On the Complexity of Membership and Counting in Height-Deterministic Pushdown Automata

Nutan Limaye, Meena Mahajan, Antoine Meyer

On the complexity of membership and counting in height-deterministic pushdown automata

Nutan Limaye[1], Meena Mahajan[1], and Antoine Meyer[2]

[1] The Institute of Mathematical Sciences, Chennai 600 113, India.
nutan,meena@imsc.res.in
[2] LIAFA, Université Paris Diderot – Paris 7, Case 7014, 75205 Paris Cedex 13, France. ameyer@liafa.jussieu.fr

Abstract. While visibly pushdown languages properly generalise regular languages and are properly contained in deterministic context-free languages, the complexity of their membership problem is equivalent to that of regular languages. However, the corresponding counting problem could be harder than counting paths in a non-deterministic finite automaton: it is only known to be in LogDCFL.
We investigate the membership and counting problems for generalisations of visibly pushdown automata, defined using the notion of height-determinism. We show that, when the stack-height of a given PDA can be computed using a finite transducer, both problems have the same complexity as for visibly pushdown languages. We also show that when allowing pushdown transducers instead of finite-state ones, both problems become LogDCFL-complete; this uses the fact that pushdown transducers are sufficient to compute the stack heights of all real-time height-deterministic pushdown automata, and yields a candidate arithmetization of LogDCFL that is no harder than LogDCFL(our main result).

1 Introduction

There is a close connection between complexity classes and formal language theory. Over the years, various language classes have been studied from the complexity theoretic perspective. Capturing the complexity of membership has been the goal in this approach. The study of language classes and their complexity under meaningful closures was first started by Sudborough [1, 2]. In [1], he showed that the *nondeterministic linear context-free languages* or LIN are complete for the complexity class NL (nondeterministic log-space). In [2], he defined two interesting complexity classes, namely LogCFL and LogDCFL, as the log-space closures of context-free languages (CFLs) and their deterministic counterparts (DCFLs) respectively. Ibarra, Jiang and Ravikumar [3] further studied subclasses of CFL such as $\mathsf{DLIN}_{LL(1)}$ (the deterministic counterpart of LIN defined by $LL(1)$ linear grammars), *Dyck2* and *bracketed expressions* and showed that they are contained in NC^1. Holzer and Lange [4] showed that deterministic linear context-free languages (DLIN), as defined via $LR(1)$ linear grammars, are equivalent to those accepted by deterministic 1-turn automata $\mathsf{DPDA}_{1\text{-turn}}$ (deterministic pushdown automata that never push after a pop move). They showed

that deciding membership in a DLIN language is complete for L, in contrast to the result of [3]. Barrington made an important contribution to the above study [5], showing that the class of regular languages, REG, is complete for the circuit complexity class NC^1 comprising of polynomial size log depth bounded fan-in AND-OR circuits. (As the classes get smaller, completeness is via not L -reductions but appropriate weaker notions such as AC^0-many-one-reductions.) See [6] for an overview of these results.

Visibly pushdown automata (VPA) are real-time pushdown automata whose stack behaviour is dictated solely by the input letter under consideration. They are also referred to as input-driven PDA. The membership problem for VPL was considered in [7–9]; [9] shows that languages accepted by such PDA are in NC^1. A rigorous language-theoretic study of VPA was done in [10], where it is shown that they can be determinised. Thus they lie properly between REG and DCFLs, and their membership problem is complete for NC^1.

A related line of study is understanding the power of counting. It is easy to see from the proof of [1] that counting the number of parse trees in a linear grammar, #LIN, is equivalent to counting accepting paths in an NL machine. At the lower end, however, though Barrington's result showed that deciding membership in REG (and hence in the language of a nondeterministic finite-state automaton or NFA) is equivalent to NC^1, counting the number of accepting paths in an NFA (#NFA) is not yet known to be equivalent to arithmetic NC^1, $\#\mathsf{NC}^1$. In [11], a one-way containment is shown: $\#\mathsf{NFA} \subseteq \#\mathsf{NC}^1$, but to this day the converse reduction remains open[3]. A natural question to ask is what generalisation of NFA can capture $\#\mathsf{NC}^1$ in this setting. In [12], it was claimed that the generalisation to VPA adds no power, #VPA is equivalent to #NFA. However, this claim was later retracted in [13], where it is shown, however, that #VPA functions can be computed in LogDCFL (and are hence presumably weaker than #PDA functions).

Our starting point in this note is a careful examination of what makes membership and path-counting easier in VPA than in general PDA. The intention is to identify a largest possible class of PDA for which the technique used for VPA can still be applied. This technique exploits the fact that, despite nondeterminism, all paths on a given input word have the same stack-profile, and furthermore, this profile is very easily computable. One can view the partitioning of the input alphabet as height advice being provided to an algorithm for deciding membership. This naturally leads to the conjecture that PDA possessing easy-to-compute height advice functions should be easier than general PDA. The *real-time height-deterministic* PDA defined by Nowotka and Srba ([14]), rhPDA, are a natural candidate: they are defined as precisely those PDA that possess height advice functions. They also very naturally generalise a subclass of the *synchronised* PDA defined by Caucal ([15]), namely the subclass where the synchronisation function is the stack-height, and, as in general synchronised PDA, is computable by a finite-state transducer. We provide a parameterised definition of rhPDA that captures this generalisation: the complexity of the transducer computing

[3] [11] shows a weaker converse: every $\#\mathsf{NC}^1$ function can be expressed as the difference of two #NFA functions.

the height advice function is the parameter. We then examine the complexity of membership and path-counting at either end of the parameterisation.

A related model equivalent to VPA is that of nested word automata (NWA), defined in [16] with a view to applications in software verification and XML document processing. [17] defines motley word automata (MWAs) as a generalisation of NWAs. Our techniques can be used to show that the equivalence is indeed very strong: deciding membership and counting accepting paths in NWA and MWA are NC^1-equivalent to the same problems over VPA.

2 Preliminaries

A *pushdown automaton* (PDA) over Σ is a tuple $P = (Q, q_0, F, \Gamma, \Sigma, \delta)$ where Q is a finite set of control states, $q_0 \in Q$ the initial state, $F \subseteq Q$ a set of accepting states, Γ a finite alphabet of stack symbols, Σ a finite alphabet of labels and δ a finite set of transition rules $pU \xrightarrow{a} qV$ with $p, q \in Q$, $U, V \in \Gamma^*$ and $a \in \Sigma$. In the following, we will only consider weak PDA, whose rules are such that $|UV| \leq 1$.[4]

A *configuration* of P is a word of the form pW with $p \in Q$ and $W \in \Gamma^*$, where p is the current control state and W is the current stack content read from top to bottom. The stack height at configuration pW is $|W|$.

The semantics of P are defined with respect to its transition graph $G_P = \{pUW \xrightarrow{a} qVW \mid pU \xrightarrow{a} qV \in \delta,\ W \in \Gamma^*\}$. A *run* of P on input word $w \in \Sigma^*$ from configuration pW is a path in G_P between vertex pW and some vertex qW', written $pW \xrightarrow{w} qW'$. Such a run is successful (or accepting) if $pW = q_0$ and q belongs to the set F of accepting states of P. By $L(G_P, S, T)$ where S, T are sets of vertices of G_P, we mean all words $w \in \Sigma^*$ such that $c \xrightarrow{w} c'$ for some configurations $c \in S$, $c' \in T$. The *language* of P is the set of all words w over which there exists an accepting run; i.e. it is the language $L(G_P, \{q_0\}, F\Gamma^*)$.

A *visibly* pushdown automaton (VPA) over Σ is a weak PDA P where Σ is partitioned as $\Sigma_c \cup \Sigma_r \cup \Sigma_i$, and for all $p \in Q$, $a \in \Sigma$, $p \xrightarrow{a} q\gamma \Rightarrow a \in \Sigma_c$ (a push move/*call*), $p\gamma \xrightarrow{a} q \Rightarrow a \in \Sigma_r$ (a pop move/*return*) and $p \xrightarrow{a} q \Rightarrow a \in \Sigma_i$ (an internal move). VPA can be nondeterministic, i.e. δ need not be a function. VPA are equivalent to the earlier notion of input-driven automata when run on well-matched strings, and their languages can be easily reduced to input-driven languages, as observed in [12, 13].

For any class $\mathcal{C}$ of automata, its arithmetic version $\#\mathcal{C}$ is defined as follows:

$$\#\mathcal{C} = \{f : \Sigma^* \to \mathbb{N} \mid \text{ for some } M \in \mathcal{C},\ f(x) = \#\mathsf{acc}_M(x) \text{ for all } x \in \Sigma^*\}$$

Proposition 1 ([11, 13]). *The following inclusions hold:*

$$\#\mathsf{NFA} \subseteq \#\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{LogDCFL} \qquad \#\mathsf{NFA} \subseteq \#\mathsf{VPA} \subseteq \mathsf{LogDCFL}$$

(A containment $\mathcal{F} \subseteq \mathcal{C}$ involving both a function class $\mathcal{F}$ and a language class $\mathcal{C}$ means: $\forall f \in \mathcal{F}, L^f \in \mathcal{C}$, where $L^f = \{\langle x, i, b\rangle \mid \text{the } i\text{th bit of } f(x) \text{ is } b\}$.)

[4] This is only for simplicity, as all results go through for arbitrary $U, V \in \Gamma^*$.

3 Revisiting the VPL in NC^1 proof

In [9], Dymond proved that the membership problem for VPL is in NC^1. Dymond's proof transforms the problem of recognition/membership to efficiently evaluating an expression whose values are binary relations on a finite set and whose operations are functional compositions and certain unary operations depending on the inputs. This transformation is done in NC^1. Containment in NC^1 follows from the result, due to Buss [18], that the evaluation of formulae involving expressions as k-ary functions over a finite domain is in NC^1.

For a VPA, on an input w, the stack height after processing i letters, $h(i, w)$ (or simply $h(w)$ if $i = |w|$), is the same across any run. Define a set of binary relations, denoted $\Rightarrow^{i,j}$ for $1 \leq i \leq j \leq |w|$, on surface configurations $(q, \gamma) \in Q \times \Gamma$ (state and stack-top pair). These relations are expected to capture all cases where surface configurations are reachable from one another without accessing the previous stack profiles. A unary operation, and a composition operation, are defined on these relations. Given a string w, the main work is to figure out the correct indices for the relations and then the appropriate operations. But that can be accomplished essentially by computing stack heights for various configurations, which is easy for VPL.

As pointed out in [9], the above transformation works for more than VPAs.

Remark 1 ([9]). Dymond's NC^1 membership algorithm works for any pushdown automaton M satisfying the following three conditions.

- There should be no ϵ-moves.
- Accepting runs should end with an empty stack (and a final state).
- There should exist an NC^1-computable function h such that for $w \in \Sigma^*$ and $0 \leq i \leq |w|$, $h(i, w)$ is the height of the stack after processing the first i symbols of w. If M is non-deterministic, then $h(i, w)$ should be consistent with some run ρ of M on w; further, if M accepts w, then ρ should be an accepting run.

Clearly, VPA satisfy these conditions. By definition, they have no ϵ-moves. Though they may not end with an empty stack, this can be achieved by appropriate padding that is computable in TC^0, see for instance [12, 13]. (TC^0 is a subclass of NC^1 consisting of polynomial size constant depth circuits where each gate is allowed unbounded fan-in, and gates compute MAJORITY and NOT.) Though VPA may be nondeterministic, all runs have the same height profile, and the function $h(i, w)$ can in fact be computed in TC^0.

Since any computation up to NC^1 can be allowed for Dymond's proof to go through, VPL do not fully exploit Dymond's argument. We explore a natural generalisation of VPA allowing us to define natural classes for which Dymond's scheme or its precursor from [8] may work for deciding membership, and then examine the power of counting in these models.

4 More general height functions: height-determinism

Adding a pushdown stack to an NFA significantly increases the complexity of both membership (regular to context-free) and path-counting (#NFA to #PDA). However, if stack operations are restricted to an input-driven discipline, as in VPA, then membership is no harder than for NFA, and path-counting seems easier than over general PDA. What is being exploited is that, despite nondeterminism, all paths on a given input word have the same stack-profile, and this profile is computable in NC^1 (and even in TC^0). One can view the partitioning of the input alphabet as height advice being provided to an algorithm for deciding membership. This naturally leads to the question: what can be deduced from the existence of such height advice, independently of how this function is computed?

The term *height-determinism*, coined by [14], captures precisely this idea. A PDA is height-deterministic if the stack height reached after any partial run depends only on the input word w which has been read so far, and not on non-deterministic choices performed by the automaton. Consequently, in any (real-time) height-deterministic pushdown automaton (rhPDA), all runs on a given input word have the same stack profile. Another way to put it is that for any rhPDA P, there should exist a *height-advice function* h from Σ^* to integers, such that $h(w)$ is the stack-height reached by P on any run over w.

Any rhPDA that accepts on an empty stack and whose height-advice function h is computable in NC^1 directly satisfies the conditions in Remark 1, and hence its membership problem lies in NC^1. In this section, we explore some sub-classes of rhPDA and discuss the complexity of their membership and counting problems.

Let us first give a formal definition of rhPDA.

Definition 1 (rhPDA, [14]). *A real-time (weak) pushdown automaton*[5] $P = (Q, q_0, F, \Gamma, \Sigma, \delta)$ *is called height-deterministic if it is complete (does not get stuck on any run), and* $\forall w \in \Sigma^*$, $q_0 \xrightarrow{w} q\alpha$ *and* $q_0 \xrightarrow{w} q\beta$ *imply* $|\alpha| = |\beta|$.

The robustness of this notion is illustrated by the fact that rhPDA retain most good properties of VPA, even when the actual nature of the height-advice function is left unspecified. This had already been obtained in [15] for a slightly different class (which the authors of [14] admittedly used as a starting point in the elaboration of their paper).

Proposition 2 ([14, 15]). *Any* rhPDA *can be determinised. Consequently, for a fixed h, the class of languages accepted by* rhPDA *and whose height advice function is h forms a boolean algebra (and properly includes regular languages). Moreover, language equivalence between two* rhPDA *with the same height-advice function is decidable.*

All these results are effective as soon as h is computable. Since any deterministic real-time PDA is also height-deterministic, another consequence of the fact that rhPDA can be determinised is that the whole class rhPDA accepts precisely the class of real-time DCFL.

[5] In [14], the definition involves rules of the form $pX \xrightarrow{a} q\alpha$ where $\alpha \in \{\epsilon, X\} \cup \{YX | Y \in \Gamma\}$. This is not an essential requirement for the results presented here.

4.1 Instances of height-deterministic PDA

The definition of a rhPDA leaves the exact nature of the height-advice function h unspecified. This is troublesome, since h could be arbitrarily complex. We consider some classes of specific height-advice functions, the simplest being VPA.

Following the framework developed by Caucal [15], we consider classes $\mathcal{T}$ of transducers mapping words to integers. A *transducer* T over Σ and $\mathbb{Z}$ is a transition system $(C, c_0, F, (\Sigma \times \mathbb{Z}), \delta)$, where c_0 denotes the initial configuration and F a set of final configurations, and whose transitions described by δ are labelled with pairs (a, k), where a is a letter and k an integer. The first component of any such label is considered as an input, and the second component as an output. A run $c_0(a_1, k_1)c_1 \dots c_{n-1}(a_n, k_n)c_n$ is associated to the pair $(w, k) = (a_1 \dots a_n, k_1 + \dots + k_n)$. Such a transducer defines a relation $g_T \subseteq \Sigma^* \times \mathbb{Z}$ defined as the set of all pairs (w, k) labelling an accepting run in T.

In our setting, we only consider both input-complete and input-deterministic transducers (i.e. transducers whose underlying Σ-labelled transition system is deterministic and complete), in which all configurations are final (in which case we omit F in the definition). Consequently, for any such transducer T the relation g_T is actually a function, and is defined over the whole set Σ^*. The transition graph G_P of a PDA P is said to be *compatible* with a transducer T if for every vertex s of G_P, if $u, v \in L(G_P, \{q_0\}, \{s\})$ then $g_T(u) = g_T(v)$.

One may consider several kinds of transducers. The simplest class is *finite-state transducers* (FST), where the configuration space C is simply a finite set of control states (often written Q). One may also consider *pushdown transducers* (PDT) whose underlying Σ-labelled transition system is a PDA transition graph, or even more complex transducers (for instance defined using Turing machines).

Definition 2. *For any class $\mathcal{T}$ of complete deterministic transducers,* rhPDA($\mathcal{T}$) *is the class of* rhPDA *whose height function h can be computed by a transducer T in $\mathcal{T}$, in the sense that $h(w) = |g_T(w)|$ (absolute value of $g_T(w)$) for all w.*

The height-advice function of any VPA running on well-matched strings can be computed by a single-state transducer, that reads letters and outputs $+1$ or -1 or 0 depending on whether the letter is in Σ_c or Σ_r or Σ_i. However, note that such single-state transducers can also compute stack-heights for languages that are provably not in VPL. Also, allowing more than one state in a FST provably enlarges the class of languages.

Example 1. The language $EQ(a, b) = \{w \mid |w|_a = |w|_b\}$ is not accepted by any VPA for any partition of $\{a, b\}$. But a single-state transducer can compute the stack-height of the obvious DPDA acceptor: it outputs $+1$ on a and -1 on b.

The language $REV = \{wcw^R \mid w \in \{a, b\}^*\}$ is not a VPL. The obvious DPDA acceptor has a height function computable by a two-state transducer: one state has $+1$ on a and b, the other has -1 on a and b, and moves to second state on c. It is easy to see that for any PDA accepting REV, two states in the transducer are essential for computing stack height.

Further, [14] provided a separating example in rhPDA but not in rhPDA(FST); from Proposition 3 below, it follows that this language is in fact in rhPDA(PDT). In the remainder of this section, we will focus on the classes rhPDA(FST) and rhPDA(PDT), and also to some extent on the class rhPDA(rDPDA$_{1\text{-turn}}$), where the transducer is a 1-turn PDT.

The class we define as rhPDA(FST) is a restricted (and simpler) subclass of the synchronised pushdown automata considered by Caucal in [15]. Even though Caucal's results require, for a PDA to be synchronised by a transducer T, that the transition graph of P' satisfy some additional geometric properties with respect to g_T, these properties are always satisfied when only considering stack-height[6]. One can thus see rhPDA(FST) as the intersection of rhPDA with synchronised PDA. As an aside, we note that [14] considers the class rhPDA(FST) as equivalent to synchronised PDA. This is not guaranteed to be true and has to be proved, since [15] also permits synchronisation by norms other than stack-height.

Finally, we note that since, by definition, rhPDA are complete, it is in fact unnecessary to consider more complex transducers than deterministic and complete PDTs. Formally:

Proposition 3. *For any* rhPDA P *whose height-advice function is* h*, there exists a deterministic and complete pushdown transducer* T *such that* $h(w) = g_T(w)$ *for all* $w \in \Sigma^*$*. That is, every* rhPDA *is in* rhPDA(PDT)*.*

4.2 Complexity of the membership problem

As we already mentioned, rhPDA have exactly the same power as real-time DPDA in terms of accepted languages. This settles the complexity of the membership question for the whole class rhPDA (and thus also for rhPDA(PDT)): it is in LogDCFL, and since the hardest DCFL ([2]) is hard for the class LogDCFL and is accepted by a real-time DPDA, it is also hard for LogDCFL.

We observe easy bounds on the complexity of the height-advice function.

Lemma 1. *For a complete deterministic transducer* T *computing function* g_T*,*

1. *If* T *is a* FST*, then* g_T *is computable in* NC^1*.*
2. *If* T *is a* rDPDA$_{1\text{-}turn}$*, then* g_T *is computable in* L*.*
3. *If* T *is a* PDT*, then* g_T *is computable in* LogDCFL*.*

This allows us to apply Dymond's algorithm for rhPDA(FST).

Lemma 2. *For any fixed* rhPDA(FST)*, the membership problem is in* NC^1*.*

This membership algorithm exploits Dymond's construction better than VPA, as the height function requires a possibly NC^1-complete computation (predicting states of the transducer). Recall that for VPA, the height function is computable in TC^0, a subclass of NC^1.

[6] In the terminology of [15], this is due to the fact that all transition graphs of pushdown automata are regular by stack height.

In [8], the membership problem for VPLs is shown to be in L. We observe that their algorithm can be more explicitly implemented as L^g where g is the height function of the VPL. In this form, it can be generalised to any rhPDA having height function g, as stated in Theorem 1 below. The proof follows from Lemmas 3 and 4, and the result, along with Lemma 1, yields the next corollary.

Theorem 1. *For any fixed* rhPDA P *with height function* g*, the membership problem is in* L^g*.*

Corollary 1. *1. The membership problem for* $\mathsf{rhPDA}(\mathsf{rDPDA}_{1\text{-turn}})$ *is in* L*.*
2. The membership problem for rhPDA *is in* LogDCFL*.*

The class $\mathsf{rhPDA}(\mathsf{rDPDA}_{1\text{-turn}})$ referred to here contains languages accepted by real-time $\mathsf{DPDA}_{1\text{-turn}}$ as well as languages accepted by rhPDA(FST). It is contained in DCFLs. The upper bound for rhPDA follows from [14], where it is shown that rhPDA as a language class equals the DCFLs accepted by DPDA with no ϵ-moves, and so is a proper subclass of DCFLs.

Lemma 4 uses the algorithm from [8] to establish the L^{g_T} bound for well-matched inputs, and Lemma 3 brings the input in that form.

Lemma 3. *For every* $\mathsf{rhPDA}(T)$ P *over an alphabet* Σ*, there is a corresponding* $\mathsf{rhPDA}(T')$ P' *over an alphabet* Σ' *and a* $\mathsf{L}^{g_{T'}}$ *many-one reduction* f *such that for every* $x \in \Sigma^*$, $\#\mathsf{acc}_P(x) = \#\mathsf{acc}_{P'}(f(x))$*, and* $f(x)$ *is well-matched.*

Lemma 4 (Algorithm 2 of [8], stated differently). *Let* $P = (Q, \Sigma, Q_{in}, \Gamma, \delta, Q_F)$ *be a* $\mathsf{rhPDA}(T)$ *accepting well-matched strings. Given an input string* x*, checking if* $x \in L(P)$ *(membership test for* $L(P)$*) can be done in* L^{g_T}*.*

4.3 Complexity of the counting problem

The aspect of rhPDA which interests us in this study is that it is a nondeterministic model capturing the deterministic class LogDCFL. It thus provides a way of arithmetising LogDCFL, simply by counting the number of accepting paths on each word in a rhPDA. We call the class of such functions #rhPDA. In particular, we consider the classes #rhPDA(FST) and #rhPDA(PDT).

We have seen that although rhPDA(FST) properly generalises VPA, the membership problem has the same complexity as that over VPA. It turns out that even the path-counting problem has the same complexity.

Theorem 2. $\#\mathsf{rhPDA}(\mathsf{FST}) \equiv \#\mathsf{VPA}$ *(via* NC^1 *many-one reductions).*

Proof Sketch. VPA are contained in rhPDA(FST), so we only need to show that computing #rhPDA(FST) functions reduces to computing #VPA functions.

Let P be an rhPDA with height-advice computed by FST T. A naive approach would be to construct a single PDA P' that simulates (P, T) by running PDA P along with transducer T. However, such a PDA P' will not necessarily be a VPA. Now consider the string rewritten using an enriched alphabet which consists of the input letter along with a tag indicating whether P should push or pop. On

this enriched alphabet, if the tags are correct, then a PDA that simulates the original PDA P (i.e. ignores the tags) behaves like a VPA. But by Lemma 1, the correct tags for any word can be computed in NC^1. □

Theorem 3 shows that membership and counting for rhPDA have the same complexity, a situation rather unusual for nondeterministic complexity classes.

Theorem 3. *#rhPDA is in LogDCFL.*

The proof of this theorem proceeds in several stages. To compute a #rhPDA function f on input x, we first compute $f(x)$ modulo several small (logarithmic) primes, and then reconstruct $f(x)$ from these residues. This is the standard Chinese remainder technique (see for instance [19]), stated formally below.

Lemma 5 (folklore). *Let P be a fixed rhPDA. There is a constant $c \geq 0$, depending only on P, such that given input x, the number of accepting paths of P on input x can be computed in logarithmic space with oracle access to the language L_{res} defined below. (Here p_i denotes the ith prime number.)*

$L_{res} = \{\langle x, i, j, b\rangle | 1 \leq i \leq |x|^c,$ *the jth bit of* $\#\mathsf{acc}_P(x) \bmod p_i$ *is* $b\ \}$

We now show that L_{res} can be computed by a polynomial time DAuxPDA machine – a deterministic polynomial time PDA with $O(\log n)$ auxiliary space, this model characterises LogDCFL – making oracle queries to the height-advice function g_T. This follows from the technique of [8] as used in [13] to show that #VPA functions are in LogDCFL.

Lemma 6. *If P is any rhPDA and T a PDT computing its height-advice function, then L_{res} is in $\mathsf{LogDCFL}^{g_T}$.*

Lemmas 1 and 6 together imply that L_{res} is in LogDCFL(LogDCFL). This is not adequate for us, since it is not known whether LogDCFL(LogDCFL) $\subseteq$ LogDCFL. (Relativising a space-bounded class is always tricky. Here, we have a pushdown class with auxiliary space, making the relativisation even more sensitive.) However, we further note that the $\mathsf{LogDCFL}^{g_T}$ machine accepting L_{res} makes oracle queries which all have short representations: each query can be written in logarithmic space. (Strictly speaking, the input x is also part of the query. But for eliminating the oracle, this plays no role.) In such a case, we can establish a better bound, which may be of independent interest:

Lemma 7. *Let $L(M^A)$ be the language accepted by a poly-time DAuxPDA M which makes $O(\log n)$-bits oracle queries to a language $A \in$ LogDCFL. Then $L(M^A) \in$ LogDCFL.*

Combining these lemmas proves Theorem 3, since L(LogDCFL) equals LogDCFL.

5 Related models: nested and motley words

In [16], Alur and Madhusudan defined nested word automata (NWA) as an equivalent model for VPA, motivated by applications in software verification and XML

document processing. In [17], Blass and Gurevich defined motley word automata (MWAs) as a generalisation of NWAs. The definitions of models of NWA and MWA are orthogonal to the notion of height-determinism. However, we observe that their complexity bounds are the same as that of VPL for both membership and counting problems.

We begin with definitions of NWA and MWA.

A *nested relation* ν of width n, for $n \geq 0$, is a binary relation over $[1, n]$ such that (1) if $\nu(i, j)$ then $i < j$; (2) if $\nu(i, j)$ and $\nu(i', j')$ then either $\{i, j\} = \{i', j'\}$ or $\{i, j\} \cap \{i', j'\} = \emptyset$, and (3) if $\nu(i, j)$ and $\nu(i', j')$ and $i < i'$ then either $j < i'$ or $j' < j$.

If ν is a nested relation with $\nu(i, j)$, then i is the *call-predecessor* of j and j is the *return-successor* of i. The definition requires that each position has at most one call-predecessor or at most one return-successor but not both.

A nested word over an alphabet Σ is a pair (w, ν) such that $w \in \Sigma^*$, and ν is a nested relation of width $|w|$. A position $k \in [1, |w|]$ of w is a *call position* if $(k, j) \in \nu$ for some j, a *return position* if $(i, k) \in \nu$ for some i, and an *internal position* otherwise.

Definition 3 (NWA). *A nested word automaton (*NWA*) A over an alphabet Σ is a tuple $(Q, q_0, F, \Sigma, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states , $\delta = \langle \delta_c, \delta_i, \delta_r \rangle$ is a set of transitions such that $\delta_c \subseteq Q \times \Sigma \times Q$, $\delta_i \subseteq Q \times \Sigma \times Q$ and $\delta_r \subseteq Q \times Q \times \Sigma \times Q$ are the transitions for call, internal and return positions respectively.*

A starts in state q_0 and reads the word left to right. At a call or internal position, the next state is determined by the current state and input symbol, while at a return position, the next state can also depend on the state just before the matching call-predecessor. A run ρ of the automaton A over a nested word $nw = (a_1 \dots a_n, \nu)$ is a sequence $q_0, \dots, q_n$ over Q such that for each $1 \leq j \leq n$,

- if j is a call position, then $(q_{j-1}, a_j, q_j) \in \delta_c$
- if j is a internal position, then $(q_{j-1}, a_j, q_j) \in \delta_i$
- if j is a return position with call-predecessor k, then $(q_{j-1}, q_{k-1}, a_j, q_j) \in \delta_r$.

A accepts the nested word nw if $q_n \in F$. The language $L(A)$ of a nested-word automaton A is the set of nested words it accepts.

A *motley word* mw of dimension d over Σ is a tuple $(w, \nu_1, \dots, \nu_d)$, where $w \in \Sigma^*$ and $\nu_1, \dots, \nu_d$ are nested relations of width $|w|$.

Definition 4 (MWA). *A motley word automaton (*MWA*) A of dimension d is a direct product $A_1 \times \dots \times A_d$ of d* NWA *$A_1, \dots, A_d$.*[7]

A *run* of A on dimension d motley word $mw = (w, \nu_1, \dots, \nu_d)$ with $|w| = n$ is a sequence $(q_0^1, \dots, q_0^d), \dots, (q_n^1, \dots, q_n^d)$ of states of A such that every $(q_0^k, \dots, q_n^k)$

[7] As NWA are in general non-deterministic, so are motley automata. A MWA $A_1 \times \dots \times A_d$ is deterministic if every nested word automata A_k is so.

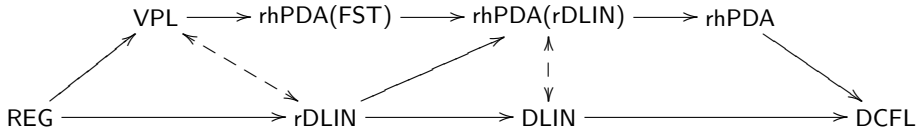

Fig. 1. Summary of language classes

Fig. 2. Summary of language classes closures

is a run of A_k on the nested word (w, ν_k). A run of A on mw is *accepting* (or mw is accepted by A) if each of the d constituent runs is. $L(A)$ is defined as usual.

The languages of nested/motley words accepted by NWA or MWA are called *regular* nested/motley languages. Regular motley languages strictly generalise regular nested languages [17], since for some $i \neq j$, the same position can be a call-position for ν_i and a return position for ν_j.

It is shown in [16] (Theorem 6) that for a fixed NWA, the membership question is in NC^1. The analogous question for a fixed MWA is easily seen to have the same complexity, since it involves answering membership questions for d different, but fixed, NWAs, where d is the dimension of the MWA.

In both models, NWA and MWA, non-determinism is allowed in the definition. We show that path-counting in NWA and MWA is equivalent to that in VPA. This does not follow from the equivalence of membership testing; rather, it requires that the equivalence be demonstrated by a parsimonious reduction.

Theorem 4. *Deciding membership and counting accepting paths in* NWA *and* MWA *are equivalent, via* NC^1*-many-one reductions, to the corresponding problems over* VPA*.*

6 Conclusion

We have studied a range of real-time height-deterministic pushdown automata lying between visibly and real-time deterministic pushdown automata. Fig. 1 depicts the relations between language classes, Fig. 2 shows their closures under appropriate reductions, and Fig. 3 shows the corresponding counting classes. (Dashed arrows indicate incomparability, dotted arrows containment, and solid arrows proper containment.)

Some open questions remain. First, it would be interesting to investigate additional classes lying between rhPDA(FST) and rhPDA(PDT). Also, the only known upper bound for #VPL, #rhPDA(FST) and #rhPDA(rDLIN), is LogDCFL. It would be interesting to refine this bound. Finally, all of our results concern

height-deterministic PDA. However, [15] allows PDA to be synchronised by functions other than stack-height. It is not clear how many of our proofs carry over.

Fig. 3. Summary of counting classes

References

1. Sudborough, I.H.: A note on tape-bounded complexity classes and linear context-free languages. JACM **22**(4) (1975) 499–500
2. Sudborough, I.: On the tape complexity of deterministic context-free language. JACM **25(3)** (1978) 405–414
3. Ibarra, O., Jiang, T., Ravikumar, B.: Some subclasses of context-free languages in NC^1. IPL **29** (1988) 111–117
4. Holzer, M., Lange, K.J.: On the complexities of linear LL(1) and LR(1) grammars. In: 9th FCT. Volume 710 of LNCS. (1993) 299–308
5. Barrington, D.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. JCSS **38**(1) (1989) 150–164
6. Lange, K.J.: Complexity and structure in formal language theory. In: 8th CoCo, IEEE Computer Society (1993) 224–238
7. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL recognition. In: 7th ICALP. Volume 85 of LNCS. (1980) 422–432
8. Braunmuhl, B.V., Verbeek, R.: Input-driven languages are recognized in log n space. In: 4th FCT. Volume 158 of LNCS. (1983) 40–51
9. Dymond, P.: Input-driven languages are in $\log n$ depth. IPL **26** (1988) 247–250
10. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th STOC, ACM (2004) 202–211
11. Caussinus, H., McKenzie, P., Thérien, D., Vollmer, H.: Nondeterministic NC^1 computation. JCSS **57(2)** (1998) 200–212
12. Limaye, N., Mahajan, M., Rao, B.V.R.: Arithmetizing classes arround NC^1 and L. In: 24th STACS. Volume 4393 of LNCS. (2007) 477–488
13. Limaye, N., Mahajan, M., Rao, B.V.R.: Arithmetizing classes arround NC^1 and L. Technical Report ECCC TR07- (2007) submitted to TCS (spl. issue for STACS'07).
14. Nowotka, D., Srba, J.: Height-deterministic pushdown automata. In: MFCS. Volume 4708 of LNCS. (2007) 125–134
15. Caucal, D.: Synchronization of pushdown automata. In: 10th DLT. Volume 4036 of LNCS. (2006) 120–132
16. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: 10th DLT. Volume 4036 of LNCS. (2006) 1–13
17. Blass, A., Gurevich, Y.: A note on nested words. Technical Report MSR-TR-2006-139, Microsoft Research (October 2006)
18. Buss, S.: The Boolean formula value problem is in ALOGTIME. In: 19th STOC, ACM (1987) 123–131
19. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer (1999)