On the Sizes of DPDAs, PDAs, LBAs

by Richard Beigel and William Gasarch

Abstract

There are languages A such that there is a Pushdown Automata (PDA) that recognizes A which is much smaller than any Deterministic Pushdown Automata (DPDA) that recognizes A. There are languages A such that there is a Linear Bounded Automata (Linear Space Turing Machine, henceforth LBA) that recognizes A which is much smaller than any PDA that recognizes A. There are languages A such that both A and $\overline{A}$ are recognizable by a PDA, but the PDA for A is much smaller than the PDA for $\overline{A}$. There are languages A_1, A_2 such that $A_1, A_2, A_1 \cap A_2$ are recognizable by a PDA, but the PDA for A_1 and A_2 are much smaller than the PDA for $A_1 \cap A_2$. We investigate these phenomema and show that, in all these cases, the size difference is captured by a function whose Turing degree is on the second level of the arithmetic hierarchy.

Our theorems lead to infinitely-often results. For example: for infinitely many n there exists a language A_n such that there is a small PDA for A_n, but any DPDA for A_n is large. We look at cases where we can get almost-all results, though with much smaller size differences.

1 Introduction

Let DPDA be the set of Deterministic Push Down Automaton, PDA be the set of Push Down Automata, and LBA be the set of Linear Bounded Automata (usually called $NSPACE(n)$). Let $L(\mathrm{DPDA})$ be the set of languages recognized by DPDAs (similar for $L(\mathrm{PDA})$ and $L(\mathrm{LBA})$). It is well known that

$$L(\mathrm{DPDA}) \subset L(\mathrm{PDA}) \subset L(\mathrm{LBA}).$$

Our concern is with the *size* of the DPDA, PDA, LBA. For example, let $A \in L(\mathrm{DPDA})$. Is it possible that there is a PDA for A that is much smaller than any DPDA for A? For all adjacent

pairs above we will consider these questions. There have been related results by Valiant [12], Schmidt [9], Meyer and Fischer [8], Hartmanis [3], and Hay [4]. We give more details on their results later.

Let Σ be a finite alphabet. All of our languages will be subsets of Σ^*.

Convention 1.1 A *device* will either be a recognizer (e.g., a DFA) or a generator (e.g., a regular expression). We will use $\mathcal{M}$ to denote a set of devices (e.g., DFAs) We will refer to an element of $\mathcal{M}$ as an $\mathcal{M}$-device. If P is an $\mathcal{M}$-device then let $L(P)$ be the language recognized or generated by P. Let $L(\mathcal{M}) = \{L(P) : P \in \mathcal{M}\}$.

Def 1.2 Let $\mathcal{M}$ and $\mathcal{M}'$ be two sets of devices such that every $L(\mathcal{M}) \subseteq L(\mathcal{M}')$. (e.g., DFAs and DPDAs). A *bounding function for* $(\mathcal{M}, \mathcal{M}')$ is a function f such that for all $A \in L(\mathcal{M})$, if $A \in L(\mathcal{M}')$ via a device of size n then $A \in L(\mathcal{M})$ via a device of size $\leq f(n)$.

Def 1.3

1. The *size of a DFA or NDFA* is the number of states it has.
2. The *size of a DPDA or PDA* is the sum of the number of states and the number of symbols in the stack alphabet.
3. The *size of a CFG or CSL* is the number of nonterminals it has.
4. The *size of an LBA* is the sum of the number of states and the number of symbols in the alphabet (note that the alphabet used by the Turing machine may have more symbols than in the input alphabet).

We now give some examples and known results.

Known Upper Bounds:

Example 1.4

1. $f(n) = 2^n$ is a bounding function for (DFA,NDFA) by the standard proof that $L(\mathrm{NDFA}) \subseteq L(\mathrm{DFA})$.

2. $f(n) = n^{n^{n^{O(n)}}}$ is a bounding function for (DFA,DPDA). This is a sophisticated construction by Stearns [10].

3. $f(n) = O(n^{O(1)})$ is a bounding function for (CFG,PDA). This can be obtained by an inspection of the proof that $L(\mathrm{CFG}) \subseteq L(\mathrm{PDA})$.

4. $f(n) = O(n)$ is a bounding function for (PDA,CFG). This can be obtained by an inspection of the proof that $L(\mathrm{PDA}) \subseteq L(\mathrm{CFG})$.

5. $f(n) = O(n)$ is a bounding function for (CSG,LBA). This can be obtained by an inspection of the proof that $L(\mathrm{CSG}) \subseteq L(\mathrm{LBA})$.

6. $f(n) = O(n)$ is a bounding function for (LBA,CSG). This can be obtained by an inspection of the proof that $L(\mathrm{LBA}) \subseteq L(\mathrm{CSG})$.

Known Lower Bounds:

Example 1.5

1. Meyer and Fischer [8] proved that (1) If f is the bounding function for (DFA,NDFA) then $2^n \leq f(n)$. (2) If f is the bounding function for (DFA,DPDA) then $2^{2^{O(n)}} \leq f(n)$. (3) If f is the bounding function for (DFA,CFG) then $HALT \leq_{\mathrm{T}} f$. The sets they used for (3) were finite.

2. Let UCFG be the set all unambiguous context free grammars. Valiant [12] showed that if f is the bounding function for (DPDA,UCFG) then $HALT \leq_{\mathrm{T}} f$

3. Schmidt [9] showed that if f is the bounding function for (UCFG,CFG) then $HALT \leq_{\mathrm{T}} f$

4. Hartmanis [3] showed that if f is the bounding function for (DPDA,PDA) then $HALT \leq_{\mathrm{T}} f$.

5. Hay [4] showed that if f is the bounding function for (DPDA,PDA) then $f \not\leq_{\mathrm{T}} HALT$. She also showed that there is a bounding function f for (DPDA,PDA) such that $f \leq_{\mathrm{T}} INF$. (INF is the set of all indices of Turing machines that halt on an infinite number of inputs. It is complete for the second level of the arithmetic hierarchy and hence strictly harder than $HALT$.)

Note 1.6 The results above that conclude $HALT \leq_{\mathrm{T}} f$ were not stated that way in the original papers. They were stated as either *f is not recursive* or *f is not recursively bounded*. However, an inspection of their proofs yields that they actually proved $HALT \leq_{\mathrm{T}} f$.

Def 1.7 Let $\mathcal{M}$ be a set of devices. A *c-bounding function for $\mathcal{M}$* is a function f such that for all A that are recognized by an $\mathcal{M}$-device of size n, if $\overline{A} \in L(\mathcal{M})$ then it is recognized by an $\mathcal{M}$-device, of size $\leq f(n)$.

We now give some examples and known results.

Example 1.8

1. $f(n) = 2^n$ is a c-bounding function for NDFA. This uses the standard proofs that $L(\mathrm{NDFA}) \subseteq L(\mathrm{DFA})$ and that $L(\mathrm{DFA})$ is closed under complementation.

2. $f(n) = O(n)$ is a c-bounding function for DPDA. This is an easy exercise in formal language theory.

3. $f(n) = O(n)$ is a c-bounding function for LBA. This can be obtained by an inspection of the proof, by Immerman-Szelepcsenyi [5, 11], that nondeterministic linear space is closed under complementation.

Def 1.9 Let $\mathcal{M}$ be a set of devices. An *i-bounding function for* $\mathcal{M}$ is a function f such that for all A_1, A_2 that are recognized by an $\mathcal{M}$-device of size n, if $A_1 \cap A_2 \in L(\mathcal{M})$ then it is recognized by an $\mathcal{M}$- device, of size $\leq f(n)$.

Example 1.10

1. Let $\mathcal{M}$ be any of DFA, DPDA, P, $\mathrm{DTIME}(T(n))$ for any reasonable $T(n)$. $f(n) = O(n)$ is an i-bounding function for $\mathcal{M}$. This uses the standard proofs that these classes are closed under complementation. For that matter $f(n) = O(n)$ is the i-bounding function for any class we can think of that is closed under complementation.

2. $f(n) = 2^{2n}$ is an i-bounding function for NDFAs. Convert both NDFAs to DFAs and then use the standard proof that $L(\mathrm{DFA})$ is closed under complementation.

Note 1.11 We will state our results in terms of DPDAs, PDAs, and LBAs. Hence you may read expressions like $L(\mathrm{PDA})$ and think *isn't that just CFLs?* It is. We do this to cut down on the number of terms this paper refers to.

2 Summary

We will need the following notation and facts to state our results.

Fact 2.1

1. $M_0, M_1, M_2, \ldots$ *is a standard numbering of all Turing Machines.*

2. $M_{e,s}(x)$ *is the result of running* $M_e(x)$ *for* s *steps.*

3. $HALT$ *is the set* $\{(e, x) : M_e(x)$ *halts* $\}$. $HALT$ *is* Σ_1*-complete. Hence any* $\exists$ *question can be phrased as a query to* $HALT$.

4. *INF is the set $\{e : (\forall x)(\exists y, s)[M_{e,s}(y)$ halts $]\}$. INF is Π_2-complete. Hence any $(\forall)(\exists)$ question can be phrased as a query to INF. Note that any $(\exists)(\forall)$ question can also be phrased as a query; however, you will have to negate the answer.*

5. *$A \leq_{\mathrm{T}} B$ means that A is decidable given complete access to set B. This can be defined formally with oracle Turing machines.*

6. *The following is well known. If $f \leq_{\mathrm{T}} HALT$ then there exists a computable g such that, for all n, $f(n) = \lim_{s\to\infty} g(n, s)$.*

We also need the following informal term.

Def 2.2 (Informal) A language is *unnatural* if it exists for the sole point of proving a theorem.

Example 2.3

1. Languages that involve Turing configurations are not natural.

2. Languages created by diagonalization are not natural.

3. The language $\{ww : |w| = n\}$ is natural.

Note 2.4 We will sometimes state theorems as follows: *there exists a (natural) language such that* .... If we do not state it that way then the language is unnatural.

The results of Hartmanis [3] and Hay [4] mentioned above leave open the exact Turing degree of the bounding function for (DPDA,PDA). In Section 3 we resolve this question by proving a general theorem from which we obtain the following:

1. If f is a bounding function for (DPDA,PDA) then $INF \leq_{\mathrm{T}} f$.

2. There exists a bounding function for (DPDA,PDA) such that $f \leq_{\mathrm{T}} INF$.

3. If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists a language A_n such that (1) any DPDA that recognizes A_n requires size $\geq f(n)$, (2) there is a PDA of size $\leq n$ that recognizes A_n. (This follows from Part 1.)

4. If f is a bounding function for (PDA,LBA) then $INF \leq_{\mathrm{T}} f$.

5. There exists a bounding function for (PDA,LBA) such that $f \leq_{\mathrm{T}} INF$.

6. If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists a language A_n such that (1) any PDA that recognizes A_n requires size $\geq f(n)$, (2) there is an LBA of size $\leq n$ that recognizes A_n. (This follows from Part 4.)

In Section 4 and 5 we find the exact Turing degree of the c-bounding function and the i-bounding function for PDAs. We obtain the following:

1. If f is a c-bounding function for PDA then $INF \leq_{\mathrm{T}} f$.

2. There exists a c-bounding function for PDA such that $f \leq_{\mathrm{T}} INF$.

3. If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists a language A_n such that (1) A_n and $\overline{A_n}$ are both PDA, (2) there is no PDA of size $\leq f(n)$ for $\overline{A_n}$, but (3) there is a PDA of size $\leq n$ for A_n. (This follows from Part 1.)

4. Results 1,2,3 but with i-bounding functions instead of c-bounding functions.

Note that we have several results of the form *for infinitely many* n .... We would like to have results of the form *for almost all* n .... In Sections 6 and 7 we obtain the following *for almost all* n result:
For almost all n there exists a (natural) language A_n such that

1. Any DPDA for A_n requires size $\geq 2^{2^{n^{\Omega(1)}}}$.

2. There is a PDA of size $O(n)$ that recognizes A_n.

For almost all n there exists a (natural) language A_n such that

1. Both A_n and $\overline{A_n}$ are recognized by PDAs.

2. Any PDA for $\overline{A_n}$ requires size $\geq 2^{2^{n^{\Omega(1)}}}$.

3. There is a PDA of size $O(n)$ that recognizes A_n.

For almost all n there exists a (natural) language A_n such that

1. Any PDA for A_n requires size $\geq 2^{2^{n^{\Omega(1)}}}$.

2. There is an LBA of size $O(n)$ that recognizes A_n.

In Section 8 we obtain[1] a *for almost all* n result for (PDA,LBA):

Let f be any function such that $f \leq_{\mathrm{T}} HALT$. For almost all n there exists a language A_n such that

1. Any PDA for A_n requires size $\geq f(n)$.

2. There is an LBA of size $O(n)$ that recognizes A_n.

3 Bounding Functions for (DPDA,PDA) and (PDA,LBA)

In this section we prove a general theorem about bounding functions and then apply it to both (DPDA,PDA) and (PDA,LBA). In both cases we show that the Turing degree of the bounding function is the second level of the arithmetic hierarchy.

We will need to deal just a bit with actual Turing Machines.

[1] Meyer originally claimed this result. See the discussion in Section 8.

Def 3.1 Let M be a Turing Machine. A *configuration* (config) of M is a string of the form $\alpha_1 \stackrel{q}{\sigma} \alpha_2$ where $\alpha_1, \alpha_2 \in \Sigma^*$, $\sigma \in \Sigma$, and $q \in Q$. We interpret this as saying that the machine has $\alpha_1 \sigma \alpha_2$ on the tape (with blanks to the left and right), is in state q, and the head is looking at the square where we put the $\stackrel{q}{\sigma}$. Note that from the configuration one can determine if the machine has halted, and also, if not, what the next configuration is.

Def 3.2 Let $e, x \in \mathsf{N}$ Let \$ be a symbol that is not in the alphabet for M_e. We assume that any halting computation of M_e takes an even number of steps.

1. $ACC_{e,x}$ be the set of all sequences of config's represented by

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

such that

- $|C_1| = |C_2| = \cdots = |C_s|$.
- The sequence $C_1, C_2, \ldots, C_s$ represents an accepting computation of $M_e(x)$.

2. Let $ACC_e = \bigcup_{x \in \mathsf{N}} ACC_{e,x}$.

Hartmanis [3] proved the following lemma.

Lemma 3.3 *For all e, x, $\overline{ACC_{e,x}} \in L(\text{PDA})$. For all e, $\overline{ACC_e} \in L(\text{PDA})$. In both cases it is computable to take the parameters ((e, x) or e) and obtain the PDA.*

Def 3.4 Let $\mathcal{M}$ and $\mathcal{M}'$ be two sets of devices.

1. $\mathcal{M} \subseteq \mathcal{M}'$ *effectively* if there is a computable function that will, given an $\mathcal{M}$-device P, output an $\mathcal{M}'$-device P' such that $L(P) = L(P')$.

2. $\mathcal{M}$ is *effectively closed under complementation* if there is a computable function that will, given an $\mathcal{M}$-device P, output an $\mathcal{M}$-device P' such that $L(P') = L(\overline{P})$.

3. The *non-emptiness problem for* $\mathcal{M}$ is the following: *given an* $\mathcal{M}$*-device* P *determine if* $L(P) \neq \emptyset$.

4. The *membership problem for* $\mathcal{M}$ is: *given an* $\mathcal{M}$*-device* P *and* $x \in \Sigma^*$ *determine if* $x \in L(P)$.

5. $\mathcal{M}$ is *size-enumerable* if there exists a list of devices $P_1, \ldots$ such that (1) $\mathcal{M} = \{L(P_i) : i \in \mathsf{N}\}$, (2) $(\forall i)[|P_i| \leq |P_{i+1}|]$, and (3) the function from i to P_i is computable. Note that DFA, NDFA, DPDA, PDA, LBA are all size-enumerable, however UCFG is not.

Theorem 3.5 *Let* $\mathcal{M}$ *and* $\mathcal{M}'$ *be two sets of devices such that the following hold.*

- $L(\mathcal{M}) \subseteq L(\text{PDA}) \subseteq L(\mathcal{M}')$ *effectively.*
- *At least one of* $\mathcal{M}$, $\mathcal{M}'$ *is effectively closed under complementation.*
- *The non-emptiness problem for* $\mathcal{M}$ *is decidable.*
- *The membership problems for* $\mathcal{M}$ *and* $\mathcal{M}'$ *are decidable.*
- *Every finite set is in* $L(\mathcal{M})$.
- $\mathcal{M}$ *is size-enumerable.*

Then

1. *If* f *is a bounding function for* $(\mathcal{M}, \mathcal{M}')$ *then* $HALT \leq_{\text{T}} f$.
2. *If* f *is a bounding function for* $(\mathcal{M}, \mathcal{M}')$ *then* $INF \leq_{\text{T}} f$.
3. *There exists a bounding function* $f \leq_{\text{T}} INF$ *for* $(\mathcal{M}, \mathcal{M}')$.

4. *If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists a language A_n such that (1) any $\mathcal{M}$-device that recognizes A_n requires size $\geq f(n)$, (2) there is an $\mathcal{M}'$-device of size $\leq n$ that recognizes A. (This follows from Part 2 so we do not prove it.)*

Proof:

1) If f is a bounding function for $(\mathcal{M}, \mathcal{M}')$ then $HALT \leq_{\mathrm{T}} f$.

Note that

- If $M_e(x)$ halts then $ACC_{e,x}$ has one string, which is the accepting computation of $M_e(x)$.
- If $M_e(x)$ does not halt then $ACC_{e,x} = \emptyset$.
- Given e, x one can construct a PDA for $\overline{ACC_{e,x}}$ by Lemma 3.3.

We give the algorithm. There will be two cases in it depending on which of $\mathcal{M}$ or $\mathcal{M}'$ is effectively closed under complementation.

ALGORITHM FOR $HALT$ THAT USES f

1. Input(e, x)
2. Construct the PDA P for $\overline{ACC_{e,x}}$. Obtain the device Q in $\mathcal{M}'$ that accepts $\overline{ACC_{e,x}}$.
3. **Case 1:** $\mathcal{M}$ is effectively closed under complementation. Compute $f(|Q|)$. Let $D_1, \ldots, D_t$ be all of the $\mathcal{M}$-devices of size $\leq f(|Q|)$. Create the $\mathcal{M}$ devices for their complements, which we denote $E_1, \ldots, E_t$.

   Case 2: $\mathcal{M}'$ is effectively closed under complementation. Find an $\mathcal{M}'$- device R for $\overline{L(Q)} = ACC_{e,x}$. Compute $f(|R|)$. Let $E_1, \ldots, E_t$ be all of the $\mathcal{M}$-devices of size $\leq f(|R|)$.

   Note that at the end of step 3, regardless of which case happened, we have a set of $\mathcal{M}$-devices $E_1, \ldots, E_t$ such that

$e \in HALT$ iff

$(\exists 1 \leq i \leq t)[L(E_i)$ is one string which represents an accepting computation of $M_e(x)]$.

4. For each $1 \leq i \leq t$ (1) determine if $L(E_i) = \emptyset$ (2) if $L(E_i) = \emptyset$ then let w_i be the empty string, and (3) if $L(E_i) \neq \emptyset$ then, in lexographical order, test strings for membership in $L(E_i)$ until you find a string in $L(E_i)$ which we denote w_i. If $\{w_1, \ldots, w_s\}$ contains a string representing an accepting computation of $M_e(x)$ then output YES. If not then output NO.

END OF ALGORITHM

2) If f is a bounding function for $(\mathcal{M}, \mathcal{M}')$ then $INF \leq_{\mathrm{T}} f$.

Note that

- If $e \in INF$ then $ACC_e \notin L(\mathrm{PDA})$ since ACC_e is infinite and every string in it begins with $\$C_1\$C_2^R\$C_3\$$ where $|C_1| = |C_2^R| = |C_3|$.
- If $e \notin INF$ then $ACC_e \in L(\mathrm{PDA})$ since ACC_e is finite.
- Given e one can construct a PDA for $\overline{ACC_e}$ by Lemma 3.3.

We give the algorithm. There will be two cases in it depending on which of $\mathcal{M}$ or $\mathcal{M}'$ is effectively closed under complementation.

In the algorithm below we freely use Fact 2.1.2 to phrase $(\exists)$-questions as queries to $HALT$, and Part 1 to answer queries to $HALT$ with calls to f.

ALGORITHM FOR INF THAT USES f

1. Input(e)
2. Construct the PDA P for $\overline{ACC_e}$. Obtain the device Q in $\mathcal{M}'$ that accepts $\overline{ACC_e}$.

3. There are two cases.

   Case 1: $\mathcal{M}$ is effectively closed under complementation. Compute $f(|Q|)$. Let $D_1, \ldots, D_t$ be all of the $\mathcal{M}$-devices of size $\leq f(|Q|)$. Create the $\mathcal{M}$ devices for their complements, which we denote $E_1, \ldots, E_t$.

   Case 2: $\mathcal{M}'$ is effectively closed under complementation. Find an $\mathcal{M}'$- device R for $\overline{L(Q)} = ACC_e$. Compute $f(|R|)$. Let $E_1, \ldots, E_t$ be all of the $\mathcal{M}$-devices of size $\leq f(|R|)$.

   Note that at the end of step 3, regardless of which case happened, we have a set of $\mathcal{M}$-devices $E_1, \ldots, E_t$ such that

   $e \in INF \implies ACC_e \notin L(\text{PDA}) \implies ACC_e \notin L(\mathcal{M}) \implies ACC_e \notin \{L(E_1), \ldots, L(E_t)\}$
   $\implies (\exists x_1, \ldots, x_t)(\forall 1 \leq i \leq t)[ACC_e(x_i) \neq E_i(x_i)]$.

   $e \notin INF \implies ACC_e$ is finite $\implies \overline{ACC_e} \in L(\mathcal{M}) \implies (\exists 1 \leq i \leq t)[L(E_i) = L(P_i)]$
   $\implies \neg(\exists x_1, \ldots, x_t)(\forall 1 \leq i \leq t)[ACC_e(x_i) = E_i(x_i)]$.

4. Ask $(\exists x_1, \ldots, x_t)(\forall 1 \leq i \leq t)[ACC_e(x_i) \neq E_i(x_i)]$. (Note that ACC_e is decidable so this is a $(\exists)$ question.) If YES then output YES. If NO then output NO.

3) There exists a bounding function $f \leq_{\text{T}} INF$ for $(\mathcal{M}, \mathcal{M}')$.

In the algorithm below we freely use Fact 2.1.3 to phrase $(\exists)(\forall)$-questions as queries to INF.

Algorithm for f

1. Input(n)
2. MAX=0.
3. For every $\mathcal{M}'$-device P of size $\leq n$ do the following
   (a) Ask $(\exists \mathcal{M}\text{-device } D)(\forall x)[P(x) = D(x)]$?

(b) If YES then for $i = 1, 2, 3, \ldots$ ask $(\exists \mathcal{M}$-device $D, |D| = i)(\forall x)[P(x) = D(x)]$? until the answer is YES.

(c) Let i be the value of i when the last step stopped. Note that $(\exists D, |D| = i)(\forall x)[P(x) = D(x)]$. If $i > MAX$ then $MAX = i$.

4. Output MAX.

∎

Corollary 3.6

1. *If f is a bounding function for (DPDA,PDA) then $INF \leq_{\mathrm{T}} f$.*
2. *There exists a bounding function for (DPDA,PDA) such that $f \leq_{\mathrm{T}} INF$.*
3. *If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists a language A_n such that (1) any DPDA that recognizes A_n requires size $\geq f(n)$ for A_n, but (2) there is a PDA of size $\leq n$ that recognizes A_n.*
4. *If f is a bounding function for (PDA,LBA) then $INF \leq_{\mathrm{T}} f$.*
5. *There exists a bounding function for (PDA,LBA) such that $f \leq_{\mathrm{T}} INF$.*
6. *If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists a language A_n such that (1) any PDA that recognizes A_n requires size $\geq f(n)$, (2) there is an LBA of size $\leq n$ that recognizes A_n.*

Proof: We can apply Theorem 3.5 to all the relevant pairs. since all of the premises needed are either obvious, well known, or follow from the comments in Definition 3.7. ∎

Note 3.7 Since deterministic time classes are effectively closed under complementation we can also apply Theorem 3.5 to get a corollaries about any deterministic time class that contains $L(\mathrm{PDA})$.

Let α be the least number such that two $n \times n$ Boolean matrices can be multiplied in time $O(n^\alpha)$. We abuse notation by letting $\mathrm{DTIME}(n^\alpha)$ be the set of all deterministic Turing machines that run in time $O(n^\alpha)$. Valiant [13] showed that that $L(\mathrm{PDA}) \subseteq L(\mathrm{DTIME}(n^\alpha))$. (Lee [6] showed that if $L(\mathrm{PDA}) \subseteq DTIME(n^{3-\epsilon})$ then $\alpha \leq 3 - (\epsilon/3)$; therefore the problems of $L(\mathrm{PDA})$ recognition and matrix multiplication are closely linked.) Hence we could obtain a corollary about the bounding function for $(\mathrm{PDA}, \mathrm{DTIME}(n^\alpha))$.

4 c-Bounding Functions for PDA

The following theorem can be proven in essentially the same way as Theorem 3.5 hence we just sketch the proof.

Theorem 4.1

1. *If f is a c-bounding function for PDA then $HALT \leq_{\mathrm{T}} f$.*
2. *If f is a c-bounding function for PDA then $INF \leq_{\mathrm{T}} f$.*
3. *There exists a c-bounding function $f \leq_{\mathrm{T}} INF$ for PDA. (This is almost identical to the proof of Theorem 3.5.3 so we do not prove it.)*
4. *If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists a language A_n such that (1) $A_n, \overline{A_n} \in L(\mathrm{PDA})$, (2) there is no PDA of size $\leq f(n)$ for $\overline{A_n}$, but (3) there is a PDA of size $\leq n$ for A_n. (This follows from Part 2 so we do not prove it.)*

Proof:

1. $P_1, P_2, \ldots,$ is a size-enumerable of PDAs.
2. f is a c-bounding function for PDAs.
3. g (when on two variables) is the computable function such that $\overline{ACC_{e,x}}$ is recognized by PDA $P_{g(e,x)}$.

4. g (when on one variable) is the computable function such that $\overline{ACC_e}$ is recognized by PDA $P_{g(e)}$.

1) Let $t = f(g(e,x))$.

$(e,x) \in HALT$ iff $(\exists 1 \leq a \leq t)[L(P_a)$ is an accepting computation of $M_e(x)]$.

Since both the nonemptiness problem and the membership problem for PDAs is decidable this condition can be checked.

2) Let $t = f(g(e))$.

$e \in INF \implies ACC_e \notin L(\text{PDA}) \implies ACC_e \notin \{L(P_1), \ldots, L(P_t)\} \implies$

$(\exists x_1, \ldots, x_t)(\forall 1 \leq i \leq t)[P_i(x_i) \neq ACC_e(x_i)]$.

$e \notin INF \implies ACC_e$ is finite $\implies ACC_e \notin \{L(P_1), \ldots, L(P_t)\} \implies$

$\neg(\exists x_1, \ldots, x_t)(\forall 1 \leq i \leq t)[P_i(x_i) \neq ACC_e(x_i)]$.

We can now use $f \leq_{\text{T}} HALT$ to determine if $(\exists x_1, \ldots, x_t)(\forall 1 \leq i \leq t)[P_i(x_i) \neq ACC_e(x_i)]$. is true or not. ∎

5 i-Bounding Functions for PDA

Def 5.1 We use the same conventions for Turing machines as in Definition 3.2 Let $e, x \in \mathsf{N}$.

1. $ODDACC_{e,x}$ be the set of all sequences of config's represented by

$$\$C_1\$C_2^R\$C_3\$C_4^R\$ \cdots \$C_s^R\$$$

such that

- $|C_1| = |C_2|$ and $|C_3| = C_4$ and ... and $|C_{s-1}| = |C_s|$.
- For all odd i, C_{i+1} is the next configuration after C_i. (Note that we have no comment on, say C_2 and C_3 and they could even be of different lengths.)

- C_s represents an accepting configuration.

2. Let $ODDACC_e = \bigcup_{x \in \mathsf{N}} ODDACC_{e,x}$.

3. $EVENACC_{e,x}$ be the set of all sequences of config's represented by

$$\$C_1\$C_2^R\$C_3\$C_4^R\$ \cdots \$C_s^R\$$$

such that

- $|C_2| = |C_3|$ and $|C_4| = |C_5|$ and ... and $|C_{s-2}| = |C_{s-1}|$.
- For all even i, C_{i+1} is the next configuration after C_i. (Note that we have no comment on, say C_3 and C_4 and they could even be of different lengths. And we have no restriction on C_1.)

4. Let $EVENACC_e = \bigcup_{x \in \mathsf{N}} EVENACC_{e,x}$.

Note that

1. $(e, x) \in HALT$ iff $ODDACC_{e,x} \cap EVENACC_{e,x}$ contains only one string and that string is an accepting computation of $M_e(x)$.

2. $e \in INF$ iff $ODDACC_e \cap EVENACC_e \notin L(\text{PDA})$.

Using these two facts you can prove the theorem below in a manner similar to the proof of Theorem 4.1.

Theorem 5.2

1. *If f is an i-bounding function for PDA then $HALT \leq_{\text{T}} f$.*

2. *If f is an i-bounding function for PDA then $INF \leq_{\text{T}} f$.*

3. *There exists an i-bounding function $f \leq_{\mathrm{T}} INF$ for PDA.*

4. *If $INF \not\leq_{\mathrm{T}} f$ then for infinitely many n there exists languages $A_{n,1}$ and $A_{n,2}$ such that (1) $A_{n,1}, A_{n_2} \in L(\mathrm{PDA})$, (2) there is no PDA of size $\leq f(n)$ for $A_{n,1} \cap A_{n,2}$, but (3) there is a PDA of size $\leq n$ for $A_{n,1} \cap A_{n,2}$.*

6 A Double-Exp For-Almost-All Result Via a Natural Language for (DPDA,PDA)

We show that for almost all n there is a (natural) language A_n such that A_n has a small PDA but $\overline{A_n}$ requires a large PDA. We then use this to show that for almost all n there is a language A_n that has a small PDA but requires a large DPDA.

Lemma 6.1 *Let X, Y, Z be nonterminals. Let Σ be a finite alphabet.*

1. *For all $n \geq 2$ there is a PDA of size $O(\log n)$ that generates $\{Y^n\}$.*

2. *For all $n \geq 2$ there is a PDA of size $O(\log n)$ that generates $\{a, b\}^n$.*

3. *For all $n \geq 2$ there is a PDA of size $O(\log n)$ that generates $\{Y^{\leq n}\}$.*

4. *For all $n \geq 2$ there is a PDA of size $O(\log n)$ that generates $\{a, b\}^{\leq n}$.*

Proof:

We present CFGs of size $O(\log n)$. By Example 1.4 this suffices to obtain PDAs of size $O(\log n)$.

1) We show that there is a CFG of size $\leq 2 \lg n$ that generates $\{Y^n\}$ by induction on n.

If $n = 2$ then the CFG for $\{YY\}$ is

$S \to YY$

which has $2 = 2 \lg 2$ nonterminals.

If $n = 3$ then the CFG for $\{YYY\}$ is

$S \to Y_1 Y \quad | \quad Y_1 \to YY$

which has $3 \leq 2 \lg 3$ nonterminals.

Assume that for all $m < n$ there is a CFG of size $\leq 2 \lg m$ for $\{Y^m\}$. We prove this for n.

- n is even. Let G' be the CFG for $\{Y^{n/2}\}$ with the start symbol replaced by S'. The CFG G for $\{Y^n\}$ is the union of G' and the one rule $S \to S'S'$. This CFG has one more nonterminal than G'. Hence the number of nonterminals in G is

$$\leq 2 \lg(n/2) + 1 = 2(\lg n - 1) + 1 = 2 \lg n - 1 \leq 2 \lg n.$$

- n is odd. Let G' be the CFG for $\{Y^{(n-1)/2}\}$ with the start symbol replaced by S'. The CFG G for $\{Y^n\}$ is the union of G' and the two rules $S \to YS''$ and $S'' \to S'S'$. This CFG has two more nonterminals than G'. Hence the number of nonterminals in G is

$$\leq 2 \lg((n-1)/2) + 2 = 2(\lg(n-1) - 1) + 2 = 2 \lg(n-1) - 2 + 2 = 2 \lg(n-1) \leq 2 \lg n.$$

2) Add the the productions $Y \to a$ and $Y \to b$ to the grammar from Part 1.

3,4) These can be obtained in a manner similar to Parts 1,2. ∎

Theorem 6.2 *For almost all n there exists a (natural) language A_n such that the following hold.*

1. $A_n, \overline{A_n} \in L(\text{PDA})$.
2. *Any PDA that recognizes $\overline{A_n}$ requires size $\geq 2^{2^{n^{\Omega(1)}}}$.*
3. *There is a PDA of size $O(n)$ that recognizes A_n.*

Proof: We show there is a language A_n such that (1) $A_n, \overline{A_n} \in L(\text{PDA})$, (2) any PDA that recognizes $\overline{A_n}$ requires size $\geq 2^{n^{\Omega(1)}}$, (3) there is a PDA of size $O(\log n)$ that recognize A_n. Rescaling this result yields the theorem.

Let $W_n = \{ww : |w| = n\}$. Let $A_n = \overline{W_n}$.

1) A_n is cofinite, so both A_n and $\overline{A_n}$ are in $L(\mathrm{PDA})$.

2) Filmus [2] showed that any CFG for W_n requires size $\geq 2^{\Omega(n)}$. Hence by Example 1.4 any PDA for $W_n = \overline{A_n}$ requires size $\geq 2^{n^{\Omega(1)}}$.

3) We present a CFG for A_n of size $O(\log n)$. By Example 1.4 this suffices to obtain a PDA of size $O(\log n)$.

Note that if $x \in A_n$ then either $|x| \leq 2n-1$ or there are two letters in x that are different and are exactly $n-1$ apart.

The CFG is the union of two CFGs. The first one generates all strings of length $\leq 2n-1$. By Lemma 6.1 there is such a CFG of size $O(\log n)$.

The second one generates all strings of length $\geq 2n$ where there are two letters that are different and exactly $n-1$ apart.

By Lemma 6.1 there is a CFG G' of size $O(\log n)$ that generates all strings of length $n-1$. Let S' be its start symbol. G' will be part of our CFG G, though S' will not be the start symbol.

Our CFG has all of the rules in G' and also the following:

$S \rightarrow UaS'bU \quad | \quad UbS'aU$

$U \rightarrow aU \quad | \quad bU \quad | \quad e$

This CFG clearly generates what we want and is of size $O(\log n)$. ∎

We can now obtain a double exponential result about (DPDA,PDA).

Theorem 6.3 *For almost all n there exists a (natural) language A_n such that the following hold.*

1. *Any DPDA that recognizes A_n requires size $\geq 2^{2^{n^{\Omega(1)}}}$.*

2. *There is a PDA of size $O(n)$ that recognizes A_n.*

Proof: Let A_n be as in Theorem 6.2. We already have that A_n has a PDA of size $O(n)$. We show that any DPDA for A_n is large. Let P be an DPDA for A_n. By Example 1.4 there is a DPDA P' for $\overline{A_n}$ of size $O(|P|)$. By Theorem 6.2 $|P'| \geq 2^{2^{n^{\Omega(1)}}}$, hence $|P| \geq 2^{2^{n^{\Omega(1)}}}$, ∎

7 A Double-Exp For-Almost-All Result Via a Natural Language for (PDA,LBA)

We show that for almost all n there is a (natural) language A_n that has a small LBA but requires a large PDA.

Theorem 7.1 *For almost all n there exists a (natural) language A_n such that the following hold.*

1. *Any PDA that recognizes A_n requires size $\geq 2^{2^{n^{\Omega(1)}}}$.*
2. *There is an LBA of size $O(n)$ that recognizes A_n.*

Proof: We show there is a language A_n such that (1) any PDA for A_n requires size $\geq 2^{n^{\Omega(1)}}$ and (2) there is an LBA of size $O(\log n)$ for A_n. Rescaling this result yields the theorem.

Let $\#_\sigma(w)$ be the number of σ's in w. Let

$$A_n = \{w \mid \#_a(w) = \#_b(w) = \#_c(w)\}.$$

1) Filmus [2] showed that any CFG for A_n requires size $\geq 2^{\Omega(n)}$. Hence, by Example 1.4, any PDA for A_n requires size $\geq 2^{n^{\Omega(1)}}$.

2) We present a CSG for A_n of size $O(\log n)$. By Example 1.4 this yields an LBA of size $O(\log n)$.

Let G_A (G_B, G_C) be the grammar for the language $\{A^n\}$ ($\{B^n\}$, $\{C^n\}$) from Lemma 6.1. Note that G_A, G_B, G_C are all of size $O(\log n)$. Let S_A (S_B, S_C) be the start symbol for G_A (G_B, G_C). Make sure that all of the nonterminals in G_A, G_B, G_C are disjoint.

The CSG G for A_n has start symbol S, all of the productions in G_A, G_B, G_C, and the following rules

$S \to S_A S_B S_C$

$AB \to BA$

$AC \to CA$

$BA \to AB$

$BC \to CB$

$CA \to AC$

$CB \to BC$

$A \to a$

$B \to b$

$C \to c$

Since G_A, G_B, G_C are of size $O(\log n)$, the CSG G is of size $O(\log n)$. ∎

8 A Ginormous For-Almost-All Result for (PDA,LBA)

Meyer and Fisher [8] say the following in their **Further Results** Section:

. . . context-sensitive grammars may be arbitrarily more succinct than context-free grammars . . .

The reference given was a paper of Meyer [7]. That paper only refers to Turing Machines. We exchanged emails with Meyer about this and he informed us that his techniques could be used to obtain the result that is Theorem 8.1 below. Rather than work through his proof we provide our own which is probably similar.

Let $P_1, P_2, \ldots$ be an easily accessible list of all PDAs. They are in order of size. We assume that P_e is of size $\geq e$.

Theorem 8.1 *Let $f \leq_{\mathrm{T}} HALT$. For almost all n there exists A_n such that the following hold.*

1. *Any PDA that recognizes A_n requires size $\geq f(n)$.*

2. *There is an LBA of size $O(n)$ that recognizes A_n.*

Proof: We construct the language A_n by describing an $NSPACE(|x|)$ algorithm. The idea is that A_n will be equivalent (up to a finite number of strings) to a large PDA and will be diagonalized against all small PDAs.

Since $f \leq_{\mathrm{T}} HALT$, by Fact 2.1.5, there exists a computable g such that $(\forall n)[f(n) = \lim_{s\to\infty} g(n,s)]$.

ALGORITHM for A_n

1. Input(x). Let $s = |x|$. Carry out the algorithm given below unless the computation uses more than s space, in which case reject and stop. Note that all steps but the last depend on s but not on x.

2. Compute $g(n,s)$. We denote it by t.

3. Compute A_n on all strings of length $\leq \lg\lg s$.

4. Simulate deterministically $P_1, P_2, \ldots, P_t$ on all inputs of length $\leq \lg\lg s$. Using the membership information about A_n that was calculated in the last step we find some $1 \leq i \leq t$ and $z \in \Sigma^{\leq \lg\lg s}$ (there may not be any) such that $P_i(z) \neq A_n(z)$. Let $ACTIVE$ be the set of i such that no z was found hence it is still possible that $L(P_i) = A_n$.

5. Search for $(a, \{x_i : i \in ACTIVE\})$ such that $a \geq t$ and for all $i \in ACTIVE$, $P_i(x_i) \neq P_a(x_i)$. When one is found goto the next step. We describe the search carefully since it will help in our proof later.

   For $a = t, t+1, t+2, \ldots, 2^t$ do the following.

   (a) $ELIM = \{1, \ldots, a-1\} - ACTIVE$. The set $ELIM$ will be all of the indices i such that we have a witness to $L(P_i) \neq A_n$.

   (b) For $y \in \Sigma^{\leq s}$ in lexographical order, (1) Simulate $P_a(y)$, (2) Simulate all $\{P_i(y) : i \leq a-1, i \notin ELIM\}$, (3) Add to $ELIM$ all i found such that there exists y with

$P_i(y) \neq P_a(y)$.

(c) If after you've gone through all such y you have $ELIM = \{1, 2, \ldots, a-1\}$ then goto the next step with this value of a. Else proceed to the next a.

6. Simulate nondeterministically the PDA for P_a on x.

END OF ALGORITHM for A_n

By the definition of g there exists s_0, t such that, for all $s, s' \geq s_0$, $g(n, s) = g(n, s_0) = t$. Let a be the least number $\geq t$ such that $L(P_a)$ differs from $L(P_1), \ldots, L(P_{t-1})$. It is easy to show that there exists an $s_1 \geq s_0$ such that (1) for all $s \geq s_1$, on inputs of length s_1, the algorithm will find the value a, and (2) for all $s \geq s_1$ the algorithm will finish, i.e., the algorithm will not terminate via using too much space.

We *do not* have that $L(P_a) = A_n$. We *do* have that $L(P_a)$ and A_n differ on only a finite number of strings, hence $L(P_a) \in L(\text{PDA})$.

We show that for all $i < a$, $A_n \neq L(P_i)$. Assume by way of contradiction that there exists $i < a$ such that $L(P_i) = A_n$. Let $s \geq s_1$ be very large (we'll say how large later). Lets look at what happens on an input of length s. Since $L(P_i) = A_n$ the index i will be put in the set $ACTIVE$. Since a is chosen the index i was put into the set $ELIM$. Hence $L(P_i)$ and $L(P_a)$ differ on some string of length $\geq \lg \lg s$. We take s large enough so that A_n and $L(P_a)$ agree on strings of length $\geq \lg \lg s$. Hence the disagreement of $L(P_i)$ and $L(P_a)$ means a disagreement of $L(P_i)$ and A_n. Therefore $L(P_i) \neq A_n$.

The machine operates in nondeterministic linear space since (1) for all but the last step we do not allow it to use more than $|x|$ space, and (2) the last step is a nondeterministic simulation of a PDA which is nondeterministic linear space.

The machine is of size $O(n)$ since the only parameter it uses that is not constant is n. In fact, we could even write the machine down with $O(\log n)$ space but this does not help us. ∎

What was it about PDAs and LBAs that made this proof work? The fact that we could simulate PDAs with LBAs with very little overhead is all we needed. We could formulate a much more general theorem; however, we do not.

9 Open Problems

In Corollary 3.6 we obtained many results of the form *for infinity many* n there is a language A_n such that (1) any BLAH that recognizes A_n requires size $\geq f(n)$, (2) there is a BLAH' of size $\leq n$ that recognizes A_n. The languages A_n were not natural. For that matter, they were not explicit. Can we obtain such results, or perhaps slightly weaker results, with explicit and/or natural languages?

Recall Theorem 6.3: for almost all n there exists a (natural) language A_n such that (1) Any DPDA that recognizes A_n requires size $\geq f(n)$, (2) there is a PDA of size $O(n)$ that recognizes A_n, where $f(n) = 2^{2^{n^{\Omega(1)}}}$. Is the same theorem true with a faster growing f? If we used a non-natural language then can we get a faster growing f? One obstacle is that there are very few techniques available to get lower bounds on DPDAs. In fact we used lower bounds on PDAs, and closure of DPDA under complementation, to get our results. Another obstacles is that PDAs are too weak to allow for a diagonalization proof in the spirit of Theorem 8.1.

Recall Theorem 7.1: for almost all n there exists a (natural) language A_n such that (1) Any PDA that recognizes A_n requires size $\geq f(n)$, (2) there is an LBA of size $O(n)$ that recognizes A_n, where $f(n) = 2^{2^{n^{\Omega(1)}}}$. Is the same theorem true with a faster growing f? If we allow non-natural A_n then we can do much better as shown in Theorem 8.1. How well can we do with natural sets? One obstacle is that there are very few techniques available to get lower bounds on PDAs. We've used the lower bounds of Filmus [2]; however, it is unlikely they can be pushed further.

Recall Theorem 8.1: Let $f \leq_{\mathrm{T}} HALT$. For almost all n there exists (unnatural) A_n such that (1) any PDA that recognizes A_n requires size $\geq f(n)$, (2) there is an LBA for A_n of size $O(n)$. Is the same theorem true with a faster growing f? One possible result would be to look at $f <_{\mathrm{T}} INF$.

Valiant [12] showed that if f is the bounding function for (DPDA,UCFG) then $HALT \leq_{\mathrm{T}} f$

and Schmidt [9] showed that if f is the bounding function for (UCFG,CFG) then $HALT \leq_{\mathrm{T}} f$. Both of these bounding functions can easily be shown to be computable-in-INF; however, it is open to determine the exact Turing degree of these bounding functions. On difficulty is that UCFG is not size-enumerable.

We have shown that it is possible to have large size differences between (say) DPDAs and PDAs. But what happens in the typical case? It is open to even state the question and what the answer is.

10 Acknowledgment

We thank Albert Meyer for help with Theorem 8.1 and for pointing us to Schmidt's paper. We thank Karthik Gopalan and Rebecca Kruskal for help with some of the proofs and for proofreading. We thank Sam Zbarsky for help with Lemma 6.1. We thank Jefferey Shallit whose paper [1] inspired this paper,

References

[1] K. Ellul, B. Krawetz, J. Shallit, and M. Wang. Regular expressions: new results and open problems. *Journal of Automata, Languages, and Combinatorics*, 9(2):233–256, 2004.

[2] Y. Filmus. Lower bounds for context-free grammars. *Information Processing Letters*, 111(18):895–898, 2011.

[3] J. Hartmanis. On the succinctness of different representations of languages. *SIAM Journal on Computing*, 9(1):114–120, 1980.

[4] L. Hay. On the recursion-theoretic complexity of relative succinctness of representations of languages. *Information and Computation*, 52(1):1–7, 1982.

[5] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. Prior version in *C*onf. on Structure in Complexity Theory, 1988.

[6] L. Lee. Fast contex-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM*, 49(1):1–15, 2002.

[7] A. Meyer. Program size in restricted programming languages. *Information and Control*, 21(4):382–394, 1972.

[8] A. Meyer and M. Fischer. Economy of description by automata, grammars and formal systems. In *Proceedings of the 12th Annual symposium on Switching and Automta Theory*, pages 188–191, Washington, DC, 1971. IEEE.

[9] E. Schmidt. Succintness of descriptions of unambigous context-free languages. Technical Report Technical Report 76-277, Cornell University, Dept of Computer Science, April 1976. `http://ecommons.library.cornell.edu/handle/1813/7319`.

[10] R. E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(2):323–340, 1967.

[11] R. Szelepcsenyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 25(3):279–284, 1988.

[12] L. Valian. A note on the succintness of descriptions of deterministic languages. *Information and Control*, 32(2):139–145, 1976.

[13] L. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975.