Fundamenta Informaticae 148 (2016) 291–308
DOI 10.3233/FI-2016-1436
IOS Press

Visibly Pushdown Automata and Transducers with Counters

Oscar H. Ibarra*
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract. We generalize the models of visibly pushdown automata (VPDAs) and visibly pushdown transducers (VPDTs) by equipping them with reversal-bounded counters. We show that some of the results for VPDAs and VPDTs (e.g., closure under intersection and decidability of emptiness for VPDA languages) carry over to the generalized models, but other results (e.g., determinization and closure under complementation) do not carry over, in general. We define a model that combines the desirable features of a VPDA and reversal-bounded counters, called 2-phase VPCM, and show that the deterministic and nondeterministic versions are equivalent and that the family of languages they define is closed under Boolean operations and has decidable emptiness, infiniteness, disjointness, containment, and equivalence problems. We also investigate the finite-ambiguity and finite-valuedness problems concerning these devices.

Keywords: pushdown automaton, pushdown transducer, visibly pushdown automaton, visibly pushdown transducer, reversal-bounded counters, ambiguous, finite-valued, decidable, undecidable.

1. Introduction

A visibly pushdown automaton (VPDA) [1], also known as input-driven pushdown automaton [2], is a restricted version of a nondeterministic pushdown automation (NPDA). In a VPDA, the input symbol determines when the automaton can push or pop. It was shown in [1] that the class of languages accepted by VPDAs enjoys many of the properties of regular languages. For example, the class is closed under union, intersection, complementation, concatenation, and Kleene-*. Unlike NPDAs, the

Address for correspondence: Department of Computer Science, University of California, Santa Barbara, CA 93106, USA.
*Supported in part by NSF Grant CCF-1117708.

containment and equivalence problems are decidable for VPDAs. It was shown in [1] that the VPDA framework unifies and generalizes many of the decision procedures in the program analysis literature, and it provides algorithmic verification of recursive programs with respect to many context-free properties. Various aspects of VPDAs (input-driven NPDAs) have been studied in the literature, e.g., the space complexity of the languages they accept [2, 3, 4], the equivalence of the deterministic and nondeterministic varieties [3, 1], their descriptional complexity [3, 1, 5], etc. Many papers have been written concerning VPDAs and their variants/extensions. One, in particular, is the visibly pushdown transducer (VPDT), which is simply a VPDA with outputs [6, 7].

In this paper, we generalize the models of VPDAs and VPDTs by equipping them with reversal-bounded counters. We investigate the closure and decidability properties of these generalized models. In particular, we show that some of the results for VPDAs and VPDTs (e.g., closure under intersection and decidability of emptiness for VPDA languages) carry over to languages accepted by VPDAs with reversal-bounded counters, but other results (e.g., determinization and closure under complementation) do not carry over, in general. We introduce a model that combines the desirable features of a VPDA and reversal-bounded counters, called 2-phase VPCM (2-phase visibly pushdown automaton with reversal-bounded counters). We show that the deterministic and nondeterministic versions of 2-phase VPCMs are equivalent and the family of languages they define is closed under Boolean operations and has decidable emptiness, infiniteness, disjointness, containment, and equivalence problems. We also look at the ambiguity questions concerning VPDAs with reversal-bounded counters and the finite-valuedness problem for VPDTs with reversal-bounded counters.

2. Preliminaries

A counter is an integer variable that can be incremented by 1, decremented by 1, left unchanged, and tested for zero. It starts at zero and can only store nonnegative integer values. (Thus, one can think of a counter as a pushdown stack with a unary alphabet, in addition to the bottom of the stack symbol which is never altered.)

An automaton (DFA, NFA, DPDA, NPDA, etc.) can be augmented with multiple counters, where the "move" of the machine also now depends on the status (zero or non-zero) of the counters, and the move can update the counters. See [8] for formal definitions.

It is well known that a DFA augmented with two counters is equivalent to a Turing machine (TM) [9]. However, when we restrict the operation of the counters so that during the computation, the number of times each counter alternates between nondecreasing mode and nonincreasing mode is at most some fixed number r (such a counter is called *reversal-bounded*), the computational power of a DFA (and even an NPDA) augmented with reversal-bounded counters is significantly weaker than a TM. Note that a counter that makes r reversals can be simulated by $\lceil \frac{r+1}{2} \rceil$ 1-reversal counters.

We will use the following notations: DFCM (resp., NFCM, DPCM, NPCM) will denote a DFA (resp., NFA, DPDA, NPDA) augmented with a finite number of reversal-bounded counters. Note that these machines are allowed to make ε-moves.

Convention: For all machines, we provide the inputs with a right end marker, \$. The language accepted by a machine M, denoted by $L(M)$, is the set of all strings w such that M, when given $w\$$, eventually enters an accepting state after processing $w\$$. However, we note that for the nondeterministic machines, the end marker is not really needed, since the machine can always guess the right end of the input and then simulates the computation on \$ with ε-moves.

An acceptor is k-ambiguous if every input has at most k distinct accepting computations. Note that 1-ambiguous is the same as unambiguous. An acceptor is finitely-ambiguous if it is k-ambiguous for some k.

We will need the following known results:

Theorem 2.1. [10] Given a DFCM M, we can construct an equivalent DFCM M' which, on any input, eventually falls off the end of the tape in an accepting or rejecting state. Hence, the complement of $L(M)$ (the language accepted by M) can also be accepted by a DFCM.

The construction of M' in Theorem 2.1 is straightforward. Let n be the number of states of M. If M makes more than n ε-moves without decrementing a counter, it has entered an infinite loop. So M' just needs to check this condition.

Theorem 2.2. [11] Given a DPCM M, we can construct an equivalent DPCM M' which, on any input, eventually falls off the end of the tape in an accepting or rejecting state. Hence, the complement of $L(M)$ can also be accepted by a DPCM.

The construction of M' in the above theorem is a generalization of the construction when M is a DPDA (i.e., has no counter) [12].

The following fundamental result was shown in [8]:

Theorem 2.3. The following problems are decidable for NPCMs:

1. Emptiness: Given an NPCM M, is $L(M) = \varnothing$?
2. Infiniteness: Given an NPCM M, is $L(M)$ infinite?
3. Disjointness: Given two NPCMs M_1 and M_2, is $L(M_1) \cap L(M_2) = \varnothing$?

3. VPDAs with reversal-bounded counters

The model of a visibly pushdown automaton (VPDA) was defined and studied in [1], although the model itself was first introduced earlier in [2] under the name input-driven pushdown automaton. It is an NPDA where the input symbol determines the (push/pop) operation of the stack. It has a distinguished symbol $\perp$ at the bottom of the stack which is never altered or occurs anywhere else. The input alphabet Σ is partitioned into three disjoint alphabets: Σ_c, Σ_r, Σ_ℓ. The machine pushes a specified symbol on the stack if it reads a *call symbol* in Σ_c on the input; it pops a specified symbol if the specified symbol is at top of the stack and it is not the bottom of the stack $\perp$ (otherwise it it does not pop $\perp$) if it reads a *return symbol* in Σ_r on the input; it does not use the (top symbol of) the stack and can only change state if it reads a *local symbol* in Σ_ℓ on the input. In the definition of VPDA, the machine has no ε-moves, i.e, it reads an input at every step. An input $x \in \Sigma^*$ is accepted if the machine, starting from one of a designated set of initial states with $\perp$ at bottom of the stack, enters an accepting state after processing all symbols in x. A language $L \subseteq \Sigma^*$ is a VPDA language (i.e., accepted by a VPDA) if there is a VPDA that accepts Σ with respect to some partitioning of Σ into $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_\ell$. We say that the family of VPDA-languages is closed under a k-ary operation OP ($k \geq 1$) if, given languages $L_1, \ldots, L_k \subseteq \Sigma^*$ accepted by VPDAs with respect to a given partitioning of Σ, $OP(L_1, \ldots, L_k)$ is also accepted by a VPDA over the same partitioning of Σ.

In this paper, we assume, without loss of generality, that the VPDA has only one initial state. We also allow ε-moves but these moves do not use the top of the stack and, hence, do not alter the stack. Thus, we assume that the state set Q of a VPDA M has a subset Q_ε (the set of ε-states). M can only make an ε-move, if it is in a state in Q_ε, and in this case, the stack is not used.

Clearly, as observed in [1], a VPDA that allows ε-moves, as described above, can be converted to an equivalent VPDA that has no ε-moves. So allowing ε-moves does not alter the power of the device.

Again, we assume that the inputs to the VPDAs, DVPDAs, VPCMs, DVPCMs have a right end marker \$, where \$ is a local symbol (i.e., in Σ_ℓ). For VPCMs, the end marker is not really needed, since the machine can always guess the right end of the input and then simulates the computation on \$ with ε-moves. Moreover, for the case of DVPDAs, the right end marker is not needed, since they are equivalent to VPDAs.

Every VPDA can be converted to an equivalent DVPDA (deterministic VPDA) which has no ε-moves, i.e., it operates in real-time [3, 1]. Clearly, a DVPDA is a special case of a DPDA. We note, however, that, in general, the ε-moves in a DPDA cannot be removed. DPDAs are more powerful than DVPDAs, since the former can accept languages like $L_1 = \{a^k \# a^k \mid k \geq 1\}$ and $L_2 = \{x \# x^R \mid x \in (a+b)^+\}$ that cannot be accepted by the latter.

Now consider a VPDA augmented with k reversal-bounded counters. We allow the machine to have ε-moves, but in such moves, the stack is not used, only the state and counters are used and updated. Acceptance of an input string is when machine eventually falls off the right end of the input in an accepting state. Thus, a VPDA M with k reversal-bounded counters operates like a VPDA but can now use reversal-bounded counters as auxiliary memory. More precisely, M's operation is defined as follows (where q, p are states, s_i = 0 or + is the status of counter i, and d_i = +1, -1, or 0 is added to counter i):

1. If the current input is $a \in \Sigma_c$, then M can only use rules of the form:

   $(q, a, s_1, \ldots, s_k) \rightarrow (p, \gamma, d_1, \ldots, d_k)$

   which means that γ (which cannot be $\perp$) is pushed, and the state and counters are updated, or of the form:

   $(q, \varepsilon, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$

   which means an ε-move: q is an ε-state, the input symbol is not read, the stack is not used, but the state and counters are updated.

2. If the current input is $a \in \Sigma_r$, then M can only use rules of the form:

   $(q, a, s_1, \ldots, s_k) \rightarrow (p, \gamma, d_1, \ldots, d_k)$

   which means that if γ is the top of the stack and $\neq \perp$, it is popped; otherwise, if $\gamma = \perp$ and it is the top of the stack, it is not popped; and the state and counters are updated, or of the form :

   $(q, \varepsilon, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$, where q is an ε-state.

3. If the current input is $a \in \Sigma_\ell$, then M can only use rules of the form:

   $(q, a, s_1 \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$

   which means the stack is not used, but the state and counters are updated, or of the form:

   $(q, \varepsilon, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$, where q is an ε-state.

Consider the language $L_1 = \{w \mid w = xy$ for some $x \in (a+b)^+, y \in (0+1)^+, h(x) = y^R\}$, where h is the homomorphism $h(a) = 0$ and $h(b) = 1$. Clearly, L_1 can be accepted by a VPDA M_1. However, the language $L_2 = \{w \mid w \in L_1$, the number of a's + number of 0's in w = the number of b's + number of 1's in $w\}$ cannot be accepted by a VPDA, since it can easily be shown that it is not even a context-free language (i,e., it cannot be accepted by any NPDA). But L_2 can be accepted by a VPDA M_2 with two 1-reversal counters C_1 and C_2 as follows: On a given input w, M_2 simulates M_1, and stores the number of a's and 0's (resp., the number of b's and 1's) it sees on the input in counter C_1 (resp., C_2). Then when M_1 accepts, M_2 (on ε-moves) decreases the counters simultaneously and accepts if the counters become zero at the same time.

Denote a VPDA augmented with a finite number of reversal-bounded counters by VPCM. Clearly, a VPCM is a special case of an NPCM. Hence, from Theorem 2.3, we have:

Theorem 3.1. The emptiness, infiniteness, and disjointness problems for VPCMs are decidable.

Unlike VPDAs:

Theorem 3.2. It is undecidable, given a VPCM M over input alphabet Σ, whether $L(M) = \Sigma^*$. Hence, the containment and equivalence problems (given M_1 sand M_2, is $L(M_1) \subseteq L(M_2)$? and is $L(M_1) = L(M_2)$?) for VPCMs are undecidable.

Proof:
It is well-known that it is undecidable, given an NFCM M with only one 1-reversal counter over input alphabet Σ, whether $L(M) = \Sigma^*$ [13]. The result follows, since such an NFCM is a special case of a VPCM where the stack is not used and all the input symbols are local. □

Theorem 3.3. The class of languages accepted by VPCMs is closed under union, intersection, and concatenation, but not under Kleene-* and complementation.

Proof:
The construction for union is obvious. For intersection and concatenation, the constructions are similar to ones for VPDAs without reversal-bounded counters in [1], with some modifications. Suppose M_1 and M_2 are two VPCMs with k_1 and k_2 reversal-bounded counters.

Intersection: We construct a VPCM with $k_1 + k_2$ reversal-bounded counters to simulate M_1 and M_2 in parallel. However, since the machines can have ε-transitions (which do not involve the stack), whenever one machine, say M_1, wants to make an ε-move but the other machine M_2 wants to make a non-ε-move, the simulation of M_2 is suspended temporarily until M_1 decides to make a non-ε-move, at which time the parallel simulation of the two machines can be resumed.

Concatenation: Assume that the stack alphabets of M_1 and M_2 are disjoint. We construct a VPCM M with $k_1 + k_2$ reversal-bounded counters which, when given input w, guesses a decomposition $w = w_1 w_2$ and simulates M_1 on w_1 and M_2 on w_2. Note that M simulates M_1 on the end marker of $w_1\$$ with ε-moves at the end of w_1 (since w_1 and w_2 are not separated by \$).

Non-closure under complementation and Kleene-* will be shown in Corollary 3.8. □

We can define a deterministic VPCM (DVPCM) as follows. For every tuple $(q, \gamma, s_1, \ldots, s_k)$:

1. For every $a \in \Sigma \cup \{\varepsilon\}$, there is at most one transition $(q, a, s_1, \ldots, s_k) \to (p, \gamma, d_1, \ldots, d_k)$.
2. If there is a transition $(q, \varepsilon, \gamma, s_1, \ldots, s_k) \to (p, d_1, \ldots, d_k)$, which means that q is an ε-state, then there is no transition $(q, a, \gamma, s_1, \ldots, s_k) \to (p, d_1, \ldots, d_k)$ for any $a \in \Sigma$. Clearly, this is automatically satisfied by the requirement that q is an ε-state.

Theorem 3.4.

1. The class of languages accepted by DVPCMs is closed under Boolean operations.
2. The containment and equivalence problems for DVPCMs are decidable.

Proof:
Closure under union and intersection is obvious. As for complementation, in a DVPCM (as in a VPCM) M, the stack is not used on ε-moves. Hence, we can use the same idea as in the proof of Theorem 2.1; we now construct a DVPCM M' accepting $L^c(M)$ (the complement of $L(M)$: Let n be number of states of M. If M makes more than n ε-moves without decrementing a counter, it has entered an infinite loop. When this happens, M' moves the input head to the right (pushing a fixed symbol, say, γ if the input is in Σ_c, popping if the input is in Σ_r and the top of the stack is not $\perp$ (otherwise, it does not pop), and does not use the stack if the input is in Σ_ℓ. During this process, M' remains in a rejecting state. It follows that $L^c(M)$ is accepted by the DVPCM M'.

The second part follows from the first part using Theorem 3.1. □

We will need the two lemmas below.

Lemma 3.5. $L = \{0^n\#0^n\# \mid n \geq 1\}$ is accepted by a DCM with only one 1-reversal counter, but L^* cannot be accepted by any NFCM.

Proof:
The first part is obvious. Now suppose L^* can be accepted by an NFCM M. Consider the following languages:

$L_1 = \{0^n\#0^{n+1}\# \mid n \geq 1\}^+\{0^n\# \mid n \geq 1\}$,

$L_2 = \{0^1\}\{\#0^n\#0^{n+1}\# \mid n \geq 1\}^+$.

Clearly, we can construct from M, NFCMs M_1 and M_2 accepting L_1 and L_2, respectively. Since the family of NFCM languages is closed under intersection, $L_3 = L_1 \cap L_2 = \{0^1\#0^2\# \cdots \#0^n\#0^{n+1}\# 0^{n+2}\# \mid n \geq 1\}$ is also accepted by an NFCM. The result follows, since the Parikh map of any NFCM language is semilinear [8], but the Parikh map of L_3 is not semilinear. □

Lemma 3.6. There is a language L_1 accepted by a DFCM with only two 1-reversal counters and a regular language L_2 such that L_1L_2 cannot be accepted by any DFCM.

Proof:
Consider the following languages:

$L_1 = \{\#0\#0\#0w\#0\#0\#0 \mid w \in \{\#0\#0\#0, \#00\#, \#\#000\#\}^*, |w|_{\#00\#} \neq |w|_{\#\#000\#}\}$
(where $|w|_u$ denotes the number of occurrences of string u in w),

$L_2 = \{\#0\#0\#0, \#00\#, \#\#000\#\}^*$.

Let $L'_1 = \{\#w\# \mid w \in \{\#, a, b\}^*, |w|_a \neq |w|_b\}$, and $L'_2 = \{\#, a, b\}^*$. Clearly, L'_1 can be accepted by a DFCM with only two 1-reversal counters, and L'_2 is regular. However, it was shown in [14] that $L'_1 L'_2$ cannot be accepted by any DFCM.

Now let h be a homomorphism defined by: $h(\#) = \#0\#0\#0$, $h(a) = \#00\#$, and $h(b) = \#\#000\#$. Then $L_1 = h(L'_1)$ and $L_2 = h(L'_2)$. It follows from the above that L_1 can be accepted by DFCMs with two 1-reversal counters and L_2 is regular, but $L_1 L_2$ cannot be accepted by any DFCM. □

Theorem 3.7.

1. There is a language L that can be accepted by a DVPCM such that L^* cannot be accepted by any VPCM under any partitioning of the input alphabet (into call, return, local symbols).
2. There is a language L_1 accepted by a DVPCM and a regular language L_2 such that $L_1 L_2$ cannot be accepted by any DVPCM under any partitioning of the input alphabet.

Proof:
For Part 1, let $L = \{0^n\#0^n\# \mid n \geq 1\}$. Clearly, L can be accepted by a DVPCM with only one 1-reversal counter under one possible partition: both 0 and # are local symbols.

Now consider L^*. Let M_* be a VPCM accepting L^*. We first look at the possible partitioning of the input alphabet $\{0, \#\}$ into call, return, and local symbols. Note that at the start of any computation, the stack contains only $\perp$ (the bottom of the stack symbol) which is never deleted nor altered. We consider three cases:

1. 0 is a return symbol, and # is a local, return, or call symbol. Clearly, in this case, the stack can be removed and simulated in the finite control that just keeps track of the top two symbols (resp., top symbol) in the stack if there are at least two symbols (resp., $\perp$ is the only symbol) in the stack.

   The other cases are:

2. 0 is a local symbol, and # is a local, return, or call symbol.
3. 0 is a call symbol, and # is a local, return, or call symbol.

   Clearly, for these two cases, the stack can also be simulated in the finite control in the same manner as in Case 1.

It follows that, for any possible partitioning of the input alphabet, we can always construct an NFCM M'_* that simulates the NPCM M_* to accept L^*. This is a contradiction, since L^* cannot be accepted by an NFCM by Lemma 3.5.

For Part 2, let
$L_1 = \{\#0\#0\#0w\#0\#0\#0 \mid w \in \{\#0\#0\#0, \#00\#, \#\#000\#\}^*, |w|_{\#00\#} \neq |w|_{\#\#000\#}\}$,
$L_2 = \{\#0\#0\#0, \#00\#, \#\#000\#\}^*$.
Then by making # and 0 as local symbols, and using two 1-reversal counters, we can construct a DVPCM accepting L_1.

Suppose $L_1 L_2$ can be accepted a DVPCM M. Then, as in Part 1, it is easy to verify that for any possible partitioning of the input alphabet, the stack can be simulated in the finite control by just

keeping track of the top three symbols (resp., top two symbols, top symbol) of the stack if there are at least three symbols (resp., there are exactly two symbols, $\perp$ is the only symbol) in the stack. It follows that for any possible partitioning of the input alphabet, we can always construct a DFCM M' to simulate the DVPCM M accepting L_1L_2. This is a contradiction, since L_1L_2 cannot be accepted by an DFCM by Lemma 3.6. □

Corollary 3.8.

1. The class of languages accepted by VPCMs (resp., DVPCMs) is not closed under Kleene-*.
2. The class of languages accepted by DVPCMs is not closed under concatenation.
3. The class of languages accepted by VPCMs is not closed under complementation.

Proof:
Parts 1 and 2 follow from Theorem 3.7 parts 1 and 2, respectively. For part 3, let $L = \{0^n\#0^n\# \mid n \geq 1\}^*$. From Theorem 3.7 part 1, L cannot be accepted by any VPCM for any partitioning of the input alphabet. However, its complement, L^c, can be accepted by a VPCM M^c where 0 and $\#$ are local symbols. M^c checks that the input is of the form $0^{i_1}\#0^{j_1}\cdots 0^{i_k}\#0^{j_k}\#$ for some $k \geq 1$, and nondeterministically guesses an $1 \leq r \leq k$ and verifies, using a 1-reversal counter, that $i_r \neq j_r$. □

Every VPDA can be converted to an equivalent DVPDA (i.e., deterministic VPDA) [3, 1]. In contrast:

Corollary 3.9. There are VPCMs that cannot be converted to equivalent DVPCMs.

Proof:
This follows from Corollary 3.8 part 3, and the fact the class of languages accepted by DVPCMs is closed under complementation by Theorem 3.4. □

4. Separation results

In this section, we compare the accepting power of DVPCMs and NFCMs. In particular, we show that DVPCMs and NFCMs are incomparable.

A DVPDA (DPDA) is 1-reversal if the stack makes only one reversal, i.e., once it pops it can no longer push.

Theorem 4.1. There is a language accepted by a 1-reversal DVPDA (hence, by a DVPCM) that cannot be accepted by any NFCM.

Proof:
Let $L = \{w \mid w = x\#y$ for some $x \in (a+b)^+, y \in (0+1)^+, h(x) = y^R\}$. Clearly, L can be accepted by a 1-reversal DVPDA.

Suppose L can be accepted by an NFCM M. It was shown in [13] that for any NFCM, there is a constant c (depending only on the specification of the machine) such that any input of length n it accepts can be accepted in a computation of length at most cn. It follows that NFCM M can only distinguish a polynomial number of inputs of length n. Since the number of strings of length n in L is exponential, it follows that L cannot be accepted by M, a contradiction. □

Note that the above result obviously holds for 1-reversal DPDA. Below, we describe another proof of the above theorem (that already appeared in [10]), since the proof technique will be used later. For notational convenience, we first consider the 1-reversal DPDA case.

Theorem 4.2. There is a language accepted by a 1-reversal DPDA (hence, by a DPCM) that cannot be accepted by any NFCM.

Proof:
Let $L \subseteq \{a\}^*$ be a unary recursively enumerable language that is not recursive, i.e., not decidable (such an L exists), and Z be a Turing machine (TM) accepting L. Let Q and Σ be the state set and worktape alphabet of Z and let $q_0 \in Q$ be the initial state of Z. Let $\Sigma' = Q \cup \Sigma \cup \{\#, \$\}$. Note that a is in Σ. The halting computation of Z on input a^d can be represented by the string $ID_1\#ID_3 \cdots \#ID_{2k-1}\$ID_{2k}^R \cdots \#$
$ID_4^R\#ID_2^R$ for some $k \geq 2$ (without loss of generality, we can assume that the length of a computation is even), where $ID_1 = q_0a^d$ and ID_{2k} are the initial and halting configurations of Z, and $(ID_1, ID_2, \cdots, ID_{2k})$ is a sequence of configurations of Z on input a^d, i.e., configuration ID_{i+1} is a valid successor of ID_i. Now consider the languages:

$$\begin{aligned} L_1 &= \{ID_1\# \cdots \#ID_{2k-1}\$ID_{2k}^R \cdots \#ID_2^R \mid ID_{2k} \text{ is a halting configuration,} \\ &\quad k \geq 2, ID_1 = q_0a^p (p \geq 1), \text{and } ID_{i+1} \text{ is a valid successor of } ID_i \text{ for odd } i\}, \\ L_2 &= \{ID_1\# \cdots \#ID_{2k-1}\$ID_{2k}^R \cdots \#ID_2^R \mid ID_{2k} \text{ is a halting configuration,} \\ &\quad k \geq 2, ID_1 = q_0a^p (p \geq 1), \text{and } ID_{i+1} \text{ is a valid successor of } ID_i \text{ for even } i\}. \end{aligned}$$

Clearly, L_1 and L_2 can be accepted by 1-reversal-DPDAs. We claim that L_1 or L_2 cannot be accepted by any NFCM. For suppose L_1 and L_2 can be accepted by NFCMs M_1 and M_2. We construct from M_1 and M_2 an NFCM M accepting the language $L_1 \cap L_2 = \{ID_1\# \cdots \#ID_{2k-1}\$ID_{2k}^R \cdots \#ID_2^R \mid k \geq 2, ID_1 = q_0a^p, p \geq 1, ID_{2k}$ is a halting configuration, and ID_{i+1} is a valid successor of ID_i for $i \geq 1\}$. (M simulates M_1 and M_2 in parallel.)

Finally, we show (using M) that there exists an algorithm to decide L, which would make L recursive, hence, a contradiction. The algorithm works as follows:

1. On an input a^d, construct a finite automaton accepting $q_0a^d\#\Sigma'^*$.
2. From the finite automaton and the NFCM M (accepting $L_1 \cap L_2$), construct an NFCM M' which accepts $q_0a^d\#\Sigma'^* \cap L_1 \cap L_2$.
3. Check if the language accepted by M' is empty. This is possible since the emptiness problem for NPCMs (hence, also for NFCMs) is decidable by Theorem 2.3.

Clearly, $a^d \notin L$ if and only if the language accepted by M' is empty. The result follows. □

Corollary 4.3. DPCMs are strictly more powerful than DFCMs.

Here is another proof of Theorem 4.1.

Proof:
The languages L_1 and L_2 as defined in the proof of Theorem 4.2 cannot be accepted by DVPDAs. We modify the languages by using a different set of symbols to represent the second half of the strings in L_1 and L_2 (after the middle symbol \$). Specifically, the string:

$ID_1\# \cdots \#ID_{2k-1}\$ID_{2k}^R \cdots \#ID_2^R$

now becomes:

$ID_1\# \cdots \#ID_{2k-1}\$\overline{ID}_{2k}^R \cdots \#\overline{ID}_2^R$

where $\overline{ID}_{2k}^R, \ldots, \overline{ID}_2^R$ are now using a different set of symbols. Thus, we enlarge the input alphabet so that the symbols appearing in $ID_1, \ldots, ID_{2k-1}$ are call symbols (i.e., push symbols) and the symbols appearing in $\overline{ID}_{2k}^R, \ldots, \overline{ID}_2^R$ are return symbols (i.e., pop symbols), and $\#, \$$ are local symbols. □

Corollary 4.4. DVPCMs are strictly more powerful than DFCMs.

We will prove the converse of Theorem 4.1. Since a DVPCM is a special case of a DPCM, we need only show that there is a language accepted by an NFCM that is not accepted by a DPCM.

Call an NFCM with only one 1-reversal counter a 1-reversal NFCM(1). The next theorem also answers in the affirmative an open question in [10]:

Theorem 4.5. There is a language accepted by a 1-reversal NFCM(1) (hence, by an NFCM) that cannot be accepted by a DPCM.

Proof:
We use the notations in the proof of Theorem 4.2. But now, we represent the halting computation of Z on the input a^d by the string $ID_1\#ID_2\# \cdots \#ID_k$ for some $k \geq 2$, where $ID_1 = q_0a^d$ and ID_k are the initial and halting configurations of Z, and $(ID_1, ID_2, \cdots, ID_k)$ is a sequence of configurations of Z on input a^d, i.e., configuration ID_{i+1} is a valid successor of ID_i.

Consider the language $L' = \{w \mid w \neq ID_1\#ID_2\# \cdots \#ID_k, k \geq 2, ID_1 = q_0a^p (p \geq 1), ID_{i+1}$ is a valid successor of ID_i for $1 \leq i \leq k-1$, ID_k is the halting configuration of $Z\}$.

Clearly, we can construct a 1-reversal NFCM(1) to accept L'. (The idea is as follows: M_1 accepts all strings w that are not in the correct format. If w is in the correct format, M_1 guesses an i and checks that ID_{i+1} is not a successor of ID_i .)

We claim that L' cannot be accepted by a DPCM. For suppose L' can be accepted by a DPCM M_2. Then, by Theorem 2.2, we can construct a DPCM M_3 accepting $L^c(M_2)$ (the complement of $L(M_2)$). Now using M_3, an algorithm to decide the non-recursive recursively enumerable unary language L could be constructed as follows, given input a^d:

1. Let $R = \{q_0a^d\#\ ID_2\# \cdots \#ID_k \mid k \geq 2, ID_i$ is any configuration of Z for $1 \leq i \leq k-1\}$. R is regular and can be accepted by a DFA M_4.
2. Construct a DPCM M_5 accepting $L(M_3) \cap L(M_4)$.
3. Check if the language accepted by M_5 is empty. This is possible since the emptiness problem for NCPMs (hence also for DPCMs) is decidable by Theorem 2.3.

Clearly, $a^d \notin L$ if and only if the language accepted by $L(M_5)$ is empty. It follows that L' cannot be accepted by any DPCM. □

From Theorem 4.1 and Theorem 4.5, we have:

Corollary 4.6.

1. There is a language accepted by a 1-reversal DVPDA (hence, by a DVPCM) that cannot be accepted by any NFCM.
2. There is a language accepted by a 1-reversal NFCM(1) (hence, by an NFCM) that cannot be accepted by any DVPCM.

5. VPDTs with reversal-bounded counters

A transducer T is an acceptor with outputs. Thus, an NFT (resp., NPDT, NPCMT, VPDT, VPCMT, DVPCMT, etc.) is an NFA (resp, NPDA, NPCM, VPDA, VPDCM, DVPCM, etc.) with outputs. The output at each step is a (possibly null) string over an output alphabet.

We say that (u, v) is a transduction accepted by T if, when given input string u (which can be ε), T eventually The set of transductions accepted by T is denoted by $R(T)$.

A transducer T is k-ambiguous ($k \geq 1$) if T with outputs ignored is a k-ambiguous acceptor (1-ambiguous is the same as unambiguous). T is finitely-ambiguous if it is k-ambiguous for some k.

A transducer T is k-valued ($k \geq 1$) if for every u, there are at most k distinct strings v such that (u, v) is in $R(T)$ (1-valued is the same as single-valued). T is finite-valued if T is k-valued for some k.

VPDTs were studied in [6, 7]. In [7] it was shown that VPDTs that do not allow ε-transitions have a decidable k-valuedness problem. Here, we generalize this result for VPCMTs (i.e., VPDTs with reversal-bounded counters). Just as in VPCMs, we allow VCMTs to have ε-transitions, i.e., ε-moves on the input (where the stack is not used), and on each such transition, the machine can output any string, not necessarily ε.

Theorem 5.1. It is decidable, given a VPCMT T and $k \geq 1$, whether T is k-valued.

Proof:
Consider the case $k = 1$, Given T with m reversal-bounded counters, we construct a VPCM M with two sets of m reversal-bounded counters and two additional 1-reversal counters C_{12} and C_{21}. Hence, M will have $2(m + 1)$ reversal-bounded counters. M on input x simulates T suppressing the outputs and accepts x if it finds two outputs y_1 and y_2 such that y_1 and y_2 disagree in some position p and x is accepted by T. In order to do this, during the simulation, M uses C_{12} and C_{21} to record the positions i and j (chosen nondeterministically) in y_1 and y_2, respectively, and the symbols a and b in these positions, such that $i = j$ and $a \neq b$. Clearly, a and b can be remembered in the state. For storing i and j, only incrementing C_{12} and C_{21} is needed during the simulation. To check that $i = j$, after the simulation, M decrements C_{12} and C_{21} simultaneously and verifies that they reach zero at the same time.

Note that since T is a VPCMT, M can simulate two distinct accepting computations P_1 and P_2 (if they exist) in parallel using only one stack but using m counters for process P_1 and m counters for process P_2. If in a step, both P_1 and P_1 move right on an input symbol, M moves right on the symbol and simulates the actions of P_1 and P_2 on their respective counters accordingly. Since T is a VPCM, the actions on the stack by P_1 and P_2 are synchronized: both push, pop, or 'no change'; so only one stack is needed. If both P_1 and P_2 make ε-moves (hence, the stack is not used), then M also makes an ε-move and simulates the actions of P_1 and P_2 on their respective counters. If only one of P_1 and P_2

makes an ε-move, e.g., if P_1 makes an ε-move and P_2 wants to read an input symbol, the simulation of P_2 is suspended temporarily until P_1 decides to read an input symbol, at which time the parallel simulation of P_1 and P_2 on the input symbol read can be resumed.

Clearly, the procedure just described can be accomplished by M in a single accepting run on the input. Now, T is 1-valued if and only if $L(M) = \varnothing$, which is decidable, since emptiness of VPCMs is decidable (by Theorem 3.1).

The above construction generalizes for any $k \geq 1$. Now M, on input x, checks that there are at least $k+1$ distinct outputs $y_1, \ldots, y_{k+1}$. M uses $k+1$ sets of m counters to simulate $k+1$ accepting computations and $k(k+1)$ additional 1-reversal counters (so now M will have $(k+1)(m+k)$ reversal-bounded counters). In the simulation, for $1 \leq i \leq k+1$, M nondeterministically selects k positions $p_{i1}, \ldots, p_{i(i-1)}, p_{i(i+1)}, \ldots, p_{i(k+1)}$ in output y_i and records these positions in counters $C_{i1}, \ldots, C_{i(i-1)}, C_{i(i+1)}, \ldots, C_{i(k+1)}$ and the symbols at these positions in the state. At the end of the simulation, M verifies that for all $1 \leq i, j \leq k+1$ such that $i \neq j$, the symbol in position p_{ij} is different from the symbol in position p_{ji} and the value of counter C_{ij} is the same as the value of C_{ji}. □

Recall that a VPCM is k-ambiguous ($k \geq 1$) if every input is accepted in at most k distinct accepting computations.

Corollary 5.2. It is decidable, given a VPCM M and $k \geq 1$, whether M is k-ambiguous.

Proof:
Given a VPCM M, let $t_1, ..., t_r$ be its transition rules. Construct a VPCM T which simulates M and outputs the transition rules used in the computation. Clearly T is k-valued if and only if M is k-ambiguous. The result follows from Theorem 5.1. □

The above corollary contrasts the undecidability of k-ambiguity (even for $k = 1$) for NPDAs whose stack makes at most one reversal, i.e., when it pops, it can no longer push [15, 16].

The construction in the proof of Theorem 5.1 above does not work when T is an NPCMT. In fact, it does not work even when T is k-ambiguous for any $k \geq 2$. This is because the computation of T on an input x may not be unique, so it is possible, e.g., that one accepting run on x produces output y_1 and a different accepting run on x produces output y_2. So to determine if $y_1 \neq y_2$, we need to simulate two runs on input x, and the behavior of the two runs on the stack may not be synchronized.

In fact, the following was shown in [16]:

Theorem 5.3. For any $k \geq 1$, it is undecidable, given a $(k+1)$-ambiguous 1-reversal NPDT T (i.e., its underlying NPDA is $(k+1)$-ambiguous and the stack makes only one reversal), whether T is k-valued.

On the other hand, when the NPCMT T is 1-ambiguous (i.e., its underlying NPCM is 1-ambiguous), the NPCMT has at most one accepting computation on any given input, although it may generate different outputs. Then multiple runs can be simulated using only one stack, and the construction in the proof of Theorem 5.1 would work. Hence, we have:

Theorem 5.4. It is decidable, given a 1-ambiguous NPCMT T and $k \geq 1$, whether T is k-valued.

It is known that the equivalence problem for DFTs is decidable [17]. However, for NFTs, the problem is undecidable even for one-way nondeterministic finite transducers (NFTs) [18]. The undecidability holds even for NFTs operating on a unary input (or output) alphabet [19]. For single-valued (i.e., 1-valued) NFTs, the problem becomes decidable [20], and the decidability result was later extended to finite-valued NFTs in [21]. The complexity of the problem was subsequently derived in [22].

Since the equivalence, hence, also containment problems for NFTs are undecidable, these problems are also undecidable for VPCMTs.

Now VPCMTs and DVPCMTs are allowed to output any string on an ε-move. Since the machine can enter an accepting state many times at the end of the input on epsilon moves, a DVPCMT may not necessarily be single-valued. Obviously, a DVPCMT that outputs only ε's on ε-moves is single-valued. However, the converse is not true.

Theorem 5.5. The following problems are decidable:

1. Given a VPCMT T_1 and a single-valued DVPCMT T_2, is $R(T_1) \subseteq R(T_2)$?
2. Given two single-valued DVPCMTs T_1 and T_2, is $R(T_1) = R(T_2)$?

Proof:
Clearly, we only need to prove (1). Let M_1 be the underlying VPCM of T_1 and M_2 be the underlying DVPCM of T_2. Thus, $L(M_1) = domain(R(T_1))$ and $L(M_2) = domain(R(T_2))$.

First we determine if $L(M_1) \subseteq L(M_2)$. This is decidable, since from Theorem 3.4, we can construct a DVPCM M_3 to accept the complement of $L(M_2)$. Then from Theorem 3.3 we can construct a VPCM M_4 to accept $L(M_1) \cap L(M_3)$. Clearly $L(M_1) \subseteq L(M_2)$ if and only if $L(M_4) = \varnothing$, which is decidable by Theorem 3.1.

Obviously, if $L(M_1) \not\subseteq L(M_2)$, then $R(T_1) \not\subseteq R(T_2)$. Otherwise, $R(T_1) \not\subseteq R(T_2)$ if and only if there exists an x such that the following two conditions are satisfied:

(a) For some y, (x, y) is in $R(T_1)$, and

(b) x is in $domain(R(T_2))$ and the (unique) z such that (x, z) is in $R(T_2)$ is different from y.

We will construct a VPCM M such that $L(M) \neq \varnothing$ if and only if the conditions above are satisfied. The result would then follow since we can decide if $L(M) = \varnothing$ by Theorem 3.1. We construct M from T_1 and T_2. Given an input x, M simulates T_1 and T_2 using separate sets of reversal-bounded counters but the same stack and two additional 1-reversal counters C_1 and C_2. M guesses that x is in the domains of T_1 and T_2 and some y and (unique) z are generated by the transducers. In the simulation, M suppresses the outputs but stores in C_1 a nondeterministically chosen position p_1 in y and remembers (in the finite control) the symbol a at this position. Similarly, M stores in C_2 a nondeterministically chosen position p_2 in z and remembers the symbol $b \neq a$ at this position. At the end of the input, when both T_1 and T_2 accept, M accepts (by decrementing the counters C_1 and C_2 simultaneously) if $p_1 = p_2$. □

Note that Theorem 5.5 is not valid when DVPCMT is replaced by VPCMT. Let M_1 be a VPCM over input alphabet Σ. Convert M_1 to a VPCMT T_1 which, on any input x , outputs 1 if M_1 accepts x. Clearly T_1 is single-valued. Now let T_2 be the trivial VPCMT which accepts Σ^* and outputs 1 on any input x. Again T_2 is single-valued. Then $T_1 = T_2$ if and only if $L(M_1) = \Sigma^*$, which is undecidable by Theorem 3.2.

6. Determinization

We have seen that, in general, a VPCM cannot be determinized. In this section, we consider a different model that is more powerful than a VPDA and a DFCM, which can be determinized and the class of languages it defines is closed under Boolean operations.

A *2-phase VPCM* M is a nondeterministic automaton with a pushdown stack and reversal-bounded counters whose input tape has a right end marker, \$ (which is a local symbol). It operates in two phases:

1. In Phase 1, M makes a left-to-right sweep of the input and operates like a VPDA without using the counters. (Note that the VPDA is nondeterministic and can have ε-moves). When M reaches the right end marker \$, it eventually resets its input head to the left end of the input in some state r (there may be several such reset states) to begin the second phase. We assume that the reset states are distinguished and are only used for this purpose.

2. In Phase 2, M makes a second left-to-right sweep of the input and operates deterministically starting in state r using the reversal-bounded counters (which are initially set to zero) but not the stack. The input is accepted if M eventually falls off the right end marker in an accepting state.

If in Phase 1, the machine operates like a DVPDA, then we call the machine a deterministic 2-phase VPCM.

Example: Let $L = \{w \mid w = xy$ for some $x \in (a+b)^+, y \in (0+1)^+, h(x) = y^R$, the number of a's + number of 0's in w $=$ the number of b's + number of 1's in $w\}$. Clearly, L can be accepted by a deterministic 2-phase VPCM with two 1-reversal counters.

Theorem 6.1.

1. Every 2-phase VPCM can be converted to an equivalent deterministic 2-phase VPCM.

2. The class of languages accepted by 2-phase VPCMs is closed under Boolean operations.

Proof:
Let M be a 2-phase VPCM with reset states $r_1, \ldots, r_k$. Let M_{r_i} be the VPDA component of M that is used in the first phase that resets to state r_i. Thus on input $w\$$, M_{r_i} makes a left-to-right sweep and after reading \$ eventually enters reset state r_i. Note that M_{r_i} can have ε-moves. In fact, it can have ε-moves after reading \$ before entering the reset state r_i. Construct a DVPDA M'_{r_i} equivalent to M_{r_i}, i.e., M'_{r_i} accepts an input $w\$$ if and only if M_{r_i} on input $w\$$ enters reset state r_i. This can be done, since VPDAs can be determinized. Since the input ends in \$, we can construct M'_{r_i} so that when it is given input $w\$$, it operates in real-time and enters a unique accepting state, call it also r_i, and a unique rejecting state s_i if $w\$$ is accepted or rejected, respectively.

Let A_{r_i} be the DFCM component of M (i.e., the machine in the second phase) whose initial state is set to r_i. Construct a DFCM A'_{r_i} equivalent to A_{r_i} such that A'_{r_i} eventually falls off the right end marker of input $w\$$ in an accepting or rejecting state. This can be done as in Theorem 2.1. (Note that A_{r_i} and, hence, A'_{r_i}, may have ε-moves.) Now for any $S \subseteq \{r_1, \ldots, r_k\}$, construct a DFCM A'_S which simulates in parallel the DFCMs whose subscripts are in S and accepts if at least one of the machines accepts.

Finally, construct a deterministic 2-phase VPCM M' which, on input $w\$$, simulates $M'_{r_1}, \ldots, M'_{r_k}$ in parallel in Phase 1. Thus the states of M' are k-tuples $(q_1, \ldots, q_k)$, where each q_i is a state of M'_{r_i}. Suppose M' enters state $(f_1, \ldots, f_k)$ after reading the end marker $\$$ of the input. Note that f_i is either r_i or s_i. Then M' resets to $S = \{r_i \mid f_i = r_i\}$, and simulates A'_S. It is straightforward to verify that M' is equivalent to M.

For Part 2, closure under complementation follows from the construction above. Closure under union is obvious. From De Morgan's law, closure under intersection also follows. □

Corollary 6.2. The emptiness, infiniteness, disjointness, containment, and equivalence problems for 2-phase VPCMs are decidable.

Proof:
This follows from Theorem 6.1 and the fact that emptiness, infiniteness, disjointness for NPCMs are decidable (Theorem 3.1). □

Theorem 6.3. Every 2-phase VPCM M can be converted to an equivalent DVPCM M'.

Proof:
Following the notation in the proof of Theorem 6.1, let M be a 2-phase VPCM with reset states $r_1, \ldots, r_k$. Let M_{r_i} be the VPDA component of M (i.e., the machine in the first phase) whose accepting state is the reset state r_i. Construct a DVPDA M'_{r_i} equivalent to M_{r_i}.

Let A_{r_i} be the DFCM component of M (i.e., the machine in the second phase) whose initial state is r_i. Construct a DFCM A'_{r_i} equivalent to A_{r_i} such that A'_{r_i} eventually falls off the right end marker in an accepting or rejecting state.

Next, construct for each $1 \leq i \leq k$, a DVPCM M_i to accept $L(M'_{r_i}) \cap L(A'_{r_i})$ by simulating the computation of M'_{r_i} and A'_{r_i} in parallel.

Finally, construct a DVPCM M to accept $L(M_1) \cup L(M_2) \cup \cdots \cup L(M_k)$ by simulating the M_i's in parallel. Note that separate sets of reversal-bounded counters are used in the simulation, but only one stack is needed. □

A language L is called a DVPDA-DFCM language if $L = L_1 \cap L_2$, where L_1 is accepted by a DVPDA and L_2 is accepted by a DFCM.

Theorem 6.4. L is accepted by a 2-phase VPCM if and only if L is a finite union of DVPDA-DFCM languages.

Proof:
The "only if" part follows from the proof of Theorem 6.3. For the "if" part, suppose $L = L_1 \cup \cdots \cup L_k$, where each L_i is a DVPDA-DFCM lanquage. Let $L_i = L_{i1} \cap L_{i2}$, where L_{i1} (resp., L_{i2}) is accepted by a DVPDA M_{i1} (resp., DFCM M_{i2}). We may assume that M_{i1} has a unique accepting state r_i and that the initial state of M_{i2} is r_i. We can then construct a 2-phase VPCM M which has reset states $r_1, \ldots, r_k$. When given an input, M on ε-move nondeterministically guesses an $1 \leq i \leq k$ and simulates M_{i1} on the first phase and resets to r_i to then simulate M_{i2} on the second phase. Clearly, $L(M) = L_1 \cup \cdots \cup L_k$. □

The converse of Theorem 6.3 seems unlikely. For consider the following language over the input alphabet $\{a, b, 0, 1, c\}$:

$$L = \{xy^R c^k \mid k \geq 0, x \in (a+b)^+, y \in (0+1))^+, |x| = |y|, x \text{ and } h(y) \text{ disagree in } k \text{ positions } \}$$

where h is a homomorphism that maps a to 0 and b to 1.

L can be accepted by a DPCM (with only one 1-reversal counter) by defining a, b to be call symbols, $0, 1$ to be return symbols, and c to be a local symbol: The stack is used to check that $|x| = |y|$ and locate the positions of discrepancies while the counter is incremented after finding each such discrepancy. Then the counter is decremented to check that the k in c^k is equal to the the number of discrepancies. However, it does not seem that L can be accepted by a 2-phase VPCM.

Finally, we look at 2-phase VPCMs with outputs. Call these transducers 2-phase VPCMTs. Thus, at each step, the 2-phase VPCMT T outputs a string (possibly ε). However, T does not output when it resets its head to the left end of the input.

Theorem 6.5. It is decidable, given a 2-phase VPCMT T and $k \geq 1$, whether T is k-valued.

Proof:
As in the proof of Theorem 5.1, we can construct a 2-phase VPCM M from T such that T is k-valued if and only if $L(M) = \varnothing$, which is decidable by Corollary 6.2. □

Theorem 6.6. The following problems are decidable:

1. Given a 2-phase VPCMT T_1 and a single-valued 2-phase VPCMT T_2, is $R(T_1) \subseteq R(T_2)$?

2. Given two single-valued 2-phase VPCMTs T_1 and T_2, is $R(T_1) = R(T_2)$?

Proof:
We refer to the notations and the proof of Theorem 5.5. The proof there applies, since we can decide whether $L(M_1) = domain(R(T_1)) \subseteq domain(R(T_2)) = L(M_2)$ by Corollary 6.2. Then, a 2-phase VPCM M can be constructed such that $L(M) = \varnothing$ if and only if the conditions (a) and (b) in that proof are satisfied. □

7. Conclusion

The emptiness problem for NPDAs with reversal-bounded counters has recently been shown to be NP-complete [23]. Using this result, one can derive lower and upper bounds for many decidable problems discussed in the paper. There are a number of interesting questions that we are currently working on, two of which are:

1. Is it decidable, given a VPCM, whether it is finitely-ambiguous?

2. Is it decidable, given a VPCMT, whether it is finite-valued?

References

[1] Alur R, Madhusudan P. Visibly Pushdown Languages. In: Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing. STOC '04. New York, NY, USA: ACM; 2004. p. 202–211. Available from: http://doi.acm.org/10.1145/1007352.1007390. doi:10.1145/1007352.1007390.

[2] Mehlhorn K. Pebbling Moutain Ranges and Its Application of DCFL-Recognition. In: Proceedings of the 7th Colloquium on Automata, Languages and Programming. London, UK, UK: Springer-Verlag; 1980. p. 422–435. Available from: http://dl.acm.org/citation.cfm?id=646234.682537.

[3] von Braunmühl B, Verbeek R. Input Driven Languages Are Recognized in Log N Space. In: Selected Papers of the International Conference on "Foundations of Computation Theory" on Topics in the Theory of Computation. New York, NY, USA: Elsevier North-Holland, Inc.; 1985. p. 1–19. Available from: http://dl.acm.org/citation.cfm?id=4030.4031.

[4] Rytter W. An Application of Mehlhorn's Algorithm for Bracket Languages to Log(N) Space Recognition of Input-driven Languages. Inf Process Lett. 1986 Aug; 23(2):81–84. Available from: http://dx.doi.org/10.1016/0020-0190(86)90047-5. doi:10.1016/0020-0190(86)90047-5.

[5] Okhotin A, Salomaa K. Complexity of Input-driven Pushdown Automata. SIGACT News. 2014 Jun; 45(2):47–67. Available from: http://doi.acm.org/10.1145/2636805.2636821. doi:10.1145/2636805.2636821.

[6] Raskin JF, Servais F. Visibly Pushdown Transducers. In: ICALP (2); 2008. p. 386–397.

[7] Filiot E, Raskin JF, Reynier PA, Servais F, Talbot JM. Properties of Visibly Pushdown Transducers. In: Hlinený P, Kucera A, editors. Mathematical Foundations of Computer Science (MFCS). vol. 6281 of Lecture Notes in Computer Science. Springer; 2010. p. 355–367.

[8] Ibarra OH. Reversal-Bounded Multicounter Machines and Their Decision Problems. J ACM. 1978 Jan; 25(1):116–133. Available from: http://doi.acm.org/10.1145/322047.322058. doi:10.1145/322047.322058.

[9] Minsky ML. Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. Annals of Mathematics. 1961;74(2):437–455.

[10] Chiniforooshan E, Daley M, Ibarra OH, Kari L, Seki S. One-reversal Counter Machines and Multihead Automata: Revisited. Theor Comput Sci. 2012 Oct;454:81–87. Available from: http://dx.doi.org/10.1016/j.tcs.2012.04.002. doi:10.1016/j.tcs.2012.04.002.

[11] Ibarra OH, Yen HC. On the Containment and Equivalence Problems for Two-way Transducers. Theor Comput Sci. 2012;429:155–163. Available from: http://dblp.uni-trier.de/db/journals/tcs/tcs429.html#IbarraY12.

[12] Hopcroft JE, Ullman JD. Introduction To Automata Theory, Languages, And Computation. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 1990.

[13] Baker BS, Book RV. Reversal-bounded Multipushdown Machines. J Comput Syst Sci. 1974 Jun;8(3):315–332. Available from: http://dx.doi.org/10.1016/S0022-0000(74)80027-9. doi:10.1016/S0022-0000(74)80027-9.

[14] Eremondi J, Ibarra OH, McQuillan I. Insertion Operations on Deterministic Reversal-Bounded Counter Machines. In: Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings; 2015. p. 200–211. Available from: http://dx.doi.org/10.1007/978-3-319-15579-1_15. doi:10.1007/978-3-319-15579-1_15.

[15] Wich K. Exponential Ambiguity of Context-Free Grammars. In: Developments in Language Theory, Foundations, Applications, and Perspectives, Aachen, Germany, 6-9 July 1999; 1999. p. 125–138.

[16] Ibarra OH. On the Ambiguity, Finite-Valuedness, and Lossiness Problems in Acceptors and Transducers. In: Implementation and Application of Automata - 19th International Conference, CIAA 2014, Giessen, Germany, July 30 - August 2, 2014. Proceedings; 2014. p. 211–225. Available from: http://dx.doi.org/10.1007/978-3-319-08846-4_16. doi:10.1007/978-3-319-08846-4_16.

[17] Blattner M, Head T. The Decidability of Equivalence for Deterministic Finite Transducers. Journal of Computer and System Sciences. 1979;19(1):45–49. Available from: http://www.sciencedirect.com/science/article/pii/0022000079900126. doi:http://dx.doi.org/10.1016/0022-0000(79)90012-6.

[18] Griffiths TV. The Unsolvability of the Equivalence Problem for Epsilon-Free Nondeterministic Generalized Machines. J ACM. 1968 Jul;15(3):409–413. Available from: http://doi.acm.org/10.1145/321466.321473. doi:10.1145/321466.321473.

[19] Ibarra OH. The Unsolvability of the Equivalence Problem for Epsilon-Free NGSM's with Unary Input (Output) Alphabet and Applications. SIAM J Comput. 1978;7(4):524–532. Available from: http://dx.doi.org/10.1137/0207042. doi:10.1137/0207042.

[20] Gurari EM, Ibarra OH. A Note on Finite-Valued and Finitely Ambiguous Transducers. Theory of Computing Systems / Mathematical Systems Theory. 1983;16:61–66. doi:10.1007/BF01744569.

[21] II KC, Karhumäki J. The Equivalence of Finite-Valued Transducers (On HDT0L Languages) is Decidable. Theor Comput Sci. 1986;47(3):71–84. Available from: http://dx.doi.org/10.1016/0304-3975(86)90134-9. doi:10.1016/0304-3975(86)90134-9.

[22] Weber A. Decomposing Finite-Valued Transducers and Deciding Their Equivalence. SIAM J Comput. 1993;22(1):175–202. Available from: http://dblp.uni-trier.de/db/journals/siamcomp/siamcomp22.html#Weber93.

[23] Hague M, Lin AW. Model Checking Recursive Programs with Numeric Data Types. In: Computer Aided Verification (CAV); 2011. .