TWO WAY DETERMINISTIC PUSHDOWN AUTOMATON LANGUAGES AND SOME OPEN PROBLEMS IN THE THEORY OF COMPUTATION

Zvi Galil

TR 74-204

April 1974

Department of Computer Science
Cornell University
Ithaca, New York 14850

TWO WAY DETERMINISTIC PUSHDOWN AUTOMATON LANGUAGES AND SOME OPEN PROBLEMS IN THE THEORY OF COMPUTATION

Zvi Galil

Abstract

We consider some of the important unsolved problems in the theory of computation concerning the relationship between deterministic and nondeterministic computations, and between tape and time bounded computations. For each such problem we find an equivalent problem concerning two way deterministic pushdown automaton languages. This is the first time many of the open problems have been reduced to questions about one class of automata.

Keywords and Phrases:

Two way deterministic pushdown automata, open problems, determinism versus nondeterminism, space bounded computations, time bounded computations, Turing machines, multihead pushdown automata, two way counter machines, auxiliary pushdown machines.

CR Categories: 5.22, 5.23

0. Introduction

In the theory of computation we can find several open problems. One of them the LBA problem [10], has been around more than ten years. These problems concern the relationship between tape and time bounded computations and the relationship between deterministic and nondeterministic computations.

Each of these problems is of the following form: There are two classes of languages, $\mathcal{L}_A$ and $\mathcal{L}_B$, and two classes of machines $\mathcal{M}_A$ and $\mathcal{M}_B$, which characterize them. In each case it is easy to prove that $\mathcal{L}_A \subseteq \mathcal{L}_B$ and the open problem is whether $\mathcal{L}_A$ equals $\mathcal{L}_B$. It is usually the case that there are languages in $\mathcal{L}_B$ that are easily shown to be there, but nobody can prove that they are not in $\mathcal{L}_A$; there is still a chance that although $\mathcal{L}_B$ looks much richer, each machine in $\mathcal{M}_B$ can be simulated by some machine in $\mathcal{M}_A$, perhaps using clever tricks.

Although unable to solve these open problems, researchers have recently reduced them to various other, seemingly simpler problems. One of the most common approaches was to construct $L \in \mathcal{L}_B$ such that $L \in \mathcal{L}_A$ iff $\mathcal{L}_B = \mathcal{L}_A$. Thus, the question 'can we simulate all machines in $\mathcal{M}_B$ by machines in $\mathcal{M}_A$?' was reduced to the question 'can we accept L by a machine in $\mathcal{M}_A$?', which seems much easier. Moreover, we try to construct an L as simple as possible and thus to reduce the original problem to the simplest form.

In this paper we follow this approach and reduce some of the most important open problems of the theory of computation to questions about two-way pushdown automaton languages (2DPDA).

The idea of reducing open problems to other, seemingly simpler problems is not new. But this is the first time most of the open problems about the relationship between tape and time bounded computations and between nondeterministic and deterministic computations are reduced to questions about one class of automata. So far every open problem has been dealt with separately. Moreover, this class is a model for simple computations, since S. Cook proved in [5] that every two-way deterministic pushdown automaton (2dpda) can be simulated in linear time by a random access machine (RAM). Note that we cannot say the same about any class of languages (or automata) to which the open problems were reduced in the past. Another innovation is the rather surprising fact that we can reduce problems about nondeterminism versus determinism to questions about 2dpda's which don't mention nondeterminism at all. So far, the question 'Does non-determinism buy us more computational power?' was reduced either to the same question for simpler machines,† or to the question whether a specific nondeterministic language can be accepted deterministically as well.

We are mainly interested in 2dpda's, but we will also consider two-way deterministic counter machines (2dc's) and two-way nondeterministic counter machines (2nc's).

In Section 1 we list the open problems of the theory of computation, that are discussed in the paper. In Section 2 we

---

†e.g. in [21] it was reduced to the question whether every two-head, one-way, nondeterministic finite automaton can be simulated by a k-head, two-way deterministic finite automaton.

describe briefly the classes of pushdown automaton languages and some unsolved questions about them. In the first two sections we also introduce our notation. In Section 3 we relate some questions about pushdown automaton languages and the relationship between deterministic and nondeterministic tape bounded computations. In Section 4 we relate some questions about two-way deterministic pushdown automaton languages and open problems involving the class of languages which can be accepted in polynomial time. In Section 5 we relate some open problems to questions about two-way deterministic counter machines.

1. Some Open Problems of the Theory of Computation

In this section we list six of the most important unsolved problems of the theory of computation. In the sequel we will relate these problems to questions about languages accepted by two-way deterministic pushdown automata. First we introduce the notation we use to describe these open problems.

By a TM we shall mean an off-line multi-tape Turing machine[†] defined as a recognition device as in [12]. By convention, such an automaton usually receives its input (a member of Σ^+, where Σ is the input alphabet) on its input tape. We will write DTM (NTM) for a deterministic (nondeterministic) TM.

A TM, M, accepts the word $x \in \Sigma^+$ if some computation by M on x halts in accepting state. M accepts the set $L(M) = \{x \in \Sigma^+ |$ M accepts $x\}$. M accepts x within time t (within space s) if some

[†]When we discuss tape bounded computations we assume w.l.o.g. that the TM has only one working tape.

computation by M on x halts in accepting state after at most t steps (using at most s tape squares). For convenience we shall say that a TM, M, operates within t steps on x if either M accepts x within t steps or M does not accept x.

M operates within time T(n) if it operates on each $x \in \Sigma^+$ within time $T(|x|)$.[†]

Define NTIME(T(n)) = $\{L \subseteq \Sigma^+ | L = L(M)$ for some NTM M which operates within time T(n)$\}$ and DTIME(T(n)) = $\{L \subseteq \Sigma^+ | L = L(M)$ for some DTM M which operates within time T(n)$\}$.

We define similarly the notions of operation within space S(n) and the classes NSPACE(S(n)) and DSPACE(S(n)).

We also define P-TIME = $\bigcup_{k=1}^{\infty}$ DTIME(n^k), NP-TIME = $\bigcup_{k=1}^{\infty}$ NTIME(n^k), P-SPACE = $\bigcup_{k=1}^{\infty}$ DSPACE(n^k), DLBA = DSPACE(n), NLBA = NSPACE(n), DLOG = DSPACE(ℓogn) and NLOG = NSPACE(ℓogn). We are ready now to give a list of the open problems which are the subject of this paper.

A Determinism versus nondeterminism in tape bounded computations.

1. Is DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq$ ℓogn?[††] The special case S(n) = n is called the LBA problem and has already been around for ten years ([10]). The problem was reduced to the question of whether a specific language L in NLOG (NLBA),

---

[†]For any string $w = a_1 \ldots a_n$, |w| denotes the length n of w.

[††]All logarithms in this paper are of base 2. Also $f(n) \geq g(n)$ means that there is a constant $c > 0$ such that $f(n) > cg(n)$ for all n.

is in DLOG (DLBA)?[†] ([18],[10]). Among examples of such an L are a context-free language and a nondeterministic one way two head finite automaton language ([21]).

B Determinism versus nondeterminism in time bounded computations.

2. Is P-TIME = NP-TIME?[††]

This problem is probably the most important one, since many believe that P-TIME is a reasonable candidate for the class of feasible computations. Many combinatorial problems are easily shown to be in NP-TIME, but nobody knows if they are in P-TIME. Some of them[†††] have the property that they belong to P-TIME iff P-TIME = NP-TIME ([3],[16]). This is again reducing the question about a class of languages to a similar question about one language in the class.

In [2] the question is reduced to the question: "Is P-TIME closed under nonerasing homormorphism?" or "Is NTIME(n) $\subseteq$ P-TIME?"

C Space bounded computations versus time bounded computations.

3. Is P-TIME = P-SPACE?

4. Is P-TIME = DLOG?

[†]These are the $<_{LOG}$ complete in NLOG. We preferred not to use this terminlogy since we deal with different kinds of reducibilities.

[††]The corresponding problem for space is solved [18], and the answer is positive. However, most researchers conjecture that the answer is negative for time.

[†††]The so-called polynomial complete problems.

The last two problems show very clearly how poor is our knowledge about the relationship between tape and time bounded computations (since DLOG $\subsetneq$ DLBA $\subsetneq$ P-SPACE).[†]

Another problem is:

5. Is P-TIME $\subseteq$ DSPACE($(\log n)^i$)[††] for some i?
   Cook, and Cook and Sethi have proved ([6],[7]) that the answer is negative when we consider very restricted computations. Jones and Laaser [15] reduced the last two problems to questions about specific languages in P-TIME.[†††]

D Does an auxiliary stack add to the computational power of tape bounded TM's?

In [4] Cook has introduced the notion of PDM(S(n)), i.e. a TM using a pushdown stack and operating within S(n) space (not counting the space used by the stack). He then proved the surprising result that NPDM(S(n)) = DPDM(S(n)) for all S(n) $\geq$ &log n. The last open problem in our list is

6. Is PDM(S(n)) = DSPACE(S(n)) for all S(n) $\geq$ &log n?

Note that the six problems are not independent. For example P-TIME = DLOG => DSPACE(S(n)) = NSPACE(S(n)) for all S(n) $\geq$ &log n and P-SPACE = P-TIME => P-TIME = NP-TIME. In Section 4 we will see that problems 6 and 4 are actually the same.

[†] Book [2] has proved that P-TIME $\neq$ DLBA.

[††] Note that problem 4 is the same as problem 5 when i = 1 since DLOG $\subseteq$ P-TIME.

[†††] The $<_{LOG}$ complete problems for P-TIME.

Figure 1.1 describes some of the classes mentioned above. Circle A is inside circle B if $A \subseteq B$. The open problems are: Are these inclusions strict? We only know that DLOG and NLOG (the two innermost circles) are strictly contained in P-SPACE (the outermost circle).

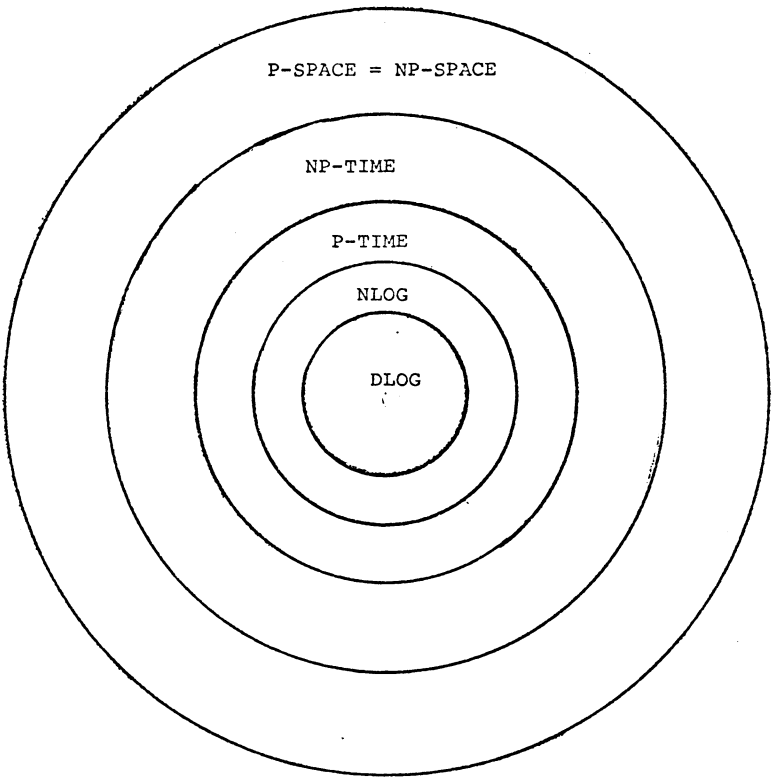

Figure 1.1

2. Pushdown Automaton Languages and Some Unsolved Questions About Them

Although most of our results concern two-way deterministic pushdown automaton languages, some of the proofs and some of the results deal with much more general definitions from which we derive all other classes as special cases. The definitions (and hence part of the proofs) are informal. For formal definitions see [14].

A two-way nondeterministic k-head pushdown automaton, k-npda, is a finite control attached to a read only input tape and a pushdown stack. The input tape will contain a string ¢x$ where $x \in \Sigma^+$, where Σ is a finite alphabet, and ¢ and $ are left and right endmarkers, respectively. There are k heads on the input tape which communicate with the finite control as follows: The automaton depending on its state, the k symbols scanned by its heads and its top stack symbol can choose one of a finite number of possible moves to perform. A move consists of 1) changing state, 2) popping off the top symbol of the stack, 3) pushing some symbols onto the stack and 4) moving some of the heads one square to the left and some others one to the right.† The k-npda starts in a starting state with all its heads scanning the ¢. It accepts the input if some choice of moves takes it to an accepting state. The language accepted by a k-npda is defined as for TM's. A k-dpda is defined as above, except that at most one choice of move is defined for each k-tuple of input symbols, state and stack symbol.

---

†No head can move left (right) when it scans ¢ ($).

Let DPDA(k) = {L|L = L(A) for some k-dpda A}.

We will mostly be interested in the case k = 1. In this case the automaton is a 2dpda and the class of languages is 2DPDA.†

Similarly we define all the classes related to 2npda. We will be interested in the counter machines (always with one head). Namely, the 2dc (2nc) which is a 2dpda (2npda) which uses only one stack symbol (in addition to one extra symbol which marks the bottom of the stack). The class 2DC (2NC) is defined in the obvious way.

We will mention also a k-head deterministic (nondeterministic) finite automaton, k-dfa (k-nfa). It is a k-dpda (k-npda) without a stack and its next move is determined by its state and the k symbols which are scanned by its heads. The corresponding classes are defined in the obvious way.

2dpda's were introduced in [8]. We don't find much about them in the literature and all the open problems which were mentioned in the introductory paper are still unsolved. In [5] Cook proved that the membership problem for 2dpda's can be solved on a RAM (Random access machine) in linear time. This gives a proof that some pattern matching type problems can be solved in linear time (e.g. recognition of palindromes). A 2dpda can perform some computations in a very strange way. It is an amusing exercise to show that each of $L_1 = \{ww^R u \quad | w \in \Sigma^+\}$,†† $L_2 = \{w_1 w_1^R w_2 w_2^R u \quad | w_1, w_2 \in \Sigma^+\}$ and $L_3 = \{0^n 1^{n^2} | n \geq 1\}$ is in 2DPDA.

---

† The 2 stands for 'two-way' and is used to prevent confusion with the one way device. For k > 1 "k-dpda" ("k-npda") will always mean two-way device.

†† If $w = a_1 \ldots a_n$ then $w^R = a_n \ldots a_1$.

Thus a 2dpda can simulate, in some sense, a bounded number of guesses and can perform some integer arithmetic. Therefore, Cook's result is of a great significance and justifies further effort to study this class (We don't know even how to simulate a 2-dfa in linear time on a RAM).

The main reason which, in our opinion, accounts for the fact that not much is known about 2DPDA is that the head can move in two ways.† Therefore, a 2dpda cannot be characterized by a simple grammar†† and we cannot apply any sort of 'Pumping lemmas' to produce a language not in 2DPDA. Actually, the only languages, that we know, not to be in 2DPDA either are obtained by a straightforward diagonalization or are languages which are shown not to be in P-SPACE ([17],[13]).

We are ready to describe the unsolved questions which we are going to relate to the open problems mentioned in Section 1. Figure 2.1 shows the current state of knowledge about some of the classes we defined earlier.

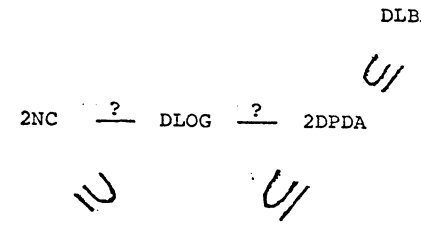

Figure 2.1

'$\subseteq$' is inclusion and '$\subseteq$? $\subseteq$' stands for an unknown relation.

†The same difficulty implies that we cannot prove that some simple languages cannot be accepted by 2-dfa. Sudborough has proved recently ([20]) that some simple languages cannot be accepted by any k-dfa which makes a bounded number of head reversals.

††From what follows we will see that every $L \in$ 2DPDA is generated by some context-sensitive grammar, but this does not help much.

Inclusions 2 and 3 follow trivially from the definitions. For completeness we prove below inclusion 1, which is a well known fact ([8]). We also need the proof, since we will refer to it in Section 5.

Lemma 2.2: 2DPDA $\subseteq$ DLBA .

Proof: Let A be a 2dpda with s states and t stack symbols which can push at most ℓ symbols on its stack in one move. Assume that the input is x with $|\not{c}x\$| = n$ and at some point of the computation the length of the stack is bigger than $s.t.\ell.n$. Consider the computation up to this point. In the process of growing the stack there must be two points at which

1. The lengths of the stack are ℓ_1 and ℓ_2, $\ell_1 < \ell_2$.
2. A scans the same symbol, is in the same state and has the same top symbol on its stack, and
3. During the computation from the first point to the second the length of the stack is never shorter than ℓ_1.

This implies that the computation must diverge (overflowing the stack). Thus $x \in L(A)$ => the length of the stack never exceeds $st\ell n$ => an LBA (linear bounded automaton) can simulate A using linear tape instead of the stack => $L(A) \in$ DLBA.

As in the situation with the open problems, which we mentioned in Section 1, all inclusions in Figure 2.1 are probably strict (and all '$\subseteq$' stand probably for the fact that the classes are incomparable), but we cannot prove it.

We do not solve here any of the problems which arise from Figure 2.1. In Section 4 we will get as a corollary that 2DPDA $\neq$ DLOG. It is also easy to see that one of inclusions 1 and 2 must be strict, since DLBA $\supsetneq$ 2DC. (In Section 5 we prove that 2DC $\subseteq$ DLOG and thus the claim follows from the hierarchy of tape bounded computation established in [19]).

We are ready now to describe briefly the main results of this paper. In Section 3 we construct a 2dpda-language (1nc-language), L, such that L $\in$ DLOG iff DSPACE(S(n))= NSPACE(S(n)) for all S(n) $\geq$ ℓogn. In Section 4 we construct a family of languages $\{L_k\}_{k=0}^{\infty}$ ($\{L_k'\}_{k=0}^{\infty}$) such that $L_k \in$ 2DPDA ($L_k' \in$ 2DPDA) for some k iff P-SPACE = P-TIME (P-TIME = NP-TIME). We also construct a 2dpda-language, L, such that L $\in$ DSPACE($(\ell ogn)^i$) for some i (L $\in$ DLOG) iff P-TIME $\subseteq$ DSPACE($(\ell ogn)^i$) (P-TIME = DLOG or PDM(S(n)) = DSPACE(S(n)) for all S(n) $\geq$ ℓogn). In Section 5 we construct several families of 2dpda-languages (2nc-languages) $\{L_k\}_{k=0}^{\infty}$. We prove in the case of 2dpda-languages that $L_k \in$ 2DC for some k iff NSPACE(S(n)) = DSPACE(S(n)) for all S(n) $\geq$ ℓogn (or iff P-TIME = DLOG, or iff PDM(S(n)) = DSPACE(S(n)) for all S(n) $\geq$ ℓogn for other families) and in the case of 2nc-languages that $L_k \in$ 2DC for some k iff NSPACE(S(N)) = DSPACE(S(n)) for all S(n) $\geq$ ℓogn.

As we mentioned above, Figure 2.1 includes only those questions which we relate to the six open problems we described in Section 1. A very important question —namely 'is 2NPDA = 2DPDA?' — is missing. We could not relate it to any of the open problems.

3. Pushdown Automaton Languages and the Relationship between Deterministic and Nondeterministic Tape Bounded Computations

In [18], Savitch defined the set M_Σ of threadable mazes over Σ. He proved that the existence of a DTM that operates in $\log n$ space which recognized M_Σ would imply that DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \log n$. We review briefly the definition of M_Σ.[†]

Definition 3.1: Let [,], #, and Δ be four symbols not in Σ. A maze over Σ is a string $\mathcal{M}$ of the form

$$s\,[x_1 \# y_1^{(1)} \# y_2^{(1)} \ldots \# y_{n(1)}^{(1)}] \ldots [x_t \# y_1^{(t)} \ldots \# y_{n(t)}^{(t)}]$$

where $n(i) \geq 0$, $s, y_j^{(i)} \in \Sigma^+$ and $x_i \in \Sigma^+ \cup \Sigma^+\Delta$, $1 \leq i \leq t$, $1 \leq j \leq n(i)$.

Definition 3.2: A maze $\mathcal{M}$ is threadable if there is a sequence $i_1, i_2, \ldots, i_k$ such that $s = x_{i_1}$, $x_{i_k} \in \Sigma^+\Delta$ and for each ℓ, $1 \leq \ell \leq k-1$, there is a j_ℓ, $1 \leq j_\ell \leq n(i_\ell)$, with $x_{i_{\ell+1}} = y_j^{(i_\ell)}$.

Notation: M_Σ denotes the set of all threadable mazes over Σ.

Savitch proved the following result in [18]:

Theorem 3.3: $M_\Sigma \in$ DLOG iff DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \log n$.

[†] Our definition differs from the definition given in [18] in two ways: 1) We identify mazes and their encodings and 2) In [18] the string is followed by the string $u_1 \# u_2 \ldots \# u_r$ which is a list of what he called the goal rooms. In our definition a goal room is denoted by x_i, $x_i \in \Sigma^+\Delta$.

Proof: Athough we made some changes in the definition of M_Σ, the proof in [18] still holds. We sketch below the essential elements in that proof: If an NTM Z which operates within space $S(n) \geq \ell ogn$ and an input x are given, then one can define a sequence of mazes $\{\mathcal{M}^k_{Z,x} \mid k=1,2...\}$ such that

a) The x_i's are the i.d.'s[†] of Z on x under the assumption that k tape cells are used,

b) $x_i \in \Sigma^+\Delta$ iff it is an accepting i.d.,

c) s is the initial i.d. of Z on x and

d) $y^{(i)}_j$ $1 \leq j \leq n(i)$ are all the i.d.'s which follow from x_i by one move of Z.

For all $k \geq 1$ $\mathcal{M}^k_{Z,x}$ satisfies

(1) $\mathcal{M}^k_{Z,x}$ is threadable iff Z accepts x within space k.

(2) Given x as an input, a DTM can generate the i-th symbol of $\mathcal{M}^k_{Z,x}$ $(1 \leq i \leq |\mathcal{M}^k_{Z,x}|)$ by using no more than max(ℓogn,k) tape cells.

We will modify M_Σ to get a 2dpda-language M'_Σ and a lnc-language M''_a for which a theorem analogous to Theorem 3.3 still holds. Hence we will be able to establish the existence of a language $L \in$ 2DPDA (1NC) such that $L \in$ DLOG iff DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \ell ogn$.

Definition 3.4: A maze is binary if for every i, $1 \leq i \leq t$ $n(i) \leq 2$ in Definition 3.1.

[†]i.d. stands for instantenous description.

Definition 3.5: A maze is monotone if a) $x_{i'} > x_i$ for $i' > i$ and b) $y_j^{(i)} > x_i$ for $1 \leq j \leq n(i)$.[†]

Lemma 3.6: Given an NTM, Z, which operates within space $S(n) \geq \ell\log n$ and an input x, we can define a sequence of monotone binary mazes $\{\mathcal{M}'^{k}_{Z,x} | k=1,2...\}$ which all satisfy properties 1) and 2) of $\mathcal{M}^{k}_{Z,x}$ in the proof of Theorem 3.3.

Proof: We first construct a sequence of NTM's, $\{Z'_k | k=1,2...\}$, which satisfy

a) For all k, Z'_k has at most two choices for the next move (This can easily be achieved by adding more states.);

b) Z'_k simulates Z on one track of its working tape as long as it uses less than or equal to k tape cells while counting its steps on a lower track;

c) Z'_k always halts, since it can detect when Z loops on k tape-cells; and

d) Z'_k accepts x iff Z accepts x within space k.

For every $k \geq 1$ let $\mathcal{M}'^{k}_{Z,x}$ be the maze which corresponds to Z'_k, x and k (i.e. replace Z by Z'_k in the definition of $\mathcal{M}'^{k}_{Z,x}$ in the proof of Theorem 3.3). By property (a) of Z'_k, $\mathcal{M}'^{k}_{Z,x}$ is binary. By property (b) of Z'_k $\mathcal{M}'^{k}_{Z,x}$ can be choosen to be monotone. This is done by ordering the tape symbols giving high priority to the symbols of the second track and adding some symbols in such a way that the process of updating the counter in the second track

[†] If $|\Sigma| = m$ and we assign some arbitrary order to Σ, then we can consider $x \in \Sigma^+$ as an integer in m-ary notation.

preserves monotonicity. We omit the tedious details. It is easy to verify that $\mathcal{M}'^{k}_{Z,x}$ satisfies properties 1) and 2) as was claimed above.

Notation: M'_{Σ} denotes the set of all monotone binary threadable mazes.

Theorem 3.7: $M'_{\Sigma} \in$ DLOG iff DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \ell ogn$.

Proof: The proof is the same as the proof of Theorem 3.3, except that $\mathcal{M}'^{k}_{Z,x}$ is used instead of $\mathcal{M}^{k}_{Z,x}$ for k = 1,2...

We are ready now to prove the main theorem of this section.

Theorem 3.8: There exists a language L, L $\in$ 2DPDA, such that L $\in$ DLOG iff DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \ell ogn$.

Proof: We show that $L = M'_{\Sigma} \in$ 2DPDA and the above claim will follow directly from Theorem 3.7. We give below a program for a 2dpda, A, that checks if its input is in M'_{Σ}. It will use its stack to keep track of the path it is trying to find from s to some $x \in \Sigma^{+}\Delta$. At any stage of the computation the stack will contain the string $\bar{3}x_{i_1}\bar{j}_1x_{i_2}\bar{j}_2x_{i_3}\ldots\bar{j}_rx_{i_{r+1}}$, $1 \leq i_{\ell} \leq t_{(i_{\ell})}$ and $1 \leq j_{\ell} \leq n(i_{\ell}) \leq 2$ for $1 \leq \ell \leq r$, $x_{i_1} = s$ and $x_{i_{\ell+1}} = y_{j_{\ell}}$ $1 \leq \ell \leq r$.†

†t and $n(i_{\ell})$ are those of Definition 3.1 and Definition 3.2, $\bar{j}$ $1 \leq j \leq 3$ are three symbols not in Σ (j=3 is a bottom of stack marker).

1) Initialization

Check if the input is of the right format, i.e. if it is a binary monotone maze.†

a If it is not - reject the input.

b Otherwise, push $\bar{3}$s on the stack ($\bar{3}$ marks the bottom of the stack).

2) Continue the path

Scan the input from left to right and look for x_i ,†† where $x_i \in \{x, x\Delta\}$ and $\bar{j}x$ is on top of the stack.†

a If there is no such x_i then go to 3.

b) If there is such x_i, $x_i = x\Delta$, then accept the input.

Otherwise,

c If there is such x_i and $n(i)=0$ then go to 3 (A dead end was found - try another path).

d If there is such x_i and $n(i) \neq 0$ (The path may be continued) then push $\bar{1}\ y_1^{(i)}$†† onto the stack and go to 2.

---

†Note that s, $\bar{k}x_i$, $y_j^{(i)}$, can be long strings, but a 2dpda can match strings and can check monotonicity.

††x_i is identified by the endmarkers [*, $y_1^{(i)}$ by * * and $y_2^{(i)}$ by *].

3) Change the Path

Let $\bar{j}x$ be on top of the stack, $1 \leq j \leq 3$.

a If $j = 3$ - reject the input ($x = s$ and all possibilities have been tried).

Otherwise scan the input, moving the head to the right, until you find $x = y_j^{(i)}$.

b Record j in the finite control and erase $\bar{j}x$. Now $\bar{k}y$ is on the top of the stack.

(i) If $y \neq x_i$, then restore the stack, returning $\bar{k}y$ and $\bar{j}y_j^{(i)}$ to it (j is in the finite control) and go to 3 (continue the search for x_i; the search must eventually succeed since x_i was put on the stack).

(ii) If $y = x_i$, then return only $\bar{k}y$ to the stack. Now, if $j=1$ and $n(i)=2$ then push $\bar{2}y_2^{(i)}$ onto the stack and goto 2. Otherwise go to 3.

Since the maze is monotone the 2dpda, A, can never loop and can check, if necessary, all paths. It essentially uses a depth-first search (c.f. [22]).

Corollary 3.9: 2DPDA $\subseteq$ DLOG => DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \ell ogn$.

In Section 4 we will prove that we get a much stronger result if we assume that 2DPDA $\subseteq$ DLOG, namely that P-TIME = DLOG.

Definition 3.10: A maze $\mathcal{M}$ is one-way threadable if there are $1 \leq i_1 < i_2 \ldots i_k \leq t$ such that $s = x_{i_1}, x_{i_k} \in \Sigma^+\Delta$ and for each ℓ, $1 \leq \ell \leq k-1$, there is a j_ℓ, $1 \leq j_\ell \leq n(i_\ell)$,† with $x_{i_{\ell+1}} = y^{(i_\ell)}_{j_\ell}$.†

Notation: $M^{"}_a$ denotes the set of all one way threadable mazes over $\Sigma = \{a\}$. (i.e. $s, x_i, y^{(i)}_j$ are strings over one letter alphabet).

Theorem 3.11:†† There exists a language, L, $L \in 1NC$, such that $L \in DLOG$ iff $DSPACE(S(n)) = NSPACE(S(n))$ for all $S(n) \geq \ell ogn$.

Proof: Theorem 3.3 and hence Theorem 3.7 are still true if we assume that $\Sigma = \{a\}$. We construct mazes, $\{\mathcal{M}^{"k}_{Z,x} | k = 1,2\ldots\}$ which are the same as those in the proof of Theorem 3.7, except that now they are represented by strings of a's. If r symbols are used to describe the i.d.'s then an i.d. x is represented by $a^{P(x)}$ where P(x) is the value of x interpreted as an r-ary numeral. We still use [,*,],Δ as extra symbols. For each k $\mathcal{M}^{"k}_{Z,x}$ satisfies the following properties:

1) Since it is monotone, $\mathcal{M}^{"k}_{Z,x}$ is one way threadable iff it is threadable iff Z accepts x within space k.

2) Given x as an input, a DTM can generate the i-th symbol of $\mathcal{M}^{"k}_{Z,x}$ ($1 \leq i \leq |\mathcal{M}^{"k}_{Z,x}|$) by using no more than max(ℓogn,k) tape cells. This is done in a manner

---

†t and $n(i_\ell)$ are as in Definition 3.1 and Definition 3.2.

††It has been brought to our attention that Theorem 3.11 was proved independently by Sudborough in a recent paper (which is an extended version of [21] and will appear in JCSS). In that paper he uses more complicated constructions (reduction to k head one way nfa) to obtain what we call here monotonicity.

2) continued.

similar to that in [18] using several r-ary counters. The only difference is that any time an i.d. is generated in one of the counters, the corresponding change of the counter which contains the input head position is not the length of the i.d. but the r-ary number it represents. Note that $|\mathcal{M}'^{k}_{Z,x}| = O(nr^k \log(nr^k))$[†] and $|\mathcal{M}''^{k}_{Z,x}| = O((nr^k)^2)$ and thus $\log|\mathcal{M}'^{k}_{Z,x}| = O(k + \log n) = \log|\mathcal{M}''^{k}_{Z,x}|$. Thus $\max(\log n, k)$ of tape cells will suffice for the counters. Thus Theorem 3.3 (Theorem 3.7) holds with M''_a replacing M_Σ (M'_Σ). To complete the proof we show that $L = M''_a \in$ 1NC. A 1-nc, A, that accepts L operates as follows: First, A pushes s onto its counter. Let $s = y_1^{(i_0)}$. When some $y_j^{(i_\ell)}$ is in the counter, A guesses at some point, while moving its head to the right, that $x_{i_{\ell+1}} = y_j^{(i_\ell)}$ (or $x_{i_{\ell+1}} = y_j^{(i_\ell)}\Delta$). It checks its guess (since it has just started scanning $x_{i_{\ell+1}}$) by popping out the counter. If its guess was wrong, then A rejects. Otherwise, if a Δ follows then A accepts; if not, then if $n(i_{\ell+1}) = 0$ A rejects and if $n(i_{\ell+1}) > 0$ then it nondeterministically chooses $y_{j_\ell}^{(i_\ell)}$ and pushes it onto its stack. It repeats the process described above and rejects if it hits the right endmarker (\$).

[†] $f(n) = O(g(n))$ if there are $c_2 \geq c_1 \geq 0$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n.

Corollary 3.12: 1NC $\subseteq$ DLOG iff DSPACE(S(n)) = NSPACE(S(n)) for all S(n) $\geq$ &ogn.

One way to interpret the results of this section is the following: If we could determine whether a given 2dpda-language (lnc-language), belongs to DLOG, then we could find out whether nondeterminism buys us more computational power in space bounded computation.

4. 2DPDA and Some Open Problems involving P-TIME

In this section we relate questions about 2DPDA to some open problems involving P-TIME. Our strongest result is the construction of a 2dpda-language, L, such that for every i, $L \in DSPACE((\ell ogn)^i)$ iff P-TIME $\subseteq DSPACE((\ell ogn)^i)$. We will make use of the following result, by Cook, which relates P-TIME and the languages accepted by multi-head dpda's.

Lemma 4.1: $\text{P-TIME} = \bigcup_{k=1}^{\infty} DPDA(k) = \bigcup_{k=1}^{\infty} NPDA(k)$.

Proof: See [4].

We now define an operator ϕ, with domain and range 2^{Σ^*}. This operator will have the following property: If $L \in$ P-TIME, then $\phi^i(L) \in$ 2DPDA for some i.†

Definition 4.2: Let a and b be two symbols not in Σ. Then $\phi(L) = \{(axb)^{|axb|} \mid x \in L\}$. We will refer to each copy of axb as a block of $(axb)^{|axb|}$.

Lemma 4.3: $L \in DPDA(2k)$ iff $\phi(L) \in DPDA(k)$ for $k \geq 1$.††

Proof: First we assume that $L \in DPDA(2k)$ i.e. there is a 2k-dpda, A, which accepts L. We construct a k-dpda, B, that accepts $\phi(L)$ and thus prove that $\phi(L) \in DPDA(k)$. B operates as follows: it first checks if the input is of the form $(axb)^n$, $n = |axb|$ and

---

† ϕ^i means applying ϕ i times, i.e. $\phi^0(L) = L$ and $\phi^{i+1}(L) = \phi(\phi^i(L))$.

††Our techniques were introduced by Ibarra in [14]. For $k \geq 2$, Lemma 4.3 follows from a similar lemma in [14].

$x \in \Sigma^+$. Obviously, even a (one-head) 2dpda can do this. If the input is not of this form, B, rejects it. Otherwise, on input $(axb)^n$,B, simulates the action of A on x: The state of A is stored in B's finite control. Each head of B represents two heads of A as follows: The head of B scans the i-th symbol of the j-th block of $(axb)^n$ ($1 \leq i, j \leq n$) iff the two heads it represents scan the i-th and the j-th symbols of ¢x$.

In order to look at the j-th symbol of axb (This is needed for simulating A),† B does the following actions: 1) It stacks P^i †† by moving its head to the beginning of the block (one P per symbol). 2) It stacks Q^j †† by moving its head to the left end-marker, ¢, (one Q per block). 3) It moves its head to the j-th symbol of the first block (while popping off the Q's) and stores that symbol in its finite control. 4) Finally, B reverses this process and returns to the configuration it was in before action number 1. In a similar way, B can change its head position to reflect a corresponding change of the two heads it represents. Thus, in order to simulate one move of A, B acts as follows: 1) It first records in its finite control the symbols which are scanned by the 2k heads of A (by repeating k-times the process we described above). 2) If A accepts or rejects so does B. 3) Otherwise, B records the change of state and changes the positions of its k heads to reflect the change of the 2k heads of A. Hence B accepts ¢(L).

---

†Note that the i-th symbol is scanned by the head of B.

††P and Q are new stack symbols.

To prove the converse, just note that if a k-dpda accepts $\phi(L)$, then a 2k-dpda accepts L. The latter uses two-heads for each head of the former. The first head represents the number of the block and the second represents the position in the block. We omit the obvious details of the simulation.

Lemma 4.4: If $L \subseteq \Sigma^*$, then $L \in$ P-TIME iff $L_k \equiv \phi^k(L) \in$ 2DPDA for some k.

Proof: By Lemma 4.3 $L_k \in$ 2DPDA for some k iff $L \in$ 2DPDA(2^k) for some k. By Lemma 4.1 the latter occurs iff $L \in$ P-TIME.

Theorem 4.5: There is a family of languages, $\{L_k\}_{k=0}^{\infty}$ ($L_k \in$ NSPACE$(n^{1/2^k})$), such that P-TIME = P-SPACE iff $L_k \in$ 2DPDA for some k.

Proof: Let $L_0 = \{R | R$ is a regular expression over Σ and $L(R) = \Sigma^*\}$.† And let $L_k \equiv \phi^k(L_0)$ for $k \geq 0$. In [10], Hartmanis and Hunt proved that given a TM,M, which operates in polynomial space and $x \in \Sigma^+$ we can construct, in polynomial time, a regular expression $R = R_{M,x}$ such that 1) The length of R is bounded by a polynomial in $|x|$, and 2) $L(R) \neq \Sigma^*$ ($R \in L_0$) iff $x \in T(M)$.†† Thus P-TIME = P-SPACE iff $L_0 \in$ P-TIME. But by Lemma 4.4 $L_0 \in$ P-TIME iff $L_k \in$ 2DPDA for some k. $L_k \in$ NSPACE$(n^{1/2^k})$. This is because 1) To check the format we need O(log n) space (we need several counters) and 2) To check that $x \in L$ we need

†L(R) is the language that R describes.

††L_0 is $<_{\text{P-TIME}}$ complete in P-SPACE.

$O(n^{1/2^k})$ space, since $L_0 \in$ NLBA and x appears n^{2^k} times in a word in L_k which corresponds to it.

Theorem 4.6: There is a family of languages, $\{L_k\}_{k=0}^{\infty}$, such that P-TIME = NP-TIME iff $L_k \in$ 2DPDA for some k.

Proof: Let L_0 be the set of satisfiable formulas of the propositional calculus and let $L_k = \phi^k(L_0)$ for $k \geq 0$. By [3] P-TIME = NP-TIME iff $L_0 \in$ P-TIME. But by Lemma 4.4 $L_0 \in$ P-TIME iff $L_k \in$ 2DPDA for some k.

Note that to recognize L_k a TM needs linear deterministic time (to check the format) and then $O(n^{1/2^k})$ time to check nondeterministically whether $x \in L_0$. ($L_0 \in$ NTIME(n)).

By Theorem 4.6 DLBA = 2DPDA => P-SPACE = P-TIME. We don't have the converse, since a much weaker condition implies the same, namely for every $\varepsilon > 0$ there is $L \in$ DSPACE(n^{ε}) such that $L \in$ 2DPDA => P-SPACE = P-TIME.

One way to interpret the results of Theorem 4.6 (Theorem 4.5) is the following: If we could determine when at least one of a given family of languages could be accepted by a 2dpda, we could find out if P-TIME = P-SPACE (P-TIME = NP-TIME).

In order to prove some stronger results we first prove some auxiliary lemmas.

Lemma 4.7: $L \in$ DSPACE($(\ell ogn)^i$) iff $\phi(L) \in$ DSPACE($(\ell ogn)^i$).

Proof: Assume $L \in$ DSPACE($(\ell ogn)^i$) and let A be a DTM which accepts L and operates within space $(\ell ogn)^i$. Let B be a DTM that first checks whether its input is $(axb)^k$ and $k = |axb|$ for $x \in \Sigma^*$. If it is not of this form, B rejects it; otherwise

it simulates A on x. B accepts $\phi(L)$ within space $(\ell ogn)^i$ because the check requires $\ell ogk < \ell ogn$ space (using a binary counter) and the simulation requires $(\ell ogk)^i < (\ell ogn)^i$ space.

Now assume $\phi(L) \in DSPACE((\ell ogn)^i)$ and let C be a DTM which accepts it and operates within space $(\ell ogn)^i$.

Let D be a DTM which, on input x, simulates C on $(axb)^k$ $(k = |axb|)$, by using a binary counter to keep track of the number of the block which is currently scanned by C. D accepts L within $(\ell ogn)^i$ space. This is because the counter needs ℓogn space and the simulation needs $(\ell ogn^2)^i = O((\ell ogn)^i)$ space.

Lemma 4.8: $P\text{-TIME} \subseteq DSPACE((\ell ogn)^i)$ iff $DTIME(n) \subseteq DSPACE((\ell ogn)^i)$.

Proof: The proof uses the padding argument which was used in [2]. The 'only if' part is trivially true. For the 'if' part, assume $DTIME(n) \subseteq DSPACE((\ell ogn)^i)$. Let $L \in P\text{-TIME}$. $L \in DTIME(n^k)$ for some $k \Rightarrow L' \equiv \{xc^{n^k} | x \in L\} \in DTIME(n) \Rightarrow L' \in DSPACE((\ell ogn)^i) \Rightarrow L \in DSPACE((\ell ogn)^i)$. The last implication follows from the fact that a DTM on input x can simulate the DTM which accepts L' within $O((\ell ogn)^i)$ space: It has a binary counter which counts up to n^k in order to keep track of the simulation in the c's part (the counter needs $O(\ell ogn)$ space). It uses $O((\ell ogn^k)^i) = O((\ell ogn)^i)$ space.

Theorem 4.9: There exists a 2dpda-language, L, such that $L \in DSPACE((\ell ogn)^i)$ iff $P\text{-TIME} \subseteq DSPACE((\ell ogn)^i)$.

Proof: Consider $L' = \{code(M_i)\#x \mid M_i$ is a DTM and it accepts x within $|x|^2$ steps$\}$.†

†Code(M_i) is any standard binary encoding of TM's.

Fact 1: $L' \in$ P-TIME. (Obvious)

Fact 2: $\exists k_0$ such that $L' \in DPDA(2^{k_0})$ (by Lemma 4.1)

Fact 3: $L' \in DSPACE((\ell ogn)^i) \Rightarrow DTIME(n) \subseteq DSPACE((\ell ogn)^i)$.

Proof of Fact 3: If M_i is a DTM which operates in linear time, then $code(M_i)\#x \in L'$ for all $x \in L(M_i)$ sufficiently long. Thus a DTM, A_i, can accept $L(M_i)$ by accepting the short strings by a table look-up and the long strings by simulating the DTM which accepts L' within space $(\ell ogn)^i$. A_i uses its finite control for the simulation on the $code(M_i)\#$ part (since its input is x). If $x \in L(M_i)$, then A_i accepts x within $(\ell og\ |cod(M_i)\#x|)^i = O(\ell og|x|)^i$ space. Hence $L(M_i) \in$ DSPACE((ℓogn)

Fact 4: $L \equiv \phi^{k_0}(L')$ satisfies the requirements of the theorem.

Proof of Fact 4: $L \in$ 2DPDA by Fact 2 and Lemma 4.3. Since $L \in$ P-TIME the 'only if' part is trivially true. Assume $L \in DSPACE((\ell ogn)^i)$. By Lemma 4.7 $L' \in DSPACE((\ell ogn)^i)$, by Fact 3 $DTIME(n) \subseteq DSPACE((\ell ogn)^i)$ and by Lemma 4.8 $P\text{-TIME} \subseteq DSPACE((\ell ogn)^i)$.

Corollary 4.10: $2DPDA \subseteq DSPACE((\ell ogn)^i)$ iff P-TIME $\subseteq$ DSPACE((ℓogn)

Corollary 4.11: There is a 2dpda-language, L, such that the following three statements are equivalent:

a) $L \in$ DLOG

b) P-TIME = DLOG

c) $2DPDA \subsetneq$ DLOG.

Proof: a => b by Theorem 4.9 (i=1) and the fact that DLOG $\subseteq$ P-TIME.

b => c since by [1] 2DPDA $\subseteq$ DTIME($n^2 \ell ogn$) and by [11] DTIME(n^k) $\subsetneq$ P-TIME for all k.

c => a is trivial.

Corollary 4.12: DLOG $\neq$ 2DPDA

In order to prove the last result in this section, concerning the question 'does the auxiliary push down stack add to the computational power of a TM?', we need one more lemma.

Lemma 4.13: PDM(S(n)) = DSPACE(S(n)) for all S(n) $\geq$ ℓogn iff P-TIME = DLOG.

Proof: By [4] PDM(ℓogn) = P-TIME. Hence we must show that PDM(S(n)) = DSPACE(S(n)) for all S(n) $\geq$ ℓogn iff it is true for S(n) = ℓogn. The 'only if' part is trivial and a simple padding argument (like in Lemma 4.8) proves the 'if part': Assume PDM(ℓogn) = DLOG. Let L $\in$ PDM(S(n)), S(n) $\geq$ ℓogn, and let M be a PDM which operates within space S(n) and accepts L. Define $L' = \{xc^k | M \text{ accepts } x \text{ within space } \ell og(n+k)\}$. Let M' be a PDM which, on input xc^k, first lays off $\ell og(n+k)$ tape cells and then simulates M on x and accepts x iff M accepts x within that amount of space. Obviously M' accepts L' and operates within ℓogn space. (By the definition of L', $xc^k \in L'$ => $\ell og(|xc^k|) = \ell og(n+k)$ will suffice for the simulation). Thus, L' $\in$ PDM(ℓogn) and by our assumption L' $\in$ DLOG. Hence there is a TM, $\bar{M}'$, which accepts L' within ℓogn space and halts for all inputs. (A TM which operates in ℓogn space can be made to halt for all inputs). Let $\bar{M}$ be a TM which on input x simulates $\bar{M}'$ on $xc, xc^2, \ldots$ and accepts x iff $\bar{M}'$ accepts

xc^k for some k. Now, $x \in L$ iff $xc^{k_0} \in L'$ for some k_0 with $\ell og(n+k_0) \leq 2S(n)$. Thus $\bar{M}$ accepts L and uses $\ell og(n+k_0)$ space which $\bar{M}'$ uses plus ℓogk_0 for counting the c's. Altogether $\leq O(S(n))$ space is used. Thus $PDM(S(n)) \subseteq DSPACE(S(n))$ and hence $PDM(S(n)) = DSPACE(S(n))$ for all $S(n) \geq \ell ogn$.

Theorem 4.9 (i=1) and Lemma 4.13 imply

Theorem 4.14: There exists a 2dpda-language, L, such that $L \in DLOG$ iff $PDM(S(n)) = DSPACE(S(n))$ for all $S(n) \geq \ell ogn$.

Note that all the theorems in this section are true if we replace 2DPDA by 2NPDA or if we replace DLOG by NLOG (and DSPACE(S(n)) by NSPACE(S(n)) in Theorem 4.14). They are stated and proved in the strongest form.

One way to interpret the results of Theorem 4.9 and Theorem 4.14 is the following: If we could determine whether a given 2dpda-language, L, is in DLOG ($DSPACE((\ell ogn)^i)$), then we could find out whether P-TIME = DLOG and whether the auxiliary pushdown stack does add more computational power (whether P-TIME $\subseteq DSPACE((\ell ogn)^i)$).

5. The Relationship Between Some Open Problems and Questions About 2DC

In this section we relate some open problems of the theory of computation to the questions a) Is 2DC = 2NC? and b) Is 2DC = 2DPDA? First we prove in Lemma 5.1 that 2DC $\subseteq$ DLOG. Thus, by corollary 3.12 2DC = 2NC => DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \log n$ and by Corollary 4.11 2DC = 2DPDA => P-TIME = DLOG. We cannot prove the converse, and we cannot construct a 2nc-language (2dpda-language), L, such that $L \in$ 2DC iff 2DC = 2NC (2DC = 2DPDA). However, we can prove several weaker results like those in Theorem 4.5 and Theorem 4.6.

Note that a 2dc is one of the simplest forms of automaton. It is even simpler than a 2dfa, since a 2dfa can simulate a given 2dc using its second head for counting.

Lemma 5.1: 2DC $\subseteq$ DLOG.

Proof: In Lemma 1.1 we saw that given a 2dpda, A, there is a constant, c, such that if A accepts x, its stack never has more than $c|x|$ symbols in it. Thus if A is a 2dc we can construct a TM,B, that simulates it using a binary counter, whose maximum value is $c|x|$, to simulate A's counter.† Obviously, B operates within $\log n$ space and the proof is completed.

Remark: In fact, 2DC $\subsetneq$ DLOG, but we don't need that fact here.

Given a language, L, we now show how to construct a family of languages, $\{L_k\}_{k=0}^{\infty}$, such that $L \in$ DLOG iff $L_k \in$ 2DC for some k. This will enable us to translate some theorems in Section 3 and Section 4, dealing with DLOG, into some weaker theorems about 2DC.

---

†The stack of a counter machine is called a counter.

Definition 5.2: Let $\Sigma' = \{a,b,(,),[,],<,>\}$ be a set of eight symbols not in Σ. Given x, $|x| = n-2$, and k let $x_0 = axb$,

$$x_{i+1} = \underbrace{((x_i)(x_i)\ldots(x_i))}_{n \text{ times}} \text{ for } i \geq 0,$$

$$y_0 = x_{k-1}, \quad y_{i+1} = \underbrace{[[y_i][y_i]\ldots[y_i]]}_{n \text{ times}} \text{ for } i \geq 0 \text{ and}$$

$$z(x,k) = \underbrace{y_k \; y_k \cdots y_k}_{k \text{ times}}.^{\dagger}$$

Note z(x,k) consists of kn^k "big blocks" (the y_0's), and each big block consists of n^{k-1} "small blocks"(the axb's). Thus, $|z(x,k)| = 0(n^{2k})$.

Lemma 5.3: Given w and k a 2dc can check if $w = z(x,k)$ for some $x \in \Sigma^+$.

Proof: (sketch) The proof follows from three observations:

a) $L = \{z\#z \mid z \in \Sigma^+\} \in$ 2DC (left to the reader). Hence, a 2dc can check if w is of the form $(<(\Sigma'^{+}x\Sigma'^{*})^{+}>)^{k}$†† for some $x \in \Sigma^*$.

b) Balanced parentheses (of all kinds) can be checked by the finite control, since they are nested only up to depth k.

c) For each i, $1 \leq i \leq k$, a 2dc can locate all matching parentheses (of all kinds) of depth i (by its finite control) and check whether there are exactly n(=|axb|)

---

†Actually the x_i's depend on x and the y_i's depend on x and k, but we omit them for clarity.

††The '(' and ')' belong to the regular expression and are not the symbols of Σ'.

occurrences of matching parentheses of depth i+1 in it. The counting up to n is done by using the counter and one of the small blocks, and the matching of parentheses is done by the finite control.

Definition 5.4: Given L and k , let

$$L_k \equiv \{z(x,k) \mid x \in L\}.$$

Lemma 5.5: $L \in$ DLOG iff $L_k \in$ 2DC for some $k \geq 0$.

Proof: Assume $L_k \in$ 2DC for some k. Then by Lemma 5.1 $L_k \in$ DLOG. Let A be a TM which accepts L_k and operates within space ℓogn. We construct a TM, B, which operates in ℓogn space and accepts L and thus we show that $L \in$ DLOG. B on input x simulates A on z(x,k). B needs several binary counters to keep track which big block and which small block A is currently scanning. B will know that A is scanning some symbol in Σ' (one of the parentheses), by the fact that one of the counters has reached n. Thus, $L \in$ DLOG.

Now, assume $L \in$ DLOG. Then L is accepted by a k-dfa, C. (This is a well-known fact, see, for example, [9]). We now construct a 2dc, D, which accepts L_k. On input w, D first checks if $w = z(x,k)$ for some $x \in \Sigma^+$. It can do so by Lemma 5.3. If the check fails, then D rejects the input. Otherwise, D proceeds to simulate C on input x. We show below how D can simulate one step of C. At the beginning of the simulation, the head of D scans the leftmost symbol of w, the counter contains P iff $P = \sum_{i=0}^{k-1} a_i n^i$, and the (i+1)th head of A scans the a_i-th symbol

of ¢x$, $0 \leq i \leq k-1$. We need only to show how D can scan the a_0-th symbol of ¢x$ and change the contents of the counter to $P' = a_0 n^{k-1} + \sum_{i=0}^{k-2} a_{i+1} n^i = a_0 n^{k-1} + [\frac{P}{n}]^{\dagger}$. This corresponds to a cyclic right shift of the a_i's.[††] By repeating this process, D can record all symbols which are scanned by the heads of C. If C accepts or rejects so does D. Otherwise D records the change of state and the directions of movement of each of the heads of C. By repeating the process of visiting the a_i-th symbol of ¢x$, $0 \leq i \leq k-1$ (with minor changes), D updates P to reflect the new values of the a_i's.

D accomplishes the task of looking at the a_0-th symbol and rotating the coefficients as follows: D moves its head from left to right popping off n symbols for each big block it passes. It identifies each big block by '[(a' and uses the first small block in a big block to count up to n. When the counter is empty its head scans the a_0-th symbol of the first small block in the $([\frac{P}{n}]+1)$-th big block.[†††] D records this symbol, moves its head left to the '(' before the axb, pushing a_0 onto its counter. Now, D can identify the beginning and end of each x_{k-2} by counting depth of parentheses with its finite control. D moves its head to the right, popping one symbol off its counter for each x_{k-2} it traverses. When the counter is empty, D moves its head left to the beginning of the big block, counting the a's, the b's and the symbols of Σ on its way. Note that

---

[†] $[\frac{a}{b}]$ is the largest integer c such that $cb \leq a$.

[††] They are shifted from $(a_{k-1} \cdots a_0)$ to $(a_0 a_{k-1} \cdots a_1)$

[†††] Note that $w = z(x,k)$ has enough big blocks.

(a) When it reaches the beginning of the block the counter contains $a_0 n^{k-1}$, and

(b) during this last stage, D has been scanning the same big block.

Now D moves its head left counting the big blocks. It is not hard to see that when it reaches the leftmost symbol of w the value of the counter is $P' = a_0 n^{k-1} + [\frac{P}{n}]$, as was claimed above. This completes the proof of the lemma.

Lemma 5.6: Let $L \subseteq \Sigma^*$ and $\{L_k\}_{k=0}^{\infty}$ be those of Definition 5.4, then

(1) If $L \in$ 2DPDA, then $L_k \in$ 2DPDA for all k.

(2) If $L \in$ 2NC, then $L_k \in$ 2NC for all k.

Proof of (1): Assume $L \in$ 2DPDA. Since 2DC $\subseteq$ 2DPDA a 2dpda can check if the format is right, by Lemma 5.3. Thus a 2dpda which accepts L_k first checks the format, and if it is right it simulates the 2dpda which accepts L on one of the small blocks.

The proof of (2) is identical.

Lemma 5.5 and Lemma 5.6 enable us to obtain another version of Theorem 3.8, Theorem 3.12, corollary 4.11 and Theorem 4.14. Namely

Theorem 5.7: There exists a family of 2dpda-languages, $\{L_k\}_{k=0}^{\infty}$, such that $L_k \in$ 2DC for some k iff DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \ell ogn$.

Theorem 5.8: There exists a family of 2nc-languages, $\{L_k\}_{k=0}^{\infty}$, such that $L_k \in$ 2DC for some k iff DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \ell ogn$.

Theorem 5.9: There exists a family of 2dpda-languages, $\{L_k\}_{k=0}^{\infty}$, such that $L_k \in$ 2DC for some k iff P-TIME = DLOG.

Theorem 5.10: There exists a family of 2dpda-languages, $\{L_k\}_{k=0}^{\infty}$, such that $L_k \in$ 2DC for some k iff PDM(S(n)) = DSPACE(S(n)) for all $S(n) \geq \ell ogn$.

Note that Theorem 3.10 which deals with 1NC is translated into Theorem 5.8 which deals with 2NC. This is because Lemma 5.6 does not hold for 1NC (The L_k's are not context-free).

One way to interpret these theorems is the following: If we could determine when at least one of a given family of 2dpda-languages (2nc-languages) can be accepted by a 2dc, then we could find out if DSPACE(S(n)) = NSPACE(S(n)), P-TIME = DLOG and PDM(S(n)) = DSPACE(S(n) for all $S(n) \geq \ell ogn$ (whether DSPACE(S(n)) = NSPACE(S(n)) for all $S(n) \geq \ell ogn$).

6. Conclusion

In this paper we relate some of the most important open problems of the theory of computation to questions about 2dpda-languages. There can be two interpretations to our results. The first, the optimistic one, was expressed throughout the paper: We hope that we will be able to solve questions such as 1) Does a given 2dpda-language (1nc-language), L, belong to DLOG? 2) Does one of a given family of languages belong to 2DPDA? 3) Does one of a family of 2dpda-languages (2nc-languages) belong to 2DC? Being able to solve them we will be able to solve the six open problems listed in Section 1.

The second interpretation, perhaps a more realistic one, is the following: 2DPDA is a class of languages whose membership problem can be solved in linear time (by a RAM), and thus is perhaps the only natural model for the concept of "simple computations". Nevertheless, 2dpda-languages are not so simple. Even though 2dpda's can only perform simple computations, open problems concerning space versus time and determinism versus nondeterminism can be found (in disguised form) among 2dpda-languages.

No matter which interpretation is correct, we hope that the results of this paper will add to our insight into the relationship between determinism and nondeterminism, and between space and time.

Acknowledgements:

I wish to thank my advisor Professor J. Hopcroft and Professor J. Hartmanis for their encouragement, advice and suggestions. I also would like to thank K. Mehlhorn, J. Simon and M. Solomon for useful discussions and suggestions.

References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, "Time and Tape Complexity of Pushdown Automaton Languages", Information and Control 13, 3 (1968), 186-206.

[2] Book, R.V., "On Languages Accepted in Polynomial Time", SIAM J. Computing 1 (1972), 281-287.

[3] Cook, S.A., "The complexity of Theorem Proving Procedures", Proceedings of 3rd Annual ACM Symposium of Theory of Computing, 1971, 151-158.

[4] Cook, S.A., "Characterization of Pushdown Machines in Terms of Time-Bounded Computers", JACM 18 (1971), 4-18.

[5] Cook, S.A., "Linear Time Simulation of Deterministic Two-Way Pushdown Automata", Information Processing 71, North Holland Publishing Co., (1972), 75-80.

[6] Cook, S.A., "An Observation of Time-Storage Trade Off", Proceeding of 5th Annual ACM Symposium on Theory of Computing, (1973), 29-33.

[7] Cook, S.A. and R. Sethi, "Storage Requirements for Deterministic Polynomial Time Recognizable Languages", Proceedings of 6th Annual ACM Symposium on Theory of Computing (1974).

[8] Gray, J.W., M.A. Harrison, and O.H. Ibarra, "Two-Way Pushdown Automata", Information and Control 13,5 (1967), 433-470.

[9] Hartmanis, J., "On Non-Determinancy in Simple Computing Devices", Acta Informatica 1 (1972), 336-344.

[10] Hartmanis, J. and H.B. Hunt, III., "The LBA Problem and Its Importance in the Theory of Computing", Cornell University Technical Report 73-171.

[11] Hartmanis, J. and R.E. Stearns, "On the Computational Complexity of Algorithms", Trans. of AMS, May 1965.

[12] Hopcroft, J.E. and J.D. Ullman, "Formal Languages and Their Relation to Automata", Addison-Wesley, 1969.

[13] Hunt, H.B. III., "On Time and Tape Complexity of Languages I", Proceedings of 5th Annual ACM Symposium on the Theory of Computing , (1973), 10-19.

[14] Ibarra, O.H., "On Two-Way Multi-Head Automata", JCSS 7,1 (1973), 28-36.

[15] Jones, N.D. and W.T. Laaser, "Some Simplified Polynomial Complete Problems", Proceedings of 6th Annual ACM Symposium on Theory of Computing", (1974)

[16] Karp, R.M., "Reducibilities Among Combinatorial Problems" in R. Miller and J. Thatcher (eds), Complexity of Computer Computation, Plenum Press, (1972), 85-104.

[17] Meyer, A.R. and L.J. Stockmeyer, "Word Problems Requiring Exponential Time", Proceedings of 5th Annual ACM Symposium on Theory of Computing, (1973), 1-9.

[18] Savitch, W.J., "Relationship Between Nondeterministic and Deterministic Tape Complexities", JCSS 4,2 (1970), 177-192.

[19] Stearns, R.E., J. Hartmanis, and P.M. Lewis II, "Hierarchies of Memory Limited Computations", IEEE Conference Record on Switching Circuit Theory and Logical Design, (1965), 179-190.

[20] Sudborough, I.H., "Bounded Reversal Multi-Head Finite Automata Languages", to appear in Information and Control.

[21] Sudborough, I.H., "On Tape-Bounded Complexity Classes and Multi-Head Finite Automata", IEEE Conference Record of Thirteenth Annual Symposium on Switching and Automata Theory, (1973), 138-144.

[22] Tarjan, R.E., "Depth First Search and Linear Graph Algorithms", SIAM J. Computing 1 (1972), 146-159.