Store Languages of Turing Machines and Counter Machines

Noah Friesen^a, Oscar H. Ibarra^b, Jozef Jirásek^{c,1}, Ian McQuillan^{c,1,*}

^aDepartment of Mathematics and Statistics
University of Saskatchewan
Saskatoon, SK S7N 5C9, Canada
^bDepartment of Computer Science
University of California, Santa Barbara, CA 93106, USA
^cDepartment of Computer Science, University of Saskatchewan
Saskatoon, SK S7N 5C9, Canada

Abstract

The store language of an automaton is the set of store configurations (state and store contents, but not the input) that can appear as an intermediate step in an accepting computation. A one-way nondeterministic finite-visit Turing machine (fvNTM) is a Turing machine with a one-way read-only input tape, and a single worktape, where there is some number k such that in every accepting computation, each worktape cell is visited at most k times. We show that the store language of every fvNTM is a regular language. Furthermore, we show that the store language of every fvNTM augmented by reversal-bounded counters can be accepted by a machine with only reversal-bounded counters and no worktape. Several applications are given to problems in the areas of verification and fault tolerance, and to the study of right quotients. We also continue the investigation of the store languages of one-way and two-way machine models where we present some conditions under which their store languages are recursive or non-recursive.

Keywords: Store Languages, Finite-Visit Turing Machines, Decidability Questions, Counter Machines

1. Introduction

A useful concept in formal language and automata theory is that of the store language of a machine. The store language is the set of store configurations (state and store contents) that can appear as an intermediate step in an accepting computation. For example, the store language of a one-way nondeterministic pushdown automaton consists of all words of the form $q\gamma$, where there is some accepting computation that passes through a configuration in state q and with γ as the pushdown contents. It is well-known that the store language of any pushdown automaton is a regular language [1, 2]. Furthermore, it is known that a nondeterministic finite automaton accepting the store language of a given pushdown automaton can always be constructed in polynomial time [3]. In addition, the optimal number of states needed in the nondeterministic finite automaton constructed from the given pushdown automaton has been determined [4].

There are several other models where the store languages have been characterized as well. One-way stack automata are one-way pushdown automata with the additional ability to read from the inside of the stack in a two-way read-only mode, but pushing and popping are only available when at the top of the stack [5]; finite-turn Turing machines (denoted here by ftNTM) have a one-way read-only input tape and a single worktape where there is a bound on the number of changes in direction of the worktape head on the worktape [6]; and finite-flip pushdown automata are pushdown automata which can "reverse" (or flip) their

^{*}Corresponding author

Email addresses: noah.friesen@usask.ca (Noah Friesen), ibarra@cs.ucsb.edu (Oscar H. Ibarra),

jirasek.jozef@usask.ca (Jozef Jirásek), mcquillan@cs.usask.ca (Ian McQuillan)

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

pushdown store up to a bounded number of times [7]. To note, finite-turn Turing machines are also known as reversal-bounded Turing machines [6] but we use the term finite-turn as we use the term reversal-bounded for counters instead. It has been proven that the store languages of all of these models are also always regular ([8, 9] for stack automata, [9] for ftNTM, and [10] for finite-flip pushdown automata). One-way counter machines have some finite number of counters that each contains a non-negative integer, and transitions can test whether the value in each counter is zero or not. While it is well known that these machines can accept all recursively enumerable languages [11], restrictions can limit their power. A counter machine is reversalbounded if in every accepting computation, for each counter, the number of switches between executing sequences of transitions that do not decrease the counter and transitions that do not increase the counter, is bounded [12, 13]. The set of one-way nondeterministic reversal-bounded multi-counter machines is denoted by NCM, and the set of deterministic machines in NCM is denoted by DCM. The store languages of NCMs can be non-regular, but can always be accepted by DCMs [9]. If we augment any of pushdown automata [9], finite-turn Turing machines [9], or finite-flip pushdown automata [10] with any number of reversal-bounded counters, then their store languages can always be accepted by NCMs. Lest one speculate that this would be true for any model that only produces regular store languages, it is not so as stack automata augmented by one reversal-bounded counter can produce store languages that cannot be accepted by a NCM [9] despite stack automata only having regular store languages.

When all store languages of a one-way nondeterministic machine model \mathcal{M} (under some simple conditions described in the paper) are regular, this implies that the languages accepted by the deterministic machines in \mathcal{M} are closed under right quotient with regular languages [9]. This technique can completely replace the often lengthy and ad-hoc proofs of closure under right quotient with regular languages for deterministic classes of machines in the literature (e.g. for deterministic pushdown automata [14] or deterministic stack automata [15]).

Also, a class of machines either having store languages that are regular or that can be accepted by NCM machines has applications to problems in the area of model checking, reachability, and verification. Two commonly studied operations are: given a set of store configurations C, the set of store configurations that can follow in zero or more moves of a machine M from those in C is called $post_M^*(C)$, and the set of store configurations that can lead in zero or more moves to those in C is called $\operatorname{pre}_{M}^{*}(C)$. It is known that given a pushdown automaton M and a regular set of configurations C, $\operatorname{pre}_M^*(C)$ and $\operatorname{post}_M^*(C)$ are both regular languages [16]. It is also known that for an NCM M and a set C of configurations accepted by an NCM, both $\operatorname{pre}_{M}^{*}(C)$ and $\operatorname{post}_{M}^{*}(C)$ can be accepted by DCMs [17]. These operations have also been studied for other machine models, e.g. [18, 19, 20, 21, 22]. More recently, in [10], connections were made between store languages and these operations, whereby determining that the store languages of a class of machines are always in certain families of languages implies that $\operatorname{pre}_M^*(C)$ and $\operatorname{post}_M^*(C)$ are in the same family, and vice versa. More specifically, it was shown that under some simple conditions, any class of machines \mathcal{M} with only regular store languages always satisfies the following: given a regular set C, then both $\operatorname{pre}_{M}^{*}(C)$ and $post_M^*(C)$ are also regular; further, for a class where the store languages can always be accepted by an NCM, then given C accepted by an NCM, both $\operatorname{pre}_M^*(C)$ and $\operatorname{post}_M^*(C)$ can be accepted by an NCM. Also, consider the following problem, called the common configurations problem for a class of machines \mathcal{M} : "given two machines $M_1, M_2 \in \mathcal{M}$, are there any non-initial store configurations in common between M_1 and M_2 ?". The same paper [10] shows that as long as store languages of a class can always be accepted by an NCM, then the common configurations problem is always decidable. This problem is related to fault-tolerance: if one machine, M_1 , has a store language that describes all faulty configurations, then computations of the other machine M_2 may lead to a fault configuration if and only if they have configurations in common.

In this paper, we are interested in one-way nondeterministic Turing machines with a single two-way read/write worktape. Such a machine is k-turn (respectively k-visit, k-crossing) if, in every accepting computation, the number of changes in direction of movement of the read/write from left-to-right or right-to-left (respectively the number of visits to a worktape cell, or the number of crosses between the boundary of any two adjacent worktape cells) is at most k. It is finite-turn (respectively finite-visit, finite-crossing), if it is k-turn (respectively k-visit, k-crossing) for some k. The class of one-way finite-turn nondeterministic (respectively finite-visit, finite-crossing) Turing machines is denoted by ftNTM (respectively fvNTM, fcNTM). All ftNTM machines are fvNTM (but not vice versa), and all fvNTM machines are fcNTM but not vice versa

(since staying on a cell counts as a visit but not a cross). However, it is known that the languages accepted by ftNTM are properly contained in the languages accepted by fvNTM, which are equal to the languages accepted by fcNTM. Indeed, the languages accepted by fcNTM and fvNTM are closed under Kleene-* but those accepted by ftNTM are not [6]; also fvNTM and fcNTM precisely characterize an important family that has been studied in the formal language theory literature. Greibach showed [6] that the languages accepted by fvNTM and fcNTM are equal to the languages generated by absolutely parallel grammars [23]. And it was shown by Latteux [24] that the languages generated by absolutely parallel grammars are identical to the languages generated by many different types of grammar models that are so-called finite-index [25], where there is some integer k such that every word generated by the grammar has a derivation with at most k nonterminals. The languages generated by finite-index grammars of the following types (which we will not define here) were all shown to coincide: ET0L systems, EDT0L systems (two types of Lindenmayer systems), context-free programmed grammars, ordered grammars, matrix grammars, and matrix grammars with appearance checking [25]. Thus, finite-crossing and finite-visit Turing machines provide an automata model that accepts exactly the same languages as all of these grammar systems can generate. Hence, fvNTM and fcNTM are an important and natural class of machines.

While it is already known that the store languages of all ftNTM are regular [9], in this paper, we show that the store languages of the more powerful finite-visit and finite-crossing Turing machines are regular languages. This immediately shows that the languages accepted by deterministic finite-visit and deterministic finite-crossing Turing machines are closed under right quotient with regular languages, which was not previously known. Furthermore, if we augment finite-crossing Turing machines with reversal-bounded counters, then the store languages of all such machines can be accepted by NCMs. These store language results have applications to the pre* and post* operations, and the common configuration problem, with the common configurations problem being decidable for fcNTM augmented with reversal-bounded counters.

Next, we investigate decidability problems, such as, given Turing machine M and $k \geq 0$, is M k-turn, (respectively k-visit, k-crossing)? Also, given M, is M finite-turn (respectively finite-visit, finite-crossing)? This would thereby guarantee that its store language is a regular language. Then, we describe the subtle difference between defining finite-crossing as "in every accepting computation, each boundary between a pair of adjacent cells is crossed at most k times", and what we call weak finite-crossing: "for every accepted word k, there is some accepting computation in which the boundary of each boundary between a pair of adjacent cells is crossed at most k times". We use the former notion here, but it is the latter notion that is called finite-crossing in [6]. Despite the two notions being equivalent in terms of languages accepted, this is not so for store languages. The store languages of the first class of machines are all regular languages, however we show that for the second notion, the store languages contain non-recursive languages. This shows that there are non-recursive store languages for these machines. This shows that even though two classes of machines accept the same family of languages, they can be dramatically different in terms of store languages. The subtleties of the machine model definition are crucial.

Finally, we continue the study of store languages of classes of two-way machines. Previously, very little was known besides some examples of non-regular store languages being demonstrated [9]. We show that several models have non-recursive store languages, including two-way deterministic pushdown automata, and two-way deterministic 1-counter machines. We also find other classes that have store languages with an undecidable membership problem.

2. Preliminaries

We denote the set of integers by \mathbb{Z} , the set of non-negative integers by \mathbb{N}_0 , and the set of positive integers by \mathbb{N} . Given $n \in \mathbb{N}_0$, let $\pi(n)$ be 0 if n = 0 and 1 otherwise.

We assume a familiarity with the basics of formal language and automata theory, including one-way and two-way nondeterministic finite automata, regular languages, and Turing machines [11]. An alphabet Σ is a finite set of symbols. The empty word is denoted by λ . Given a word $w \in \Sigma^*$, the length of w is denoted by |w|. Given a word $w = a_1 \cdots a_n, a_i \in \Sigma, 1 \le i \le n$, the reverse of w, $w^R = a_n \cdots a_1$. The set of all non-empty words over Σ is denoted by Σ^+ , and the set of all words over Σ , including the empty word, is denoted by Σ^* . A language over Σ is any subset of Σ^* . Given $L_1, L_2 \subseteq \Sigma^*$, the left quotient of L_2 by L_1 ,

 $L_1^{-1}L_2 = \{v \mid uv \in L_2, u \in L_1\}, \text{ and the right quotient of } L_1 \text{ by } L_2, L_1L_2^{-1} = \{u \mid uv \in L_1, v \in L_2\}.$ A language $L \subseteq \Sigma^*$ is bounded if $L \subseteq w_1^* \cdots w_n^*$ for some non-empty words $w_1, \ldots, w_n \in \Sigma^*$; and L is letter-bounded if $L \subseteq a_1^* \cdots a_n^*$ where a_1, \ldots, a_n are letters of Σ .

In this paper, we are going to primarily focus on three types of automata: multi-counter machines, Turing machines, and Turing machines augmented by counters. All of the machines are assumed to be nondeterministic, and use a one-way read-only input tape. Hence, we will simply define the model combining together both storage types, and plain Turing machines will be restricted to only use the worktape, and counter machines will only use the counters.

Definition 1. A one-way nondeterministic Turing machine with t counters is denoted by a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q is the finite set of states, Σ is the finite input alphabet, Γ is the finite worktape alphabet containing the blank symbol \cup , $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a finite subset of $\Omega_0 \cup \cdots \cup \Omega_t$, where

$$\Omega_0 = Q \times (\Sigma \cup \{\lambda, \triangleleft\}) \times \{0\} \times \Gamma \times Q \times \Gamma \times \{L, S, R\},$$

$$\Omega_i = Q \times (\Sigma \cup \{\lambda, \triangleleft\}) \times \{i\} \times \{0, 1\} \times Q \times \{-1, 0, +1\} \text{ for } 1 \le i \le t.$$

A configuration of M is a tuple

$$(q, w, x \not | y, z_1, \dots, z_t), \tag{1}$$

where $q \in Q$ is the current state, $w \in \Sigma^* \triangleleft \cup \{\lambda\}$ is the remaining input, $x \not = y$ is the worktape contents with $x \in \Gamma^+, y \in \Gamma^*$, $(J \text{ is a new symbol denoting the position of the read/write head, which is scanning the symbol immediately to its left), and <math>z_i \in \mathbb{N}_0$ is the current contents of counter i for $1 \le i \le t$. The store configuration of the configuration (1) is the string $qx \not = yC_1^{z_1} \cdots C_t^{z_t}$ where C_1, \ldots, C_t are new symbols, and conf(M) is the set of all store configurations. Configurations will change as follows.

- $(q, aw, xc \not y, z_1, \dots, z_t) \vdash_M (q', w, xd \not y, z_1, \dots, z_t)$, if $(q, a, 0, c, q', d, S) \in \delta$, (a stay transition on the worktape),
- $(q, aw, xc \not \exists y, z_1, \ldots, z_t) \vdash_M (q', w, x' \not \exists d'y, z_1, \ldots, z_t)$, if $(q, a, 0, c, q', d, L) \in \delta$, $(x = \lambda \implies x' = \sqcup, otherwise x' = x)$, $(y = \lambda \land d = \sqcup \implies d' = \lambda, otherwise d' = d)$, (a left transition on the worktape),
- $(q, aw, xc \lor y, z_1, \ldots, z_t) \vdash_M (q', w, xd'c' \lor y', z_1, \ldots, z_t)$, if $(q, a, 0, c, q', d, R) \in \delta$, $(y = \lambda \implies y' = \lambda, c' = \sqcup$, otherwise $y = c'y', c' \in \Gamma$), $(x = \lambda \land d = \sqcup \implies d' = \lambda$, otherwise d' = d), (a right transition on the worktape),
- $(q, aw, xc \triangleleft y, z_1, \ldots, z_t) \vdash_M (q', w, xc \triangleleft y, z_1, \ldots, z_{i-1}, z_i + z, z_{i+1}, \ldots, z_t)$, if $(q, a, i, \pi(z_i), q', z) \in \delta$ where $1 \leq i \leq t$, (a counter transition).

A computation of M on w is a derivation $(p_0, w_0, \gamma_0, z_{0,1}, \ldots, z_{0,t}) \vdash \cdots \vdash (p_n, w_n, \gamma_n, z_{n,1}, \ldots, z_{n,t})$ where $p_0 = q_0, w_0 = w \triangleleft$, and $\gamma_0 = \sqcup \lrcorner$. This computation is accepting if $p_n \in F$ and $w_n = \lambda$. Given such a computation, the worktape address of configuration $j, c_j = (p_j, w_j, \gamma_j, z_{j,1}, \ldots, z_{j,t}),$ for $0 \leq j \leq n$, denoted add (c_j) is defined inductively to be 1 if j = 0; and otherwise it is the address of c_{j-1} plus 1 (resp. -1, 0) if $c_{j-1} \vdash c_j$ with a transition that moves right on the worktape (resp. moves left on the worktape, stays or uses a counter). Let \vdash_M^* be the reflexive and transitive closure of \vdash_M .

The language accepted by M, denoted by L(M), is defined to be the set of all $w \in \Sigma^*$ such that there is an accepting computation of M on w. Furthermore, the store language of M is defined to be:

$$S(M) = \left\{ qxC_1^{z_1} \cdots C_t^{z_t} \middle| \begin{array}{l} (q_0, w \triangleleft, \sqcup \triangleleft, 0, \ldots, 0) \vdash_M^* (q, w', x, z_1, \ldots, z_t) \vdash_M^* (q_f, \lambda, x', z_1', \ldots, z_t'), \\ q_f \in F, w \in \Sigma^*, w' \in \Sigma^* \triangleleft \cup \{\lambda\}. \end{array} \right\}$$

Intuitively, Ω_0 contains all possible transitions involving the worktape, and Ω_i contains all possible transitions involving the ith counter, for $1 \le i \le t$. Therefore, each transition only uses one store at a time (either the worktape, or one of the counters), and it would require multiple transitions to read from multiple stores. Here the symbol \triangleleft is the right end-of-input symbol. We will sometimes omit the end-marker for one-way nondeterministic machines, as these machines can guess that they are at the end of the input, as is common for say one-way nondeterministic pushdown automata. The first component is always the current state, the second component is the input letter being read (or the empty word, or the end-marker), and the third component is the current store being accessed. For the worktape store (so a transition in Ω_0), the fourth component is the current symbol under the read/write head of the store, the fifth component is the state to switch to, the sixth component is the worktape symbol to replace the current symbol, and the seventh component is either L, S, R which provides the direction to move the read/write head on the worktape (left, stay, or right respectively). For the counter stores in Ω_i , $i \ge 1$, the fourth component is 0 if counter i is currently zero and 1 otherwise, the fifth component is the state to switch to, and the sixth component is the value to add to counter i.

Notice that in such a machine with 0 counters, the store language only encodes the state and worktape and does not use the symbols C_1, \ldots, C_t . In such a case, these are simply one-way nondeterministic Turing machines, which we refer to by NTM. In this case, we leave off the third component of the transitions as they are all 0. Notice that the read/write head is encoded within the worktape contents encoded in the store language. We sometimes examine the restriction where we leave off the worktape and only have counters. We denote by COUNTER(t) machines with t counters, and COUNTER all machines with some number of counters.

Next, we study a restriction of the counters, and then three different restrictions of the worktape. It is well known that Turing machines with one worktape accept all recursively enumerable languages, and likewise one-way deterministic machines with 2 counters also accept all recursively enumerable languages [11]. Hence, restrictions are necessary for certain problems to be decidable or to have simpler store languages. All restrictions studied will limit their power.

A one-way nondeterministic Turing machine with t counters $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is r-reversal-bounded if, in each accepting computation, the number of changes between sequences of non-decreasing transitions (where transitions can add 1 or 0) and sequences of non-increasing transitions (where transitions can add -1 or 0), or vice versa, in each counter is at most r. The machine is reversal-bounded if it is r-reversal-bounded for some r. The class of all one-way nondeterministic Turing machines with some number of reversal-bounded counters is denoted by NTCM. If the machine does not use the worktape, so $\Omega_0 \cap \delta = \emptyset$, then the class of all one-way nondeterministic machines with some number of reversal-bounded counters (no worktape) is denoted by NCM.

All three restrictions of the worktape were studied in [6] (finite-turn was called reversal-bounded in their paper, but we use finite-turn here to disambiguate with our use of counters).

Definition 2. Let $M=(Q,\Sigma,\Gamma,\delta,q_0,F)$ be an NTM (respectively an NTCM). Let $(p_0,w_0,\gamma_0,0,\ldots,0) \vdash \cdots \vdash (p_n,w_n,\gamma_n,z_1,\ldots,z_n)$, be an accepting computation, where i_j is the address of the jth configuration, for $0 \le j \le n$. We say the accepting computation makes a turn in the jth configuration where $j \ge 1$, if either $i_j < i_{j-1}$ and there exists a largest $1 \le l \le j-1$ such that $i_l \ne i_{j-1}$ which has $i_l < i_{j-1}$; or $i_j > i_{j-1}$ and there exists a largest $1 \le l \le j-1$ such that $i_l \ne i_{j-1}$ which has $i_l > i_{j-1}$. Then, we say the accepting computation is

- k-turn if the number of turns is at most k,
- k-crossing if, for each $r \in \mathbb{Z}$, $|\{j \mid i_j = r, i_{j+1} = r+1\} \cup \{j \mid i_j = r+1, i_{j+1} = r\}| \le k$,
- k-visit if, for each $r \in \mathbb{Z}$, $|\{j \mid i_j = r\}| \le k$.

Further, we say M is k-turn (respectively k-visit, k-crossing) if every accepting computation is k-turn (respectively k-visit, k-crossing). Also, M is finite-turn (respectively finite-visit, finite-crossing) if it is k-turn (respectively k-visit, k-crossing) for some $k \ge 0$.

Given a class of machines \mathcal{M} , let $\mathcal{L}(\mathcal{M})$ be the family of languages accepted by machines in \mathcal{M} , and let $\mathcal{S}(\mathcal{M})$ be the family of store languages of machines in \mathcal{M} .

We denote the class of one-way nondeterministic finite automata by NFA, and class of two-way nondeterministic finite automata by 2NFA, and as a short-form we denote the class of regular languages by REG and the class of recursively enumerable languages by RE. We denote the class of all finite-turn (respectively finite-visit, finite-crossing) Turing machines by ftNTM (respectively fvNTM, fcNTM). We denote the class of all finite-turn (respectively finite-visit, finite-crossing) Turing machines with some number of reversal-bounded counters by ftNTCM (respectively fvNTCM, fcNTCM).

As noted in Section 1, every finite-turn Turing machines is finite-visit, but not vice versa (see Example 2 below), and all finite-visit machines are finite-crossing, but not vice versa (each transition that stays on the same worktape cell increases the visit count but not the crossing count). In terms of languages accepted, however, it was shown in [6] that $\mathcal{L}(\mathsf{ftNTM}) \subsetneq \mathcal{L}(\mathsf{fvNTM}) = \mathcal{L}(\mathsf{fcNTM})$. In particular, $\mathcal{L}(\mathsf{ftNTM})$ is not closed under Kleene-*, but $\mathcal{L}(\mathsf{fcNTM})$ is closed under it. Also, as mentioned in Section 1, $\mathcal{L}(\mathsf{fcNTM})$ is equal to the family of languages generated by many types of finite-index grammar systems, such as finite-index matrix grammars. Furthermore, $\mathcal{L}(\mathsf{fcNTCM})$ was studied in [26] where it was shown that all languages in this family are semilinear (with an effective procedure) implying a decidable emptiness, membership, and finiteness problem.

3. Store Languages of Finite-Visit and Finite-Crossing Turing Machines

We give two examples of Turing machines and their store languages.

Example 1. Consider the NTM $M=(Q,\Sigma,\Gamma,\delta,q_0,F)$, where $\Sigma=\{a,b,\$\}$, $\overline{\Sigma}=\{a,b\}$, $\Gamma=\{a,b,\sqcup\}$, $F=\{q_4\}$, and δ contains the following transitions, for all $c\in\overline{\Sigma}$:

$$(q_0, c, 0, \sqcup, q_1, c, \mathbf{R}), (q_1, c, 0, \sqcup, q_1, c, \mathbf{R}), (q_1, \$, 0, \sqcup, q_2, \sqcup, \mathbf{L}), (q_2, \lambda, 0, c, q_2, c, \mathbf{L}), \\ (q_2, \lambda, 0, \sqcup, q_3, \sqcup, \mathbf{R}), (q_3, c, 0, c, q_3, c, \mathbf{R}), (q_3, \triangleleft, 0, \sqcup, q_4, \sqcup, \mathbf{S}).$$

Note that the third components are all 0's as it does not have any counters. This machine accepts the language $\{w\$w \mid w \in \overline{\Sigma}^+\}$, which is a well-known non-regular and non-context-free language [11]. It operates by copying the first part of the input (up to \$) to the tape, moving the read/write head back to the leftmost end of the tape, then matching the second part of the input with the contents of the tape. Hence this machine is 2-turn. It is also 3-crossing, and 3-visit.

The store language of this machine, S(M), is

$$\{q_0\sqcup \downarrow\} \cup \{qx\sqcup \downarrow \mid q\in \{q_1,q_3,q_4\}, x\in \overline{\Sigma}^+\} \cup \{q_2\sqcup \downarrow x\mid x\in \overline{\Sigma}^+\} \cup \{qx \downarrow y\mid q\in \{q_2,q_3\}, x\in \overline{\Sigma}^+, y\in \overline{\Sigma}^*\},$$

which is a regular language.

This example is indicative of a general important property of store languages of ftNTM Turing machines. We have the following from [9].

Proposition 3 ([9]). $\mathcal{S}(\mathsf{ftNTM}) \subseteq \mathsf{REG}$.

Whether this was true for finite-visit and finite-crossing Turing machines was left open.

Example 2. Consider the following Turing machine which is a modified version of the machine given in Example 1, $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where $\Sigma = \{a, b, \$, \#\}$, $\overline{\Sigma} = \{a, b\}$, $\Gamma = \{a, b, \#, \sqcup\}$, $F = \{q_0\}$, and δ contains the following transitions, for all $c \in \overline{\Sigma}$ (leaving off all third components as they are all 0's):

$$(q_0, \lambda, \sqcup, q_1, \#, R), (q_1, c, \sqcup, q_1, c, R), (q_1, \$, \sqcup, q_2, \sqcup, L), (q_2, \lambda, c, q_2, c, L),$$

 $(q_2, \lambda, \#, q_3, \#, R), (q_3, c, c, q_3, c, R), (q_3, \#, \sqcup, q_0, \sqcup, S).$

This machine accepts the language $\{w\$w\# \mid w \in \overline{\Sigma}^*\}^*$. For each segment u\$v of the input separated by #'s, it writes #, copies the string u to the tape, moves the read/write head left until it reaches #, then matches the string v from the input with the contents of the tape to the right of the head. It then repeats this for the next segment of the input using a new section of the worktape.

This machine is not finite-turn, since it makes 2n turns for an input with n segments. However, M is 4-visit and 3-crossing. The store language of this machine is:

$$S(M) = \{q_0 x \sqcup \exists \mid x \in (\#\overline{\Sigma}^*)^*\} \cup \{q x \sqcup \exists \mid q \in \{q_1, q_3\}, x \in (\#\overline{\Sigma}^*)^+\}$$

$$\cup \{q_2 x \exists y \mid x \in (\#\overline{\Sigma}^*)^+, y \in \overline{\Sigma}^*\} \cup \{q_3 x \exists y \mid x \in (\#\overline{\Sigma}^*)^* \#\overline{\Sigma}^+, y \in \overline{\Sigma}^*\},$$

which is a regular language.

Next we shall prove that this is true in general for any finite-crossing Turing machine.

Theorem 4. $\mathcal{S}(\mathsf{ftNTM}) \subseteq \mathsf{REG} \ and \ \mathcal{S}(\mathsf{fvNTM}) \subseteq \mathsf{REG}.$

PROOF. It is enough to prove this for fcNTM since all finite-visit machines are finite-crossing.

We use a construction that is somewhat similar to the one found in the proof of the previous proposition for ftNTM given in [9], however, it is quite a bit more complex. Let $M=(Q,\Sigma,\Gamma,\delta,q_0,F)$ be an r-crossing Turing machine. Let k=2r. Notice that if M is r-crossing, then each cell is involved in at most 2r=k crossings, where at most r come from the cell to the left, and at most r from the cell to the right. Let $\overline{\Gamma}=\{\overline{a}\mid a\in\Gamma\}$ be a new alphabet. Symbols in Γ are said to be unmarked, and those in $\overline{\Gamma}$ are said to be marked. We are going to construct a 2NFA M' that accepts words that represent a "history" of a computation of M. First, we will describe the alphabet C. Intuitively, each symbol of w to w is a column representing the history of a single cell of the worktape of w, and each component of the symbols of w is a "track" (we call these tracks 0 through w). Track 0 will contain a representation of the worktape on some single point of time from the history of this computation, with the state in the first cell, and with a single marked symbol which represents the position of the read/write head at that point of time. In tracks 1 through w, the numbers and arrows encode a doubly linked list structure representing actions that w performs on its tape given some input.

Formally, define a new alphabet C whose elements are of the form (c_0, \ldots, c_k) , where $c_0 \in Q \cup \Gamma \cup \overline{\Gamma}$ and for each $i = 1, \ldots, k$, either $c_i = \emptyset$ or

$$c_i \in (\Gamma \cup \overline{\Gamma}) \times \{\leftarrow, \rightarrow, \varnothing\} \times \{1, \dots, k, \varnothing\} \times \{\leftarrow, \rightarrow, \varnothing\} \times \{1, \dots, k, \varnothing\}.$$

For such a $c_j = (x, s_d, s_t, p_d, p_t)$, $1 \le j \le k$, we call x the tape symbol, s_d the successor direction, s_t the successor track, p_d the predecessor direction, and p_t the predecessor track. The new symbol \varnothing is used as an "unused" marker.

Next, construct a 2NFA $M' = (Q', C, \delta', q'_0, F')$, where we will describe the construction of Q' and δ' below, with input delimited by left and right end-markers \triangleright and \triangleleft , *i.e.*, the input to M' is $\triangleright w \triangleleft$ where $w \in C^*$. Figure 1 shows a visualization of an example input given to M' which encodes a simulation of the following sequence of instructions of M. The encoding below shows the symbol to write in the current cell with a transition that moves to another cell (possibly after a sequence of stay transitions) along with the direction to move as superscript, corresponding to the computation from this example:

$$x_0^{\rightarrow}, \ x_1^{\rightarrow}, \ x_2^{\rightarrow}, \ x_3^{\leftarrow}, \ x_4^{\leftarrow}, \ x_5^{\rightarrow}, \ x_6^{\leftarrow}, \ x_7^{\leftarrow}, \ x_8^{\rightarrow}, \ x_9^{\rightarrow}, \ x_{10}^{\rightarrow}, \ x_{11}^{\rightarrow}, \ x_{12},$$

which corresponds to the following sequence of worktape contents of M:

$$\sqcup \downarrow$$
, $x_0 \sqcup \downarrow$, $x_0 x_1 \sqcup \downarrow$, $x_0 x_1 x_2 \sqcup \downarrow$, $x_0 x_1 x_2 \downarrow x_3$, $x_0 x_1 \downarrow x_4 x_3$, $x_0 x_5 x_4 \downarrow x_3$, $x_0 x_5 \downarrow x_6 x_3$, $x_0 \downarrow x_7 x_6 x_3$, $x_8 x_7 \downarrow x_6 x_3$, $x_8 x_9 x_6 \downarrow x_3$, $x_8 x_9 x_{10} x_3 \downarrow$, $x_8 x_9 x_{10} x_{11} \sqcup \downarrow$, $x_8 x_9 x_{10} x_{11} \sqcup \downarrow$.

The bolded configuration corresponds to the figure. In Figure 1, we only visually show the arrows encoded by the second and third components of each track (the second component encodes arrow direction and the

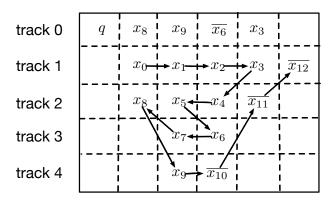


Figure 1: Example input to the 2NFA.

third component encodes the track of the arrow destination), and for each track there is an arrow encoded by the fourth and fifth components that exactly follows the shown arrow backwards. For example, the first three characters of the input corresponding to this visualization would be:

$$\begin{split} (q,\varnothing,\varnothing,\varnothing,\varnothing) \\ (x_8,(x_0,\rightarrow,1,\varnothing,\varnothing),(x_8,\rightarrow,4,\rightarrow,3),\varnothing,\varnothing) \\ (x_9,(x_1,\rightarrow,1,\leftarrow,1),(x_5,\rightarrow,3,\rightarrow,2),(x_7,\leftarrow,2,\rightarrow,3),(x_9,\rightarrow,4,\leftarrow,2)). \end{split}$$

The last component of the third character is encoded as $(x_9, \to, 4, \leftarrow, 2)$, since the next node in the list is the fifth track of the column to the right, and the previous node in the list is the third track of the column to the left.

Intuitively, as the arrows describe the movement between worktape cells of the computation, they fill in the next available unused track, from track 1 to k. In this way, a linked list representing any r-crossing computation of M can be stored in a string w within these k tracks. By following the arrows of the linked list, it is possible to follow the simulation of M represented by w. At some nondeterministically chosen point in the simulation, track 0 will be verified to contain the current tape contents at that point. Since that point is chosen nondeterministically, we get an encoding of each possible store language word on the first track with the read/write head marked. From then on, it continues the simulation in tracks 1 to k, but we switch from using unmarked symbols to marked symbols. We can apply a generalized sequential machine (gsm) to extract the word contained in track 0 (erasing blanks at the ends, and replacing the only marked symbol in track 1 $\overline{a} \in \overline{\Gamma}$ with $a \triangleleft 1$) to obtain a word in S(M).

The input to M' is accepted if and only if M' is able to successfully verify all of the following conditions:

- 1. Track 0 of the input word contains a word in $Q\Gamma^*\overline{\Gamma}\Gamma^*$, *i.e.*, a word that could possibly be in S(M) after applying the gsm mentioned above. In particular, track 0 contains exactly one marked symbol.
- 2. In each column, tracks 1 to k are filled contiguously from track 1 downwards from the start to the end of the linked list. This implies that if some track i is \emptyset , then tracks $i+1,\ldots,k$ are also \emptyset . Thus, consider a node (x, s_d, s_t, p_d, p_t) in the ith track, $i \geq 1$. Examining the column in direction s_d , the first track after s_t with direction opposite s_d (if it exists) must be to track i+1.
- 3. There is exactly one node whose predecessor track and direction are both \varnothing , and this node is located in track 1. This node is called the beginning node. There is exactly one node whose successor track and direction are both \varnothing , and this node is located in the bottom-most non-empty track of its column. This node is called the end node. The predecessor and successor tracks and directions of all other nodes are not \varnothing .

- 4. Consider a node (x, s_d, s_t, p_d, p_t) in the *i*th track. Then the predecessor track of the node in the s_t th track of the column to the left (if $s_d = \leftarrow$) or to the right (if $s_d = \rightarrow$) must be set to *i*, and the predecessor direction of this node must be set to the opposite direction of s_d . The successor link from the predecessor of every node is verified similarly.
- 5. In each column, if the symbol in track 0 is unmarked, then it must match the bottom-most unmarked symbol of the nodes in tracks $1, \ldots, k$.
- 6. The linked list represents an accepting computation of M as follows: M' starts by reading and storing the state q written at the beginning of track 0 in the finite control of M', and then M' finds the beginning node in track 1. M' then simulates transitions of M by matching instructions with the symbols in the linked list rather than writing to a tape. M' keeps track of the current track as it simulates. The most recent symbol that M writes while staying on the same tape cell can also be tracked using the state of M', and the symbols in the linked list represent the last symbol written in a cell before moving off of the tape cell (so the history does not contain any intermediate values written to a cell as it executes a sequence of stay transitions). The point in the linked list sequence where the first marked symbol occurs represents the point during the computation of M where the word in track 0 matches the tape of M. This means that at some nondeterministically guessed point before the simulation of M moves off of this tape cell (possibly in a sequence of stay transitions), the state of the simulated M must match the state q in track 0, and the symbol M has just written to this tape cell (remembered in the state of M') must match the marked symbol in track 0 of the same column. If this is successful, the simulation continues with only marked symbols until M accepts, at which point M' accepts.

These conditions ensure that each input word represents a valid sequence of actions that M could perform. In particular, they ensure that the directed graph described by the successor relation is a single directed path with no cycles. To see this, suppose that the list contains a cycle. Then there must be a node in the leftmost column of the cycle with both a predecessor and successor in the adjacent column to the right. Similarly, there is a node in the rightmost column of the cycle with both a predecessor and successor in the adjacent column to the left. Condition 2 implies that the successors of these nodes are in lower tracks than the predecessors of these nodes. Since these nodes are in the cycle, there must exist directed paths from the successor of the leftmost node to the predecessor of the rightmost node, and from the successor of the rightmost node to the predecessor of the leftmost node. However, these directed paths must cross at some point since they cannot extend beyond the leftmost and rightmost nodes, contradicting condition 2.

Thus, M' accepts words which describe an accepting computation of M in tracks 1 through k; and track 0 contains the store word of M in one specific configuration of this computation, after adjusting the head symbol, and possibly erasing blank symbols at the beginning and the end. It is known that the language accepted by any 2NFA is regular [11], hence L(M') is a regular language. We can then apply the gsm that extracts the word from track 0 of every word in L(M'), erasing blanks as appropriate and replacing $\overline{a} \in \overline{\Gamma}$ with $a \triangleleft$, to obtain S(M). Since regular languages are closed under gsms [11], it follows that S(M) is a regular language.

Note that in all three cases of ftNTM, fvNTM, and fcNTM, the store languages are "essentially" equal to the regular languages, except for the state at the beginning of each word, and the read/write head. For all three restrictions, given a regular language R, it is easy to construct a Turing machine M that would just put a word in R on the worktape, then switch to some new fixed state q, and then switch to a final state. Let $S_q(M) = \{u \mid qu \neq S(M)\}$. For any subclass M of Turing machines, let $S_{\text{state}}(M)$ be the family of all $S_q(M)$ for $M \in M$ and states q of M. Because S(M) is always regular, $S_q(M)$ must be as well, for any state q of M. This implies the following:

 $\mathbf{Corollary~5.~} \mathcal{S}_{\mathrm{state}}(\mathsf{ftNTM}) = \mathcal{S}_{\mathrm{state}}(\mathsf{fvNTM}) = \mathcal{S}_{\mathrm{state}}(\mathsf{fcNTM}) = \mathsf{REG}.$

Next, we consider each of ftNTCM, fvNTCM, and fcNTCM. Each of these models uses t counters, and the words of the store languages are of the form $qxC_1^{i_1}\cdots C_t^{i_t}$ where q is the current state, x represents the

contents of the worktape, and i_j is the value of the jth counter, for $1 \leq j \leq t$ where C_1, \ldots, C_t are fixed characters.

It was already shown that for finite-turn Turing machines augmented by reversal-bounded counters, all store languages can be accepted by NCM machines:

Proposition 6. [9] $\mathcal{S}(\mathsf{ftNTCM}) \subseteq \mathcal{L}(\mathsf{NCM})$.

But it remains open as to what occurs with finite-visit and finite-crossing Turing machines augmented by counters. We see that a similar construction to that of Theorem 4 holds. For this, we need two-way nondeterministic reversal-bounded multi-counter machines, denoted by 2NCM [12].

Proposition 7. $\mathcal{S}(\mathsf{fcNTCM}) \subseteq \mathcal{L}(\mathsf{NCM})$ and $\mathcal{S}(\mathsf{fvNTCM}) \subseteq \mathcal{L}(\mathsf{NCM})$.

PROOF. We describe a modification of the construction of Theorem 4. Let M be an fcNTCM with t counters. Instead of building a 2NFA M', we build a 2NCM with 3t counters. We refer to the counters of M' by the names $c_1, \ldots, c_t, d_1, \ldots, d_t, e_1, \ldots, e_t$. The machine operates just as in the simulation in the proof of Theorem 4, except the following:

- In point 1, track 0 of the input word contains a word in $Q\Gamma^*\overline{\Gamma}\Gamma^*C_1^*\cdots C_t^*$, *i.e.*, a word that could possibly be in S(M) after applying the gsm (that also fixes letters of C_1,\ldots,C_t that appear in the store language) mentioned above. Namely, track 0 contains exactly one marked symbol.
- In point 6, it simulates the transitions of M by using counters c_1, \ldots, c_t of M' to simulate the counters of M faithfully until it hits the first marked symbol. Then in the nondeterministically guessed position where the marked symbol and state in track 0 matches the current worktape cell contents and state, it copies c_j to both d_j and e_j for each j, $1 \le j \le t$ (so that now d_j and e_j have identical contents to what was in c_j , and c_j is now 0). It then continues the simulation using d_j , for each j until it verifies that M accepts. Finally, M' verifies that the number of each of the characters C_i at the end of the input word is equal to the value of the corresponding counter e_i .

It follows that S(M) = g(L(M')) where g is the gsm. However, before applying g to L(M'), we first apply another construction to M', which is a 2NCM. Notice that M' satisfies the property that for every $x \in L(M')$, every accepting computation of x crosses the boundary of each pair of adjacent cells of the **input tape** a bounded number of times. This restriction of 2NCM is a known model referred to by the name finite-crossing two-way nondeterministic reversal-bounded multi-counter machines [27]. Furthermore, it is known that each such machine can be converted to a one-way nondeterministic reversal-bounded multi-counter machine [27] that accepts the same language. Therefore, M' can be converted to a (one-way) NCM M'' that accepts L(M'). Also, $\mathcal{L}(NCM)$ is closed under gsms [12] and therefore $g(L(M'')) = S(M) \in \mathcal{L}(NCM)$.

As with the case without counters, given any $M \in \mathsf{NCM}$, we can build a machine $M' \in \mathsf{ftNTCM}$ (resp. fcNTCM , fvNTCM) that copies the input to the worktape, switches to some new fixed state q, goes back to the beginning of the worktape, then simulates M by reading the worktape to simulate the reading of the input, and using the counters faithfully. Then if we look at $S_q(M') = \{u \mid qu \not \in S(M')\}$, this is exactly L(M). Again, considering any such class of machines \mathcal{M} and letting $\mathcal{S}_{\text{state}}(\mathcal{M})$ be the family of all $S_q(M)$ where $M \in \mathcal{M}$, we get the following:

$$\mathbf{Corollary~8.~~} \mathcal{S}_{\mathrm{state}}(\mathsf{ftNTCM}) = \mathcal{S}_{\mathrm{state}}(\mathsf{fvNTCM}) = \mathcal{S}_{\mathrm{state}}(\mathsf{fcNTCM}) = \mathcal{L}(\mathsf{NCM}).$$

Theorem 4 and Proposition 7 immediately provide several applications using results in [9, 10]. First, denote the deterministic machines in fcNTM (respectively fvNTM) by fcDTM (respectively fvDTM). In this case, a machine is deterministic if given any state q, worktape symbol x, and $a \in \Sigma \cup \{ \triangleleft \}$, there is at most one transition in the transition relation from q, with worktape symbol x, and on either input symbol a or a.

Proposition 9. The following are true:

 \bullet $\mathcal{L}(\mathsf{fcDTM})$ and $\mathcal{L}(\mathsf{fvDTM})$ are closed under right quotient by regular languages.

- For any machine M in any of fcNTM, fvNTM and regular sets of store configurations $C \in \mathsf{REG}$, then both $\mathsf{post}_M^*(C)$ and $\mathsf{pre}_M^*(C)$ are regular languages, and a finite automaton can be constructed to accept each of them.
- For any machine M in any of fcNTCM, fvNTCM and sets of store configurations $C \in \mathcal{L}(\mathsf{NCM})$, then both $\mathsf{post}_M^*(C)$ and $\mathsf{pre}_M^*(C)$ are in $\mathcal{L}(\mathsf{NCM})$, and an NCM can be constructed to accept each of them.
- For any of fcNTM, fvNTM, fcNTCM, fvNTCM, the common configurations problem is decidable.

PROOF. For the first point, right quotient was studied in [9]. There, a data store was called readable if it is possible for such a machine to have a "stay" that can be executed from any configuration (i.e. any machine of that type could have an instruction designed to not to change its store; e.g. pushdown automata have a stay instruction that does not change its contents, and pushdown automata are allowed to define their transitions to use it from any state); and at any point, if the store contents are represented by y, it is possible to switch to a configuration where y can be read one letter at a time, either from left-to-right (like a queue, which can repeatedly dequeue) or right-to-left (like a pushdown, which can repeatedly pop). Certainly both a finite-crossing worktape and a finite-visit worktape satisfy this property as they can read the contents until hitting blanks. Then given any readable store types, if we consider the class of all one-way nondeterministic machines \mathcal{M} with those readable store types where $\mathcal{S}(\mathcal{M}) \subseteq \mathsf{REG}$, and we let \mathcal{M}_D be the deterministic machines in \mathcal{M} , then $\mathcal{L}(\mathcal{M}_D)$ are closed under right quotient with regular languages. Hence the first point follows.

For the second and third points, we need two technical properties studied in [10] — they are used to establish the connection between store languages and the pre* and post* operations. We say that a machine model \mathcal{M} can be loaded by a set of configurations C, if for all $M \in \mathcal{M}$ with state set Q and $C \subseteq \operatorname{conf}(M)$, there is a machine $M' \in \mathcal{M}$ with $Q' \supseteq Q$ that, on input $q\gamma$ \$x\$ where $c = q\gamma \in \operatorname{conf}(M), q \in Q, x \in \Sigma^*$, and \$ is a new symbol, operates as follows:

- 1. M' reads $c = q\gamma$ while using states in $Q' \setminus Q$ and initializing the store configuration to γ .
- 2. After reading the first \$, it switches to state q (the first state of Q hit) and configuration c if $c \in C$, and switches to a unique state $q_N \in Q' \setminus Q$ if $c \notin C$.
- 3. From c, M' simulates M on x if and only if $c \in C$, as M is defined on states of Q.

It is said that \mathcal{M} can be loaded by sets from some family \mathcal{L} if, for all $M \in \mathcal{M}$ and $C \subseteq \text{conf}(M)$ with $C \in \mathcal{L}$, then \mathcal{M} can be loaded by C.

We say that a machine model \mathcal{M} can be *unloaded* by a set of configurations C if, for all $M \in \mathcal{M}$ with state set Q and $C \subseteq \operatorname{conf}(M)$, there is a machine $M' \in \mathcal{M}$ that, on input $c\$x\$, c = q\gamma \in \operatorname{conf}(M), q \in Q, x \in \Sigma^*$, operates as follows:

- 1. M' reads $q\gamma$ using states not in Q while putting γ on the stores.
- 2. Upon reading the first \$, it switches to state q and configuration c.
- 3. It then simulates M on x starting from c.
- 4. Upon reading the second \$, it verifies that the current configuration of M' is in C and accepts only in this case.

It is said that \mathcal{M} can be unloaded by sets from a family \mathcal{L} if, for all $M \in \mathcal{M}$ and $C \subseteq \text{conf}(M)$ with $C \in \mathcal{L}$, then C can be unloaded by \mathcal{M} .

We will show that fcNTCM (respectively fvNTCM) can be loaded and unloaded by sets from $\mathcal{L}(\mathsf{NCM})$ (this is already known for ftNTCM, whose proof is essentially the same). The proofs for fcNTM, fvNTM, and ftNTM being loaded and unloaded by regular languages are simpler. First we will show it can be loaded by sets from $\mathcal{L}(\mathsf{NCM})$. Consider an fvNTCM $M = (Q, \Sigma, \Gamma, q_0, F)$ with t counters, and consider a language C

accepted by an NCM M_C with t_C counters. From this, we will describe how to build a C-loaded version $M' \in \mathsf{fvNTCM}$ of M with $t + t_C$ counters. Throughout the simulation, the first t counters will be used to simulate the counters of M, while the last t_C counters will be used to simulate M_C . The input will be verified to be of the form $q\gamma C_1^{i_1} \cdots C_t^{i_t} \$x\$$ where $c = q\gamma C_1^{i_1} \cdots C_t^{i_t} \in \mathsf{conf}(M), q \in Q, \gamma \in \Gamma^*, x \in \Sigma^*$, which is processed as follows:

- 1. When reading $q\gamma$, M' puts γ on the worktape while marking the position of the read/write head.
- 2. When reading $C_1^{i_1} \cdots C_t^{i_t}$, M' puts i_j on counter j.
- 3. In parallel to executing steps 1 and 2, M' verifies that $c \in C$ using the last t_C counters.
- 4. If $c \in C$, then it returns to the position of the encoded read/write head on the worktape and switches to state q (if $c \notin C$, it rejects). From this configuration, M' simulates M on x from q.

Hence, fcNTCM can be loaded by sets from $\mathcal{L}(NCM)$.

For the unloaded case, again let $M=(Q,\Sigma,\Gamma,q_0,F)$ be an fvNTCM with t counters, and consider an NCM M_C with t_C counters accepting C. From this, we can build a C-unloaded version $M'\in \mathsf{fvNTCM}$ of M with $t+t_C$ counters. Again, throughout the simulation, the first t counters will be used to simulate the counters of M, while the last t_C counters will be used to simulate M_C . The input will be verified to be of the form $q\gamma C_1^{i_1}\cdots C_t^{i_t}\$x\$$ where $c=q\gamma C_1^{i_1}\cdots C_t^{i_t}\in \mathsf{conf}(M), q\in Q, \gamma\in \Gamma^*, x\in \Sigma^*$, which is processed as follows:

- 1. When reading $q\gamma$, it puts γ on the worktape while marking the read/write head.
- 2. When reading $C_1^{i_1} \cdots C_t^{i_t}$, M' puts i_i on counter j.
- 3. At the first \$, it then returns to the correct first position of the read/write head, and switches to q and configuration c.
- 4. From there, it simulates M starting at q on x.
- 5. When it hits the second \$, it can verify that the current configuration is in C by examining the current state p, scanning the current worktape γ' from left-to-right, and the current counter contents i'_1, \ldots, i'_t to verify that $p\gamma'C_t^{i'_1}\cdots C_t^{i'_t} \in C$.

Therefore, fcNTCM can be unloaded by sets from $\mathcal{L}(NCM)$.

From Theorem 20 of [10], it follows that the third property is true.

For the last property, Proposition 24 of [10] says that for any machine model \mathcal{M} and language family \mathcal{L} such that:

- $\mathcal{S}(\mathcal{M}) \subseteq \mathcal{L}$ with an effective construction,
- L has a decidable emptiness problem,
- and \mathcal{L} is effectively closed under intersection,

then \mathcal{M} has a decidable common configurations problem. Both REG [11] and $\mathcal{L}(\mathsf{NCM})$ [12] have a decidable emptiness problem and are closed under intersection. The result follows from Theorem 4 and Proposition 7.

The results in Proposition 9 were previously unknown. The closure of the deterministic classes under right quotient by regular languages is interesting from a theoretical perspective. Furthermore, the result that $\operatorname{pre}_M^*(C)$ and $\operatorname{post}^*(C)$ can always be accepted by a finite automaton when M is in either fcNTM or fvNTM and $C \in \mathsf{REG}$, is interesting towards the areas of model checking, reachability, and verification (similarly to the analogous result on pushdown automata [16]), as is the decidability of the common configuration problem for all of these models even with reversal-bounded counters.

4. Decidability Questions Regarding Turing Machine Restrictions

Even though we now know that the store language of every ftNTM, fvNTM, fcNTM machine is regular, the question arises of whether we can determine if a given NTM is an ftNTM (respectively fvNTM, fcNTM) in the first place (which would guarantee that it has a regular store language). We prove next that this cannot be determined in general. The following result is shown for deterministic Turing machines (denoted by DTM), and therefore it is also true for NTMs.

Proposition 10. It is undecidable whether a DTM is 0-turn (resp., 0-crossing, 1-visit).

PROOF. We will show the 0-turn case. Let M be a single-tape two-way deterministic Turing machine (with a combined input and two-way worktape as commonly defined [11]) with an initially blank tape. Construct a one-way DTM M' with input alphabet $\{a\}$ and one read/write worktape. Then M' on input x operates as follows:

- If x is λ , M' accepts without using the worktape tape.
- If $x = a^n, n \ge 1$, M' first simulates M on the worktape. There are two cases. Case 1: M does not halt. Then M' does not halt also. Case 2: M halts. Then M' makes n dummy turns and accepts x.

Clearly, if M does not halt on blank tape, M' would only accept $x = \lambda$ with 0-turns. If M halts, M' would accept not only a^n with n additional turns.

It follows that M' is 0-turn if and only if M does not halt on blank tape. The result follows from the undecidability of the halting problem for DTMs on an initially blank tape. The proof is identical for 0-crossings (except in case 2 above, M' would make n dummy crosses). For 1-visit, M' makes one visit immediately with the first move, and in case 2, M' makes n dummy visits.

Corollary 11. It is undecidable whether a DTM is an ftNTM (respectively, fvNTM, fcNTM).

This follows from the previous proposition, since M' is finite-turn if and only if it is 0-turn.

The next two propositions study decidability of two related problems: 1) Given an NTM M, determining whether there is a word x whereby all accepting computations on x are k-turn (respectively k-visit, k-crossing), and 2) determining whether there is a word x whereby there exists an accepting computation on x that is k-turn (respectively k-visit, k-crossing). We will see that the first problem is undecidable, while the second is decidable. This second property implies that both problems are decidable for deterministic (or unambiguous, meaning there is at most one accepting computation on each word) machines.

Proposition 12. It is undecidable, given an NTM M and $k \ge 0$, whether there is an an input $x \in L(M)$ such that all accepting computations on x are k-turn (respectively, k-crossing, $k \ge 1$ -visit).

PROOF. Again, it is sufficient to show the k-turn case.

Let M be a single-tape deterministic Turing machine (with a combined input and two-way worktape [11]) with an initially blank tape. Construct a one-way NTM M' with input alphabet $\{a\}$. First, M' when given input x makes k turns on the worktape. Then it nondeterministically does one of the following:

- 1. M' accepts x if x = a (taking 0 turns).
- 2. M' simulates M on the worktape and if M halts, M' makes a nondeterministically guessed number of turns and accepts x if x = a and rejects all other strings.

Clearly, M' only accepts a, and not all accepting computations of M on x are within k-turns if and only if M halts on blank tape.

Corollary 13. It is undecidable, given an NTM and $k \ge 0$, whether there is an an input x such that only a finite number of accepting computations on x are not k-turn (respectively not k-visit, not k-crossing).

This follows from the proof of the proposition above, since if M halts, M' has an unbounded number of accepting computations with arbitrarily many turns.

Now we examine the second problem.

Proposition 14. It is decidable, given an NTM M and $k \ge 0$, whether there is an an input x such that there is an accepting computation on x that is k-turn (respectively k-crossing, $k \ge 1$ -visit). Furthermore, an ftNTM (respectively fvNTM, fcNTM) can be effectively constructed to accept all words w where there is an accepting computation of M on w that is k-turn (respectively k-visit, k-crossing), and whereby every accepting computation is k-turn (respectively k-visit, k-crossing).

Both statements are also true for NTCM by constructing an ftNTCM (respectively fvNTCM, fcNTCM).

PROOF. We prove the k-crossing case as it is the most complicated. Notice that if an fcNTM (respectively fcNTCM) can be constructed to accept all words w where there is an accepting computation of w by M that is k-crossing, then this implies that the problem is decidable, because emptiness is decidable for fcNTCM [26], and this machine is not empty if and only if M accepts an input that is k-crossing.

Let M be an NTCM. We construct another NTCM M' that simulates M as follows. Each cell of the worktape keeps either a worktape letter of M or a non-negative integer, in an alternating fashion (between every two cells labelled by letters is a single non-negative integer). Then M' simulates M, but as it crosses to the right say, M' moves right to an integer m which it increases to m+1 before proceeding to the next cell. M' also stores a variable called S in its finite control which stores the maximum integer stored in any cell so far. All of these values only store integers up to k.

When M accepts, M' accepts if $S \leq k$; else it rejects. Therefore, if M accepts w with at most k crossings, then so does M', and every accepting computation of M with at most k crossings has a corresponding at most k crossing accepting computation of M'. Any accepting computation using more than k crossings does not have a corresponding accepting computation of M'.

For the k-turn case, we only need to keep track of the number of turns so far in S. For the k-visit case, we need to store the number of visits to each cell as a second component of each cell along with the maximum visit number.

To note, Proposition 14 does not contradict Proposition 12 because in Proposition 12, there could be additional accepting computations that are not k-turn, but the machine in Proposition 14 has accepting computations corresponding with all accepting computations of M with at most k turns.

An NTCM M is considered unambiguous, if for each $w \in \Sigma^*$, M has at most one accepting computation on w. The proof of the next result is similar to the one above, noting that since the machine is unambiguous, for any input, there is at most one accepting computation. To note, every deterministic machine is unambiguous. This corollary is meant to contrast Proposition 12.

Corollary 15. It is decidable, given an unambiguous NTCM M and k, whether there is an an input x such that the accepting computation on x is k-turn (resp., k-crossing, k-visit).

It is also possible to construct an explicit example of a string x with the desired property by testing non-emptiness, and then if it is non-empty, testing non-emptiness of $L(M') \cap \{w\}$ (M' in proof above) for all w from the shortest until one accepts.

5. Discussion Regarding Definitions of Turing Machine Restrictions

Consider the following example of a non-finite-crossing NTM.

Example 3. Consider an NTM $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where $\Sigma = \{a, b\}$, $\Gamma = \{a, b, a', b', \$, \sqcup\}$, $F = \{q_7\}$, and δ contains the following transitions, for all $x \in \{a, b\}$ and $y \in \{a, b, \$\}$:

$$(q_{0}, x, \sqcup, q_{0}, x, R), (q_{0}, \triangleleft, \sqcup, q_{1}, \$, L), (q_{1}, \lambda, x, q_{1}, x, L), (q_{1}, \lambda, \sqcup, q_{2}, \sqcup, R), (q_{2}, \lambda, a, q_{3}, a', R), (q_{2}, \lambda, b, q_{4}, b', R), \\ (q_{3}, \lambda, y, q_{3}, y, R), (q_{3}, \lambda, \sqcup, q_{5}, a, L), (q_{4}, \lambda, y, q_{4}, y, R), (q_{4}, \lambda, \sqcup, q_{5}, b, L), (q_{5}, \lambda, y, q_{5}, y, L), (q_{5}, \lambda, x', q_{6}, x, R), \\ (q_{6}, \lambda, x, q_{2}, x, S), (q_{6}, \lambda, \$, q_{7}, \$, S).$$

This machine accepts the simple regular language Σ^+ , and when given an input w, produces w\$w on the tape. It operates by first copying the input string to the tape followed by \$, then moving to the leftmost symbol on the tape. It marks the symbol using a prime symbol (') and remembers it in the state, moves right to the first blank, writes the symbol, moves left to the marked symbol, erases the mark, moves right, then repeats until all symbols before \$ have been copied.

Notice that when given an input of length n, performing the duplication on the tape requires the read/write head to cross the cell containing \$ at least 2n times. Thus this machine is not finite-crossing.

Since this machine only reaches the final state q_7 after completing its operation, the only store configurations involving q_7 are those in which the tape contains a duplicated string, i.e., if we intersect S(M) with the regular language $q_7\Gamma^*$, we obtain the language

$$\{q_7w\$ \downarrow w \mid w \in \Sigma^+\}$$

which is not regular. Thus S(M) is not regular, since regular languages are closed under intersection [11].

We will refer to this example in discussing several points below.

In the definition of k-crossing machines we use in this paper, a machine is k-crossing if every accepting computation is k-crossing. Elsewhere in the literature, it was defined that a machine M is k-crossing if, for every $w \in L(M)$, there is some accepting computation that is k-crossing [6]. Here, we call this notion weakly k-crossing (and weakly finite-crossing). The languages accepted by k-crossing and weakly k-crossing NTMs coincide by Proposition 14. But there is a large difference between the store languages of the two models. The following proof is reminiscent of Example 3. We use the definition of $S_q(M)$ and $S_{\text{state}}(\mathcal{M})$ from Section 3. We add "weak" as a superscript to fcNTM to represent weakly finite-crossing machines.

Proposition 16. $S_{\text{state}}(\text{fcNTM}^{\text{weak}}) = \text{RE}$. Furthermore, there are non-recursive languages that are store languages of weakly 0-crossing NTMs.

PROOF. Let M be a single-tape deterministic Turing machine (one combined input and read/write worktape) accepting L over Σ accepting an arbitrary RE language. Build M' as follows: M' does one of two things nondeterministically on input $w \in \Sigma^*$. Either 1) M' simply reads w and accepts, or 2) M' copies w to the worktape, switches to a new special state q (which it only enters once), moves back to the first worktape cell before w, and then simulates M on w, accepting if M' accepts. Thus, $L(M') = \Sigma^*$, and M' is weakly 0-crossing because for every $w \in L(M')$, there is some accepting computation (of type 1) that is 0-crossing. But if one looks at $S_q(M')$, this is equal to L(M).

For the second point, let L be a non-recursive language accepted by a single-tape deterministic Turing machine M. By following the procedure of the first paragraph, we can construct M such that $S_q(M') = L(M)$. Assume S(M') were recursive. But then $(q^{-1}S(M'))(A)^{-1} = S_q(M')$ would be as well since the recursive languages are closed under left and right quotient with a symbol, a contradiction.

This illustrates that even if two classes of machines accept the same family of languages, this does not imply that their classes of store languages are equal. Indeed, in this case, finite-crossing NTM and weakly finite-crossing NTM accept the same family of languages. However, the first class of machines always have regular store languages, while the second has non-recursive store languages. Notice that the machine M' constructed in this proof uses nondeterminism. For unambiguous NTM, where each word has at most one accepting computation, there is no difference in store languages between the two notions, and they are always regular in both cases.

6. Non-Bounded Visit and Crossing Turing Machines

We have shown that for an NTM, a constant bound on turns, crossings, or visits always produces a regular store language. From Example 3, we can see that given an input word of length n, a bound of O(n) on the number of turns (respectively visits, crossings) can produce non-regular store languages. This motivates us to consider other bounds on turns, crossings, and visits based on the length of the input to the Turing machine.

Example 4. Consider the Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where $\Sigma = \{a\}$, $\Gamma = \{a, b, \sqcup\}$, $F = \{q_7\}$, and δ contains the following transitions:

$$(q_0, a, \sqcup, q_0, a, R), (q_0, \triangleleft, \sqcup, q_1, \sqcup, L), (q_1, \lambda, a, q_1, a, L), (q_1, \lambda, \sqcup, q_2, \sqcup, R), (q_2, \lambda, a, q_3, b, R), (q_2, \lambda, b, q_2, b, R), \\ (q_3, \lambda, a, q_4, a, R), (q_3, \lambda, b, q_3, b, R), (q_3, \lambda, \sqcup, q_7, \sqcup, L), (q_4, \lambda, a, q_5, b, R), (q_4, \lambda, b, q_4, b, R), (q_4, \lambda, \sqcup, q_6, \sqcup, L), \\ (q_5, \lambda, a, q_4, a, R), (q_5, \lambda, b, q_5, b, R), (q_6, \lambda, a, q_6, a, L), (q_6, \lambda, b, q_6, b, L), (q_6, \lambda, \sqcup, q_2, \sqcup, R).$$

This machine accepts the language $\{a^{2^k} \mid k > 0\}$. It operates by copying the input to the tape, moving the head back to the leftmost symbol, then sweeping from left to right while changing every other a to b. After reaching the rightmost symbol, if the number of a's seen was odd and greater than 1, then the length of the input could not have been a power of two. If a single a was seen, then the length of the input must have been a power of two, so M accepts. If the number of a's seen was even, then the head is returned to the leftmost symbol and another sweep is performed.

Since each sweep halves the number of a's on the tape, and M only accepts when all a's have been changed to b's, this machine makes 2k + 2 turns for input of length 2^k . Hence if the length of the input is n, then the number of turns is $O(\log n)$. Since this machine always sweeps the full length of the tape after a turn and has no transitions that stay on a tape cell, the number of crossings and visits are also both $O(\log n)$.

This machine only reaches the final state q_7 after changing every a on the tape to b, so if we intersect the store language S(M) with the regular language $q_7\Gamma^*$, we obtain the language

$$\{q_7b^{2^k} \ | \ | \ k > 0\}$$

which is not regular. Thus S(M) is not regular (nor semilinear [28]).

From this example, it is clear that for a Turing machine given input of length n, a bound of $O(\log n)$ on turns, crossings, or visits is not sufficient to guarantee that the store language is regular. This result can be strengthened as follows:

Proposition 17. Let $f: \mathbb{N} \to \mathbb{N}$ be an unbounded function, and let M be an NTM which is not finite-turn, but on every input of length n, it makes at most f(n) turns. Then there exists an NTM M', which on every input of length n makes at most O(f(n)) turns, but S(M') is not regular.

PROOF. Choose an arbitrary number k_0 . Since M is not finite-turn, it has some accepting computation on a string w, |w| = n, which makes at least k_0 turns. Denote the number of turns in this computation by k, where $k_0 \le k \le f(n)$. Now construct M' as the following: M' has a two-track worktape, and it simulates a computation of M on w using the first track, but whenever M would make a turn, M' puts a new symbol x on the second track. However, if this cell already contains the symbol x, M' changes this symbol to a new symbol y, continues in the same direction as it was moving previously until it finds a cell with an empty second track, puts the symbol x there, turns around, moves back to the cell with symbol y, changes this symbol back to x, and continues simulating the computation of M.

Note that after the simulation of M ends, the second track of the tape of M' contains exactly k symbols x; and M' has made k turns so far.

M' now moves its head to the left end of the tape (using up to one more turn), and sweeps the tape k more times. In every sweep, the machine replaces one of the symbols x on the second track by the symbol y, and appends one cell with the symbol z in the second track at the very end of the tape. This continues until all symbols x are erased, taking 2k more turns. Finally, M' appends one more symbol # to the end of the tape and accepts. (This is a slightly modified variant of the construction from Example 3.)

Observe that the computation of M' has used 3k plus a constant number of turns, and since $k \leq f(n)$, the number of turns is in O(f(n)). Further, the second track of the tape contains the string $x^k y^k \#$, potentially interleaved with blank spaces.

Let h be a homomorphism which preserves the second track of the tape, and erases all tape cells which have an empty second track, and all non-tape symbols (state, head). Thus the image of the last configuration of M' under h is $x^ky^k\#$.

If S(M') was regular, then so should be L = (h(S(M'))) intersected with $x^*y^*\#$. However, by the above construction, since we were able to choose an arbitrarily large k, we obtain an infinite number of strings $x^ky^k\#$ which should be in L. On the other hand, no string where the number of symbols x and y differ is in L, as the symbol # was only appended in the last step of M'. This is a contradiction with regularity of L, and thus also S(M').

What this means is that for any function f, as long as some machine which makes f(n) turns on an input of length n exists, then O(f(n)) turns is not a sufficient bound to guarantee a regular store language. Example 4 then demonstrates an NTM which makes $\log(n)$ turns on an input of length n.

An open question is: does there exist an unbounded function f, such that every f(n)-turn-bounded machine is also constant-turn-bounded? If this was answered negatively, it would mean that the only possible limit on number of turns which guarantees a regular store language is constant.

7. Store Languages of One-Way Multi-Counter Machines

It is known that the store language of every NCM is in $\mathcal{L}(\mathsf{DCM})$, where DCM are the deterministic machines in NCM [9]. But, how complex can the store languages of multi-counter machines get, when we do not start with the assumption that the counters are reversal-bounded? Certainly not all recursively enumerable languages can be store languages of multi-counter machines (even after taking the left quotient by a state) since the store language of every t counter machine over state set Q is a subset of $QC_1^*\cdots C_t^*$, which is a letter-bounded language. It is known that even the simple language Σ^* where $|\Sigma| \geq 2$ is not a bounded language [28], and therefore Σ^* cannot be the store language of a multi-counter machine (even after taking any sort of quotient with the state set). However, we get non-recursive store languages. Let DCOUNTER(t) be the deterministic COUNTER(t) machines.

Proposition 18. S(DCOUNTER(2)) contains non-recursive languages.

PROOF. First, we will note the following well-known strategy. If we have a 2-counter machine with counters called A and B, with a number c in A, and 0 in B, and we have a fixed number d, we can compute dc and ultimately store it in A. We can do this by adding d to B for every decrease of A by 1, and then when A is zero, move the contents of B back to A. Similarly, if we have a number c in A and 0 in B, and we have a fixed number d, we can determine if c is divisible by d, and compute c divided by d if it is. This can be done by adding 1 to B for every decrease of d by A. If it is not divisible, then it is also clearly possible to recover c.

It is known that that there are non-recursive unary languages that can be accepted by one-way deterministic 2-counter machines [11]. Let L be such a language accepted by a machine M, where the counters are called X and Y. Construct a deterministic 2-counter machine M' with counters called A and B. Then M' on unary input $a^n \triangleleft$ first computes on A the number 2^n using the doubling strategy above for every input letter read. When M' reaches the end marker, M' switches to special state p, and then simulates M in the following altered fashion. After simulating each move of M, counter A of M' will be of the form $2^{c_1}3^{c_2}5^{c_3}$, where c_1 is the remaining number of a's to read from the simulated computation, c_2 is the current value in the simulated counter X, and c_3 is the current value in the simulated counter Y.

Before simulating each transition of M, we can test whether there are any input letters left to read by checking if A is divisible by 2, and we can test whether any of the two simulated counters X or Y are non-zero by checking if A is divisible by 3 and by 5. Then M' can choose the appropriate transition, and can simulate the reading of an a with a division by 2, and an increase to X (resp. Y) by multiplying by 3 (resp. 5), and can simulate a decrease to X (resp. Y) by dividing by 3 (resp. 5). Then M' accepts if M accepts. Notice that $S(M') \cap pC_1^*C_2^* = pL'$, where $L' = \{a^{2^n} \mid a^n \in L\}$. Clearly, pL' is non-recursive, since recursive languages are closed under left quotient with a symbol, and L can easily be computed from L'. \square

The above is true for deterministic machines. For nondeterministic machines, notice the following:

Corollary 19. There are non-recursive store languages of machines in COUNTER(2) accepting $\{\lambda\}$.

This is evident from the proof above as we can start by using nondeterminism to guess the value n.

8. Store Languages of Two-Way Machines

There has been comparatively little study of the store languages of two-way machines. Two-way machines can be defined similarly to one-way machines, except each transition reads a letter from Σ or the left or right end marker \triangleright or \triangleleft , and each transition has a final component that controls whether the input head position moves to the left by one cell, stays in the same position, or moves to the right by one cell. Configurations also include the input head position. We define acceptance by hitting a final state and falling off the input tape past the right end marker. Please refer to the formal definitions for two-way machines [9]. We use the same model names as with one-way machines, but preface each class name with the number 2 (like 2NCM). As in the proof of Proposition 7, two-way input tape is called *finite-crossing* if in every accepting computation, the input head crosses the boundary of any two adjacent cells of the input tape at most c times for some c. In the next results, we also need the well-known class of one-way pushdown automata (resp. augmented with reversal-bounded counters) NPDA (resp. NPCM), and replace N with D for the deterministic restriction.

The only study of store languages of two-way machines that has been done in the past was in Section 4 of [9]. There, it was shown that all store languages of finite-crossing 2NCMs are in $\mathcal{L}(\mathsf{DCM})$ (that is, they can be accepted by one-way DCM machines). If the input is not finite crossing, it was shown that there are 2DCM machines M with one 1-reversal-bounded counters (called 2DCM(1,1)) accepting bounded languages such that the store languages are not in $\mathcal{L}(\mathsf{NPCM})$. Similarly, there are 2NCM(1,1) machines M accepting unary languages such that S(M) are not in $\mathcal{L}(\mathsf{NPCM})$ nor semilinear. Furthermore, some general connections between store languages of one-way and two-way acceptors was made. If a family of one-way acceptors \mathcal{M}_1 , and \mathcal{M}_2 is the corresponding family of two-way acceptors, then \mathcal{M}_1 has store languages contained in some family \mathcal{L} that is closed under homomorphism if and only if \mathcal{M}_2 machines accepting finite languages have store languages contained in \mathcal{L} .

In the next proposition, a DPDA^{1t} is a DPDA that is one-turn, which means that it does not push after popping, and finite-crossing refers to crossings on the two-way input tape.

Proposition 20. There are non-recursive store languages of $2\mathsf{DCOUNTER}(1)$ machines accepting bounded languages. Also, there are non-recursive store languages of machines in both finite-crossing $2\mathsf{DCOUNTER}(1)$ and finite-crossing $2\mathsf{DPDA}^{1t}$.

PROOF. To start, we show the result for 2DCOUNTER(1). Let M be the 2-counter machine accepting a unary non-recursive language L over some alphabet $\{a\}$ in the proof of Proposition 18. Let # be a new symbol. We construct a two-way deterministic machine M' with one counter C which, when given an input $a^n\#^d$ (for some $n,d\geq 1$), first converts n to 2^n and stores it in counter C using the doubling strategy (by using the two-way input to act as another counter). This can be done if d is large enough. Then it simulates M as in Proposition 18, using C and the input tape, rejecting if d is not large enough. Notice that M' accepts a bounded language that is a subset of $a^*\#^*$. For the case where M' is finite-crossing, a similar construction is used which does not accept a bounded language. Let B be a new symbol. We construct a two-way deterministic machine M' with one counter C which, when given an input $a^n\#^{d_1}B \cdots \#^{d_m}B$ (for some $n, d_1, \ldots, d_m \geq 1$), first converts n to 2^n and stores it in counter C using the doubling strategy (by using the two-way input to act as another counter). This can be done if d is large enough. M' uses a new block $\#^{d_i}B$ in simulating a counter every time it goes from increasing to decreasing. Clearly M' is c-crossing for some c.

For 2DPDA, we use an arbitrary non-recursive language L accepted by a single-tape DTM Z with initial state q_0 that can only accept after an even number of configurations. On input $w, w \in L$ if and only if there exists $x = ID_1 \# ID_3 \# \cdots \# ID_{2k-1} \# \# ID_{2k}^R \# ID_{2k-2}^R \# \cdots \# ID_2^R$ where $ID_1 \vdash \cdots \vdash ID_{2k}$ is an accepting computation of Z on w and $ID_1 = q_0w$. A 2DPDA M' with pushdown alphabet Γ can be constructed which on input x, pushes w onto the pushdown, enters a special state p (which it only enters once). Then it continues to push $ID_1 \# ID_1 \# ID_3 \# ID_3 \# \cdots \# ID_{2k-1} \# ID_{2k-1}$ (doubling each configuration which it can do with the finite-crossing input tape) onto the pushdown until it reads # #. Then for each i odd on the pushdown, as it is being popped, it is possible to verify that ID_i can derive ID_{i+1} and ID_{i-1} can derive ID_i by scanning each configuration after # # at most two times. Thus, $S(M') \cap p\Gamma^* = pL$, which is not recursive, otherwise L would be as well.

Note that for the 2DPDA, it is also possible to build M' with two turns on the input and two turns on the pushdown by first verifying that odd configurations produce even configurations, and then verifying that even configurations produce odd configurations.

We showed in Proposition 20 that there exist non-recursive store languages of 2DCOUNTER(1) machines. In the proof, the machines accept bounded languages (subsets of $a^*\#^*$). A similar result is given in Proposition 20, where the machines are finite-crossing, but in that case, the machines accept non-bounded languages. However, when the machines are finite-crossing and accept bounded languages, we can show that their store languages are recursive. In fact, we prove a much stronger result. We will need Theorem 2 in [29].

Proposition 21. [29] A finite-crossing 2DPCM accepting a bounded language is effectively equivalent to a DCM. Hence, its emptiness problem is decidable.

Notice here that the DCM constructed is a one-way machine.

Proposition 22. The store language of every finite-crossing 2DPCM accepting a bounded language is recursive.

PROOF. Let M be a finite-crossing 2DPCM with t reversal-bounded counters accepting a bounded language $L \subseteq \Sigma^*$, for some alphabet Σ . Every string in the store language S(M) is of the form

$$y = pzC_1^{n_1} \cdots C_t^{n_t},$$

where p is the current state, z is the content of the stack, and n_1, \ldots, n_t are the values of the counters.

We describe an algorithm to determine whether a given string y is in S(M). Construct a finite-crossing 2DPCM M_y with a pushdown stack P and counters C_1, \ldots, C_t . M_y stores the target configuration y in its finite control.

On input w, M_y simulates M on w, maintaining in its finite control the simulated stack content as long as the height is less than or equal to |z|, and likewise keeping track of each counter C_i as long as its value does not exceed n_i . It only begins using the actual pushdown stack P when the stack height exceeds |z|, and uses counter C_i only if its value exceeds n_i .

If, during this simulation, M reaches the configuration y, then M_y records this in its control. It then continues simulating M, and accepts if and only if M accepts w. If M never enters the configuration y, then M_y rejects (regardless of whether M accepts or not).

Clearly, the counters of M_y are still reversal-bounded. Thus, $y \in S(M)$ if and only if the language $L(M_y)$ is non-empty. The result follows since the emptiness problem for 2DPCM accepting bounded languages is decidable.

In Proposition 20, we showed that S(2DCOUNTER(1)) contains non-recursive languages. However, if the counter is reversal-bounded, we have:

Proposition 23. S(2DCM(1)) are recursive, and membership is decidable in S(M) for each $M \in 2DCM(1)$.

PROOF. Let $M \in 2DCM(1)$. Every string in S(M) (assuming S(M) is non-empty) is of the form

$$y = pC_1^d,$$

where p is a state of M and d represents the counter value.

We construct a two-way deterministic 1-counter machine M_y , whose counter is reversal-bounded, such that $y \in S(M)$ if and only if the language $L(M_y)$ is non-empty. The construction of M_y is analogous to that in the proof of Proposition 22.

The result follows since the emptiness problem for 2DCM(1) is decidable [30].

Example 5. Let $L = \{a^ib^j \mid i \geq 4, j \geq 2, i \text{ is divisible by } j\}$. Clearly, L can be accepted by a 2DCM(1) M which reads and stores i in the counter and enters a special state s. Then it makes multiple passes on b^j while decrementing i and checks and accepts if i is a multiple of j. Note that M makes only one counter reversal. Then S(M) is not even a context-free language; otherwise, $S(M) \cap sC_1^+ = \{sC_1^i \mid i \geq 4, j \text{ is composite}\}$ would be context-free.

We do not know if Proposition 23 holds when M is nondeterministic. However, if M accepts a bounded language, Proposition 23 holds, since the emptiness problem for 2NCM(1) that accepts bounded languages is decidable [31]. In general however, the following holds:

Proposition 24. The membership problem for the store language for machines in 2NCM(1) is decidable if and only if the emptiness problem for NCM(1) is decidable.

PROOF. Let M be in $2\mathsf{NCM}(1)$ and f be the unique accepting state of M, and assume without loss of generality that all counters are zero before switching to f. Then f is in the store language S(M) of M if and only if L(M) is not empty. It follows that if we can decide membership in S(M), then we can decide the emptiness of L(M). Now suppose we can decide the emptiness of L(M). Then, as illustrated in the proof of Proposition 23, we can decide membership in S(M).

Clearly, if the emptiness problem for machines in $2\mathsf{NCM}(1)$ is decidable (hence, the membership problem for store languages is decidable), then the store languages for machines in \mathcal{M} are recursive. However, if the emptiness problem is undecidable, it does not necessarily mean that there are store languages that are not recursive.

The ideas in the proofs of Proposition 24 can be applied to other counter machine models such as 2DCM(2). It is known that the emptiness problem for these machines is undecidable, even when restricted to the special case when the machines accept letter-bounded languages [12].

Proposition 25. Testing membership in S(M) for $M \in 2DCM(2)$ (even where L(M) is letter-bounded) is undecidable.

Remark 1. In general, store languages can sometimes be more complicated than accepted languages, and can sometimes be simpler. The simpler case can be seen from the many cases where the store languages of models that accept more than regular languages only produce regular store languages. This is the case for one-way pushdown automata, and for fcNTM. The more complicated case occurs, for example if M is a 2NPCM or a 2NPDA, and therefore L(M) is recursive. Indeed, to decide whether y is in L(M), construct an NPCM M_y which has y in its state and simulates M on λ input, Then y is L(M) if and only if M_y accepts λ . Since emptiness (hence, membership) for NPCM is decidable, L(M) is recursive. However, store languages of 2NPCM, and even finite-crossing 2DPDA where the pushdown is one-turn, and finite-crossing 2COUNTER(1) are sometimes non-recursive by Proposition 20.

To note that all languages accepted by 2NCM are recursive, using the same argument above with decidability of emptiness for NCM. Further, testing membership in a store language of $M \in 2DCM(2)$ is undecidable by the proposition above. Despite this, it is open whether or not all store languages of 2DCM(2) (or even 2NCM) are recursive. It is possible that membership could be undecidable within a recursive store language.

A two-way input tape is called finite-turn if the input head makes at most c turns (reversals) on the input tape for some c. Clearly, finite-turn is a special case of finite-crossing. We will show that Proposition 20 does not hold if finite-crossing is replaced by finite-turn. We denote by 2NCCM to be the same as 2NPCM but the pushdown is replaced with an unrestricted counter.

Proposition 26. If M is a finite-turn 2NCCM, then S(M) is an effectively computable semilinear set that is in $\mathcal{L}(\mathsf{DCM})$.

PROOF. It was shown in [12] that if M is a finite-turn 2NPCM, then the Parikh image of L(M) is an effectively computable semilinear set. Obviously, this result holds for finite-turn 2NCCM. Let Σ be the input alphabet of M and # be a new symbol. If M has t reversal-bounded counters, then the strings in S(M), if non-empty, are of the form $1^sC_1^{i_1}C_2^{i_2}\cdots C_{t+1}^{i_{t+1}}$, where s is the state, i_1 is the value of the unrestricted counter, and i_2,\ldots,i_{t+1} are the values of the reversal-bounded counters (we assume for convenience that the state set is $\{1,\ldots,m\}$ for some m).

Given M, we can construct a finite-turn 2NCCM M' which, when given input $y = w \# 1^s C_1^{i_1} C_2^{i_2} \cdots C_{t+1}^{i_{t+1}}$, simulates M on w. At some nondeterministically chosen point, M' checks that the current configuration

(i.e., the state and counter values) is represented on the input. So it scans the input and it checks that the state is s, and the counters have values i_1, \ldots, i_{k+1} . If not, it rejects. (Note that M' can restore the values of the counters after checking). Then it continues the simulation of M on w. M' accepts if M accepts w. Then the Parikh image of L(M') is an effectively computable semilinear set S'. It follows, by deleting the components associated with the symbols in Σ and # from the vectors generating S', we get the semilinear set for S(M). Finally, it is indeed known that all bounded-semilinear languages are in $\mathcal{L}(\mathsf{DCM})$ by [29]. \square

9. Conclusions and Future Directions

We have shown that store languages of finite-crossing and finite-visit Turing machines are regular, which was previously known for the less powerful finite-turn Turing machines. These two classes are important as they exactly accept the languages accepted by many types of finite-index grammars. From this, it is proven that the languages accepted by deterministic finite-visit and deterministic finite-crossing Turing machines are closed under right quotient with regular languages. Also it is determined that given a regular set of configurations C, the set of configurations that may follow in zero or more steps from M is regular, and the set of configurations that may lead in zero or more steps to a configuration in C is regular. Furthermore, adding reversal-bounded counters to either type of Turing machine produces store languages that can be accepted by machines with only reversal-bounded counters. This leads to more general reachability results for Turing machines (optionally with reversal-bounded counters). In addition, for either type of Turing machine (optionally with reversal-bounded counters), given two machines of this type, it is decidable whether they have some non-initial configuration in common — a problem that has applications in fault tolerance.

Store languages of two-way machines are further studied, and many classes are shown to either have non-recursive store languages, or at least have store languages with an undecidable membership problem. In particular, store languages of two-way deterministic pushdown automata can be non-recursive.

In the future, it is worth considering if there is yet another larger class of Turing machines with regular store languages, or if there are other unrelated restrictions that can be imposed on Turing machines to produce regular store languages. Examples 3 and 4 present an interesting problem: how complex do the store languages become if we allow for a number of turns, crossings, or visits bounded by the size of the input. There are also many other machine models whose store languages have yet to be studied. A problem for future work is to characterize the store languages possible as a function of the number of turns, crossings, and visits, where it is larger than any constant. Finally, in [32] it was shown that it is decidable, given a finite-turn Turing machine M, whether the set of subwords of L(M) is equal to Σ^* . The key property used there is that the store language of all finite-turn Turing machines are regular. Does a similar property hold for finite-crossing Turing machines, and hence many types of finite-index grammars? Other properties of finite-crossing Turing machines are of interest. For example, it was recently shown that it is decidable to determine whether a given finite-turn Turing machine accepts a bounded language. Is this also true for finite-crossing Turing machines (and hence finite-index grammars)?

References

- [1] S. Greibach, A note on pushdown store automata and regular systems, Proceedings of the American Mathematical Society 18 (1967) 263–268.
- J. Autebert, J. Berstel, L. Boasson, Handbook of Formal Languages, Vol. 1, Springer-Verlag, Berlin, 1997, Ch. Context-Free Languages and Pushdown Automata.
- [3] A. Malcher, K. Meckel, C. Mereghetti, B. Palano, Descriptional complexity of pushdown store languages, in: M. Kutrib, N. Moreira, R. Reis (Eds.), Lecture Notes in Computer Science, Vol. 7386 of Descriptional Complexity of Formal Systems, DCFS 2012, Portugal, Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 209–221.
- [4] V. Geffert, A. Malcher, K. Meckel, C. Mereghetti, B. Palano, Descriptional complexity of pushdown store languages, in: M. Kutrib, N. Moreira, R. Reis (Eds.), Lecture Notes in Computer Science, Vol. 8031 of Descriptional Complexity of Formal Systems, DCFS 2013, London, Canada, July 23-25, Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 90-101.
- [5] S. Ginsburg, S. Greibach, M. Harrison, One-way stack automata, Journal of the ACM 14 (2) (1967) 389–418.
- [6] S. A. Greibach, One way finite visit automata, Theoretical Computer Science 6 (1978) 175–221.
- [7] M. Holzer, M. Kutrib, Flip-pushdown automata: Nondeterminism is better than determinism, in: Z. Ésik, Z. Fülöp (Eds.), Developments in Language Theory, Vol. 2710 of Lecture Notes in Computer Science, 2003, pp. 361–372.

- [8] S. Bensch, J. Björklund, M. Kutrib, Deterministic stack transducers, International Journal of Foundations of Computer Science 28 (05) (2017) 583-601.
- [9] O. Ibarra, I. McQuillan, On store languages of language acceptors, Theoretical Computer Science 745 (2018) 114–132.
- [10] O. H. Ibarra, I. McQuillan, On store languages and applications, Information and Computation 267 (2019) 28–48.
- [11] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
- [12] O. Ibarra, Reversal-bounded multicounter machines and their decision problems, Journal of the ACM 25 (1) (1978) 116–133.
- [13] B. Baker, R. Book, Reversal-bounded multipushdown machines, Journal of Computer and System Sciences 8 (3) (1974) 315–332.
- [14] S. Ginsburg, S. Greibach, Deterministic context free languages, Information and Control 9 (6) (1966) 620-648.
- [15] J. Hopcroft, J. Ullman, Deterministic stack automata and the quotient operator, Journal of Computer and System Sciences 2 (1) (1968) 1–12.
- [16] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: Application to model-checking, in: A. Mazurkiewicz, J. Winkowski (Eds.), CONCUR '97: Concurrency Theory: 8th International Conference Warsaw, Poland, July 1–4, 1997 Proceedings, Vol. 1243 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1997, pp. 135–150.
- [17] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, R. Kemmerer, Counter machines: Decidable properties and applications to verification problems, in: M. Nielsen, B. Rovan (Eds.), Mathematical Foundations of Computer Science 2000: 25th International Symposium, MFCS 2000 Bratislava, Slovakia, August 28 — September 1, 2000 Proceedings, Springer-Verlag, Berlin, Heidelberg, 2000, pp. 426–435.
- [18] M. F. Atig, Global Model Checking of Ordered Multi-Pushdown Systems, in: K. Lodaya, M. Mahajan (Eds.), IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010), Vol. 8 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2010, pp. 216–227.
- [19] A. Seth, Global reachability in bounded phase multi-stack pushdown systems, in: Proceedings of the 22nd International Conference on Computer Aided Verification, CAV'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 615–628.
- [20] M. F. Atig, From multi to single stack automata, in: Proceedings of the 21st International Conference on Concurrency Theory, CONCUR'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 117–131.
- [21] A. Bouajjani, M. Müller-Olm, T. Touili, Regular symbolic analysis of dynamic networks of pushdown systems, in: M. Abadi, L. de Alfaro (Eds.), CONCUR 2005 — Concurrency Theory, CONCUR '08, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 473–487.
- [22] A. Finkel, G. Sutre, Decidability of reachability problems for classes of two counters automata, in: H. Reichel, S. Tison (Eds.), STACS 2000: 17th Annual Symposium on Theoretical Aspects of Computer Science Lille, France, February 17–19, 2000 Proceedings, Springer-Verlag, Berlin, Heidelberg, 2000, pp. 346–357.
- [23] V. Rajlich, Absolutely parallel grammars and two-way finite-state transducers, Journal of Computer and System Sciences 6 (4) (1972) 324–342.
- [24] M. Latteux, Substitutions dans les EDT0L systèmes ultralinéaires, Information and Control 42 (1979) 194-260.
- [25] G. Rozenberg, D. Vermeir, On the effect of the finite index restriction on several families of grammars, Information and Control 39 (1978) 284–302.
- [26] T. Harju, O. Ibarra, J. Karhumäki, A. Salomaa, Some decision problems concerning semilinearity and commutation, Journal of Computer and System Sciences 65 (2) (2002) 278–294.
- [27] E. Gurari, O. Ibarra, The complexity of decision problems for finite-turn multicounter machines, Journal of Computer and System Sciences 22 (2) (1981) 220–229.
- [28] S. Ginsburg, The Mathematical Theory of Context-Free Languages, McGraw-Hill, Inc., New York, NY, USA, 1966.
- [29] O. Ibarra, S. Seki, Characterizations of bounded semilinear languages by one-way and two-way deterministic machines, International Journal of Foundations of Computer Science 23 (6) (2012) 1291–1306.
- [30] O. Ibarra, T. Jiang, N. Tran, H. Wang, New decidability results concerning two-way counter machines, SIAM J. Comput. 23 (1) (1995) 123–137.
- [31] O. Ibarra, Z. Dang, L. Yang, On counter machines, reachability problems and diophantine equations, International Journal of Foundations of Computer Science 19 (4) (2008) 919–934.
- [32] O. H. Ibarra, I. McQuillan, On the density of languages accepted by turing machines and other machine models, Journal of Automata, Languages and Combinatorics 23 (2018) 189–199.