Input-Position-Restricted Models of Language Acceptors*

Oscar H. Ibarra and Ian McQuillan

Abstract Machines of various types are studied with some restriction on the moves that can be made either on or before the end of the input. For example, for machine models such as deterministic reversal-bounded multicounter machines, one restriction is the class of all machines that do not subtract from any counters before the end the input. Similar restrictions are defined on different combinations of stores with many machine models (nondeterministic and deterministic), and their families studied.

1 Introduction

In this paper, various one-way machine models are studied where there is a restriction on the instructions that are permitted on or before hitting the end of the one-way input (on a right input end-marker $\lhd$). For example, one such model is a pushdown automaton that cannot pop until hitting the end of the input, or a pushdown automaton that cannot pop until hitting the end of the input and also cannot push after hitting the end of the input. A preliminary investigation started regarding such concepts on reversal-bounded multicounter machines (NCM) in [13]. This model consists of an NFA augmented by some number of reversal-bounded counters (each counter stores a non-negative integer that can be increased or decreased by one, or zero, and tested for being zero or non-zero, and there is a bound on the number of changes between non-decreasing and non-increasing), and DCM is the same type of machine that is deterministic. For example, in [13], it was shown that every NCM can be converted to another that does not decrease before hitting the end-marker. For the deterministic case however, this is not true, as there is a DCM with only one counter that cannot be accepted by any DCM machine that does not decrease before hitting the end of the input. Furthermore, the lan-

Oscar H. 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

* Published in In: Adamatzky A. (eds) Reversibility and Universality. Emergence, Complexity and Computation, vol 30. Springer, Cham. The final authenticated version is available online at https://doi.org/10.1007/978-3-319-73216-9_17.

guages accepted by this same restricted model of DCM were shown to coincide with the languages accepted by deterministic Parikh automata [3].

A key tool used in this paper for studying many of these machine models with input-position-based restrictions is the *store language* of a machine. The store language of a machine M is an encoding of the set of all configurations (state plus concatenated store contents) that can appear in an accepting computation. For example, the store language of a pushdown automaton M is the set of all words $q\gamma$ where there is an accepting computation of M that passes through state q with pushdown contents γ. It is known that the store language of every pushdown automaton is in fact a regular language [1], and the store language of the more general stack automata (similar to pushdown automata with the additional ability to read the contents of the pushdown in read-only mode) are all regular languages [2]. The store languages of several other models were also recently studied in [15, 14]. For example, the store languages of reversal-bounded queue automata (this is an NFA augmented by a queue data structure with a bound on the number of switches between enqueuing and dequeueing) and the store language of r-flip pushdown automata (pushdown automata with the additional ability to reverse their pushdown contents at most r times) are also all regular [15].

Here, at first nondeterministic machines are investigated. The restriction that a store cannot decrease in size until the end-marker is a major focus. When restricting the pushdown of a nondeterministic pushdown automaton in this fashion, the machines coincide with the regular languages, and similarly for reversal-bounded queue automata and r-flip nondeterministic pushdown automata. When augmenting any of these three models with reversal-bounded counters, and the decreasing property is only applied to the pushdown or queue, then these machines coincide with NCM. For any of these models, also enforcing that the reversal-bounded counters cannot decrease until the end-marker does not even reduce the capacity. For deterministic machines, the situation is more complicated. In particular, DCM and DCM augmented by an unrestricted pushdown (DPCM) are studied with the decreasing restriction placed on the counters and pushdown separately. Several witnesses are found to separate deterministic classes. Also, a bridging technique is created to determine that languages are not in DCM and DPCM from the decreasing-restricted restriction.

2 Preliminaries

We assume an introductory background in the area of formal language and automata theory; see e.g. [11].

Let $\mathbb{N}_0$ be the set of non-negative integers. Given a set X and $k \in \mathbb{N}_0$, let $[X]^k$ be the set of k-tuples over X.

An *alphabet* is a finite set of symbols, with Σ^* being the set of all words over Σ^*. The empty word is denoted by λ. A *language* is any $L \subseteq \Sigma^*$. Given a word w, w^R is the word obtained by reversing the letters of w, which can be extended to languages L^R in the natural way, and also to families of languages. Given $w \in \Sigma^*$, then u is a prefix (resp. suffix) of w if $w = ux, x \in \Sigma^*$ (resp. $w = xu, x \in \Sigma^*$). The length of w is denoted by $|w|$, and the number of a's in w, $a \in \Sigma$, is $|w|_a$.

A one-way nondeterministic k-pushdown automaton is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the input alphabet, Γ is the pushdown alphabet, containing the bottom-of-stack marker Z_0, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a finite relation from $Q \times (\Sigma \cup \{\lambda, \lhd\})^* \times [\Gamma]^k$ to $Q \times [\Gamma^*]^k$, where $\lhd$ is the right input end-marker. A configuration of M is a tuple $(q, w, \gamma_1, \ldots, \gamma_k)$, where $q \in Q$ is the current state, $w \in \Sigma^* \lhd \cup \{\lambda\}$ is the remaining input, and $\gamma_i \in Z_0(\Gamma - \{Z_0\})^*$ is the contents of the i'th pushdown, for $1 \leq i \leq k$. The derivation relation between configurations, $\vdash_M$, is such that $(q, aw, v_1b_1, \ldots, v_kb_k) \vdash_M (p, w, v_1u_1, \ldots, v_ku_k)$, where $(p, u_1, \ldots, u_k) \in \delta(q, a, b_1, \ldots, b_k)$. Then $\vdash_M^*$ is the reflexive and transitive closure of $\vdash_M$.

The language accepted by M,

$$L(M) = \{w \mid (q_0, w\lhd, Z_0, \ldots, Z_0) \vdash_M^* (q_f, \lambda, \gamma_1, \ldots, \gamma_k), q_f \in F, \gamma_i \in \Gamma^*, 1 \le i \le k\}.$$

Furthermore, the store language of M,

$$S(M) = \{q\gamma_1 \cdots \gamma_k \mid (q_0, w\lhd, Z_0, \ldots, Z_0) \vdash_M^* (q, v, \gamma_1, \ldots, \gamma_k) \vdash_M^* (q_f, \lambda, \gamma_1', \ldots, \gamma_k'), q_f \in F\}.$$

Also, M is deterministic if $|\delta(q, a, b_1, \ldots, b_k) \cup \delta(q, \lambda, b_1, \ldots, b_k)| \le 1$, for all $q \in Q, a \in \Sigma \cup \{\lhd\}, b_i \in \Gamma, 1 \le i \le k$.

Instead of a pushdown, other data structures, such as a queue can be attached to such a machine.

Often in the literature, one-way machines are defined without the right input end-marker $\lhd$. For nondeterministic machines, $\lhd$ is not necessary as machines can guess when they have hit the end of the input. And for some types of deterministic machines, such as DPDAs, it is not needed [8], but for others, such as DCM, this is not the case (see [13] for further discussion). For consistency, we will define all machines using $\lhd$. This is also useful as we will study restrictions of machines based on whether $\lhd$ has been scanned yet or not.

A counter is a pushdown stack that uses only one stack symbol, in addition to a distinguished bottom-of-stack symbol, Z_0, which is never altered. It is known that deterministic machines with only two pushdowns that are both counters have the same power as Turing machines [11]. When there is only one pushdown, this is the well-known nondeterministic pushdown automaton, denoted by NPDA (DPDA for the deterministic variant). Also, a counter is l-reversal-bounded if the machine makes at most l changes between non-decreasing and non-increasing (and vice versa) on the counter. Nondeterministic (resp. deterministic) finite automata with some number of reversal-bounded counters are denoted by NCM (resp. DCM). These machines have been extensively investigated in the literature, e.g. [17], and they are closed under intersection and have a decidable emptiness problem (resp. containment problem for deterministic machines). Furthermore, machines with one unrestricted pushdown plus some number of reversal-bounded counters also have a decidable emptiness problem, and are denoted by NPCM (DPCM for the deterministic variant).

By a slight abuse of notation, we will denote each family of machines synonymously with the family of languages they accept. Therefore, DCM will denote both the reversal-bounded counter machines, and also the family of languages they accept.

3 Restrictions on/before the end of input

In this section, we will restrict the operation of different classes of machines so that any instruction that reduces the size of the store can only occur when the input tape has read the right input end-marker $\lhd$. Notice that this can be studied separately for each store. For example, NPCMs could be studied where the pushdown cannot decrease in size before the end-marker, and separately, they could be studied where the counters cannot be decreased before hitting the end-marker, and lastly, they can be studied where both stores have this restriction. This restriction will be denoted on the family by placing a "bar" on top of the letter denoting the store with this restriction. For example, $\mathsf{DP\overline{C}M}$ are machines where the pushdown is unrestricted, but the counters cannot decrease before hitting the end-marker, and $\mathsf{D\overline{PC}M}$ is where the restriction is on both the counters and the pushdown.

In addition, a stronger notion whereby no decreasing of storage before the end-marker, plus no increasing once the end-marker is hit is studied, and for this notion, two bars are placed over the appropriate store letter, such as $\mathsf{DP\overline{\overline{C}}M}$.

We first recall a result from [13], where the families $\mathsf{N\overline{C}M}$ and $\mathsf{D\overline{C}M}$ were introduced (eNCM and eDCM was the notation used in [13]). For NCM, it was found that restricting the counters to not decrease until the end-marker did not change the family accepted. It was also found that $\mathsf{D\overline{C}M}$ coincides with the family of languages accepted by deterministic Parikh automata [3], and that this family is a strict subset of DCM. We will extend this proof by adding in $\mathsf{N\overline{\overline{C}}M}$.

Proposition 1. $\mathsf{D\overline{C}M} \subsetneq \mathsf{DCM} \subsetneq \mathsf{NCM} = \mathsf{N\overline{C}M} = \mathsf{N\overline{\overline{C}}M}$.

Proof. All but the last equality were shown in [13]. But we will briefly describe the construction of $\mathsf{NCM} = \mathsf{N\overline{C}M}$ here for use later in the paper.

Let M be a k counter NCM over Σ, with the counters labelled by $c_1,\dots,c_k$. It can be assumed without loss of generality that all counters are 1-reversal-bounded [17].

Then, construct a $\mathsf{N\overline{C}M}$ M' with counters labelled by $c_1,d_1,\dots,c_k,d_k$. On input $w \in \Sigma^*$ followed by the end-marker, M' simulates M exactly using counters $c_1,\dots,c_k$ so long as they are non-decreasing. If a counter c_i attempts to decrease before hitting the end-marker (nothing is required to be changed on the end-marker), counter d_i is instead increased and thus records the number of decrements of c_i in M. At some nondeterministically guessed point after a counter decrease, M' verifies that the contents of d_i and c_i are equal, and then continues the simulation, simulating transitions on the counter being zero. Then $L(M') = L(M)$, and M' does not decrease any counter before hitting the end-marker.

This proof can be modified slightly to create a machine M'' in $\mathsf{N\overline{\overline{C}}M}$. M' uses a third set of counters $e_1,\dots,e_k$. Then M'' similarly simulates M exactly using $c_1,\dots,c_k$ as long as they are non-decreasing and before the end-marker. Then, at some nondeterministically guessed spot before the end-marker, on λ-transitions, for all counters c_i that have not already started decrementing in M, M'' increases e_i and d_i to the same arbitrary number. Then, M'' verifies that the next character is the end-marker. On the end-marker, for every increase of c_i in M, M'' instead decreases e_i, verifying that the simulated increasing ends when counter e_i hits zero, and then when simulating the decreases of c_i in M, it decreases d_i until empty, then decreases c_i until empty. Essentially then, M' guesses right before it reaches $\lhd$, and does all the remaining additions nondeterministically instead. Then, it decreases instead of increases at the end-marker. But it therefore needs two identical copies, d_i and e_i, one to simulate the increases of M and one to simulate decreases of M. $\square$

Next, we study these restrictions on standard NPDAs. Both restrictions induce a large collapse in contrast to NCM.

Proposition 2. $\mathsf{N\overline{P}DA} = \mathsf{N\overline{\overline{P}}DA} = \mathsf{REG}$.

Proof. It is enough to show it for $\mathsf{N\overline{P}DA}$.

Let $M = (Q,\Sigma,\Gamma,\delta,q_0,F)$ be an $\mathsf{N\overline{P}DA}$. Assume without loss of generality that M empties the pushdown in every computation on the end-marker. Assume also that Q is partitioned into $Q^{\leftarrow}, Q^{\lhd}$, and $Q^{\rightarrow}$, whereby $Q^{\leftarrow}$ are all states that are used before the end-marker, $Q^{\lhd}$ are the states that can be used on the end-marker (which immediately read the end-marker), and then $Q^{\rightarrow}$ are all states that can be used after the end-marker has been read. Lastly, assume without loss of generality that all transitions before the end-marker (on states of $Q^{\leftarrow}$), either replace the top of the pushdown X with XY, where $X,Y \in \Gamma$, or replaces X with X. Indeed, we know that the pushdown does not decrease before the end-marker, it is easy to see that the machine only needs to push one symbol at a time, and also it never needs to replace the top symbol by storing the topmost symbol in the finite control and not pushing it until another symbol is pushed.

From [1], for each state $q \in Q$, the set $co\text{-}Acc(q) = \{\beta \in \Gamma^* \mid (q, v\lhd, \beta) \vdash_M^* (q_f, \lambda, p), q_f \in F\}$ is a regular language. Let $R = \bigcup_{q \in Q^{\lhd}} (co\text{-}Acc(q) \cdot q)$ (over the alphabet of $\Gamma \cup Q^{\lhd}$), which also must be regular. Let $M_R = (Q_R, \Gamma \cup Q^{\lhd}, \delta_R, q_0^R, F_R)$ be an NFA accepting R.

Next, let $M' = (Q', \Sigma, \delta', q_0', F')$ be an NFA with λ transitions with state set $Q' = (Q^{\leftarrow} \cup Q^{\lhd}) \times Q_R$, $q_0' = (q_0, q_0^R)$, $F' = Q^{\lhd} \times F_R$ that operates as follows. If M has a transition $(p, \gamma) \in \delta(q, a, X), q \in Q^{\leftarrow}, p \in Q^{\leftarrow} \cup Q^{\lhd}, a \in \Sigma \cup \{\lambda\}, X \in \Gamma$, then create:

$$\begin{aligned}
&(p, p_R) \in \delta'((q, q_R), a) \;\text{ if } \gamma = XY, \text{ and } p_R \in \delta_R(q_R, Y),\\
&(p, q_R) \in \delta'((q, q_R), a) \;\text{ if } \gamma = X.
\end{aligned}$$

Also, create $((q, p_R) \in \delta'((q, q_R), \lambda)$ if $q \in Q^{\lhd}$, and $p_R \in \delta_R(q_R, q)$.

Let $w \in L(M)$. Then

$$(q_0, w\lhd, Z_0) \vdash_M^* (q', \lhd, \beta) \vdash_M^* (q_f, \lambda, Z_0),$$

where $q' \in Q^{\lhd}, q_f \in F$. Then, $\beta \in co\text{-}Acc(q')$, and so $\beta q' \in L(M_R)$, and there exists $p \in F_R$ such that $p \in \hat{\delta}_R(q_0^R, \beta q')$. Then, by the construction, $(q', p) \in \hat{\delta}'((q_0, q_0^R), w)$, and $(q', p) \in Q^{\lhd} \times F_R = F'$, and hence $w \in L(M')$.

Let $w \in L(M')$. Then $(q', p) \in \hat{\delta}'((q_0, q_0^R), w)$, where $q' \in Q^{\lhd}, p \in F_R$. By the construction, $(q_0, w\lhd, Z_0) \vdash_M^* (q', \lhd, \beta)$, where $q' \in Q^{\lhd}, \beta q' \in L(M_R)$. Hence, there exists $q_f \in F$ such that $(q', \lhd, \beta) \vdash_M^* (q_f, \lambda, Z_0)$. Hence, $w \in L(M)$.

Then $L(M') = L(M)$. $\square$

Similarly, a reversal-bounded queue automaton, NQA, is an NFA augmented with a queue store, with a bound on the number of switches between enqueuing and dequeueing, and let NQCM be the same system augmented by reversal-bounded counters. Although it is known that queue automata without a reversal-bound on the queue has the same power as a Turing machine, it is known that both NQA and NQCM are more limited, and indeed only accept semilinear languages [10]. It was shown in [15] that the store languages of all NQA are regular languages, and the store languages of all NQCM are all in NCM.

Essentially the same proof of Proposition 2 can be used for NQA as well, ie. when enqueueing before the end-marker, verify in parallel that whatever would be enqueued is in the store language. Therefore,

Corollary 1. $\mathsf{N\overline{Q}A} = \mathsf{N\overline{\overline{Q}}A} = \mathsf{REG}$.

This same proof technique can be used for $\mathsf{N\overline{P}CM}$ and $\mathsf{N\overline{Q}CM}$, showing the resulting languages are all NCM languages. Indeed, take an input $\mathsf{N\overline{P}CM}$ M with k counters. The store language of every NPCM is an NCM language [14]. Let $M_s \in$ NCM with l counters be the store language of M. And in the store language of an NPCM, the word on the pushdown comes first, and then the counters (such as $xc_1^{i_1} \cdots c_k^{i_k}$ where x is the contents of the pushdown, and i_j is the contents of counter j). So, build an NCM M' machine accepting $L(M)$ with $k+l$ counters as follows: M' simulates M with k counters, but if M pushes y onto the pushdown, M' runs it through the store language in parallel using the other l counters. Then, when M' reaches the end of the input, it just needs to verify that the pushdown contents read this far, concatenated with the counter contents are in the store language. So, it subtracts from each counter from c_1 to c_k, while still simulating the machine accepting the store language. Hence,

Proposition 3. $\mathsf{N\overline{P}CM} = \mathsf{N\overline{\overline{P}}CM} = \mathsf{N\overline{Q}CM} = \mathsf{N\overline{\overline{Q}}CM} = \mathsf{NCM} = \mathsf{N\overline{C}M} = \mathsf{N\overline{\overline{C}}M}$.

Next, consider r-flip pushdown automata. These machines are similar to pushdown automata with the additional ability to "flip" the pushdown stack at most r times (more precisely, they flip everything above the bottom-of-stack marker Z_0, transforming pushdown contents $Z_0\gamma$, with γ over the pushdown alphabet, to $Z_0\gamma^R$). Let r-NPDA be this family, and let r-NPCM be the same type of machines augmented by reversal-bounded counters. In [15], it was shown that the store languages of all r-NPDA are regular, and in [14], it was shown that the store languages of r-NPCM are all in NCM. When restricting it to not decrease before the end-marker, these machines can therefore only push or flip before the end-marker, but not pop.

Proposition 4. *r-$\mathsf{N\overline{P}DA} = r$-$\mathsf{N\overline{\overline{P}}DA} = \mathsf{REG}$, and r-$\mathsf{N\overline{P}CM} = r$-$\mathsf{N\overline{\overline{P}}CM} = \mathsf{NCM}$.*

Proof. First, for r-$\mathsf{N\overline{P}DA}$, let M be such a machine with input alphabet Σ, and stack alphabet Γ. Then build a 2NFA (a two-way NFA [11] where there is a left and right end-marker on the input) M' over alphabet $\Sigma \cup \Gamma \cup \{\$_1,\ldots,\$_r\}$ such that M' reads $a \in \Gamma$ of M as input instead of pushing it, and also reads a new character $\$_i$ when flipping the pushdown for the ith time. When M' reaches the end of the input, it now needs to verify that the pushdown letters are a "representation" of a word in the store language of M. Let $Z_0 v_0 \$_1 \cdots \$_i v_i, v_j \in \Gamma^*, 0 \leq j \leq i$, be the sequence of letters read on the input from $\Gamma \cup \{\$_i \mid 1 \leq i \leq r\}$ (ignoring letters of Σ). Then, in M, the pushdown contents would be $x = Z_0(\cdots((v_0)^R v_1)^R \cdots v_{i-1})^R v_i$. By using the two-way input, M' can verify that x is in the store language of M since $S(M)$ is a regular language. Then, creating a homomorphism h that erases all letters not in Σ, and fixes all others, leaves $L(M) = h(L(M'))$, and the regular languages are closed under homomorphisms.

Similarly for r-$\mathsf{N\overline{P}CM}$, build a 2NCM (a two-way machine with reversal-bounded counters [9]) M' that simulates M while reading symbols of $\Gamma \cup \{\$_i \mid 1 \leq i \leq r\}$ but uses counters of M' to do so faithfully. And then, at the end of the input, it needs to verify that the representation of the pushdown letters is in the store language. For this, it uses the two-way input as above together with additional counters as the store language of M is in NCM. Further, M' makes a bounded number of turns on the input since r is fixed, and therefore M' can be converted to a one-way NCM [9]. Similarly again, since NCM is closed under homomorphism [17], the proposition follows. □

We can similarly study the increasing and decreasing restrictions when applied to the counters. Notice that in the proof of Proposition 1, this same proof would hold to show that $\mathsf{NPCM} = \mathsf{NP\overline{C}M} = \mathsf{NP\overline{\overline{C}}M}$ since the pushdown is not used. In the same way, for example, $\mathsf{N\overline{P}CM} = \mathsf{N\overline{P}\overline{C}M}$ since the counters are independent of the other stores. Similarly for all other types of stores. Hence:

Proposition 5. $\mathsf{NPCM} = \mathsf{NP\overline{C}M} = \mathsf{NP\overline{\overline{C}}M}$, *and* $\mathsf{N\overline{P}CM} = \mathsf{N\overline{P}\overline{C}M} = \mathsf{N\overline{P}\overline{\overline{C}}M} = \mathsf{NCM}$. *Similarly for* NQCM *and* r-NPCM.

Next, we will turn our attention to deterministic machines. First, a result on reversal is needed. Although closure of DCM under reversal has not been formally studied to the best of our knowledge, it follows relatively easily that DCM is not closed under reversal from a recent paper.

Proposition 6. DCM *is not closed under reversal. Hence,* 2DCM *that makes one turn on the input is strictly more powerful than* DCM.

Proof. Assume that DCM is closed under reversal. In [7], it was shown that DCM is closed under the prefix operator, but not closed under the suffix operator. Let $L \in \mathsf{DCM}$ such that the suffix closure of L is not in DCM. By assumption, and by the closure of DCM under prefix, $(\mathrm{pref}(L^R))^R \in \mathsf{DCM}$ (where pref is the prefix operator). But this is equal to the suffix closure of L, a contradiction.

From this it follows that 2DCM that makes one turn on the input is strictly more powerful than DCM. □

The above proposition is quite interesting as the previous candidate witness language conjectured to separate 2DCM from DCM was significantly more complex, being accepted by a 5-crossing 2DCM (ie. the boundary of each input cell is crossed at most five times, a more general notion than turns on the input) [16], and the proof that it is not in DCM did not appear in the text.

Next, it will be shown that $\mathsf{D\overline{C}M}$ is no more general than $\mathsf{D\overline{\overline{C}}M}$.

Lemma 1. $\mathsf{D\overline{C}M} = \mathsf{D\overline{\overline{C}}M}$.

Proof. Let $M \in \mathsf{D\overline{C}M}$ with k counters called $c_1, \ldots, c_k$, and with m states. First, on an input of size n, each counter can increase until it is at most $m \cdot n$ by the time the right input end-marker is hit, otherwise M enters an infinite loop and does not accept. Then at the end-marker, if one counter is increasing, another must be decreasing after m transitions, otherwise an infinite loop is entered. Then for every decrease, there is at most m increases of some other counter. Thus, other counters reach a value of at most $m^2 n$. Continuing across all counters, the most M can store in any counter of an accepting computation is $f(n) = m^k n$.

Then, a $3k$-counter $\mathsf{D\overline{\overline{C}}M}$ M' machine will be built accepting $L(M)$. M' simulates M using counters $c_1, \ldots, c_k$, but in parallel, M' increases counters d_i, e_i, for each $1 \leq i \leq k$, to $f(n)$ by the end of the input, which is possible by adding additional states. Then, at the right input end-marker, instead of increasing counter c_i say, it instead decreases counter d_i. Then, when M would start decreasing counter c_i, say M has increased c_i by x_i since hitting the end-marker. Then d_i holds $f(n) - x_i$ (and indeed, $f(n) \geq x_i$ in any accepting computation by the calculation of $f(n)$). At this point, M' subtracts both d_i and e_i in parallel until both are zero. Then e_i holds $f(n) - (f(n) - x_i) = x_i$. Then M' can continue to simulate c_i using counters e_i until empty, then c_i, as their combined length is the same as counter c_i in M at the point where counter c_i starts to decrease. □

Thus, every language by a deterministic Parikh automaton (equal to $\mathsf{D\overline{C}M}$) can be accepted by a DCM where the machine only adds until the end-marker, and then at the end-marker, only subtracts.

Next, the families of $\mathsf{D\overline{P}CM}$ and $\mathsf{D\overline{PC}M}$ will be studied.

Lemma 2. $\mathsf{DCM}^R \subseteq \mathsf{D\overline{PC}M} \subseteq \mathsf{D\overline{P}CM}$.

Proof. The second inclusion is immediate.

For the first, given $M \in \mathsf{DCM}$, a machine $M' \in \mathsf{D\overline{PC}M}$ can be built, that on input $x \in \Sigma^*$, pushes x on the pushdown, and then it simulates M on x^R by popping from the pushdown instead of reading from the input, while simulating the counters exactly (all counter operations are performed after the end-marker of M' has been reached). Thus, $L(M)^R \in \mathsf{D\overline{PC}M}$. □

From this, the following can be shown:

Proposition 7. $\mathsf{D\overline{\overline{C}}M} = \mathsf{D\overline{C}M} \subsetneq \mathsf{DCM} \subsetneq \mathsf{D\overline{P}CM}$.

Proof. The first equality is from Lemma 1, and the first strict inclusion follows from Proposition 1. The second inclusion follows trivially by not using the pushdown. Strictness follows since DCM is not closed under reversal by Proposition 6, and since $\mathsf{DCM}^R \subseteq \mathsf{D\overline{P}CM}$ by Lemma 2. □

Next, it will be shown that $\mathsf{D\overline{C}M}$ and its reverse are in $\mathsf{D\overline{PC}M}$.

Lemma 3. $\mathsf{D\overline{C}M} \cup \mathsf{D\overline{C}M}^R \subseteq \mathsf{D\overline{PC}M}$.

Proof. $\mathsf{D\overline{C}M} \subseteq \mathsf{D\overline{PC}M}$ follows by simulating the counters verbatim without using the pushdown. And $\mathsf{D\overline{C}M}^R$ follows from Lemma 2. □

Next, we will show that this containment is strict. A technique in [5] was used to find languages that could not be accepted by deterministic reversal-bounded multicounter machines, and also deterministic machines with a pushdown augmented by counters. Essentially, it was shown that if L is a DPCM, then there exists $w \in \Sigma^*$ such that $L \cap w\Sigma^*$ is in DPDA, and similarly if L is a DCM, then there exists $w \in \Sigma^*$ such that $L \cap w\Sigma^*$ is a regular language. Then this property can be used to find languages not in DCM or DPCM. However, a close reading of this paper shows that the definition of DPCM and DCM used in this paper does not have an end-marker on the right end of the input. However, at least for DCM, it is known that one-way deterministic machines with reversal-bounded counters, accept strictly less languages when an end-marker is not used (unlike deterministic pushdown automata) [13]. And, a careful reading of the proof technique used to find languages outside of DCM and DPCM, illustrates that the technique only works on machines without the end-marker (as defined in the paper). Here though, we are using the more general definition with an end-marker. The same technique though can be used to show that if $L \in \mathsf{DCM}$, then there exists $w \in \Sigma^*$ such that $L \cap w\Sigma^*$ is in $\mathsf{D\overline{C}M}$. Similarly with DPCM and $\mathsf{DP\overline{C}M}$. This can be used as a type of "bridge" to show languages are not in DCM. And indeed, $\mathsf{D\overline{C}M}$ coincides with deterministic Parikh automata where witness languages are known [4] and can be used with this property. This will be shown next.

First, a definition is needed. Let $M \in \mathsf{DPCM}$ (resp. DCM) with k 1-reversal-bounded counters (without loss of generality [6]). For an integer $1 \leq i \leq k$, then w is i-decreasing if, while M is reading w, then the ith counter is decreased before reading the right end-marker.

The key here is that, if there is an i-decreasing word w, then for all $y \in \Sigma^*$, then counter i decreases on wy as well. If instead a word w decreases counter i on the end-marker, then there is no guarantee that wy decreases for all $y \in \Sigma^*$ before the end-marker. The first lemma is immediate.

Lemma 4. *Let $M \in \mathsf{DPCM}$ (resp. DCM) with k-counters. If there exists $1 \leq i \leq k$ such that no word in $L(M)$ is i-decreasing, then counter i only decreases on the end-marker in an accepting computation, and another machine M' of the same type can be created where all transitions that decrease counter i before the end-marker are removed, and $L(M) = L(M')$.*

Lemma 5. *Let $M \in \mathsf{DPCM}$ (resp. DCM) with k counters and $L(M) \neq \emptyset$, such that, by Lemma 4, for all counters $c_1, \ldots, c_l$ ($l \leq k$) that decrease before the end-marker, there is some $w \in L(M)$ that is i-decreasing, for each $1 \leq i \leq l$. Then, for each such w, there is a machine of the same type accepting $L(M) \cap w\Sigma^* \neq \emptyset$ that has at most $l-1$ counters that decrease before the end-marker.*

Proof. By applying Lemma 4 on all counters i whereby there is no i-decreasing word, the only counters that can decrease before the end-marker are those that can do so on at least one word in the language, in an accepting computation. Let i be such a counter, and $w \in L(M)$ be an i-decreasing word. Then, build a DPCM M' that simulates M but enforces using the states that the input must start with w. Also, it does not need to include counter i as this counter has already started decreasing for any $L(M) \cap w\Sigma^*$ in M by the time w is read, and therefore, the counter can be stored in the finite control. Therefore, $L(M') \neq \emptyset$. By another application of Lemma 4, another M'' can be created whereby at most $l-1$ counters can decrease before the end-marker. □

By applying Lemma 5 iteratively, the following is true:

Proposition 8. *Let $L \in \mathsf{DPCM}$ (resp. DCM) be non-empty. Then there exists $w \in \Sigma^*$ such that $L \cap w\Sigma^*$ is a non-empty $\mathsf{DP\overline{C}M}$ (resp. $\mathsf{D\overline{C}M}$).*

This same technique also clearly works for all other deterministic machine models augmented by reversal-bounded counters considered in this paper. Then this can serve as a "bridge" where witnesses known for versions of machines with counters that only decrease on the end-marker can possibly be used to show a witness in the more general model where the counter restriction does not occur.

Furthermore, it is known that $\mathsf{D\overline{C}M}$ coincides with deterministic Parikh automata [13]. And, it is known that

$$L = \{v \in \{a,b\}^* \mid v[|v|_a] = b\},$$

where $v[j]$ is the jth letter of v, cannot be accepted by deterministic Parikh automata [4]. Assume by way of contradiction that $L \in \mathsf{DCM}$. Then, by Proposition 8, there exists $w \in \Sigma^*$ such that $L \cap w\Sigma^* \in \mathsf{D\overline{C}M}$. Let w be such a word, let i be the length of w, and let j be the number of a's in w. Then $L \cap w\Sigma^* = \{wv \in \{a,b\}^* \mid (wv)[|wv|_a] = b\} \in \mathsf{D\overline{C}M}$. Since $\mathsf{D\overline{C}M}$ is closed under left quotient with a fixed word, it follows that $L' = \{v \in \{a,b\}^* \mid (wv)[|v|_a + j] = b\} \in \mathsf{D\overline{C}M}$. Let $x = a^{i-j}$. Let $L'' = (L')(x)^{-1} = \{v \in \{a,b\}^* \mid (wvx)[|v|_a + j + i - j] = b\} = \{v \in \{a,b\}^* \mid (wvx)[|v|_a + i] = b\} = \{v \in \{a,b\}^* \mid v[|v|_a] = b\}$. But $\mathsf{D\overline{C}M}$ is closed under right quotient with words [13], and $L'' = L$, a contradiction.

Lemma 6. $L = \{v \in \{a,b\}^* \mid v[|v|_a] = b\} \notin \mathsf{DCM}$.

Next, we will show that $L \notin \mathsf{DCM}^R$. This is equivalent to showing that $L^R \notin \mathsf{DCM}$. Then $L^R = \{v \in \{a,b\}^* \mid v[|v| - |v|_a + 1] = b\} \notin \mathsf{DCM}$. Assume, by contradiction that $L^R \in \mathsf{DCM}$. Given $v \in L^R$, notice that $|v| - |v|_a$ is equal to $|v|_b$. Therefore, $L^R = \{v \in \{a,b\} \mid v[|v|_b + 1] = b\}$. But, this language can also be shown to not be accepted by a deterministic Parikh automaton, similar to the proof that L cannot in [4], a contradiction. Hence:

Proposition 9. *There exists* $L \in \mathsf{D\overline{PC}M}$ *such that* $L \notin (\mathsf{DCM} \cup \mathsf{DCM}^R)$.

Proof. It has been shown already that L above is not in $\mathsf{DCM} \cup \mathsf{DCM}^R$.

Also, $L \in \mathsf{D\overline{PC}M}$, as a $\mathsf{D\overline{PC}M}$ M can be built with 2 counters c_1, c_2, that on input v, pushes v to the pushdown while in parallel, recording $|v|_a$ in counter c_1, and recording $|v|$ in counter c_2. Then at the end of the input, M subtracts the value of c_1 from c_2, so that c_2 now contains $|v| - |v|_a$. Then, M pops $|v| - |v|_a$ characters from the pushdown, and then verifies that the next character on the pushdown is a b, which then has the effect of verifying that position $|v|_a$ of v contains a b. $\square$

Corollary 2. $\mathsf{D\overline{C}M} \cup \mathsf{D\overline{C}M}^R \subsetneq \mathsf{D\overline{PC}M}$.

It is still open as to whether DCM is a subset of $\mathsf{D\overline{PC}M}$, and whether $\mathsf{D\overline{PC}M}$ is a strict subset of $\mathsf{D\overline{P}CM}$. Though, the language $L = \{c^i \$ w a^j b^j v \mid w, v \in \{a,b\}^*, j > 0, |w| = i\}$ is in DCM and $\mathsf{D\overline{P}CM}$. But we conjecture that L is not in $\mathsf{D\overline{PC}M}$, which would resolve both open problems. Even though L can be accepted by a DCM with one counter and three reversals, we also conjecture that all languages accepted by DCM with one counter and one reversal are in $\mathsf{D\overline{PC}M}$.

4 Restrictions when reading/not reading input letters

In this section, we generalize the concept of machines that can only decrease the store on the end-marker. For example, in an $\mathsf{N\overline{P}CM}$, the first stack reversal only occurs on the end-marker. Here, we will create a more general model that restricts what can happen when non-λ transitions are used on the input.

Definition 1. An sNPCM M is an NPCM with the restriction that the pushdown stack can only pop on a λ transition.

An sNPCM is an slNPCM if all transitions that keep the same size of pushdown (ie. the top of the pushdown symbol X is replaced with a symbol Y with potentially $X = Y$) are λ transitions.

For both types, M is reversal-bounded if the pushdown makes at most k alternations between non-increasing and non-decreasing the size of its pushdown, for some odd k.

Definition 2. An NPCM M is in simple normal form if at every step, M can only do one of the following:

1. reads an input symbol and pushes exactly one symbol on the stack,
2. reads λ and pops one symbol (i.e. the top symbol) from the stack,
3. reads λ and does not change the stack.

Note that for any machine in simple normal form, any transition that does not change the contents of the pushdown must be a λ transition. Also, note that a simple normal form NPCM machine is an slNPCM.

Lemma 7. *An* NPCM *M over Σ can be converted to a sl*NPCM *M' over $\Sigma \cup \{\#\}$ in simple normal form such that $L(M) = h(L(M'))$, where h is a homomorphism that erases # and fixes all letters of Σ.*

Proof. First, from M with pushdown alphabet Γ, create an intermediate M_1 as follows: For all transitions that replaces A on the stack with $B\gamma$, $A,B \in \Gamma$, replace this with transitions that replace A with B, then pushes each symbol of γ, one at a time. Then, all transitions where a letter is replaced with some $\gamma \in \Gamma^*$ on the pushdown has $|\gamma| \leq 2$.

Next, from M_1, create M_2 such that the only transitions that replaces A with B on the stack, $A,B \in \Gamma$ satisfy $A = B$, and are λ transitions. Indeed, M_2 simulates M_1, but for all transitions that replaces A with B on the top of the pushdown, M_2 instead pushes a primed symbol B' (leaving A on the stack). Then, when eventually decreasing, if a primed symbol is seen, M_2 removes all primed symbols plus one more, and continues the simulation as if the topmost primed symbol is the top of the pushdown. Next, if there is a transition that reads a letter and pops A, M_2 instead pushes A' on the letter, then on a λ, pops A' then pops A

Lastly, let # be a new input symbol. From M_2, create M' such that, if a transition pushes on a λ transition, it instead reads #.

Then M' is in simple normal form, $L(M) = h(L(M'))$, where h is a homomorphism that erases # and leaves the other symbols unchanged. □

Notice that the normal form NPCM can have a non-reversal-bounded pushdown even if the original machine has a reversal-bounded pushdown. Since NPCM is closed under homomorphism, the following is obtained:

Corollary 3. NPCM $=$ slNPCM $=$ sNPCM.

We will show that reversal-bounded slNPCMs are equivalent to NCMs. We will need two lemmas first.

Lemma 8. *Every reversal-bounded* NPCM *M in simple normal form can be converted to $M' \in$* NCM *such that $L(M) = L(M')$.*

Proof. Let M be an NPCM which makes at most k reversals on the stack for some k. Since the family of NCM languages is closed under union, it is sufficient to assume that M makes exactly k-reversals for some odd k.

We will show that we can construct a finite-crossing 2NCM M' such that $L(M) = h(L(M'))$ for some homomorphism h. The result would then follow, since finite-crossing 2NCMs are equivalent to NCMs [9], and by closure of NCM under homomorphism [17].

We illustrate the construction for $k = 3$. Thus, M makes exactly 3 reversals: pushing, popping, pushing, and popping. The input w to M' has two "tracks":

- Track 1 contains an encoding of the input x to M.
- Track 2 contains the string which represents the entire string that M pushes on its stack (from the start to accepting) where the positions when the first popping started and ended and when the second popping started and ended are marked.

Let P_1, P_2, E_1, E_2 be new symbols not in the input alphabet and stack alphabet of M. The input to M' would have two tracks. The first track would look like:

$x_1 E_2 x_2 E_1 x_3 P_1 x_4 P_2$

The second track would represent contents of the stack:

$y_1 E_2 y_2 E_1 y_3 P_1 y_4 P_2$

where $|x_i| = |y_i|$ for $1 \le i \le 4$ (so, each symbol of each x_i is in the same position of track 1 as the corresponding symbol of y_i on track 2). The 2-track input w to M' indicates that the input to M is $x = x_1 x_2 x_3 x_4$, and M performs the following processes:

1. M reads input segments $x_1 x_2 x_3$ while pushing $y_1 y_2 y_3$ on the stack.
2. Then, M pops the stack content y_3 on λ, leaving $y_1 y_2$ on the stack. (These are indicated by the markers P_1 and E_1.)
3. Then M reads input segment x_4 while pushing y_4 on the stack.
4. Finally, on λ, M pops the stack content y_4 then y_2.

The finite-crossing 2NCM M', when given the two-track input w (provided with left and right end-markers), simulates M and confirms the processes above. Then, M' makes three turns on the the input w. We also note that because the number of reversals the stack of M makes exactly is k, the NCM M' can remember the relative positions of P_1, P_2, E_1, E_2.

The 2-track input could have other forms, depending on the relative positions of the P_i's and the E_i's (e.g., E_1 P_1 E_2 P_2 is another such form), and the processes of when to read/push and start/end the popping is modified accordingly.

The above construction can easily be generalized for any k. The finite-crossing 2NCMs will need markers $P_1, \ldots, P_{(k+1)/2}$ and markers $E_1, \ldots, E_{(k+1)/2}$. □

Lemma 9. *Every reversal-bounded sl*NPCM *M can be converted to a reversal-bounded sl*NPCM *M' in simple normal form such that $h(L(M')) = L(M)$.*

Proof. First, from M with stack alphabet Γ, create M_1 as follows: For all transitions that replaces A on the stack with $B\gamma$, $A, B \in \Gamma, \gamma \in \Gamma^+$, replace this with transitions that replace A with B that does not read input, then pushes each symbol of γ, one at a time (reading the input on the first letter of γ). Then, all transitions where any $A \in \Gamma$ is replaced with γ on the pushdown has $|\gamma| \le 2$, and all transitions that keep the same size of pushdown are λ transitions. (By the definition of *sl*NPCMs, all those transitions of M that replace A with B are on λ, and all those that pop are on λ.)

Next, from M_1, we will create M_2 such that the only transitions that replaces A with B on the stack, $A, B \in \Gamma$ satisfy $A = B$. Then, for all transitions that replace A with B on the top of the stack, $A, B \in \Gamma, A \neq B$ (then these must be λ transitions), then M_2 simulates this as follows: M_1 keeps track of whether it is in "non-decreasing mode" or "non-increasing mode".

- If it is in "non-decreasing mode", then M_2 instead pushes a primed symbol B' (leaving A on the stack) and continues the simulation as if B were the top of the stack. Then, when eventually in decreasing mode, if a primed symbol is seen, M_2 removes all primed symbols plus one more, and continues the simulation as if the topmost primed symbol is the top of the pushdown.
- If the machine is "non-increasing mode", then M_2 leaves A on the top of the pushdown and remembers B in the state and continues the simulation as if B was the top of the pushdown. If this eventually pops B, then M_2 pops A and continues. If the simulation does pop B, but undergoes a reversal (so the simulation reverses), M_2 pops A on λ transition, pushes B on a λ transition, then continues the simulation.

It is clear then that $L(M_2) = L(M_1)$, and M_2 is reversal-bounded.

Lastly, introduce a new input letter #, and create M' from M_2. Then, M' simulates M_2 and for all transitions of M that pushes a symbol on a λ transition, instead it will read input #.

Then $L(M) = h(L(M'))$, where h is a homomorphism that erases # and leaves the other symbols unchanged. Furthermore M' is in simple normal form, and is reversal-bounded if M is reversal-bounded. □

From this lemma, and since every reversal-bounded NPCM in simple normal form is in NCM, and from closure of NCM under homomorphism [17], we can conclude:

Proposition 10. *Every reversal-bounded sl*NPCM *M can be converted to a* NCM *M' such that $L(M) = L(M')$.*

But this is not true with only *s*NPCM, as we see next.

Proposition 11. *Every reversal-bounded* NPCM *can be accepted by a reversal-bounded s*NPCM.

Proof. Let M be a reversal-bounded NPCM. Then, for every pop transition that moves right on input letter d, replace it with a transition that moves right on d that keeps the pushdown the same, followed by a transition that pops without reading any input letter. □

In this construction, the number of counters, and the reversal-bounds on the pushdown remain unchanged.

Then, if an *sl*NPDA is an *sl*NPCM without reversal-bounded counters:

Corollary 4. *If M is a reversal-bounded sl*NPDA*, then $L(M)$ is regular.*

Proof. This is true as the constructions in the proofs above do not introduce any new counters. □

We now show that Proposition 11 and Corollary 4 are not true when the pushdown stack is not reversal-bounded.

Proposition 12. *There is a non-regular language L such that:*

1. *L can be accepted by a* DPDA *in simple normal form (but the stack is not reversal-bounded).*
2. *L cannot be accepted by an* NCM.

Proof. Let $L = \{x\#x^R \mid x \in \{0,1\}^*\}$. Clearly, L is non-regular. For Part 1, we construct a DPDA M which operates as follows, when given input w. M reads the symbols and pushes them in the stack until it sees #, which it pushes on the stack, Then M pops the top of the stack (which is #), and repeats the following process: It reads the next input symbol, say a, and remembers it in the finite control and pushes a fixed dummy symbol D on top of the stack. Then it makes two consecutive λ-moves, where on the first λ move, it pops D, and on the second, it verifies that the symbol on top of the stack is the same as the symbol a remembered in the finite control. M accepts if it finds no discrepancy during the process. Note that M is not reversal-bounded.

Part 2 follows from the fact that L cannot be accepted by an NCM [18]. □

We note that restriction (2) in Definition 2 of an NPCM M in simple normal form is essential, since if we remove this restriction, i.e., we allow M to read a symbol on the input while popping the top of the stack, a DPDA whose stack makes only 1 reversal can clearly accept $L = \{x\#x^R \mid x \in \{0,1\}^*\}$, which cannot be accepted by an NCM. Hence, Proposition 10 and Corollary 4 are not valid without restriction (2).

Now NPCMs are closed under union. But, they are not closed under intersection. In fact, it can be shown using the proof of Theorem 4.2 in [12] that there are languages L_1 and L_2 accepted by 1-reversal DPDAs such that $L_1 \cap L_2$ cannot be accepted by any NPCM. However, from Lemma 8, NPCMs in simple normal form are effectively closed under union and interesection since NCMs are clearly closed under these operations.

Acknowledgements The research of O. H. Ibarra was supported, in part, by NSF Grant CCF-1117708. The research of I. McQuillan was supported, in part, by Natural Sciences and Engineering Research Council of Canada Grant 2016-06172.

References

1. J.M. Autebert, J. Berstel, and L. Boasson. *Handbook of Formal Languages*, volume 1, chapter Context-Free Languages and Pushdown Automata. Springer-Verlag, Berlin, 1997.
2. Suna Bensch, Johanna Björklund, and Martin Kutrib. Deterministic stack transducers. In Yo-Sub Han and Kai Salomaa, editors, *Implementation and Application of Automata: 21st International Conference, CIAA 2016, Seoul, South Korea, July 19-22, 2016, Proceedings*, volume 9705 of *Lecture Notes in Computer Science*, pages 27–38. Springer International Publishing, 2016.
3. Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Bounded Parikh automata. *International Journal of Foundations of Computer Science*, 23(08):1691–1709, 2012.
4. Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Affine parikh automata. *RAIRO - Theoretical Informatics and Applications*, 46:511–545, 2012.
5. Ehsan Chiniforooshan, Mark Daley, Oscar H. Ibarra, Lila Kari, and Shinnosuke Seki. One-reversal counter machines and multihead automata: Revisited. In *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM'11, pages 166–177, Berlin, Heidelberg, 2011. Springer-Verlag.
6. Ehsan Chiniforooshan, Mark Daley, Oscar H. Ibarra, Lila Kari, and Shinnosuke Seki. One-reversal counter machines and multihead automata: Revisited. *Theoretical Computer Science*, 454:81–87, 2012.
7. Joey Eremondi, Oscar H. Ibarra, and Ian McQuillan. Deletion operations on deterministic families of automata. *Information and Computation*, TBA:1–20, 2017. Accepted.
8. Seymour Ginsburg and Sheila Greibach. Deterministic context free languages. *Information and Control*, 9(6):620–648, 1966.
9. Eitan M. Gurari and Oscar H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22(2):220–229, 1981.
10. Tero Harju, Oscar Ibarra, Juhani Karhumäki, and Arto Salomaa. Some decision problems concerning semilinearity and commutation. *Journal of Computer and System Sciences*, 65(2):278–294, 2002.

11. J E Hopcroft and J D Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.
12. O.H. Ibarra. Visibly pushdown automata and transducers with counters. *Fundamenta Informaticae*, 148(3–4):291–308, 2016.
13. O.H. Ibarra and I. McQuillan. The effect of end-markers on counter machines and commutativity. *Theoretical Computer Science*, 627:71–81, 2016.
14. O.H. Ibarra and I. McQuillan. Applications of store languages to reachability, 2017. Submitted.
15. O.H. Ibarra and I. McQuillan. On store languages of language acceptors, 2017. Submitted. A preprint appears in `https://arxiv.org/abs/1702.07388`.
16. O.H. Ibarra and H.C. Yen. On the containment and equivalence problems for two-way transducers. *Theoretical Computer Science*, 429:155–163, 2012.
17. Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.
18. Oscar H. Ibarra and Shinnosuke Seki. Characterizations of bounded semilinear languages by one-way and two-way deterministic machines. *International Journal of Foundations of Computer Science*, 23(6):1291–1306, 2012.