, each counter j, $1 \leq j \leq k$, alternates at most r times between executing maximal sequences of non-decreasing transitions (adding 0 or 1) and executing maximal sequences of non-increasing transitions (adding 0 or -1). It is *reversal-bounded* if it is r-reversal-bounded for some r. Reversal-bounded multicounter machines have been extensively studied before, e.g., [8].

Such a k-counter machine is called *partially blind* if, for each $q \in Q, a \in \Sigma \cup \{\lambda\}$, $\delta(q, a, u_1, \ldots, u_k) = \delta(q, a, v_1, \ldots, v_k)$, for all $u_i, v_i \in \{0, 1\}$. For this reason, with partially-blind machines, we usually write the transition function as being from $Q \times (\Sigma \cup \{\lambda\})$ into subsets of $Q \times \{0, +1, -1\}^k$. With partially-blind machines, the machines cannot detect whether a counter is zero and can "crash" (i.e., the computation cannot continue), if any counter goes below zero.

The language accepted by k-counter machine M is,

$$L(M) = \{w \in \Sigma^* \mid \text{there is an accepting computation of } M \text{ on } w\}.$$

The family of languages accepted by one-way nondeterministic k-counter machines (respectively, k-counter partially-blind machines, k-counter reversal-bounded machines) is denoted by $\mathsf{COUNTER}(k)$ (respectively, $\mathsf{PBLIND}(k), \mathsf{NCM}(k)$). The family of languages accepted by k-counter (respectively, k-counter partially-blind, k-counter reversal-bounded) machines, for some k, is denoted by $\mathsf{COUNTER}$ (respectively, $\mathsf{PBLIND}, \mathsf{NCM}$).

It is also known that one-way (in fact, deterministic) two-counter machines accept all recursively enumerable languages, but there are some recursively enumerable languages that are not in PBLIND [2] and, therefore,

$$\mathsf{PBLIND} \subsetneq \mathsf{COUNTER}(2) = \mathsf{COUNTER}.$$

Lastly, note $\mathsf{NCM} \subsetneq \mathsf{PBLIND}$ [2]; indeed, NCM only contains semilinear languages [8] but PBLIND does not.

3 Finite-Crossing Multicounter Languages

We start by defining the new notion that is of primary interest in this paper. Let M be a one-way nondeterministic k-counter machine

$M = (Q, \Sigma, \delta, q_0, F)$. Given an accepting computation on w,

$$(p_0, w_0, y_{0,1}, \ldots y_{0,k}) \vdash \cdots \vdash (p_n, w_n, y_{n,1}, \ldots, y_{n,k}), \tag{2}$$

where $w_0 = w, y_{0,j} = y_{n,j} = 0, 1 \leq j \leq k, w_n = \lambda, q_n \in F$, we say counter j *upward crosses* (respectively, *downward crosses*) $c \in \mathbb{N}_0$ m times, if

$$|\{i \mid y_{i,j} = c, \text{ and } y_{i+1,j} = c + 1\}| = m$$

(respectively,

$$|\{i \mid y_{i,j} = c + 1, \text{ and } y_{i+1,j} = c\}| = m).$$

We use the word *cross* to mean upward cross. So, it only counts as a cross of c if the counter switches from c to $c + 1$ in the previous step. In this definition, M may or may not be partially-blind. Given $m \geq 0$, if an accepting computation has for each counter j, $1 \leq j \leq k$, at most m crosses of each counter value, then the accepting computation is said to be *m-crossing*. For each $m \geq 0$, the *m-crossing language* accepted by M is

$$L_m(M) = \{w \in \Sigma^* \mid \exists \text{ an } m\text{-crossing accepting computation of } w\}.$$

It is clear that every accepting computation of M has some m whereby each counter value is crossed at most m times. Therefore, $\bigcup_{m=0}^{\infty} L_m(M) = L(M)$. Also note, if we had instead counted a cross of c as going from either c to $c + 1$ or $c + 1$ to c, then because every counter must end at 0 in every accepting computation, each upward cross gets uniquely matched with a downward cross and, therefore, an odd number of crosses would be impossible and the number of crosses would always be exactly two times more than our definition. Hence, we use the simpler definition of crossing from c to $c + 1$ to count as a cross of c. Using a similar argument, the number of upward crosses of c in any accepting computation is always equal to the number of downward crosses of c. Thus, we can choose to either use upward-crosses or downward-crosses for crosses without any issue. (We will sometimes say that we are referring to crosses by downward crosses.)

As previously discussed with partially-blind machines, a computation "crashes" if a counter goes below zero. With m-crossing languages,

the behaviour is similar as a computation "crashes" if they cross any counter value more than m times. Therefore, this notion is not a restriction of partially-blind machines, but rather a restriction of the accepting computations.

The family of m-crossing languages accepted by one-way nondeterministic k-counter (respectively, partially-blind) machines is denoted by $\mathsf{COUNTER}_m(k)$ (respectively, $\mathsf{PBLIND}_m(k)$), the family of languages accepted by m-crossing multicounter (respectively, partially-blind multicounter) machines is denoted by $\mathsf{COUNTER}_m$ (respectively, PBLIND_m), and the family of languages accepted by m-crossing multicounter (respectively, partially-blind multicounter) machines for any m, called *finite-crossing*, is denoted by $\mathsf{COUNTER}_{\mathrm{FIN}}$ (respectively, $\mathsf{PBLIND}_{\mathrm{FIN}}$).

Next, we analyze some basic properties of these language families. First, we study some basic inclusion properties. The following proof of $\mathsf{PBLIND}_m \subseteq \mathsf{PBLIND}_{m+1}$ and $\mathsf{COUNTER}_m \subseteq \mathsf{COUNTER}_{m+1}$ looks like it should be simple but is deceptively difficult because it is not machines that are m-crossing, but languages. If we take a PBLIND machine M, we know $L_m(M) \subseteq L_{m+1}(M)$. From M, we need to build M' such that $L_m(M) = L_{m+1}(M')$. But machines cannot keep track of which cells are crossed. If we build M' to only cross say 0 of each counter one more time from M, and then simulate M, then $L_{m+1}(M')$ might still accept more words than $L_m(M)$ as it would allow for one more cross of each number greater than 0. We need to build M' so that it has one more cross on every counter value crossed by M. This is possible by first guessing the largest number that can appear in each counter in a given computation, crossing all of the numbers up to that maximum value once, and then verifying that the guess is correct.

Proposition 1. *The following statements are true:*

- *for every* $m \geq 0$, $\mathsf{PBLIND}_m \subseteq \mathsf{PBLIND}_{m+1}$ *and* $\mathsf{COUNTER}_m \subseteq \mathsf{COUNTER}_{m+1}$.
- $\mathsf{PBLIND}_0 = \mathsf{COUNTER}_0 = \mathsf{REG}$,

Proof. We will show the proof for PBLIND as it can be made simpler with counter machines. Let M be a $\mathsf{PBLIND}(k)$ machine. We build

M' with counters called $C_1, \ldots, C_k, D_1, \ldots, D_k, E_1, \ldots, E_k$. At the beginning, M' nondeterministically guesses $(u_1, \ldots, u_k)$, the maximum values that will appear in the k counters in the simulation of M, by adding 1 u_j times to each of D_j and E_j in parallel. Next, M' simulates M using the $C_1, \ldots, C_k$ counters. In parallel, M' performs the same simulation "backwards" on both the D_j and E_j in the sense that, each decrease of C_j by 1 causes an increase of D_j and E_j by 1, and each increase of C_j causes a decrease of D_j and E_j by 1. For each counter j, at a nondeterministically guessed spot of the computation, M' guesses that E_j contains zero, and stops using it, thereby requiring that it ends with zero in any accepting computation and it was therefore zero when stopped. At the end of the simulation, M' subtracts some nondeterministically guessed amount from the D_j counters. Hence, M' accepts if it reads the input and ends in a final state of M with all counters being zero.

Let $w \in L_m(M)$, and consider an m-crossing accepting computation α of M on w. Then, for each counter j, M' guesses the maximum value u_j that can appear in counter j in α, and adds u_j to both D_j and E_j. Then M' simulates α using $C_1, \ldots, C_k$, with the simulation "backwards" in D_j and E_j as indicated above. This is accepting in M' because the D_j counters do not crash by going below 0 (which would only happen if the guessed maximum was in fact too small), and the E_j counters can stop where the maximum value in C_j is achievable and, therefore, E_j contains zero. Further, this described accepting computation is $(m+1)$-crossing because each D_j crosses every value that C_j can cross once initially, plus as the C_j and D_j counters are both simulating M (with D_j backwards), D_j crosses $u_j - c - 1$ exactly once for every cross of c by C_j. Thus, after the initial guess, each value is m-crossing as are all the other counters. In all, M' can accept w with a $(m+1)$-crossing computation.

Let $w \in L_{m+1}(M')$, and let α be an $(m+1)$-crossing accepting computation. Then, looking at the simulation of M on the C_j counters, this must be accepting in M. Clearly, the simulation of M is also $(m+1)$-crossing in M but we need to make sure it is m-crossing. Looking at the D_j counters, it crosses a value $u_j - c - 1$ one more time than C_j crosses c. Because this happens at most $m+1$ times, it implies

that c is crossed at most m times by C_j.

Lastly, $\mathsf{PBLIND}_0 = \mathsf{REG}$ as the counters cannot be used at all, thus essentially acting as a finite automaton. □

To note, property 1 above causes an increase in the number of counters. It is open for each k and m, whether $\mathsf{PBLIND}_m(k) \subseteq \mathsf{PBLIND}_{m+1}(k)$ (and, similarly, with $\mathsf{COUNTER}$).

Next, we see that 1-crossing languages and NCM are equal.

Proposition 2. $\mathsf{PBLIND}_1 = \mathsf{COUNTER}_1 = \mathsf{NCM}$.

Proof. First, to show $\mathsf{NCM} \subseteq \mathsf{PBLIND}_1$ and $\mathsf{NCM} \subseteq \mathsf{COUNTER}_1$, it is known that every NCM language can be accepted by a machine where every counter is 1-reversal-bounded [8]. Hence, every accepting computation of such a machine is 1-crossing. The only other change that needs to be made is for partially-blind counters, which can only detect a zero in each counter at the end of the computation, whereas NCM machines can detect at any point when they hit zero. Thus, when a partially-blind machine simulates the NCM machine, it needs to nondeterministically guess when each counter hits zero in each counter, then simulate only transitions on that counter being zero. Then it verifies at the end of the computation that this guess was correct.

For the converse, let M be a k-counter machine. Then create an NCM M' such that, M' keeps track, for every counter i, of whether counter i has already increased, and then decreased. If it has, then it no longer allows counter i to increase again. This accepts the same language, because any computation where counter i increases, then decreases, then increases again must not be 1-crossing. □

Proposition 3. *There is a non-semilinear language in* $\mathsf{PBLIND}_2(2)$ *and* $\mathsf{COUNTER}_2(2)$.

Proof. It is obviously enough to show this for partially-blind machines. Consider the language

$$L = \{d^{n_1}ba^{n_2}b\cdots a^{n_l}b \mid l > 1, n_i > n_{i+1} \geq 1 \text{ for } 1 \leq i < l\}.$$

First, we will show that this is not semilinear. First, let $L' = \{d^nba^{n-1}b\cdots a^1b \mid n > 1\}$. This language L' is not semilinear, as

erasing the b's and mapping d to a with a homomorphism creates a non-regular unary language $\{a^{n(n+1)/2} \mid n \geq 0\}$; and a unary language is semilinear if and only if it is regular [7]. Thus, L' is not semilinear.

Suppose, by contradiction, that L is not semilinear. Then, there must exist a regular language R with the same Parikh image as L [7]. Hence, there must also exist an NCM language with the same Parikh image as L. Let $\hat{M}$ be such a NCM machine. Let $\hat{M}'$ be obtained from $\hat{M}$ by also verifying that the number of d's is equal to the number of b's. This can be done by adding two extra counters to $\hat{M}$, counting the number of b's in one counter, and the number of d's in another counter, and then at the end of the input, verifying that they are the same. But any language with the same Parikh image as L that enforces that the number of d's is equal to the number of b's must have the same Parikh image as L'. But also $L(\hat{M}') \in \mathsf{NCM}$, and is therefore semilinear since all NCM languages are semilinear [8]. Hence, L' must be semilinear as well since it has the same Parikh image as $L(M')$, a contradiction. Hence, L is not semilinear.

We will build a machine $M \in \mathsf{PBLIND}(2)$ such that the 2-crossing language accepted by M is equal to L. So, if any value of any of the two counters is crossed going up, crossed going down, then up, then down, then up again, then M crashes. Say the input is of the form:

$$d^{n_1}ba^{n_2}b\cdots a^{n_l}b.$$

The two counters will verify that each section of a's has fewer copies than the previous section of either d's or a's. We call the first counter C_1 and the second counter C_2.

The machine first reads d^{n_1} and places n_1 in C_1. Next, the machine will verify that the first section of a's has fewer a's than the number of d's. To start this, as it reads a^{n_2}, it decreases C_1 by $n_2 + 1$, while in parallel, increasing C_2 by n_2. Clearly, $n_2 \geq n_1$ if and only if the machine crashes by going below zero. Hence, in an accepting computation, the number of a's in the first section of a's must be fewer than the number of d's.

At this point, C_1 contains $n_1 - (n_2 + 1) = n_1 - n_2 - 1$, the largest value that has ever been contained in C_1 is n_1, C_2 contains n_2, and the biggest value of C_2 where a decrease (or zero) occurred is zero (as C_2

has not yet decreased).

Next, for each i from 3 to l, the machine alternates back and forth between the following two phases where it does the first phase if i is odd and the second phase when i is even until $i = l$:

1. Say C_1 starts this phase with value c and the highest C_1 has ever reached is c' (with $c < c'$), and C_2 starts with value e' and the biggest value of C_2 where a decrease (or zero) occurred is e. It is easy to see inductively that $e + n_{i-1} = e'$. It increases C_1 to some arbitrary nondeterministically guessed amount c'', then decreases it by 1, then increases it by 1 again. If the new counter value c'' has been used by this counter before (i.e., if $c'' \leq c'$), the machine crashes since $c'' - 1$ would have been previously crossed beforehand, crossed again on the increase, and crossed a third time when subtracting once and then back again. Therefore, this new counter value must be unused by C_1 if the machine reaches this stage and eventually accepts. At this point, in parallel, M reads a^{n_i} and increases C_1 by n_i, and also in C_2 (which currently contains e') it decreases by n_i until it hits e'', say. Then e'' is verified to be more than e by finally decreasing by one more than increasing back by 1 (it would therefore crash if e'' was not more than e). This implies that any computation reaching this stage must have $n_i < n_{i-1}$. Note that at the end of this phase, C_1 has some value that is n_i larger than the largest value where a decrease occurred.

2. This is the same as phase 1 with the roles of C_1 and C_2 reversed.

Therefore, each section of a's must have strictly fewer a's than the previous section. At the end of the input, M decreases all counters by some nondeterministically guessed amount, and switches to a final state, thereby only accepting if they are all zero. □

Corollary 1. $\mathsf{PBLIND}_1 \subsetneq \mathsf{PBLIND}_2$ *and* $\mathsf{COUNTER}_1 \subsetneq \mathsf{COUNTER}_2$.

This follows since all NCM languages are semilinear [8], by Proposition 2 and since PBLIND_2 accepts non-semilinear languages.

Next we will compare $\mathsf{COUNTER}_{\mathrm{FIN}}$ to $\mathsf{PBLIND}_{\mathrm{FIN}}$. Surprisingly, we see that they are equivalent and that if we restrict accepting computations to be finite-crossing, then detecting whether counters are zero or not does not help the capacity (unlike without this restriction since $\mathsf{PBLIND} \subsetneq \mathsf{COUNTER}$). To note in the proof below, both the number of crossings and the number of counters increases, and it is not evident how the proof would work without this allowance (although the number of crossings only increases by 1 in the proof).

We start with an informal description of the proof to help the reader as the proof is quite lengthy. Intuitively, given a k-counter machine M, a partially-blind machine M' can simulate M, but it needs to know when each counter is zero or not in order to apply the correct transition. As it simulates M, M' can guess when M downward crosses to 0 and verify that it is correct with additional counters. For all situations after a downward cross where it does not guess that the counter is 0, it also needs to verify that it is not empty. This can be achieved by decreasing by one and then increasing by one again, which would cause a crash if it were zero. However, this causes an increase in the number of crosses — we do this in such a way (by using a set of new counters) that it causes an increase of exactly one cross of every counter value from 0 to the maximum value crossed in each counter minus 1. But we need to add a single cross also to the highest number crossed in M so that we always have one more cross than M. This can be accomplished using a similar technique to the proof of Proposition 1 to add a single cross to the maximum value crossed.

Proposition 4. $\mathsf{COUNTER}_{\mathrm{FIN}} = \mathsf{PBLIND}_{\mathrm{FIN}}$.

Proof. It is clear from the definition that every $\mathsf{PBLIND}_{\mathrm{FIN}}$ machine is indeed a $\mathsf{COUNTER}_{\mathrm{FIN}}$ machine, and so $\mathsf{PBLIND}_{\mathrm{FIN}} \subseteq \mathsf{COUNTER}_{\mathrm{FIN}}$.

For the opposite direction, let M be a $\mathsf{COUNTER}(k)$ machine. We assume without loss of generality that each transition of M changes at most one counter. Throughout this proof, we will equivalently use downward crosses as crosses, so crossing c means switching from $c+1$ to c. As a k-counter machine (not partially blind), M can have different transitions to be applied depending on whether each counter is 0 or not, and we need to build M' so that it can somehow know the appropriate

transition to apply even though it cannot directly detect whether a counter is zero or not.

To help explain the proof, we will start by describing a simpler special case. The special case is where, for every m-crossing accepting computation of M, for every counter j, it must cross zero on counter j **exactly** m times (instead of at most m times from the m-crossing definition). We build $M' \in \mathsf{PBLIND}$ with $(m+1)k$ counters. There are $m+1$ counters associated with each counter of M. Let $C_{j,i}, 1 \leq j \leq k, 0 \leq i \leq m$ be counter labels. Then M' simulates M and at the start, simulates the moves of counter j identically on all counters $C_{j,i}$, $0 \leq i \leq m$. But as it proceeds, for each counter j, and one at a time, from $i := 1$ to m (it does not do this for $i = 0$), it guesses the ith position directly after a decrease where counter j crosses 0. At each of these points, M' stops using $C_{j,i}$ (but keeps using all other counters that have not yet stopped), thereby verifying at the end of the accepting computation that all counters that were stopped were indeed zero when stopped. Notice that for each j, $C_{j,0}$ is never stopped and so the entire computation is simulated on those counters. As M' verifies that it does this m times for each counter and because each counter crosses zero exactly m times in any m-crossing accepting computation, this means M' knows exactly when each counter is zero, and in these cases, it can simulate a transition of M on counter j being 0 (this includes at the beginning of the computation before an increase, and after a guessed cross to 0 up until an increase); and it simulates a transition of M on the counter being positive otherwise. Thus, $L_m(M) = L_m(M')$ for this special case.

Next, we will remove this special case. It is quite a bit more difficult if accepting computations can downward cross to zero on a counter fewer than m times because the partially-blind machine cannot guess the appropriate number of times it switches to zero and then easily verify that it was correct (e.g., if it guessed that it crossed 0 less than m times, what if it hit zero other times but the machine M' was not able to detect it and execute different transitions when it was actually zero?)

We build $M' \in \mathsf{PBLIND}(k(2m+3))$, but by using $(m+1)$-crossing computations. We use counters labelled by $C_{j,i}$, $1 \leq j \leq k, 0 \leq i \leq m$,

$D_{j,i}, 1 \leq j \leq k, 1 \leq i \leq m$, and E_j, F_j, $1 \leq j \leq k$ where all counters with subscript j in the first component are associated with counter j of M.

First, we need to employ a technique similar to the proof of Proposition 1 where we need to guess the maximum value that appears in each counter and verify that it is correct. To do this, M' guesses a vector $\vec{u} = (u_1, \ldots, u_k)$, by, for each j, $1 \leq j \leq k$, adding 1 a nondeterministically guessed amount of times until it has u_j in both E_j and F_j. For the rest of the computation, M' will simulate M and it will verify that u_j was the maximum value hit by counter j using E_j and F_j in the simulated computation, as described below.

Next, M' guesses some vector $\vec{v} = (v_1, \ldots, v_k), 1 \leq v_j \leq m$ (of which there are a finite number of possible vectors), which it remembers in the state. The value v_j will be the guessed number of downward crosses to 0 of the simulation on counter j of M. Similar to the special case proof above, it will start by simulating counter j of M on all counters $C_{j,i}, 0 \leq i \leq m$, but also now identically on all counters $D_{j,i}$, $1 \leq i \leq m$.

As part of the simulation, M' will nondeterministically guess v_j positions, for each i from 1 to v_j where counter j crosses 0, and at these spots, it will stop using $C_{j,i}$, but keep using all the counters that have not yet stopped. At all positions where it guesses counter j is zero (and at the beginning before simulating an increase on counter j) and until counter j again increases, it simulates only transitions of M where counter j is 0 (this is called to 0 case). At the end of the computation, M' will verify that all stopped counters were indeed 0 when they were stopped. Also the counters $C_{j,0}$ which simulate M until the end must also end with all zeros. When simulating M, in all cases other than the 0 case, M' will simulate all transitions on counter j of M being non-zero. However, we need a way to verify that counter j was indeed non-zero at all of these positions where it simulated a transition on counter j being positive. To do this, after each downward cross on counter j to counter value c in $C_{j,0}$ where M' does not guess counter j is now 0, it immediately executes another two new transitions on one of the counters $D_{j,i}$ (where i is nondeterministically chosen), that decreases by 1 to $c-1$ causing a cross of $c-1$, and then immediately increases $D_{j,i}$ back to c again (these two transitions only change $D_{j,i}$).

Then M' simulates the next transition of M on counter j not being 0 on all counters that have not yet stopped. Thus, the cross of c by M causes M' to cross c (as in M), then $c-1$ but only in one of the counters $D_{j,i}, 1 \leq i \leq m$. It is clear that by decreasing by 1 then increasing by 1 again, that any accepting computation could not contain 0 at that point in counter j, as M' would crash if it decreased by 1 from being 0 in the counter. In addition, M' does not let the simulation continue if it tries to increase after the v_jth cross of zero.

Finally, as it is simulating M it will simulate the same computation also on both E_j and F_j as it does on $C_{j,0}$, but it does so where every increase by 1 instead decreases by 1, and every decrease by 1 will instead increase by 1 (as with Proposition 1); and at some nondeterministically guessed point for each $j, 1 \leq j \leq k$, where it guesses $C_{j,0}$ contains the maximum u_j, it will stop using F_j but continue using E_j. This stopping verifies that it must end with 0, thereby verifying that the guessed value u_j can indeed be hit in counter j. Furthermore, it must be able to hit at most this amount otherwise E_j would decrease below 0. Hence, the guessed u_j was indeed the maximum. At this same point of the computation where it guesses it is at the maximum, $C_{j,0}$ will subtract 1, then add 1, thereby crossing the maximum value crossed, $u_j - 1$ one extra time (the purpose of this will be explained below). At this point, it continues simulating M as described above. Finally, at the end of the simulation, M' decreases each E_j a nondeterministically guessed amount until zero. It remains to check that the words that can be accepted by m-crossing computations of M are precisely those that can be accepted by $(m+1)$-crossing computations of M' (proving $L_m(M) = L_{m+1}(M')$). We will show this in the remainder of this proof.

Let $w \in L_m(M)$. Consider an m-crossing accepting computation α. Let j be a counter of M. Any value c that is crossed is crossed at most m times; let c_j be the number of times value c is crossed in counter j, let v_j be the number of times 0 is crossed in counter j, and let $\vec{u} = (u_1, \ldots, u_k)$ be such that u_j is the maximum value reached in counter j. This computation can be simulated by M' as follows. First, M' guesses $\vec{u}$ by putting it on the E_j and F_j counters and then it guesses $\vec{v}$ as described, which it saves in the finite control. Then it simulates α with counter j changing all of the $C_{j,i}, D_{j,i}$ counters, and

with E_j and F_j changing "backwards" by adding 1 when the simulation subtracts 1, and subtracting 1 when the simulation adds. To simulate a transition not directly after a downward cross, it does so based on the counter status of the previous transition on that counter. After a downward cross to counter j, it guesses if its contents c is 0 or not. If it guesses $c > 0$, if it is the ith time c is downward crossed in M', M' decreases $D_{j,i}$ (certainly it could also happen in different orders as i is chosen nondeterministically, but one order that works is sufficient to produce an accepting computation in M'), then increasing it again. Thus, each time c is crossed in M causes a different counter $D_{j,i}$ to cross $c-1$ in M'. In this situation, after simulating a downward cross of c in counter j, each counter $D_{j,i}$ crosses $c-1$ at most one extra time in the accepting computation. Since we are looking at $(m+1)$-crossing computations, this simulation produces at most one more crossing than m for all c between 0 and u_j-2 (u_j is the largest value hit, which means u_j-1 is the largest value crossed, but it only decreases a $D_{j,i}$ counter after a decrease to test the counter status) and can accept. Looking at the E_j, F_j counters, when counter j reaches u_j, F_j is 0 and stops, then $C_{0,j}$ downward crosses u_j-1 (the largest crossed) one time more than the simulation of M. Thus, each number c crossed is crossed at most $m+1$ times. The E_j counters cross u_j-c-1 exactly once for every cross of c in α (which is at most m), plus they create one more cross than α as they are being decreased to 0 at the end. So this accepting computation is $(m+1)$-crossing. Hence $w \in L_{m+1}(M')$.

Let $w \in L_{m+1}(M')$. Let α be an $m+1$-crossing accepting computation of M' on w. This starts by guessing the vector $\vec{u} = (u_1, \ldots, u_k)$ of maximum values and storing them in the E_j and F_j counters, and since α accepts and is simulated with opposite operations from the simulation on the counters of M, this implies that these are indeed maximum values as being less than the maximum would cause some F_j counter to not reach 0 to accept, and being more than the maximum would cause the computation to crash. Then it guesses $\vec{v} = (v_1, \ldots, v_k)$, where v_j is the number of times counter j downward crosses to 0, and correctly is able to make sure that it is zero at the beginning before an increase, after a guessed cross (by checking that $C_{j,i}$ is 0 after the ith guessed cross of zero) and again before an increase, and it correctly simulates a

transition on counter j being positive by decreasing (and re-increasing) one of the counters $D_{j,i}$ (which has identical contents to $C_{j,0}$), which therefore must be non-zero since α is accepting.

It suffices to show that the computation of M that α simulates is m-crossing. Let j be a counter, and c be a value that is crossed $m+1$ times in counter j in α. First, assume $0 < c < u_j - 1$ ($u_j - 1$ is the biggest number crossed). Then each time after c is crossed, α must guess that $c > 0$ and decrease one of $D_{j,i}$ for some i, $1 \leq i \leq m$ and add one crossing to at least one of them, and so it will have at least one more crossing from $C_{j,0}$, but to the value $c-1$ (and not to c). But because c is crossed at least once (since $c < u_j - 1$), some counter $D_{j,i}$ for some i, say will create at least one extra cross of c and, therefore, c will be crossed at least one fewer times in the simulation than $D_{j,i}$ of M' and, therefore, the simulation of M has at least one fewer crosses of c than $m+1$, which is thus at most m crosses. Assume $c = 0$. If the simulated computation crosses 0 at most m times, we are done. Assume it crosses zero $m+1$ times (thus, α does this as well). But M' does not let the simulation continue after the v_jth cross and, therefore, this cannot happen. Assume finally that $c = u_j - 1$. But $C_{j,0}$ crossed $u_j - 1$ one more time than in the simulation (as it decreased, then increased once at its maximum value). Hence, $w \in L_m(M)$. □

4 Decidability Properties

In this section, we look at several decidability properties related to m-crossing computations of either multicounter or partially-blind multicounter machines.

Proposition 5. *It is decidable, given a* PBLIND *machine M and $m \geq 0$, whether every accepting computation is m-crossing.*

Proof. Given M with k counters and m, create a partially-blind M' with $k + 2(m+1)$ counters. We create labels for the counters of M', the first $k-1$ counters are called $D_1, \ldots, D_{k-1}$, and the remaining are $C_0, \ldots, C_{2m+1}$ and C. To start, M' nondeterministically guesses a counter j and a value x_j that is crossed at least $m+1$ times in some accepting computation. Then, M' stores x_j in $2m+1$ identical

copies on $C_1, \ldots, C_{2m+1}$ (by adding in parallel to them all). Then M' simulates M using the $D_1, \ldots, D_{k-1}$ to simulate all counters other than j, but initially uses C_0 to simulate counter j. But at $m+1$ nondeterministically guessed spots after a cross of the simulation from $i := 0$ to m, M' reduces counters C_{2i} and C_{2i+1} by the same amount and then they will not be used again (they must therefore both be zero at the same time to accept and, therefore, both contain the same value beforehand). After the $m+1$st time, it uses C for counter i and accepts if M does.

It is evident that M' will accept all words where some accepting computation that crosses some counter value at least $m+1$ times. Thus, we can use the decidable non-emptiness property of PBLIND [2] to decide the property of interest. □

Despite $\mathsf{PBLIND}_{\mathrm{FIN}}$ being equal to $\mathsf{COUNTER}_{\mathrm{FIN}}$, we get the following interesting contrast to Proposition 5.

Proposition 6. *It is undecidable, given a deterministic* COUNTER(2) *machine M and $m \geq 0$, whether every accepting computation is m-crossing.*

Proof. It is known that it is undecidable, given a deterministic COUNTER(2) machine M with an empty input and initially zero counters, whether M will halt [9]. Given such a machine M, we construct a deterministic COUNTER(2) machine M' (with right end marker) which when given any unary a^d (for some $d \geq 1$), first crosses zero on the first counter $m+1$ times and then simulates M, and accepts a^d if M halts.

If M does not halt, then M' accepts the empty language, which is obviously m-crossing (for any $m \geq 0$). If M halts, then the first counter of M' makes at least $m+1$ crosses in every accepting computation.

It follows that M' is m-crossing if and only if M does not halt, which is undecidable. □

This proposition cannot be strengthened to hold for COUNTER(1) as we show below.

Proposition 7. *It is decidable, given a* COUNTER(1) *machine M and $m \geq 0$, whether every accepting computation is m-crossing.*

Proof. The proof uses the known result that the emptiness problem for one-way nondeterministic pushdown automata with reversal-bounded counters is decidable [8]. Actually, in our proof, the pushdown will just be an unrestricted counter; call this class of machines NCCM.

Given a COUNTER(1) machine M and $m \geq 0$, we construct an NCCM M' with one unrestricted counter called C, and $2(m+1)$ 1-reversal-bounded counters called $C_1, \ldots, C_{m+1}, D_1, \ldots, D_{m+1}$. Then M' operates as follows, when given input x:

First M', on input w, without reading the input, increments the 1-reversal-bounded counters $C_1, \ldots, C_{m+1}$ to the same nondeterministically guessed value d (by adding 1 to them all in parallel). Next, M' simulates the computation of M (which has one unrestricted counter) on w using counter C. At $m+1$ nondeterministically guessed points after an upward cross in the computation from $i := 1$ to $m+1$, M guesses that the contents in C is equal to d by checking that it contains the same value as in C_i by decreasing C and C_i in parallel to check that they reach zero at the same time, while also increasing counter D_i to temporarily save the value of C. If they are equal, it restores the value in C using D_i. The process is iterated until M' has checked that d occurs after a cross at least $m+1$ times in M. Then M' accepts w if M accepts. Clearly, M is m-crossing if and only $L(M')$ is empty, which is decidable. □

For the next result, we will need the following lemma, which is almost the same as the standard proof of undecidability of whether a 1-reversal-bounded NCM(1) (a family we denote by NCM(1, 1)) accepts Σ^* [10]. NCM(1, 1) is known to be properly contained in COUNTER(1), and it is the same family as 1-crossing NCM(1).

Lemma 1. *It is undecidable, given an* NCM(1, 1) M *over input alphabet* Σ*, whether* $L(M) = \Sigma^*$*. Moreover, in the proof of this undecidability, it is the case that if* $L(M) \neq \Sigma^*$*, then* $\Sigma^* - L(M)$ *is infinite.*

Proof. This follows from the standard proof of accepting all strings that do not represent the valid sequence of IDs of a deterministic Turing machine M halting computation (on an initially blank tape) [10], which can be accepted by an NCM(1, 1). The idea is to modify M so that

when it halts on blank tape, it continues the computation by just re-entering the halting state. Then the number of valid sequences of IDs of the deterministic Turing machine is infinite. □

Proposition 8. *The following problems are undecidable:*

1. *Given a* PBLIND(1) *(respectively, an* NCM(1)*)* M *and* $m \geq 1$*, is* $L_m(M) = L(M)$*?*
2. *Given a* PBLIND(1) *(respectively, an* NCM(1)*)* M *and* $m \geq 1$*, does every input that is accepted have an accepting computation that is* m*-crossing?*
3. *Given a* PBLIND(1) *(respectively, a* COUNTER(1) M*, is there an* $m \geq 1$ *such that* $L_m(M) = L(M)$*?*

Proof. To prove Part 1, given an $M \in \mathsf{NCM}(1,1)$ (see Lemma 1), build a PBLIND(1) (or an NCM(1)) M' so that it does one of two things nondeterministically when given input x:

1. M' simulates M on x with 1 crossing.
2. M' first crosses the number 0 $(m+1)$ times (by adding 1 and subtracting 1 that many times) and then reads x and accepts.

Then $L_m(M') = L(M)$ and $L(M') = \Sigma^*$. Hence, $L_m(M') = L(M')$ if and only if $L(M) = \Sigma^*$, which is undecidable.

Part 2 is just a restatement of Part 1.

Part 3 is similar to part 1 setting $m = 1$. We alter step 2, however, so that after crossing 0 $(m+1)$ times, it then crosses 0 up and down for every symbol in x and accepts. So, if $L(M) = \Sigma^*$, then $L(M') = L_1(M')$. If $L(M) \neq \Sigma^*$, by Lemma 1, there is an infinite number of strings in Σ^* that are not in $L(M)$ and these strings are accepted by M' with an unbounded number of crossings (by the construction of M' adding 1 crossing for each symbol of x), and hence there is no $m \geq 1$ such that $L_m(M') = L(M')$. □

We note that Part 2 of Proposition 8 contrasts Proposition 5 as Proposition 5 shows that it is decidable, given a PBLIND machine and

m whether every accepting computation is m-crossing, whereas Proposition 8 says that it is undecidable if every word has an accepting computation that is m-crossing.

Proposition 8 also holds for COUNTER(2), in fact, it holds for deterministic COUNTER(2):

Proposition 9. *The following problems are undecidable:*

1. *Given a deterministic* COUNTER(2) M *and* $m \geq 0$, *is* $L_m(M) = L(M)$*?*
2. *Given a deterministic* COUNTER(2) M *and* $m \geq 0$, *does every input that is accepted have an accepting computation that is* m*-crossing?*
3. *Given a deterministic* COUNTER(2) M, *is there an* $m \geq 0$ *such that* $L_m(M) = L(M)$*?*

Proof. Again, we use the fact that it is undecidable, given a deterministic COUNTER(2) machine M with empty input and initially zero counters, whether it will eventually halt [9]. We construct a deterministic COUNTER(2) M' with input alphabet $\{a\}$. Let $w \in \{a\}^*$ be an input to M'.

- If $w = \lambda$, M' rejects without using the counters.
- If $w = a$, M' accepts without using the counters.
- If $w = aa^k$ for some $k \geq 1$, M' first simulates M after reading the first a and if M halts, the first counter of M' crosses 0 up and down for every a in a^k and accepts.

If M does not halt, then M' will only accept the language $\{a\}$ with 0 crossings. Then $L_m(M') = L(M')$ for every $m \geq 0$. If M halts, then every string of the form $aa^k, k \geq 1$ will be accepted by M' with the first counter crossing 0 a number of times that grows with k; hence M' is not m-crossing for any m, i.e., $L_m(M') \neq L(M')$. It follows that $L_m(M') = L(M')$ if and only if M does not halt, which is undecidable. Parts 2 and 3 also follow. □

To note finally, that for deterministic COUNTER(1) where each accepted word has a unique accepting computation, it follows from Proposition 7 that it is decidable, given a deterministic COUNTER(1) machine and $m \geq 0$, whether every input that is accepted has an accepting computation that is m-crossing.

5 Conclusions and Future Directions

We introduced a new notion of the m-crossing language accepted by a k-counter machine, or a k-counter partially-blind machine. We establish some basic properties, and that for both multicounter and partially-blind multicounter machines, their m-crossing languages are contained in their $(m+1)$-crossing languages. We also showed that finite-crossing multicounter languages are equal to finite-crossing partially-blind multicounter languages and, therefore, the ability to test whether counters are empty or not does increase their power.

However, we have just scratched the surface of this new topic. Indeed, we have not been able to establish any inclusions (or incomparability) between finite-crossing partially-blind languages, and partially-blind languages in general. It is also open whether the emptiness and membership problems are decidable for these languages.

References

[1] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley, 1979.

[2] S. Greibach, "Remarks on blind and partially blind one-way multicounter machines," *Theoretical Computer Science*, vol. 7, pp. 311–324, 1978.

[3] S. R. Kosaraju, "Decidability of reachability in vector addition systems," in *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '82, 1982, pp. 267–281.

[4] E. W. Mayr, "An algorithm for the general Petri net reachability problem," *SIAM Journal on Computing*, vol. 13, no. 3, pp. 441–460, 1984.

[5] O. H. Ibarra and I. McQuillan, "On the containment problem for deterministic multicounter machine models," in *Lecture Notes in Computer Science*, ser. Proceedings of the 21st International Symposium on Automated Technology for Verification and Analysis, ATVA 2023, vol. 14215, 2023, pp. 74–94.

[6] S. A. Greibach, "One way finite visit automata," *Theoretical Computer Science*, vol. 6, pp. 175–221, 1978.

[7] M. Harrison, *Introduction to Formal Language Theory*, ser. Addison-Wesley series in computer science. Addison-Wesley Pub. Co., 1978.

[8] O. H. Ibarra, "Reversal-bounded multicounter machines and their decision problems," *J. ACM*, vol. 25, no. 1, pp. 116–133, 1978.

[9] M. L. Minsky, *Computation: Finite and Infinite Machines*. Englewood Cliffs, N.J.: Prentice Hall, 1967.

[10] B. S. Baker and R. V. Book, "Reversal-bounded multipushdown machines," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 315–332, 1974.

Oscar Ibarra, Ian McQuillan

Received September 21, 2024
Revised October 22, 2024
Accepted October 23, 2024

Oscar Ibarra
Department of Computer Science, University of California,
Santa Barbara, CA 93106, USA
E–mail: ibarra@cs.ucsb.edu

Ian McQuillan
Department of Computer Science, University of Saskatchewan
Saskatoon, SK S7N 5A9, Canada
E–mail: mcquillan@cs.usask.ca