stichting
mathematisch
centrum

AFDELING INFORMATICA IW 72/76 DECEMBER
(DEPARTMENT OF COMPUTER SCIENCE)

P.M.B. VITÁNYI & W.J. SAVITCH

ON INVERSE DETERMINISTIC PUSHDOWN TRANSDUCTIONS

Prepublication

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1970): 68A20 68A25 68A30 94A30

ACM-Computing Reviews-categories: 5.23, 5.25, 5.26, 5.27

On inverse deterministic pushdown transductions *)

by

P.M.B. Vitányi & W.J. Savitch **)

ABSTRACT

Classes of source languages which can be mapped by a deterministic pushdown (DPDA-) transduction into a given object language (while their complement is mapped into the complement of the object language) are studied. Such classes of source languages are inverse DPDA transductions of the given object language. Similarly for classes of object languages.
The inverse DPDA transductions of the Dyck sets are studied in greater detail: they can be recognized by a DLBA operating in time $\mathcal{O}(n^2)$ but do not comprise all context free languages; their emptiness problem is unsolvable and their closure under homomorphism constitutes the r.e. sets. For each object language L we can exhibit a storage hardest language for the class of inverse DPDA transductions of L; similarly for the class of regular and context free object languages. Lastly, we classify the classes of inverse DPDA transductions of the regular, deterministic context free, context free and deterministic context sensitive languages.

KEY WORDS & PHRASES: *formal languages*
deterministic pushdown transductions
time- and storage complexity
hardest languages

---

*) This report will be submitted for publication elsewhere.

**) On leave from Computer Science Division, Dept. APIS, University of California, San Diego. This research was supported, in part, by NSF Grant MCS-74-02338A01.

1. INTRODUCTION

Deterministic pushdown transducers (DPDT's) are deterministic pushdown automata (DPDA's) which have been provided with an output tape. Such a device defines a mapping (DPDA transduction) from a language called the source language into another language called the object language (while the complement of the source language is mapped into the complement of the object language). DPDT's are often used as a formal model for certain important subprocedures used by compilers and even serve as idealized models for certain simple types of compilers. For example, they appear to be a good model for programs that perform syntax directed translations. (For a formal definition of DPDT's and additional discussion on the relevance of DPDT's to parsing and compiling see AHO & ULLMAN [1].) To comply with our claim that DPDT's correspond to syntax directed translations we supply DPDT's with endmarkers. We will be concerned with inverses of DPDT mappings in the following sense. Given an (object) language L we investigate properties of the class $S(L)$ of all languages of the form $T^{-1}(L)$, where T ranges over all DPDT mappings. (Notice, that T may define a partial mapping.) Hence, $S(L)$ is the class of all source languages that can be mapped into the particular object language L by means of some DPDT. If $\mathcal{L}$ is a class of object languages, then $S(\mathcal{L})$ denotes the class of source languages which can be mapped into some language in $\mathcal{L}$ by some DPDT. Since the finite control of the DPDT can be used to perform a deterministic generalized sequential machine mapping of the output we have $S(\hat{\mathcal{L}}) = S(\mathcal{L})$ where $\hat{\mathcal{L}} = \{\hat{L} \mid \hat{L} \in \text{dgsm}^{-1}(\mathcal{L})\}$. I.e. $\hat{L} \in \hat{\mathcal{L}}$ iff there is a dgsm mapping f and a $L \in \mathcal{L}$ such that $f(\hat{L}) \subseteq L$ and $f(\text{complement}(\hat{L})) \subseteq \text{complement}(L)$.

The paper is divided into two major sections. In the first section we study properties of the class $S(\mathcal{D})$ where $\mathcal{D}$ is the class of Dyck languages (i.e., languages consisting of all well formed bracket expressions over a given alphabet of left and right brackets).

Since $S(\mathcal{D}) = S(\text{dgsm}^{-1}(\mathcal{D}))$, $S(\mathcal{D}) = S(\bar{\mathcal{D}})$ where $\bar{\mathcal{D}}$ is the class of Dyck like languages: $L \in \bar{\mathcal{D}}$ if $L = \{u_1v_1u_2v_2 \ldots u_nv_n \mid u_1u_2 \ldots u_n \in D$ and $v_1v_2 \ldots v_n \in \Sigma^*\}$ where D is a Dyck language over an alphabet Δ disjoint

from Σ. Also, $S(\mathcal{D}) = S(\bar{\mathcal{D}} \cap \mathrm{REG})$ where REG denotes the regular sets. Many simple computer languages are of the form $\bar{\mathcal{D}} \cap \mathrm{REG}$; that is, a program is syntactically correct provided it has a well formed block structure and satisfies some additional regular constraints. (Furthermore, KASAI [9] shows that the closure of $\bar{\mathcal{D}} \cap \mathrm{REG}$ under homomorphisms which delete exactly the brackets and leave the remaining symbols unchanged is the class of context free languages). Note also that since $\mathcal{D} \subseteq \mathrm{dgsm}^{-1}(D_2)$, where D_2 is the Dyck set on two generators, $S(\mathcal{D}) = S(D_2)$.
To study $S(\mathcal{D})$ we use a device called a deterministic cancellation pushdown automaton (DCPDA) introduced by SAVITCH [12]. It consists of a DPDT where the output stack is used only to check that the output string is in the desired object language. The languages accepted by DCPDA's are exactly $S(\mathcal{D})$.
In order to make the model more realistic we have changed the formal definition of a DCPDA slightly from the one given in [12]. We show that the DCPDA languages are accepted by DLBA's operating in time $O(n^2)$, include the DPDA languages but are incomparable with the context free languages. Furthermore, we investigate closure properties and recursive unsolvability of various problems for the class of DCPDA languages. As a by-product we obtain some algebraic characterizations of the r.e. sets. In the second major section of this paper we show that, for any object language L, we can exhibit a storage hardest language for the class $S(L)$ of all possible source languages. Similarly, for the classes of regular object languages (REG) and context free object languages (CFL) we exhibit storage hardest languages for the classes $S(\mathrm{REG})$ and $S(\mathrm{CFL})$, respectively. Finally, we classify $S(\mathrm{REG})$, $S(\mathrm{DPDA})$, $S(\mathrm{CFL})$ and $S(\mathrm{DLBA})$. In the Appendix we prove a (anti) "pumping" Lemma for Dyck languages which is also of independent interest.

2. DETERMINISTIC CANCELLATION PUSHDOWN AUTOMATA

Cancellation pushdown automata were introduced in [12] and shown to accept the r.e. sets. These devices consist of a nondeterministic PDA with a second pushdown store, called the auxiliary pushdown store. The machine may write in the auxiliary stack but can not read in it.
The device operates just like a PDA, except that at each step it is allowed to place a symbol on top of the auxiliary pushdown store. Thus, an alternate way of describing it is to say that it consists of a PDA with an

auxiliary write-only output stack. The finite control can neither read nor erase in the auxiliary stack. However, a set of pairs of auxiliary stack symbols are specified as cancelling. Whenever such a pair occurs on the auxiliary stack, the two symbols spontaneously disappear. The device accepts just like an ordinary PDA by empty store; both stores must be empty for acceptance. In [12], the deterministic variety of these machines were shown to accept only recursive languages. In this section we show amongst other things that the deterministic versions of these machines (DCPDA's) accept only deterministic LBA languages, but not all context free languages.

DEFINITION. A deterministic cancellation pushdown automaton (DCPDA) M is specified by the following items: a finite set K of states; two finite sets of symbols, Σ and Γ, called the input and stack alphabet, respectively; a specified start state q_0 in K; a specified start stack symbol Z_0 in Γ; a transition function δ which is a partial function from $K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ into $K \times \Gamma^* \times (\Gamma \cup \{\varepsilon\})$; and a partial function E from $\Gamma \times \Gamma$ into $\{\varepsilon\}$. E is subject to the following restriction. There are disjoint subsets Δ_{left}, Δ_{right} in Γ and a one-one mapping h from Δ_{left} onto Δ_{right} such that for all $A \in \Delta_{left}$, $E(h(A),A) = \varepsilon$, and $E(B,A)$ is undefined if $B \neq h(A)$. E is called the cancellation relation for M. We insist that the transition function δ satisfy the following restriction: for each state q and pushdown symbol X, if $\delta(q,\varepsilon,X)$ is defined, then $\delta(q,a,X)$ is undefined for all input symbols $a \in \Sigma$.

The intuitive meaning of δ is similar to that of a DPDA. If $\delta(q,a,X) = (p,\gamma,Y)$, then whenever M is in state q scanning input a with X on top of the ordinary stack, it will, in one move, replace X by γ, put Y on top of the auxiliary stack, go into state p and finally advance the input head past a. Intuitively $E(B,A) = \varepsilon$ means that any time A and B are adjacent in the auxiliary stack with B on top of A, BA is replaced by ε. In actual computations this replacement always happens on top of the auxiliary stack.

DEFINITION. Let M be a DCPDA and carry over the notation from the previous definition. An instantaneous description (ID) of M is a triple (w_1qw_2,β,α) where q is a state, w_1 and w_2 are strings of input symbols $(K \cap \Sigma = \emptyset)$, and both β and α are strings of stack symbols. The interpretation is that M is in state q with input w_1w_2, that the input head is scanning the first symbol of w_2 and that β and α are the contents of the auxiliary and ordinary stacks, respectively; the leftmost symbols of β and α are considered to be the "top" symbols. The yield relation, $\vdash$, between ID's is defined by:

(i) $(w_1qw_2,BA\beta,\alpha) \vdash (w_1qw_2,\beta,\alpha)$ provided, $E(B,A) = \varepsilon$

(ii) $(w_1qaw_2,\beta,X\alpha) \vdash (w_1apw_2,Y\beta,\gamma\alpha)$ provided $\delta(q,a,X) = (p,\gamma,Y)$ and β is not of the form $BA\beta'$ where $E(B,A)$ is defined.

$\vdash^*$ denotes the reflective, transitive closure of $\vdash$. In order to make the model more realistic we will assume that our DCPDA's have a distinguished endmarker \$. So when talking about an input string w, we assume that \$ does not occur in w and that the input tape actually contains w\$.

DCPDA's accept in essentially the same way that DPDA's do but we add the additional condition that in order for an input w to be accepted, the computation on w must terminate with the auxiliary stack empty. I.e., let M be a DCPDA and retain the notation of the previous definition. M is said to <u>accept</u> the input w by <u>empty store</u> if

$$(q_0w\$,\varepsilon,Z_0) \vdash^* (w\$p,\varepsilon,\varepsilon)$$

for some state p. When talking about acceptance by final state we assume that a set F of final states have been specified; we also assume that all final states are halting states. M is said to accept the input w by final state if $(q_0w\$,\varepsilon,Z_0) \vdash^* (w\$p,\varepsilon,\alpha)$ for some final state p and some string α of stack symbols. Notice that M <u>halts</u> whenever its ordinary stack is empty or when it enters a final state. It is easy to see that acceptance by final state and empty store are equivalent in the sense that given any DCPDA that accepts by one of these conventions, we can find another DCPDA that accepts exactly the same input strings using the other acceptance convention. A language is said to be a DCPDA language if it is the set of all strings accepted by some DCPDA.

Our definition of DCPDA's differs slightly from that in [12] in that

(a) The cancellation relation E defines the Dyck set over Δ_{left}, Δ_{right} where in [12] it could define also lengthpreserving homomorphic images of Dyck sets, and

(b) in [12] the DCPDA's were not provided with an endmarker

The first restriction makes the DCPDA languages equal to $S(\mathcal{D})$ while the added power from (b) makes the DPDA transductions relatively more realistic. From the presented definition it should be clear that the DCPDA languages equal $S(\mathcal{D})$.
Note that by adding an endmaker to the input we got into the difficult, but not unrealistic situation, that our devices are not truly on-line. But neither are they truly off-line since the input is read from left to right and if a part of the stack is accessed, all above it is irretrievably lost. Hence, we cannot use lower bounds for on-line computations such as in GALLAIRE [2], but neither can we use the upper bounds from off-line Turing machines.

In the sequel it will appear that DCPDA's are rather powerfull; in terms of the Chomsky hierarchy the situation is that DPDA $\subset$ DCPDA $\subset$ DLBA but DCPDA and CFL are incomparable. CFL, DPDA, DCPDA and DLBA denote the class of context free languages, deterministic context free languages, DCPDA languages and deterministic context sensitive languages, respectively. In order to get a feel for the power of DCPDA's and to have some examples to use in later theorems, we now give a few examples of DCPDA languages.

<u>EXAMPLE 2.1.</u> $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ is accepted by a DCPDA M as follows. M checks for membership in $a^* b^* c^*$ with its finite control. All a's are pushed in the ordinary stack. When the machine starts reading b's it pushes a b on the auxiliary stack and deletes an a from the ordinary stack for every b read. For every c it pushes a $\bar{b}$ on the auxiliary stack. The cancellation relation is defined by $E(\bar{b},b) = \varepsilon$. With some minor embellishments M accepts L_1 by empty store.

<u>EXAMPLE 2.2.</u> $L_2 = \{w \not c\, w^R \mid w \in \Sigma^*\}$, where $\not c \notin \Sigma$, can be accepted without using the auxiliary stack at all; L_2 is a DPDA language.

<u>EXAMPLE 2.3</u>. $L_3 = \{w ¢ w \mid w \in \Sigma^*$ and $¢ \notin \Sigma\}$. M pushes w on the ordinary stack until it reads the marker ¢. Subsequently, M transfers the contents of the ordinary stack to the auxiliary stack until the ID $(w¢pw\$,w^R,Z_0)$ occurs. Then the remainder of the input is read and for every input symbol a M pushes $\bar{a}$ on the auxiliary stack. Upon reading \$, M enters a final state and halts. With the cancellation relation defined by $E(\bar{a},a) = \varepsilon$ for all $a \in \Sigma$ the machine accepts L_3 by final state.

We leave it to the ingenuity of the reader to ascertain that

<u>EXAMPLE 2.4</u>.

$L_5 = \{(w¢)^n \mid w \in \Sigma^*,\ ¢ \notin \Sigma$ and $n \geq 1\}$,

$L_6 = \{(a^{i-1}b)^i \mid i \geq 1\}$ and

$L_7 = \{(w¢)^n \mid w \in \Sigma^*,\ ¢ \notin \Sigma$ and $n = |w|\}$

are also accepted by DCPDA's.

We now proceed to show that DCPDA's accept in linear time. Call the <u>associated</u> DPDA M_{ass} of a DCPDA M the DPDA obtained from M by deleting the auxiliary stack mechanism. From the definition it is clear that M halts on an input word w iff (and in at most k times as many moves for some constant k) M_{ass} halts on w. Hence all DCPDA's accept in linear time iff all DPDA's accept in linear time.

<u>LEMMA 2.5</u>. DPDA's *accept in linear time. This is true both for acceptance by empty store and final state.*

<u>PROOF</u>. Let M be any DPDA accepting by empty store. Starting from a state p with Z on top of its push down stack of heigth h, M can do one of the following sequences of moves.

(i) M makes a sequence of ε-moves of which the last one makes the heigth of the stack fall for the first time below h.

(ii) M makes a sequence of ε-moves ended by a read move at which time the stack heigth is h + x for some x ≥ 0 and it never has fallen below h in the meantime.

(iii) Like (ii) but the last (read) move makes the stack height $h - 1$ for the first time during the execution of the sequence.

(iv) M enters a loop, i.e., it keeps on making ε-moves forever without the height of the stack ever falling below h.

We associate with M another DPDA M' which simulates the behavior of M such that for each (state, stack symbol) pair of M it simulates the subsequent sequence of moves of type (i)-(iii) in a single move. (state, stack symbol) pairs which start a sequence of type (iv) are left undefined in M' because if they occur M can never accept. Clearly, M' accepts exactly the same input strings that M does. Now M' never increases the height of its stack except possibly when it reads a (non-empty) input symbol. By well-known arguments (see for example HOPCROFT & ULLMAN [8]), for each M there is a constant c such that the length of the move sequences (i)-(iii) is bounded above by c. Therefore, for each M, there is a constant ℓ which bounds x in (ii) from above, and which is the length of the longest string M' can push on its stack in a single move. In reading an input of lenth n (including the end maker if there is one) M' pushes at most $n\ell$ symbols on its stack. Hence M' makes at most n reading and $n\ell + 1$ popping ε-moves during its computation (the extra one is for the original stack symbol) and hence accepts within $n + n\ell + 1$ moves.
Hence M accepts within $c(n+n\ell+1)$ moves, i.e., in linear time.

The argument for the case where M accepts by final state is essentially the same, even if we would not insist that accepting states are halting states. The above proof is more or less implicit in GINSBURG & GREIBACH [3]. □

Hence we have:

THEOREM 2.6. DCPDA'*s always accept in linear time.*

The following corollaries are immediate.

COROLLARY 2.7. DCPDA *languages are accepted in linear time by off-line deterministic Turing machines with two scratch tapes.*

COROLLARY 2.8. *The class of* DCPDA *languages are included in the class of deterministic* LBA *languages.*

COROLLARY 2.9. DCPDA *languages are accepted by one tape two-way deterministic Turing machines within time* $O(n^2)$ *and storage* $O(n)$.

The next Theorem says that the time bound in Corollary 2.9 is the best possible.

THEOREM 2.10. *There are* DCPDA *languages which cannot be accepted by one tape deterministic Turing machines in time* $T(n)$ *if* $\sup T(n)/n^2 = 0$.

PROOF. The language L_2 of Example 2.2 is a DCPDA language but can not be accepted in $T(n)$ which grows slower than $O(n^2)$, HOPCROFT & ULLMAN [8, Theorem 10.7]. □

It seems intuitively clear that languages like $L = \{ww^R | w \in \Sigma^*\}$ where an accepting DCPDA would have to "guess" where the middle of the string is cannot be accepted by a DCPDA. We do not, however, have a proof thereoff and therefore disignated the long and cumbersome proof of the next Theorem to the Appendix.

THEOREM 2.11. *There are context free languages which are not* DCPDA *languages.*

$$L_8 = \{a^i b^j \mid i \le j\} \cup \{a^i b^j c^k \mid i + j = k\} \cup \{a,b,c\}^* ¢$$

is an example.

By definition all deterministic context free languages are DCPDA languages. The inclusion relations between the family of DCPDA languages and the other relevant language families is shown in fig. 1.[1)]

Next we look at some language theoretical properties.

THEOREM 2.12. *The class of* DCPDA *languages is closed under intersection with a regular set, inverse deterministic gsm mappings, marked union, marked concatenation, marked Kleene * and marked deterministic* CFL *substitution. It is not closed under length preserving homomorphisms and union.*

PROOF. The closure results follow by routine techniques and we omit their proofs. It remains to be shown that DCPDA languages are not closed under length preserving homomorphism and union. The constituent elements of the language L_8 are all DPDA languages. Hence L_8', which is like L_8 but with all constituent sublanguages over pairwise disjoint alphabets, is a DPDA language. But according to Theorem 2.11 its length preserving homomorphism L_8 is not a DCPDA language. Since L_8 is a union of DCPDA languages, these languages are not closed under union either. □

We next consider some results which characterize the r.e. sets in terms of DCPDA's.

THEOREM 2.13. *The closure of the class of* DCPDA *languages under homomorphism is the class of* r.e. *sets.*

PROOF. In [12] it was shown that every r.e. set is accepted by some nondeterministic CPDA.[2)] So it will suffice to show that: if L is accepted by some nondeterministic CPDA, then we can find a DCPDA language L_D and a homomorphism h such that $L = h(L_D)$. With this in mind, let M be a nondeterministic CPDA and let L be the language accepted by M. Without loss of generality, we may assume that in any nondeterministic situation M has at most two choices of next moves labelled as the 0 and 1 choice. We also assume that 0 and 1 do not occur in the input alphabet of M. Let L_D be the set of all words of the form $u_0a_1u_1a_2\ldots a_nu_n$ where the a_i are symbols from the input alphabet of M, the u_i are in $\{0,1\}^*$ and the u_i determine a valid accepting computation of M on input $a_1a_2\ldots a_n$ in the following sense: there is an accepting computation of M on input $a_1a_2\ldots a_n$ that makes length (u_0) nondeterministic moves before reading a_1, makes length (u_1) nondeterministic moves from the time it reads a_1 till just before it reads a_2, makes length (u_2) nondeterministic moves from the time it reads a_2 till just before it reads a_3 and so forth; furthermore $u_0u_1\ldots u_n$ is the list of nondeterministic choices (either 0 or 1 choice) made by M in this computation. Clearly M can be modified into a DCPDA to accept L_D; the 0's and 1's determine the choice of moves and so eliminate the nondeterminism.
If we define h by $h(0) = h(1) = \varepsilon$ and $h(a) = a$ for a not equal to 0 or 1 then $L = h(L_D)$ and the proof is completed. □

By combining the techniques used to prove the previous theorem and those used to prove Theorem 5 in SAVITCH [12] [3] we can get the following characterization of the r.e. sets. The proof of Theorem 2.14 is left to the reader; the proof of Theorem 2.15. is limited to a brief sketch.

Let D denote a Dyck language over Δ and let Σ be an alphabet disjoint from Δ. Then the Dyck-like language $\bar{D}$ (with Σ understood) is shuffle (D,Σ^*).

<u>THEOREM 2.14.(i)</u> *Every* r.e. *set over* Σ *is expressible in the form* $h(\bar{D}\cap L)$ *where* L *is a deterministic context free language,* $\bar{D}$ *a Dyck-like language and* h *a homomorphism defined by* $h(a) = \varepsilon$ *for* $a \in \Delta$ *and* $h(a) = a$ *for* $a \in \Sigma$. Δ *is the alphabet for* D.

Since each context free language can be expressed as $h(\bar{D}\cap R)$ for some regular set R and $\bar{D}$ and h as above, KASAI [9], and by furthermore noting that it suffices to consider the Dyck set on two generators D_2 over $\{0,1,\bar{0},\bar{1}\}$ and a homomorphism $h : (\Sigma \cup \{a,b,\bar{a},\bar{b},0,1,\bar{0},\bar{1}\})^* \to (\Sigma \cup \{0,1,\bar{0},\bar{1}\})^*$ defined by $h(c) = c$ for $c \in \Sigma$, $h(c) = \varepsilon$ for $c \in \{0,1,\bar{0},\bar{1}\}$ and $h(a) = 0$, $h(\bar{a}) = \bar{0}$, $h(b) = 1$, $h(\bar{b}) = \bar{1}$ we can state the following

<u>THEOREM 2.14.(ii)</u> *For each* r.e. *set* L *over* Σ *there is a regular set* R *over* $\Sigma \cup \{a,b,\bar{a},\bar{b},0,1,\bar{0},\bar{1}\}$ *such that*

$$L = h(\bar{D}_2\cap h(\bar{D}_2\cap R))$$

where $\bar{D}_2$ *is the Dyck-like set*: shuffle $(D_2,(\Sigma\cup\{a,b,\bar{a},\bar{b}\})^*)$

Hint: $\bar{D}_2 \cap R$ yields the strings in R with a correct bracket structure over $\{0,1,\bar{0},\bar{1}\}$ which brackets are removed by h yielding the desired context free language. h simultaneously changes a's to 0's and b's to 1's and in doing so sets up the structure for again intersecting with $\bar{D}_2$ so that after removal of brackets $\{0,1,\bar{0},\bar{1}\}$ again by h the desired r.e. set L is derived.

<u>THEOREM 2.15.</u> *Every* r.e. *set is expressible in the form* $\tau^{-1}(L)$ *where* L *is a deterministic context free language and* τ *is a marked Dyck set substitution; that is, there is a Dyck set* D *such that* $\tau(a) = aD$ *for all* a *in the domain of* τ, *and the alphabets of* D *and the* r.e. *set are disjoint.*

PROOF. Let A be an r.e. set. In [12] it was shown that A could be generated by a phrase-structure grammar in strong normal form and that a nondeterministic CPDA could be constructed to accept A by "parsing" in this grammar. An analysis of that proof, shows that A can be represented in the form

$$A = \{a_1 a_2 \ldots a_n \mid \exists\, w_1, w_2, \ldots, w_n \in D \text{ such that } a_1 w_1 a_2 w_2 \ldots a_n w_n \in L\} \text{ where}$$

D is a Dyck set and L is the language accepted by a nondeterministic PDA, M. Without loss of generality, we may assume that at each point in a computation M has at most two choices of next moves. If, as in the proof of Theorem 2.13, we code the correct choices by interspersing 0's and 1's into the input strings of L, then L can be made deterministic. Suppose we code these 0's and 1's as two strings $\vec{X}\overleftarrow{X}$ and $\vec{Y}\overleftarrow{Y}$ and we expand the Dyck set D by allowing the two new matching pairs $\vec{X}\overleftarrow{X}$ and $\vec{Y}\overleftarrow{Y}$. Then it is not difficult to see that A can be expressed in the form

$$A = \{a_1 a_2 \ldots a_n \mid \exists\, w_1, w_2, \ldots, w_n \in D' \text{ such that } a_1 w_1 a_2 w_2 \ldots a_n w_n \in L'\} \text{ where}$$

D' is a Dyck set and L' is a deterministic context free language. The Theorem follows immediately from this. □

The last theorem of this section gives a number of undecidability results for DCPDA languages.

THEOREM 2.16. *Let* L_1 *and* L_2 *arbitrary* DCPDA *languages. All of the following questions are recursively unsolvable:*

(1) *Is* L_1 *empty?, finite?, infinite?*

(2) *Is* $L_1 = L_2$*?, Is* $L_1 \subseteq L_2$*?*

(3) *Is* $L_1 \cap L_2$ *empty?, finite?, infinite?*

(4) *Is* $L_1 \cup L_2$ *empty?, finite?, infinite?*

PROOF. (1) First consider the emptiness question. We will show that if the emptiness question for DCPDA's were solvable then the Post correspondence problem would be solvable. So it will follow immediately that the emptiness problem is unsolvable. Let $(u_1,u_2,\ldots,u_n)$ and $(v_1,v_2,\ldots,v_n)$ be an instance of the Post correspondence problem.
We will descripe a DCPDA M such that the set of words accepted by M is nonempty if and only if the Post correspondence problem is solvable for this list; that is, if and only if there are $i_1,i_2,\ldots,i_k$ such that $u_{i_1}u_{i_2}\ldots u_{i_k} = v_{i_1}v_{i_2}\ldots v_{i_k}$. M will accept exactly those words $w¢w^R$ with w in $\{1,2,\ldots n\}^*$ such that $w = i_1i_2\ldots i_k$ and $u_{i_1}u_{i_2}\ldots u_{i_k} = v_{i_1}v_{i_2}\ldots v_{i_k}$.
M works exactly like the DCPDA of Example 2.3 except that instead of pushing a digit i on the auxiliary stack it pushes the corresponding u_i and instead of pushing an "inverse" $\bar{i}$ it pushes $\bar{v}_i^R$. If $v_i = a_1a_2\ldots a_m$, then $\bar{v}_i^R = \bar{a}_m\bar{a}_{m-1}\ldots\bar{a}_1$ where $\bar{a}$ is the "inverse" of a; so $E(\bar{a},a) = \varepsilon$ for each relevant symbol a, where E is the cancellation relation. Let L be the language accepted by M. Clearly L is nonempty if and only if the given instance of the Post correspondence problem has a solution. The unsolvability of the finiteness and infiniteness questions follow easily from the observation that L is nonempty if and only if its infinite. (2) (3) and (4) follow from (1) by routine manipulations. □

We conclude this section by describing the effect on the derived results of some changes in the model.

(i) The end of input not indicated by an endmarker: the DPDT mapping becomes truly on line and acceptance by empty store and (non-halting) final state is not equivalent any more for these DCPDA's.
Theorem 2.11 now becomes easy to prove since GALLAIRE [2] shows that there is a context free language which requires at least $n^2/\log n$ time to be accepted by on-line multi tape deterministic Turing machines. Our DCPDA without endmarkers can be simulated in real time by these devices and hence DCPDA languages without endmarker can be accepted in linear time by such Turing machines. The same language could serve for showing non-closure under length preserving homomorphic mappings. The remainder of the results do not depend on the endmarker and hence go through as well. (But for some of the examples).

(ii) Suppose we keep the endmarker \$ and allow arbitrary partial cancellation relations $E : T \times T \to \{\varepsilon\}$. Under these conventions the DCPDA languages are S (length preserving homomorphic images of Dyck languages). In this case it seems hard to prove that not all context free languages are included. By the fact that there are deterministic LBA languages which can not be recognized by multitape off-line deterministic Turing machines in time $n \log/A^{-1}(n)$ where A^{-1} is the inverse of the Ackermann function, HOPCROFT, PAUL & VALIANT [7] and the fact that Theorem 2.6 still holds it follows that containment in DLBA is strict. Except for the non-closure under union and length preserving homomorphisms all results in the paper go through.

(iii) Suppose we keep the endmarker \$ and allow arbitrary partial cancellation relations $E : T \times T \to T \cup \{\varepsilon\}$. Under these conditions DCPDA languages are $S(X)$ where X is an easily describable subclass of the DPDA languages. All remarks of (ii) hold, but in addition it is now easy to prove that the class $S(X)$ is closed under complement. This is because we can cancel arbitrary long portions (up to specific stack symbols) of the auxiliary stack by long range cancellation symbols. A similar device has been used by GREIBACH [6] in her introduction of "jump" PDA's. Furthermore, it can now be shown that for $L \in S(X)$ the questions is $L = \Sigma^*$?, and $L = R$? for some given regular set R are recursively unsolvable.

Some of the problems remaining are the closure under complement and solvability of $L = \Sigma^*$?, $L = R$? for our original DCPDA languages and (non)-closure under intersection and union for the discussed language families. A more intrinsic and tantalizingly difficult open problem is to prove that not all context free languages are DCPDA languages under conventions (ii) or (iii).

3. HARDEST SOURCE LANGUAGES

The main result of this section shows that, for any object language L, the class of all source languages for L, under DPDT mappings, always contains a storage hardest language. The result is proven using techniques developed by GREIBACH [5] and SUDBOROUGH [13]. The result is in fact a generalization of Sudborough's result which states that there is a storage hardest deterministic context free language.

Recall that throughout this paper the abbreviation DPDT has been used to mean deterministic pushdown transducer with an endmarker to delimit the end of the input string and which satisfies the condition that all accepting states are halting states. Given these conventions it is easy to see that every DPDT computes a single valued partial function from input strings into strings over the output alphabet. It is also easy to see that a partial function T is computed by empty store if and only if it is computed by some DPDT by final state. We now formally introduce some notation and state the main result of this section.

DEFINITION. If L is any language, then $S(L)$ denotes the class of all languages of the form $T^{-1}(L) = \{w \mid T(w)$ exists and $T(w) \in L\}$, where T is the partial function defined by some DPDT. If $\mathcal{L}$ is a class of languages $S(\mathcal{L})$ denotes the class of all languages of the form $T^{-1}(L)$ where L is in $\mathcal{L}$ and T is the partial function defined by some DPDT.

DEFINITION. Let L_1 and L_2 be two languages. We write $L_1 \leq_{\log} L_2$ and say L_1 is $\log n$ reducible to L_2 provided there are alphabets Σ_1 and Σ_2 and a function g from Σ_1^* to Σ_2^* such that:

(i) L_1 is a subset of Σ_1^* and L_2 is a subset of Σ_2^*.

(ii) For every w in Σ_1^*, w is in L_1 if and only if g(w) is in L_2.

(iii) g is computed by some deterministic off-line Turing machine which uses at most $\log n$ storage tapesquares on inputs of length n.

THEOREM 3.1. *For any language* Y, *we can find a language* L_Y *such that:*

(1) L_Y *is in* $S(Y)$ *and*

(2) *For all* L *in* $S(Y)$, $L \leq_{\log} L_Y$.

It is easy to see that, for any non-empty Y, the class $S(Y)$ contains all deterministic context free languages. It is well known that there are deterministic context free languages that require at least $\log n$ storage for acceptance. So the language L_Y is in some sense a storage hardest language for the class $S(Y)$.

Before proving Theorem 3.1, we will derive a few corollaries.

COROLLARY 3.2. *For every language* Y *we can find a language* L_Y *such that*

(1) L_Y *is in* $S(Y)$ *and*

(2) *If* L_Y *is accepted by a deterministic (respectively nondeterministic) Turing machine within storage* $S(n)$*, then for every language* L *in* $S(Y)$*, there is a constant* c *such that* L *is accepted in deterministic (respectively nondeterministic) storage* $S(n^c)$*, provided* $S(n) \geq \log_2 n$ *and* $S(n)$ *is monotone nondecreasing.*

PROOF. Let L be any language in $S(Y)$; let L_Y be as in Theorem 3.1; let g be a function, as in the previous definition, which $\log n$ reduces L to L_Y; let M_Y and M_g be machines that accept L_Y and compute g within storage $S(n)$ and $\log n$ respectively. A machine M_L to accept L within storage $S(n^c)$ can be constructed as follows. Given input w, M_L operates by simulating M_g to compute g(w) and simulating M_Y to check if g(w) is in L_Y. If M_Y is deterministic then M_L will also be deterministic.
Since M_g runs in storage $\log n$, it runs in time n^c, for some c. Let n be the length of w, M_L uses $\log n$ storage to simulate M_g and $S(\text{length}(g(w))) \leq S(n^c)$ storage to simulate M_Y, so, except for the storage needed to hold g(w), M_L operates within storage proportional to $S(n^c)$. The length of g(w) may exceed $S(n^c)$. So M_L cannot store g(w) in the most straight forward way and still keep its storage below $S(n^c)$. However, all M_L needs to do in order to simulate M_Y on g(w) is be able to generate g(w) one symbol at a time from left to right and to keep track of the number of symbols between the end of g(w) and the current symbol generated. This can be done in storage $\log n$ and so M_L can be made to run in total storage $S(n^c)$. The details of such constructions are well known. For more details on this type of construction see, for example, SAVITCH [11]. □

If we take $S(n)$ to be a polynomial in $\log n$ and observe that $\log n^c = O(\log n)$, then Corollary 3.2 specializes to:

COROLLARY 3.3. *For any language* Y *we can find a language* L_Y *such that*

(1) L_Y *is in* $S(Y)$ *and*

(2) *If* L_Y *is accepted by some deterministic (respectively nondeterministic) Turing machine that runs in storage bounded by* $(\log n)^\alpha$, *then every language in* $S(Y)$ *is accepted by some deterministic (respectively nondeterministic) Turing machine that runs in storage bounded by* $(\log n)^\alpha$, *provided* $\alpha \geq 1$.

Theorem 3.1 can be thought of a generalization of Sudborough's result that there is a storage hardest deterministic context free language [13,14] To illustrate this point we derive Sudborough's Theorem as a corollary to Theorem 3.1.

COROLLARY 3.4. *There is a deterministic context free language* L_0 *such that: if* L *is any deterministic context free language then* $L \leq_{\log} L_0$.

PROOF. Let Y be any language with exactly one word in it. (Actually, Y may be taken to be any non-empty regular set). Then $S(Y)$ is the class of all languages, accepted by deterministic PDA's with endmarker. To see that $S(Y)$ is the class of all languages accepted by deterministic PDA's with endmarker, just observe that the finite state control of a DPDT can be modified so that it can tell if its output is in Y. The corollary now follows almost directly from Theorem 3.1. □

We can now proceed with the proof of Theorem 3.1. First we define the languages L_Y that will turn out to have the properties listed in the Theorem.

DEFINITION. Let Y be any language; let Σ be an alphabet such that $Y \subseteq \Sigma^*$; let D_3 denote the Dyck set on three letters and let $\vec{A}$, $\overleftarrow{A}$, $\vec{B}$, $\overleftarrow{B}$, $\vec{C}$ and $\overleftarrow{C}$ be the six symbols used for writing strings in D_3; More precisely, D_3 is the set of all strings which rewrite to the empty string under the rules $\vec{A}\overleftarrow{A}$, $\vec{B}\overleftarrow{B}$ and $\vec{C}\overleftarrow{C}$ each rewrite to the empty string. If $\vec{X}\overleftarrow{X}$ is rewritten as the empty string, then we say the two symbols cancel. We can and will assume that the six symbols alphabet for D_3 and the alphabet Σ are disjoint; the symbol # is yet another new symbol. If $\alpha = X_1 X_2 \dots X_\ell$ is a string such that each X_i is in {A,B,C}, then $\vec{\alpha}$ denotes $\vec{X}_1\vec{X}_2\dots\vec{X}_\ell$; $\overleftarrow{\alpha}$ is defined analogously. The language associated with Y is denoted L_Y and is defined as follows. L_Y consists of all strings of the form

$$\overleftarrow{X}_1 \vec{\alpha}_1 \beta_1 \# \overleftarrow{X}_2 \vec{\alpha}_2 \beta_2 \# \dots \# \overleftarrow{X}_m \vec{\alpha}_m \beta_m$$

where the X_i are in {A,B,C}, the α_i are in $\{A,B,C\}^*$, the β_i are in Σ^* and there are indices $i_1 < i_2 < \dots < i_\ell \leq m$ such that

(1) $\vec{A}\, \overleftarrow{X}_{i_1} \vec{\alpha}_{i_1} \overleftarrow{X}_{i_2} \vec{\alpha}_{i_2} \dots \overleftarrow{X}_{i_\ell} \vec{\alpha}_{i_\ell}$ is in D_3

(2) $\beta_{i_1} \beta_{i_2} \dots \beta_{i_\ell}$ is in Y and

(3) i_1 is the least j such that $\overleftarrow{X}_j$ is $\overleftarrow{A}$; for each $k < \ell$, i_{k+1} is the least j which is greater than i_k and is such that $\overleftarrow{X}_j$ cancels with the right most symbol of $\mathrm{red}(\vec{A}\, \overleftarrow{X}_{i_1} \vec{\alpha}_{i_1} \overleftarrow{X}_{i_2} \vec{\alpha}_{i_2} \dots \overleftarrow{X}_{i_k} \vec{\alpha}_{i_k})$.
For any string α, red(α) denotes the string obtained from α by cancelling as much as possible. That is, red(α) is obtained from α by applying the rewrite rules $\vec{A}\overleftarrow{A} \to \varepsilon$, $\vec{B}\overleftarrow{B} \to \varepsilon$ and $\vec{C}\overleftarrow{C} \to \varepsilon$ as many times as possible.

NOTATION. In order to make our notation more readable when discussing languages such as L_Y, we will, for this section, make the convention that if γ is the string of symbols in a pushdown store, then the right end of γ corresponds to the top of the stack.

LEMMA 3.5. *For any language* Y, L_Y *is in* S(Y).

PROOF . We will describe a DPDT which computes a function T such that $L_Y = T^{-1}(Y)$. In describing the DPDT, we will assume that the input string is of the form

$$\overleftarrow{X}_1 \vec{\alpha}_1 \beta_1 \# \overleftarrow{X}_2 \vec{\alpha}_2 \beta_2 \# \dots \# \overleftarrow{X}_m \vec{\alpha}_m \beta_m$$

where the X_i's are in {A,B,C}, the α_i are in $\{A,B,C\}^*$ and the β_i are in Σ^*; Σ is the alphabet for Y. There is no loss of generality in this assumption, since the DPDT can check from such strings using its finite state control. The DPDT has start stack symbol $\vec{A}$ and operates by repeatedly executing the following procedure:

If the stack is empty then go to the end of the input and ACCEPT.

Otherwise, the top stack symbol is of the form $\vec{X}$ where X is in {A,B,C}.
Let $\vec{X}$ be the top stack symbol and do the following:
Advance the input head to the first

$$\overleftarrow{X}_i \; \vec{\alpha}_i \; \beta_i \quad \text{such that } \overleftarrow{X}_i \text{ is } \overleftarrow{X} \;;$$

POP $\vec{X}$ off the stack;
PUSH $\vec{\alpha}_i$ onto the stack (the right hand end of $\vec{\alpha}_i$ on top);
OUTPUT β_i.

It is routine to show that if T is the partial function computed by the above described DPDT, then $L_Y = T^{-1}(Y)$. □

The next lemma is stated in terms of 2-way deterministic pushdown transducers (2-DPDT's). A 2-DPDT is a deterministic finite state control connected to a two-way, read only input tape with two endmarkers, a pushdown store like that of a PDA, and a one-way, write only output tape. A formal definition of two-way PDA's can be found in GRAY, HARRISON & IBARRA [4]. A 2-DPDT is obtained by adding an output tape to a two-way deterministic PDA.

<u>LEMMA 3.6</u>. *If* T *is any* DPDT *function then we can find a* 2-DPDT *M such that:*

(1) *M computes* T.

(2) *On any input string, the input head of M moves in the following regular fashion. M scans the complete input alternately from left to right, then right to left, then left to right and so forth. Furthermore, M moves its input head on every move. So if the input is* $a_1a_2 \ldots a_n$, *including endmarkers, then at successive time instances M is scanning the symbols:* $a_1, a_2, \ldots, a_{n-1}, a_n, a_{n-1}, a_{n-2}, \ldots, a_2, a_1, a_2, \ldots$ $a_{n-1}, a_n, a_{n-1}, \ldots$

(3) *M has only two stack symbols.*

(4) *M has states numbered* $0,1,2,\ldots,k$ *with* 0 *the start state.*

(5) *M accepts by empty store and, in every accepting computation the last two states entered by the finite state control are* k *and* 0.

(6) *M accepts within time* $O(n^2)$.

PROOF. In the previous section we showed that every DPDA accepts in linear time. Using this fact and the techniques of the previous section it can be shown that every DPDT computes in linear time. It is fairly straight forward to show that any DPDT that runs in linear time can be simulated by a 2-DPDT which has property (2) and which runs in time $O(n^2)$. By standard techniques the 2-DPDT can then be made to have properties (3), (4) and (5) and still retain properties (1), (2) and (6). □

In order to complete the proof Theorem 3.1, it will suffice to establish the following lemma.

LEMMA 3.7. *Let* Y *be any language, let* L *be any language in* S(Y) *and let* Σ *be an alphabet such that* $L \subseteq \Sigma^*$. *Under these conditions we can find a function* g *such that:*

(1) g *is computable by a* $\log n$ *tape bounded deterministic Turing machine and*

(2) *For any string* w *in* Σ^*, w *is in* L *if and only if* g(w) *is in* L_Y.

PROOF. $L = T^{-1}(Y)$ where T is the partial function computed by some 2-DPDT M, as in Lemma 3.6. Let A, B be the two stack symbols of M, where A, B and C are as in the definition of D_3. Let δ be the transition function for M. We will use the following notation.

(*) $\qquad \delta(i,a,X) = (j,\alpha,\beta,y)$

means that if M is in state i, scanning input symbol a and having X on top of the pushdown store, then M will go to state j, replace X by α (the right hand end of α on top), output β and shift its input head left, right or not at all depending on whether y is -1, $+1$ or 0 respectively. We will code each such instruction as a string of symbols and then use this encoding to define g. We will omit y from these encodings, since we can always easily predict the input head movement of M and so need not have this information in our encodings. We now proceed to define g in terms of codes which define larger and larger pieces of the program for M. Let $\delta(i,a,X)$ be as in (*).

$$\mathrm{code}(\delta(i,a,X)) = \overleftarrow{X}\ \overrightarrow{\alpha}\ \overrightarrow{C}^{n(i,a,X)}\beta$$

where $n(i,a,X) = (k-i) + j + 1$. (Recall that the states of M are numbered $0,1,2,\ldots,k$. The reason for coding the state transition as $n(i,a,X)$ will become apparent if the reader fills in the details of the proof.)

$$\mathrm{code}(a,i) = \mathrm{code}(\delta(i,a,A)) \# \mathrm{code}(\delta(i,a,B)) \# \overleftarrow{C}$$

$$\mathrm{code}(a) = \mathrm{code}(a,0) \# \mathrm{code}(a,1) \#\ldots\# \mathrm{code}(a,k)\#$$

code(a) is extended to a homomorphism by defining

$$\mathrm{code}(a_1a_2\ldots a_n) = \mathrm{code}(a_1)\ \mathrm{code}(a_2)\ \ldots\ \mathrm{code}(a_n).$$

Finally g is defined by

$$g(w) = \mathrm{code}((¢w\$w^R)^{cn})$$

where w^R denotes w written backwards, ¢ and \$ are the left and right input tape endmarkers, and c is a constant such that M runs in time cn^2.

It should be clear that g is computable in $\log n$ storage, since the most difficult part of computing g(w) is counting up to cn and this can easily be done in $\log n$ storage. It should also be clear that, with the exception of the input head movements, code(a) in some sense codes all possible moves of M on input a. Now the string $(¢w\$w^R)^{cn}$ gives, in order, the symbols scanned by the input head of M when the input string is w. Let $(¢w\$w^R)^{cn} = a_1a_2\ldots a_t$. Then $g(w) = \mathrm{code}(a_1)\ \mathrm{code}(a_2)\ \ldots\ \mathrm{code}(a_t)$. If M has an output for the input w, then M on input w will execute one instruction from each of the blocks $\mathrm{code}(a_1)$, $\mathrm{code}(a_2)\ \ldots\ \mathrm{code}(a_s)$, where s is the number of steps executed by M on input w. Using these facts and techniques developed by SUDBOROUGH [13], we can show that: w is in $L = T^{-1}(Y)$

if and only if $g(w)$ is in L_Y. The details are quite similar to Sudborough's proof in [13] that there is a hardest deterministic context free language, and we direct the interested reader thither. □

Before leaving the discussion of Theorem 3.1 we note that with a slight modification to the proof we can show that every language in $S_2(Y)$ is $\log n$ reducible to L_Y where $S_2(Y)$ is the class of all languages of the form $T^{-1}(Y)$ where T is the partial function computed by some 2-DPDT that runs in polynomial time.

We conclude this section with a brief study of classes of the form $S(L)$ where L ranges over some well known language families. For this purpose let DPDA denote the deterministic PDA languages.

THEOREM 3.8. $S(\text{REG}) = \text{DPDA} \subset S(\text{DPDA}) \subseteq S(\text{CFL}) \subseteq S(\text{DLBA}) = \text{DLBA}$, *and at least one of the last two inclusions is strict.*

PROOF. Clearly $\text{DPDA} \subseteq S(\text{REG})$. To see that $S(\text{REG}) \subseteq \text{DPDA}$ notice that the finite state control of a DPDT can always be modified to check if its output is in a specified regular set. Hence $S(\text{REG}) = \text{DPDA}$. Since $\text{DPDA} \subseteq S(\text{DPDA})$ and the language L_3 of Example 2.3 is in $S(\text{DPDA})$ but not even in CFL we have $\text{DPDA} \subset S(\text{DPDA})$. By definition $S(\text{DPDA}) \subseteq S(\text{CFL}) \subseteq S(\text{DLBA})$. Since there is a DLBA language which can not be recognized in time $n \log n \,/\, A^{-1}(n)$, where A^{-1} is the inverse of the Ackerman Function, [7], and by a proof completely analogous to that of Corollary 2.7 we can show that each language in $S(\text{DPDA})$ is accepted in linear time (both bounds for off-line deterministic multitape Turing machines), we have $S(\text{DPDA}) \subset S(\text{DLBA})$ and at least one of the last two inclusions must be strict. Since DPDT's run in linear time by Lemma 2.5 they can be simulated in linear space and $S(\text{DLBA}) \subseteq \text{DLBA}$ and hence $S(\text{DLBA}) = \text{DLBA}$. □

We conjecture that $S(\text{DPDA}) \subset S(\text{CFL}) \subset S(\text{DLBA})$ but have no proof for this conjecture. Certainly, by Theorem 2.11, $S(\bar{\mathcal{D}} \cap \text{REG}) \subset S(\text{CFL})$.

Our last result exhibits a storage hardest language for the class $S(\text{CFL})$. (By Theorem 3.8 and Sudborough's result (Corollary 3.4), we obtain a storage hardest language for the class $S(\text{REG}) = \text{DPDA}$).

THEOREM 3.9. *We can find a language* L_1 *such that* L_1 *is in* S(CFL) *and such that every language* L *in* S(CFL) *has the property that* $L \leq_{\log} L_1$.

PROOF. The Language L_1 is a kind of combination of Sudborough's language and Greibach's [5] hardest context free language. The definition follows.

DEFINITION. Let D_2 and D_3 denote the Dyck set on two and three letters respectively. As before, let $\vec{A}$, $\overleftarrow{A}$, $\vec{B}$, $\overleftarrow{B}$, $\vec{C}$ and $\overleftarrow{C}$ be the six symbols used for writing strings in D_3; let the first four of these be the symbols used for writing strings in D_2. Let ¢, # and £ be three additional symbols. Let R be the regular set consisting of all strings of the form $w_1 \# w_2 \# \ldots \# w_m$ where each w_i is of the form $\overleftarrow{X}_i \vec{\alpha}_i \pounds y_1^i \pounds y_2^i \pounds \ldots \pounds y_{n(i)}^i$ with X_i in $\{A,B,C\}$, α_i in $\{A,B,C\}^*$ and with each y_j^i in $\{\vec{A},\overleftarrow{A},\vec{B},\overleftarrow{B},¢\}^*$.
L_1 consists of all strings in R such that, in the above notation, there are indicies $i_1 < i_2 < i_3 < \ldots < i_\ell \leq m$ and a function c with the properties that

(1) $\vec{A}\, \overleftarrow{X}_{i_1} \vec{\alpha}_{i_1} \overleftarrow{X}_{i_2} \vec{\alpha}_{i_2} \ldots \overleftarrow{X}_{i_\ell} \vec{\alpha}_{i_\ell}$ is in D_3

(2) i_1 is the least j such that $\overleftarrow{X}_j$ is $\overleftarrow{A}$; for each $k < \ell$, i_{k+1} is the least j which is greater than i_k and is such that $\overleftarrow{X}_j$ cancels with the right most symbol of $r(\vec{A}\, \overleftarrow{X}_{i_1} \vec{\alpha}_{i_1} \overleftarrow{X}_{i_2} \vec{\alpha}_{i_2} \ldots \overleftarrow{X}_{i_k} \vec{\alpha}_{i_k})$

(3) $y^{i_1}_{c(i_1)}\, y^{i_2}_{c(i_2)} \ldots y^{i_\ell}_{c(i_\ell)}$ is in $¢D_2$.

To see that L_1 has the desired properties notice that $S(\text{CFL}) = S(L_0)$ where L_0 is Greibach's hardest context free language. The result is then immediate, since L_1 is the storage hardest language for $S(L_0)$ which we get by applying the construction of Theorem 3.1. □

It would be interesting to exhibit a nice storage hardest language for the class of S(DPDA). However, we have not yet been able to produce such a language.

APPENDIX

THEOREM 2.11. *Let* $L = \{a^i b^j \mid i \le j\} \cup \{a^i b^j c^k \mid i + j = k\}$. *Then* $L_8 = L \cup \{a,b,c\}^* ¢$ *is not accepted by any* DCPDA.

The proof of Theorem 2.11 is in the spirit of the proof of Theorem 4.1 in GINSBURG & GREIBACH [3], i.e., an exhaustive case analysis. We first present an auxiliary definition and then establish a series of Lemma's, one of which, Lemma A2, contains a result on Dyck languages which may be interesting in its own right. In this Appendix we will closely follow the notation of GINSBURG & GREIBACH [3]. So, in particular, in a PDA instantaneous description, the top of the stack is the rightmost symbol.

DEFINITION. Let M be a DPDA, f and g be deterministic gsm maps and let D_2 denote the Dyck set on two generators. Then

$$D(M,f,g) = \{w \mid (q_0,w,Z_0) \overset{d*}{\vdash} (q,\varepsilon,\gamma) \text{ and } (f(\gamma^R))^R\, g(\gamma^R) \in D_2\}.$$

As is well known, the restriction to the Dyck set on two generators does not give us less than considering Dyck sets on r generators, $r \ge 0$.

LEMMA A1. *If* $L \subseteq \Sigma^*$, $¢ \notin \Sigma$, *and* $L \cup \Sigma^* ¢ = L(M)$ *for some* DCPDA M *then there is a* DPDA M' *and two deterministic* gsm *maps* f *and* g *such that* $D(M',f,g) = L$.

PROOF. We simulate M by a DCPDA M^* which, whenever M is to read a nonempty input symbol, does the following. M^* codes its present state in the topmost symbol on the ordinary stack and enters a new distinguished state q_s. Next, M^* reads the input symbol and executes the appropriate move of M. Clearly, $L(M^*) = L(M)$. Suppose that M^* has the ID $(q_s ¢\,\$, \alpha',\gamma')$ after processing some input $w \in \Sigma^*$ and just preliminary to reading ¢. Then $(q_s ¢\,\$, \alpha',\gamma') \overset{*}{\underset{M^*}{\vdash}} (-,-,-)$ where $(-,-,-)$ is an accepting halting ID of M with empty auxiliary stack. Therefore, there is a dgsm map f' which simulates the finite control of M^* starting in state q_s with fixed input ¢ \$ and hence performing a dgsm map from $(\gamma')^R$ to the auxiliary stack such that

$\alpha' f'((\gamma')^R) \in D_2$. Similarly, for each $w \in L$ there is an ID of M^* $(q_s,\$,\alpha,\gamma)$ just preliminary to the reading of \$ such that there is a dgsm map g for which $\alpha\, g(\gamma^R) \in D_2$. But for $w \in L$ we have also $\alpha f'(\gamma^R) \in D_2$. Hence for a dgsm map $f = h f'$, where h is an isomorphism which maps all symbols to their inverses, $(f(\gamma^R))^R\, g(\gamma^R) \in D_2$. Setting $M' = M^*_{ass}$ concludes the proof. □

The next "deflating" Lemma for Dyck languages has a fleeting, but misleading, resemblance to the uvwxy -Lemma.

<u>LEMMA A2</u>. *Let* $w_{n_0} = \alpha\beta^{n_0}\gamma\delta^{n_0}\mu \in D_2$ *for some* $n_0 > |\alpha\beta\gamma\delta\mu|$. *Then* $w_n = \alpha\beta^n\gamma\delta^n\mu \in D_2$ *for all* $n \geq 1$.

<u>PROOF</u>. Let D_2 be over the alphabet $\Delta_2 = \{\overrightarrow{0},\overleftarrow{0},\overrightarrow{1},\overleftarrow{1}\}$ with $\overrightarrow{0},\overleftarrow{0}$ and $\overrightarrow{1},\overleftarrow{1}$ matched pairs. red(w), $w \in \Delta_2^*$, denotes the resultant string obtained by cancelling symbols until no occurrences of $\overrightarrow{0}\overleftarrow{0}$ or $\overrightarrow{1}\overleftarrow{1}$ are left. $w \equiv v$ if red(w) = red(v). For a word $w \in \{0,1\}^*$ we denote the corresponding word over $\{\overrightarrow{0},\overrightarrow{1}\}^*$ by $\overrightarrow{w}$ and the one over $\{\overleftarrow{0},\overleftarrow{1}\}^*$ by $\overleftarrow{w}$. It is easy to see that for any $w \in D_2$ and any substring v of w holds: red(v) = $\overrightarrow{v}_1$ if v is a prefix of w, red(v) = $\overleftarrow{v}_1$ if v is a suffix of w, and red(v) = $\overleftarrow{v}_1\overrightarrow{v}_2$ for some $v_1,v_2 \in \{0,1\}^*$.
For a substring vv of a word $w \in D_2$ we have $vv \equiv \overleftarrow{v}_1\overrightarrow{v}_2\overleftarrow{v}_1\overrightarrow{v}_2$ and therefore $\mathrm{red}(\overrightarrow{v}_2\overleftarrow{v}_1) = \overrightarrow{u}$ or $\overleftarrow{u}$, $u \in \{0,1\}^*$.
Now let, in w_{n_0} above, $\mathrm{red}(\alpha) = \overrightarrow{\alpha}_1$, $\mathrm{red}(\beta) = \overleftarrow{\beta}_1\overrightarrow{\beta}_2$, $\mathrm{red}(\gamma) = \overleftarrow{\gamma}_1\overrightarrow{\gamma}_2$, $\mathrm{red}(\delta) = \overleftarrow{\delta}_1\overrightarrow{\delta}_2$, $\mathrm{red}(\mu) = \overleftarrow{\mu}_1$, $\mathrm{red}(\overrightarrow{\beta}_2\overleftarrow{\beta}_1) = \overleftarrow{\tau}$ or $\overrightarrow{\tau}$ and $\mathrm{red}(\overrightarrow{\delta}_2\overleftarrow{\delta}_1) = \overleftarrow{\sigma}$ or $\overrightarrow{\sigma}$.

<u>CLAIM</u>. $w_{n_0} \equiv \overrightarrow{\alpha}_1\overleftarrow{\beta}_1\overrightarrow{\tau}^{n_0-1}\,\overrightarrow{\beta}_2\overleftarrow{\gamma}_1\overrightarrow{\gamma}_2\overleftarrow{\delta}_1\overleftarrow{\tau}^{n_0-1}\overrightarrow{\delta}_2\overleftarrow{\mu}_1$ with $|\tau| = |\sigma|$.

<u>PROOF OF CLAIM</u>.

<u>Case 1</u>: $\mathrm{red}(\overrightarrow{\beta}_2\overleftarrow{\beta}_1) = \overleftarrow{\tau} \neq \varepsilon$. Then $\mathrm{red}(w_{n_0}) = \overleftarrow{z}\,v$, $\overleftarrow{z}\,v \in \Delta_2^*$, since

$$|\tau^{n_0-1}| + |\beta_1| \geq |\alpha\beta\gamma\delta\mu| + |\beta_1| > |\alpha|.$$

<u>Case 2</u>: $\mathrm{red}(\overrightarrow{\delta}_2\overleftarrow{\delta}_1) = \overrightarrow{\sigma} \neq \varepsilon$: symmetric with case 1.
It remains to be proven that $|\tau| = |\sigma|$.

Case 3: $|\tau| > |\sigma|$. Since $|\tau| = |\beta_2| - |\beta_1|$ and $|\sigma| = |\delta_1| - |\delta_2|$ we have

$$|\alpha_1 \tau^{n_0-1} \beta_2 \gamma_2 \delta_2| - |\beta_1 \gamma_1 \delta_1 \sigma^{n_0-1} \mu_1| =$$

$$n_0(|\tau|-|\sigma|) + |\alpha_1 \gamma_2| - |\gamma_1 \mu_1| \neq 0$$

Since $n_0 > |\alpha \beta \gamma \delta \mu|$ and $\| |\alpha_1 \gamma_2| - |\gamma_1 \mu_1| \| < |\alpha \beta \gamma \delta \mu|$ where $\| \quad \|$ denotes absolute value. Hence the amount of left brackets is unequal to the amount of right brackets and therefore $w_{n_0} \notin D_2$.

Case 4: $|\tau| < |\sigma|$: Symmetric to case 3, which proves the claim.

In case $\tau = \sigma = \varepsilon$ the Lemma is trivially true. Assume that $|\tau| = |\sigma| \geq 1$. From the claim we see that

$$w_{n_o} \equiv \vec{\nu}_1 \vec{\tau}^{\,n_0-1} \overleftarrow{\nu}_2 \vec{\nu}_3 \overleftarrow{\sigma}^{\,n_0-1} \overleftarrow{\nu}_4$$

for some $\nu_{1,2,3,4} \in \{0,1\}^*$, and therefore

$$\vec{\nu}_1 \vec{\tau}^{\,n_0-1} \overleftarrow{\nu}_2 \equiv \vec{\nu}_4^{\,R} (\vec{\sigma}^{\,R})^{n_0-1} \overleftarrow{\nu}_3^{\,R}$$

If $\nu_1 = \nu_4^R$ then $\nu_2 = \nu_3$ and $\tau = \sigma$ and the Lemma holds.
Assume that $\nu_1 = \nu_4^R \nu_5$, $\nu_5 \in \{0,1\}^+$; then $|\nu_2| - |\nu_3| = |\nu_1| - |\nu_4|$.
(The case that $|\nu_4| > |\nu_1|$ is symmetric). Therefore,

$$\vec{\nu}_5 \vec{\tau}^{\,n_o-1} \overleftarrow{\nu}_2 \equiv (\vec{\sigma}^{\,R})^{n_0-1} \overleftarrow{\nu}_3,$$

and since

$$|\nu_2| - |\nu_3| \leq |\gamma\delta| = |\alpha\beta\gamma\delta\mu| - |\alpha\beta\mu| \leq n_0 - 3$$

$$(|\beta| \geq 1, \ |\alpha| \geq |\nu_1| \geq 1)$$

We have that $\vec{\nu}_5\vec{\tau}^2$ is a prefix of red$(\vec{\nu}_5\vec{\tau}^{n_0-1}\overleftarrow{\nu}_2)$, and therefore $\vec{\tau} = \vec{\tau}_1\vec{\tau}_2$, $\tau_1,\tau_2 \in \{0,1\}^*$, such that $\vec{\sigma}^R = \vec{\tau}_2\vec{\tau}_1$.
Hence $\vec{\nu}_5 = \vec{\tau}_2(\vec{\tau}_1\vec{\tau}_2)^c$ and $\overleftarrow{\nu}_2 = \overleftarrow{\nu}_5^R\overleftarrow{\nu}_3$, $c \le n_0-3$, and therefore for all $n \ge 1$:

$$\vec{\nu}_1\vec{\tau}^{n-1}\overleftarrow{\nu}_2 = \vec{\nu}_4^R\,\vec{\tau}_2\,(\vec{\tau}_1\vec{\tau}_2)^{n+c-1}(\overleftarrow{\tau}_2^R\overleftarrow{\tau}_1^R)^c\,\overleftarrow{\tau}_2^R\,\overleftarrow{\nu}_3$$

$$\equiv \vec{\nu}_4^R\,\vec{\tau}_2\,(\vec{\tau}_1\vec{\tau}_2)^{n-2}\,\vec{\tau}_1\,\overleftarrow{\nu}_3$$

$$= \vec{\nu}_4^R\,(\vec{\sigma}^R)^{n-1}\,\overleftarrow{\nu}_3$$

It is easy to see that, for $n \ge 1$,

$$w_n \equiv \vec{\nu}_1\,\vec{\tau}^{n-1}\,\overleftarrow{\nu}_2\,\vec{\nu}_3\,\overleftarrow{\sigma}^{n-1}\,\overleftarrow{\nu}_4$$

Hence, under the assumptions made, $w_n \in D_2$, which proves the Lemma. □

LEMMA A3. *Let* x,y *be fixed words and let* f,g *be deterministic* gsm *maps. There are positive* b,d *such that for all words* z *we can find an* n_0 *such that the following holds: if* $c \ge n_0$ *and* $(f(xy^{b+cd}z))^R\, g(xy^{b+cd}z) \in D_2$ *then for all* $i \ge 1$ $(f(xy^{b+id}z))^R\, g(xy^{b+id}z) \in D_2$.

PROOF. Suppose M_h is a dgsm transducer which transduces xy^iz to $h(xy^iz)$; in particular M_h reads all of its inputs of the form xy^iz. By standard arguments concerning the cyclic behavior of deterministic finite state automata under constant input there are positive b_h, d_h for M_h such that $h(xy^{b_h+id_h}z) = \alpha\beta^i\gamma_z$ for some γ_z depending on z. Set b_h, d_h to b_f, d_f for M_f and to b_g, d_g for M_g. By choosing $b = \max(b_f,b_g)$ and $d = \ell.c.m(d_f,d_g)$ the Lemma follows by Lemma A2. □

LEMMA A4. *Let* M *be a loopfree* DPDA. *Then for an* ID (q,a^m,γ_0) (i) *or* (ii) *below must hold.*

(i) *There exists* $n \ge 1$ *such that for all* $m \ge 1$ *if* $(q,a^m,\gamma_0) \vdash^* (p,\varepsilon,\gamma)$ *for some* p *and* γ *then* $|\gamma| \le n$ *; or,*

(ii) *There exist positive integers* m,e *; words* w,y *; a symbol* Z *and a state* p *such that*

(a) $(q, a^{m+he}, \gamma_0) \overset{*d}{\vdash} (p, \varepsilon, wy^h Z)$, and,

(b) $(p, a^k, wy^h Z) \overset{*}{\vdash} (p', \varepsilon, \gamma)$ implies that $\gamma = wy^h\gamma'$, $\gamma' \neq \varepsilon$ for $k \geq 0$.

PROOF. Ginsburg and Greibach prove this result for the special case where γ_0 is a single symbol. (Lemma 4.1 in [3]). The reduction of the above Lemma to this special case is trivial. □

LEMMA A5. *If* M *is a* DPDA *and* f *and* g *are* dgsm *maps then*

$$D(M,f,g) \neq L = \{a^i b^j \mid i \leq j\} \cup \{a^i b^j c^k \mid i + j = k\}.$$

PROOF. Suppose $D(M,f,g) = L$. We will derive a contradiction. Without loss of generality we assume that M is loopfree. According to Lemma A4 either case (i) or case (ii) holds.

Case (i). For all $i \geq 0$ these are p_i and γ_i such that:

$$(q_0, a^i, Z_0) \overset{*}{\vdash} (p_i, \varepsilon, \gamma_i)$$

$|\gamma_i| \leq n$ for a fixed constant n. Hence then are i_1, i_2 $(i_1 > i_2)$ such that $p_{i_1} = p_{i_2}$ and $\gamma_{i_1} = \gamma_{i_2}$. Since $a^{i_2} b^{i_2} \in D(M,f,g)$ also $a^{i_1} b^{i_2} \in D(M,f,g)$: contradiction.

Case (ii). There exist positive integers m,e; a state q; a symbol Z, and strings y and w such that for all $h \geq 0$.

(1) $(q_0, a^{m+he}, Z_0) \overset{*d}{\vdash} (q, \varepsilon, wy^h Z)$ and

(2) $(q, a^k, wy^h Z) \overset{*}{\vdash} (q', \varepsilon, \gamma)$ implies $\gamma = wy^h\gamma'$, $\gamma' \neq \varepsilon$ $(k \geq 0)$.

Again we consider two subcases: either the stack pops all y's under constant input of b's or it does not.

Subcase 1. For each h there are j and q" such that

$$(q,b^j,wy^hZ) \vdash^* (q'',\varepsilon,w)$$

Since the state set is finite there are h_1,j_1 and h_2,j_2, such that $0 \neq m + h_1e + j_1 \neq m + h_2e + j_2 \neq 0$, which lead to the same state q". Since $a^{m+h_1e}b^{j_1}c^{m+h_1e+j_1}$ is in $D(M,f,g)$ also $a^{m+h_2e}b^{j_2}c^{m+h_1e+j_1}$ is in $D(M,f,g)$: contradiction.

Subcase 2. There are s,j,k,γ_2,q_2 such that for all $h \geq s$,

(3) $$(q,b^j,wy^hZ) \vdash^* (q_2,\varepsilon,wy^{h-k}\gamma_2) \text{ and,}$$

(4) $$(q_2,b^i,wy^{h-k}\gamma_2) \vdash^* (q_3,\varepsilon,\gamma) \text{ implies}$$

$$\gamma = wy^{h-k}v \quad \text{for some } v \neq \varepsilon \qquad (i \geq 0).$$

Now suppose $|v| \leq n$ for some constant n and all i. Then, similarly to subcase 1 above, we can ascertain that there are h,j_1,j_2, $j_1 \neq j_2$, such that $a^{m+he}b^{j_1}$ and $a^{m+he}b^{j_2}$ drive M into the same ID. Hence, since $a^{m+he}b^{j_1}c^{m+he+j_1} \in D(M,f,g)$ also $a^{m+he}b^{j_2}c^{m+he+j_1} \in D(M,f,g)$: contradiction. Therefore, we may assume that input b^i with M in state q_2 and $wy^{h-k}\gamma_2$ on its stack will cause the stack to grow arbitrarily large if i grows arbitrarily large and (by Lemma A4) the following must hold: there exist m_2,e_2,w_2,y_2,Z_2 and q_4 such that the following holds, for all $h \geq s$ and $h' \geq 0$.

(5) $$(q_2,b^{m_2+h'e_2}, wy^h\gamma_2) \vdash^{d*} (q_4, ,wy^{h-k}w_2y_2^{h'}Z_2) \text{ and}$$

(6) $$(q_4,b^i,wy^{h-k}w_2y_2^{h'}Z_2) \vdash^* (q_3,\varepsilon,\gamma) \text{ implies}$$

$$\gamma = wy^{h-k}w_2y_2^{h'}\gamma' \text{ with } \gamma' \neq \varepsilon \qquad (i \geq 0).$$

Now set $x = Z_2$ and $y = y_2^R$ in Lemma A3 and choose t,d as b,d in Lemma A3. Next choose $h \geq s$ such that

$$(7) \qquad m + he > m_2 + (t+d)\, e_2.$$

set $z = (wy^{h-k}w_2)^R$ in Lemma A3 and let n_0 be as in Lemma A3. Finally choose $c \geq n_0$ such that

$$(8) \qquad m_2 + (t+cd)e_2 > m + he.$$

By (8) $a^{m+he}b^{m_2+(t+cd)e_2} \in L = D(M,f,g)$. Then $(f(xy^{t+cd}z))^R g(xy^{t+cd}z) \notin D_2$. But then, by Lemma A3, also $(f(xy^{t+d}z))^R g(xy^{t+d}z) \in D_2$ and therefore $a^{m+he}b^{m_2+(t+d)e_2} \in D(M,f,g)$ which is impossible by (7), and Lemma A5 is proven. □

PROOF of Theorem: Immediate from Lemma's A1 and A5. □

REFERENCES

[1] AHO, A.V. & J.D. ULLMAN, *The Theory of Parsing, Translation and Compiling, Vols. I and II*, (Prentice Hall, Englewood Cliffs, N.J., 1973).

[2] GALLAIRE, H., *Recognition time of context-free languages by on-line Turing machines*, Information and Control, 15 (1969), 288-295.

[3] GINSBURG, S. & S.A. GREIBACH, *Deterministic context-free languages* Information and Control, 9 (1966) 563-582.

[4] GRAY, J.N., M.A. HARRISON & O. IBARRA, *Two-way push down automata*, Information and Control, 11 (1967) 30-70.

[5] GREIBACH, S.A., *The hardest context-free language*, SIAM J. Comput. 2 (1973) 304-310.

[6] GREIBACH, S.A., *Jump PDA's and hierarchies of deterministic context-free languages*, SIAM J. Comput. 3 (1974) 111-127.

[7] HOPCROFT, J.E., W. PAUL & L. VALIANT, *On time versus space*, Tech. Rept. Comp. Sci. Dept. Cornell University TR 75-264, Cornell University, Ithaca, N.Y. (1975).

[8] HOPCROFT, J.E. & J.D. ULLMAN, *Formal languages and their Relation to Automata*, (Addison-Wesley, Reading, Mass, 1969).

[9] JONES, N.D., *Space-bounded reducibility among combinatorical problems*, J. Comp. Syst. Sci. 11 (1975) 68-85.

[10] KASAI, T., *A universal context free grammar*, Inf. Contr. 28 (1975) 30-34.

[11] SAVITCH, W.J., *Relationships between nondeterministic and deterministic tape complexities*, J. Comp. Syst. Sci. 4 (1970), 177-192.

[12] SAVITCH, W.J., *How to make arbitrary grammars look like context-free grammars*, SIAM J. Comput. 2 (1973) 174-182.

[13] SUDBOROUGH, I.H., *On deterministic context-free languages, multihead automata and the power of an auxiliary pushdown store*, Proceedings 1976 ACM Symposium on the Theory of Computing, 141-148.

[14] SUDBOROUGH, I.H., *On the tape complexity of deterministic context-free languages*, Northwestern University Technical Report (1976).

FOOTNOTES

1. $L_9 = \{a^n b^n c \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$.
The proof that L_9 is not in DPDA is the same as the proof of Theorem 4.1 in [3]. $L_9 \in$ DCPDA is proven as follows. The accepting machine pushes the input word on the ordinary stack until it reads the endmarker, meanwhile checking for inclusion in $\{a\}^* \{b\}^* \{c,\varepsilon\}$ by its finite control. Depending on whether or not the last symbol on the ordinary stack was a c it then chooses one of the two obvious dgsm maps from the ordinary stack to the auxiliary stack so as to accept L_9.

2. The result and its proof remain valid even though we have changed the definition of CPDA slightly from the definition in [12].

3. Theorem 5 in [12] is stated incorrectly. The two homomorphisms that occur there should be different homomorphisms.

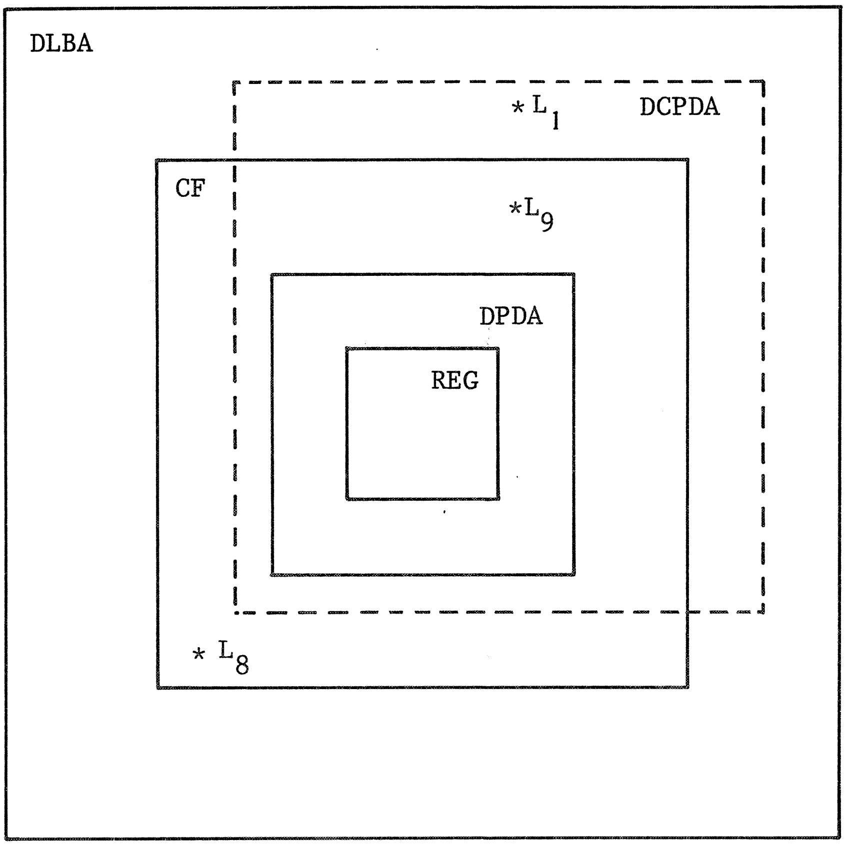

Figure 1.