1974

Linear Bounded Tree Automata

Carol Chrisman

Report Number:
74-109

Linear Bounded Tree Automata

Carol Chrisman

January 1974

CSD TR 109

This work was partially supported by NSF grant GJ 980.

Introduction

Automata with finite trees as input rather than strings were first used by Doner (1966, 1970) and Thatcher and Wright (1968). They defined a tree automaton which processed finite trees in a frontier to root fashion, ending in a final accepting or nonaccepting state. These tree automata are a natural generalization of finite automata working on strings and define a class of tree sets which are called the recognizable sets. Doner (1966, 1970) used the fact that it is decidable if a particular tree automaton accepts any tree at all to prove the decidability of the weak monadic second order theory of two successors.

Thatcher and Wright (1968) gave a characterization of the recognizable sets which generalizes the $\cup$, $\cdot$, $*$ characterization of regular sets of strings. A characterization of the recognizable sets as a class of tree sets containing some simple sets of trees and closed under $\cup$, $\cap$, complement and projection was given by Doner (1970). A third characterization was given by Brainerd (1969) in terms of the sets of trees generated by tree grammars, called regular systems, with productions of the form $\phi \rightarrow \psi$ where ϕ and ψ are finite trees. A production is applied to a tree by replacing an occurrence of a subtree ϕ by the tree ψ. More complicated tree grammars, called tree adjunct grammars or Tag's, were studied by Joshi, Takahashi, and Levy (1973). Tag grammars allow insertions of special adjunct trees at various places in a tree and can produce tree sets that are not recognizable. These tag grammars are a special case of the context-free dendrogrammars introduced by Rounds (1970) which produce complicated tree languages by allowing the subtrees of a node to be rearranged, repeated, or eliminated by the use of a tree production.

The question of whether other types of automata, besides finite automata, have natural generalizations to trees has not been considered in the literature. In this paper linear bounded automata are generalized to machines which have finite trees as input, called linear bounded tree automata or LBTA's. A LBTA is visualized as a machine which when in a particular state and reading the label on some node of the input tree can change the label on that node, change the state of the machine, and leave the reading head untouched or move it backwards to the previous node or forward to the left or right successor nodes. For simplicity we restrict our attention to binary trees (at most 2 branches from any node) and say an input tree is accepted if the LBTA ever enters one of the designated accepting states when started in an initial state at the top node of the tree.

A LBTA cannot alter the structure of the tree but only change the labels on the nodes and move back and forth in the tree. So deterministic linear bounded tree automata (DLBTA's) model the processing done by computers with trees when additions or deletions of nodes in the tree are not permitted but the information stored at any node of the tree can be altered.

Most of the theorems proved about LBTA's parallel theorems about linear bounded automata (lba's) proved by Myhill (1960), Landweber (1963), and Korado (1964). In particular, DLBTA's are shown to be closed under complement. The question of equivalence between LBTA's and DLBTA's is shown to be equivalent to the same question about lba's, providing a "tree viewpoint" for this old unsolved lba problem.

This close relationship between LBTA's and lba's is demonstrated by showing that one type of automata can simulate the other. The action of a LBTA M on a tree is simulated by a lba N on the string which is

the Polish notation of this tree in such a way that if a tree with n nodes is processed by the LBTA M in f(n) steps then the simulation by the lba will take at most $f(n)\cdot n^2$ steps. Also a lba N which is such that every string accepted by N is the Polish notation of a tree over some alphabet, can be simulated by a LBTA on the trees themselves. If on input of length n the lba N takes g(n) steps then the simulation by the LBTA takes at most $g(n)\cdot n$ steps. These simulations are then reasonably efficient.

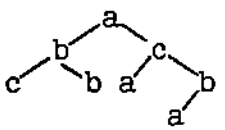
Section I: Definitions and Basic Facts

We begin with some necessary definitions and explanations of terminology to be used. The notation for languages and automata is that of Hopcroft and Ullman (1969) while that used for trees is from Doner (1970).

An alphabet Σ is a nonempty finite set of symbols. A word over Σ is a finite sequence of elements of Σ; ϵ denotes the empty word. The set of all words over Σ is denoted Σ^*.

A tree domain D is a finite subset of $\{1,2\}^*$ which is closed under initial segment (i.e., $uv \in D$ implies $u \in D$). A tree τ is a mapping $\tau: D \to \Sigma$ where D is a tree domain and Σ is a finite alphabet. For $w \in D$, $\tau(w)$ or τ_w denotes the label of the tree τ at node w. The domain D of τ is denoted dom τ. The empty tree which is a function with empty domain is denoted Λ and the set of all trees over Σ by $\Sigma^\#$.

As an example, let $\Sigma = \{a,b,c\}$ and $\tau = \{(\epsilon,a), (1,b), (2,c), (11,c), (12,b), (21,a), (22,b), (221,a)\}$. τ is generally pictured as



The subtree of τ beginning at w is denoted τ/w and defined as the tree τ' with $\tau'(u) = \tau(wu)$ for each $u \in \{1,2\}^*$. For u, $v \in \{1,2\}^*$ we say $u < v$ if for some $w \in \{1,2\}^*$ $uw = v$. For τ, τ' trees over Σ, $[\tau\ w/\tau']$ denotes the tree obtained by replacing the subtree of τ at w with the tree τ'. So $\tau[w/\tau']$ is the function Π such that

$$\Pi(u) = \tau(u) \text{ if } w \not< u$$

$$\tau'(v) \text{ if } u = wv \qquad \text{for } u \in \{1,2\}^*,$$

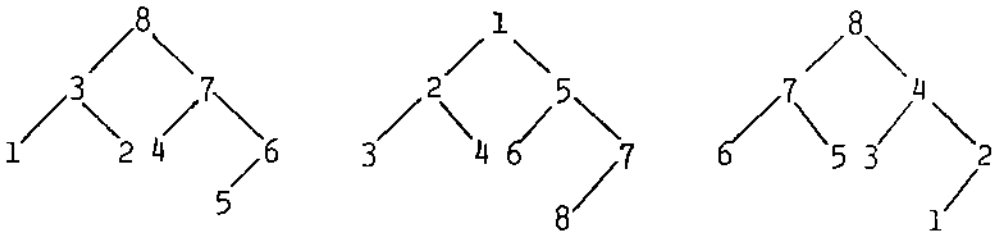
and $\text{dom}\ \Pi = \text{dom}\ \tau \cup \{wv \mid v \in \text{dom}\ \tau'\}$. $\tau[w/\tau']$ is only a tree if $w \in \{u1, u2 : u \in \text{dom}\ \tau\} \cup \{\varepsilon\}$. For $\sigma \in \Sigma$, τ, $\tau' \in \Sigma\#$ $\sigma[\tau,\tau'] = (\sigma\ [1/\tau])\ [2/\tau']$. Every tree except Λ can be expressed $\sigma[\tau,\tau']$ for some σ, τ, τ'.

The <u>depth</u> of a tree τ is denoted $||\tau||$ and is $1+n$ where n is the length of the longest word in the domain of τ. A word w such that neither $w1$ nor $w2$ is in dom τ is called a <u>terminal</u> node of τ. The set of all terminals is called the <u>frontier</u> of τ. The set of all nonterminals of τ is called the <u>interior</u> of τ. The <u>yield</u> of a tree τ, denoted yld (τ), is defined as the concatenation of the symbols appearing at the terminals of τ, taking the terminals in lexicographical order. For the tree τ in the example above yld (τ) = cbaa.

Sometimes we wish to process the tree in such a way that every node of the tree is visited. This is called traversing the tree. There are several standard ways to traverse a binary tree (see Knuth (1968)). Each method has the effect of ordering the nodes of the tree into a linear sequence. There will be several occasions when we want to process all the nodes in a tree in some organized fashion. The two ways of traversing a tree which we will use are endorder and reverse preorder. These methods process the nodes of a tree in the following way.

Endorder Traversal	Preorder Traversal
1. Traverse the left subtree	1. Visit the root
2. Traverse the right subtree	2. Traverse the left subtree
3. Visit the root	3. Traverse the right subtree

Each subtree of the tree is traversed in the same way. Reverse Preorder just reverses the order of Preorder traversal. Traversing the tree of the first example is illustrated below where the numbers on the nodes indicate that node's place in the ordering.



Endorder Preorder Reverse Preorder

We now are ready to define a linear bounded tree automaton.

Def. 1: A Linear Bounded Tree Automaton (LBTA) is a six-tuple $M = < K, \Sigma, \Gamma, d, Q, F >$ where $\Sigma \subset \Gamma$, $Q, F \subseteq K$ and $d: K \times \Gamma \to P(K \times \Gamma \times \{-1,0,1,2\})$. Σ is called the input alphabet, Γ the working or tape alphabet, Q the set of initial states and F the set of final states. A LBTA M is called deterministic if $d(q,\sigma)$ contains at most one element for each $q \in K$, $\sigma \in \Gamma$ and $|Q| = 1$. We use the abbreviation DLBTA for a deterministic linear bounded tree automaton. A triple (q, Π, w) with $q \in K$, $\Pi \in \Gamma\#$, $w \in \{1,2\}^*$ is called a configuration of the LBTA M and is interpreted as meaning that M is in state q with its reading head at the w^{th} node in the tree Π. The relation $\vdash_M$ which relates two configurations if the second can be derived from the first by an application of a transition of M is defined as follows:

Def. 2: $(q, \tau, w) \vdash_M (q', \tau', w')$ iff $(q', \tau'(w), m) \in d(q, \tau(w))$

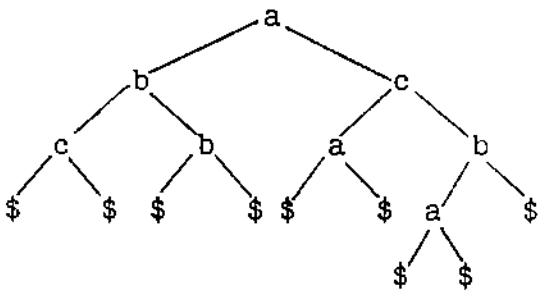
and for $m = -1$, $w'n = w, n = 1$ or 2
$m = 0$, $w' = w$
$m = 1$ or 2, $w' = wm$
and $\tau'(u) = \tau(u)$ for $u \neq w$.

When the LBTA M is understood, the relation $\vdash_M$ will be denoted $\vdash$. $\vdash_n$ will denote a derivation of n steps. Let $\vdash^*$ denote the transitive closure of $\vdash$.

One way in which LBTA's differ from linear bounded automata (lba's) is in the use of endmarkers. For lba's endmarkers are just a convenience and can be eliminated while for LBTA's endmarkers on the leaves of the tree are very useful and cannot be eliminated by the same technique used for lba's (see Landweber (1963)). The technique for lba's involves constructing a new lba (without endmarkers) which assumes as each symbol is read that it is the right end of the input and so has the lba process the string received so far as though it were the whole input. Then when the lba has determined if the original lba would accept this string the new lba tries to read the next character on the tape. If a character is present the lba now assumes this slightly longer string is the whole input and starts over to determine if the original lba would accept this string, continuing in this way until the lba actually moves past the right end of the input. The same construction will not work for trees because there are many places (not just one) where the LBTA could move out of the input tree. For example, the set of trees over {a,b} with each node labeled "a" cannot be accepted by a LBTA without endmarkers. Since all branches in a tree need not be the same length, no matter how such a LBTA M would try to traverse a tree τ before accepting it, there is another tree which is like τ except that a "b" labels a node not checked by M on τ and this tree would also be accepted by M.

A special symbol is then needed as an endmarker so throughout this paper the symbol \$ will be so used. We consider \$ an element of $\Gamma - \Sigma$ but restrict the use of this symbol by assuming that if $(q', \sigma, m) \in d(q, \$)$ for $\sigma \in \Gamma$ then $\sigma = \$$ and if $(q', \$, m) \in d(q, \sigma)$ for $\sigma \in \Gamma$ then $\sigma = \$$. Given a tree $\tau \in \Sigma\#$ we need to add endmarkers

before the LBTA processes the tree so we define a tree $\bar{\tau}$ which is just like τ but has an extra node labeled by a \$ at the end of every path in the tree. $\bar{\tau}$ is defined by $\bar{\tau}(w) = \tau(w)$ for $w \in \text{dom } \tau$, $w1 \notin \text{dom } \tau$ implies $\bar{\tau}(w1) = \$$ and $w2 \notin \text{dom } \tau$ implies $\bar{\tau}(w2) = \$$. We will use the convention that when a $\bar{}$ is placed over a tree name, the corresponding tree with endmarkers is meant. The tree given in the first example looks as follows with endmarkers:



A special symbol to designate the top of the tree is not needed because the LBTA always starts on the top node of the tree and so if desired can mark it specially so that it can be identified as the root of the tree in later processing.

The set of trees accepted or the tree language recognized by a LBTA M will consist of all trees $\tau \in \Sigma\#$ such that when M is started in an initial state on the top node of $\bar{\tau}$, M eventually goes into a final state. We denote the set T(M). Formally, we define T(M).

Def. 3: $\tau \in T(M)$ iff $(q, \bar{\tau}, \varepsilon) \vdash^* (q', \Pi, w)$ for some $q \in Q$,

$$q' \in F,\ \Pi \in \Gamma\#,\ w \in \{1,2\}^* \qquad (1)$$

Note that a LBTA cannot alter the structure of the tree but only change the labels on the nodes and move back and forth in the tree.

There are other possible acceptance conditions to be considered. Two such possibilities for a LBTA M are

$\tau \in T_2$ (M) iff M moves past an endmarker in a final state when inputed τ (2)

i.e., $(q, \overline{\tau}, \varepsilon) \vdash^* (q', \Pi, v)$ for some $q \in Q$ and $\Pi(v) = \$$ and $(q'', \$, i)\ d(q', \$)$ with $q'' \in F$ and $i = 1$ or 2.

T_3 (M) iff M backs up off the top of the tree $\overline{\tau}$ in a final state (3)

i.e., $(q, \overline{\tau}, \varepsilon) \vdash^* (q', \Pi, \varepsilon)$ for some $q \in Q$ and $(q'', \sigma, -1) \in d(q', \Pi(\varepsilon))$ and $\sigma \in \Gamma$.

Statement (2) corresponds to the acceptance condition for an lba that the lba moves past the right end of the input string. Statement (3) is what we mean when we say that an automaton runs off the top of the tree in a final state and corresponds to the idea of a lba moving to the left past the beginning of the input.

In the next lemma we will demonstrate that these three acceptance conditions are equivalent. That is, we will show that if a set of trees is accepted by a LBTA using one condition there is another LBTA using one of the other conditions which accepts the same set of trees.

Lemma 1: Acceptance conditions 1, 2, and 3 for LBTA are equivalent.

Proof: Let $M = \langle K, \Sigma, \Gamma, d, Q, F \rangle$ be any LBTA. An LBTA N such that $T_2(N) = T(M)$ is obtained by letting $N = \langle K - F \cup \{p\}, \Sigma, \Gamma, d', Q, \{p\} \rangle$ with d' defined by

$d'(q, \sigma) = d(q, \sigma)$ if $(q', \sigma', n) \notin d(q, \sigma)$ for $q' \in F$

$\{(p, \sigma, 2)\}$ if $(q', \sigma', n) \in d(q, \sigma)$ for some $q' \in F$

for each $q \in K - F$ and $d'(p, \sigma) = (p, \sigma, 2)$. Then T_2 (N) = T (M) since if M enters a final state on a tree τ then N would enter state p and move past an endmarker and the only way to get into state p is

if M would enter a final state on that tree. An LBTA N' such that $T_3(N') = T_2(M)$ is $N' = \langle K \cup \{p\}, \Sigma, \Gamma, d', Q, \{p\}\rangle$, with $d'(q, \sigma) = d(q, \sigma)$ for $\sigma \neq \$$ and $d'(q, \$) = (p, \$, -1)$ if $d(q, \$) \ni (q', \$, 1$ or $2)$ with $q' \in F$ and $d'(q, \$) = d(q, \$)$ otherwise. Also $d'(p, \sigma) = (p, \sigma, -1)$ for all $\sigma \in \Gamma$. Then $T_3(N') = T_2(M)$.

A LBTA N" such that $T(N") = T_3(M)$ is $N" = \langle K \cup \{p,s\}, \Sigma, \Gamma", d', \{p\}, \{s\}\rangle$ where $\Gamma' = \{\sigma' : \sigma \in \Gamma\}$ is a marked copy of the alphabet Γ and $\Gamma" = \Gamma \cup \Gamma'$ and

$d'(q, \sigma) = d(q, \sigma)$ for any $q \in K$ and $\sigma \in \Gamma$

$d'(p, \sigma) = \{(q, \sigma', 0): q \in Q\}$ for $\sigma \in \Gamma$

$d'(q, \sigma') = \{(s, \sigma', 0)$ if $(q', \xi, -1) \in d(q,\sigma)$ with $q' \in F\}$

$\cup$ $\{(q', \xi', n)$ if $(q', \xi, n) \in d(q, \sigma)$ for $n = 0, 1$ or $2\}$

Then $T(N") = T_3(M)$ since N" accepts a tree only if M would go off the top in a final state and the only time N" can accept a tree is if it is at the marked top node and M would then back up.

If M is a DLBTA then the corresponding LBTA N, N', N" are also deterministic.

Since these acceptance conditions are equivalent we will use whichever one is most convenient at a given time. If the condition is not specified assume that acceptance by entering a final state is meant.

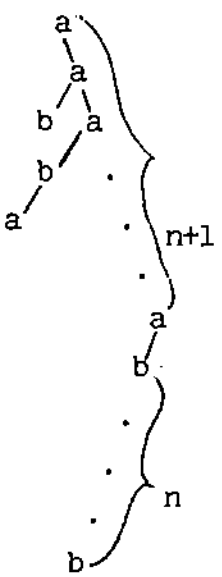
Since recognizable sets of trees correspond to regular sets of strings, if LBTA's are to properly generalize lba's the tree languages defined by LBTA's should contain the recognizable sets. We will now show that in fact the sets of trees accepted by DLBTA's properly contain the recognizable sets. We do this by showing that a DLBTA can simulate a frontier to root deterministic tree automaton. Recall the definition of a deterministic frontier to root tree automaton given by Doner (1970):

Def. 4: A tree automaton is a five tuple $\mathcal{A} = \langle S, \Sigma, t, s, D\rangle$ where S is a finite set of states, $t: S \times S \times \Sigma \to S$, $s \in S$ and $D \subseteq S$. The function t is extended to $\Sigma\#$ through the definition

$\bar{t}(\Lambda) = s$, $t(\sigma[\tau,\tau']) = t(\bar{t}(\tau), \bar{t}(\tau'), \sigma)$ for all $\sigma \in \Sigma$ and $\tau, \tau' \in \Sigma\#$.

$\mathcal{A}$ accepts a tree $\tau \in \Sigma\#$ if $\bar{t}(\tau) \in D$. Let $T(\mathcal{A})$ denote the set of Σ-trees accepted by $\mathcal{A}$. A set $A \subseteq \Sigma\#$ is recognizable if $A = T(\mathcal{A})$ for some tree automaton $\mathcal{A}$.

That LBTA are more powerful than tree automata is demonstrated by the following example of a set of trees that is not recognizable but is accepted by a DLBTA.

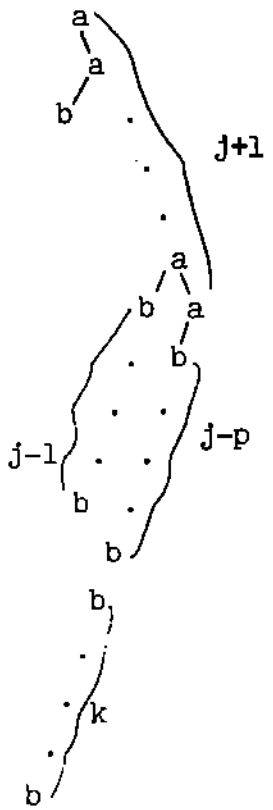
Let $\Pi = \{\tau_n : \tau_n =$  for $n \geq 0\}$

Theorem 1: Π is not recognizable but is accepted by DLBTA.

Proof: Suppose $\mathcal{A} = \langle S, \Sigma, t, s, D\rangle$ is a tree automaton such that $T(\mathcal{A}) = \Pi$ and S has m states. Let $j > m$ and consider the tree τ_j. Since the number of b's on the lowest branch of τ_j is greater than m when $\mathcal{A}$ is run on τ_j two states must be repeated on this branch, say at the k th and p th b's, with $k < p$.

But then the tree

τ =



would also be accepted by $\mathcal{DL}$ but this tree τ is not in Π since the number of b's on the lowest branch is less than j. So Π is not recognizable.

The idea of DLBTA to accept Π is that it should work down the right hand branch of a tree and at each node check to see that the correct number of b's is to the left of that node. In doing this checking the DLBTA must move back and forth on the portion of the right branch up to that current node and the branch to the left of this node matching the a's and b's. Such a DLBTA is given by

$$M = \langle \{q_0, q_1, p_1, \ldots, p_{11}, p, h\}, \{a,b\}, \{a,b,\$,A,B,X,S\},$$
$$d, \{q_0\}, p\rangle$$

where d is defined by the following table

$d(q_0, a) = (q_1, S, 1)$ change top symbol to S

$d(q_1, \$) = (q_0, S, -1)$

$d(q_0, S) = (p_1, S, 2)$

$d(p_1, a) = (p_1, X, -1)$ mark next level on right branch

$d(p_1, X) = (p_1, a, -1)$ } fix a's for counting

$d(p_1, A) = (p_1, a, -1)$ } at this level - remove old marks

$d(p_1, S) = (p_2, S, 2)$

$d(p_2, a) = (p_2, a, 2)$

$d(p_2, X) = (p_3, X, 1)$ move into left branch

$d(p_3, b) = (p_3, b, 1)$ } find next b

$d(p_3, \$) = (p_4, \$, -1)$ }

$d(p_3, B) = (p_4, B, -1)$ }

$d(p_4, b) = (p_5, B, 2)$ mark next b off

$d(p_4, X) = (h, X, 0)$ not enough b's on branch - halt

$d(p_5, \$) = (p_6, \$, -1)$

$d(p_6, B) = (p_6, B, -1)$ }

$d(p_6, b) = (p_6, b, -1)$ } start back for next a

$d(p_6, X) = (p_7, X, -1)$ }

$d(p_7, a) = (p_8, a, -1)$ }

$d(p_8, A) = (p_9, A, 2)$ }

$d(p_8, S) = (p_9, S, 2)$ } there is another a to mark off

$d(p_9, a) = (p_2, A, 2)$ }

$d(p_7, A) = (p_{10}, A, 2)$ }

$d(p_7, S) = (p_{10}, S, 2)$ } no more a's

$d(p_{10}, X) = (p_{10}, A, 1)$ }

$d(p_{10}, b) = (h, b, 0)$ too many b's on branch

$d(p_{10}, B) = (p_{11}, B, -1)$ right number of b's on branch

$d(p_{11}, X) = (p_1, X, 2)$ go to next level

$d(p_1, \$) = (p, \$, 0)$ accepts tree

Next we will show that tree automata can be simulated by DLBTA. Since nondeterministic and deterministic tree automata are equivalent (see Doner 1970) we know that every recognizable set is accepted by a deterministic tree automaton. The idea then is to make the DLBTA process the tree in endorder fashion determining at each node what state the tree automaton would be in after processing the subtree at that node. The DLBTA will then accept a tree if it goes off the top of the tree in one of the accepting states of the tree automaton (acceptance condition 3). The following lemma is needed for the proof of this result.

Lemma 2: If $(q, \Pi, \varepsilon) \vdash_M^* (q', \Pi', \varepsilon)$ and $\tau \upharpoonright w = \Pi$, $\tau' \upharpoonright w = \Pi'$ then $(q, \tau, w) \vdash_M^* (q', \tau', w)$.

Conversely, if $(q, \tau, w) \vdash_M^* (q', \tau', w)$ and in this derivation no transition is applied at $u < w$ then for $\Pi = \tau \upharpoonright w$ and $\Pi' = \tau' \upharpoonright w$ $(q, \Pi, \varepsilon) \vdash_M^* (q', \Pi', \varepsilon)$.

Proof: The same steps in the same order of the first derivation can be used to obtain the second derivation.

Theorem 2: Every recognizable set is accepted by a DLBTA.

Proof: Suppose $\mathcal{A} = \langle S, \Sigma, t, s_0 D\rangle$ is a deterministic frontier to root tree automaton. The following DLBTA will accept $t(\mathcal{A})$ by going off the top of the tree in an accepting state.

Let $M = \langle S \cup \{s\}, \Sigma, \Gamma, d, s, D\rangle$ where $\Gamma = \Sigma \cup \{\$\} \cup \{\overset{a}{q} : a \in \Sigma, q \in S\}$ and d is defined by $d(s, a) = (s, a, 1)$ $a \in \Sigma$ $d(s, \$) = (s_0, \$, -1)$, $d(q, a) = (s, \overset{a}{q}, 2)$ for $q \in S$ and $a \in \Sigma$, and $d(q, \overset{a}{q'}) = (t(q', q, a), a, -1)$ for $q, q' \in S$ and $a \in \Sigma$. Now by induction on the depth of a tree $\tau \neq \Lambda$, we will show $\bar{t}(\tau) = q$ iff $(s, \bar{\tau}, \varepsilon) \vdash_M^* (q_2, \tau', \varepsilon)$ where $\tau'(\varepsilon) = \overset{\tau(\varepsilon)}{q_1}$

for some q_1 and $\tau'(w) = \bar{\tau}(w)$ for $w \neq \varepsilon$

and $d(q_2, \tau'(\varepsilon)) = (q, \tau(\varepsilon), -1)$.

First assume $\bar{t}(\tau) = q$ for some $q \in S$.

If $||\tau|| = 1$ then $\tau = a \in \Sigma$ and since $\bar{t}(\tau) = t(\bar{t}(\Lambda), \bar{t}(\Lambda), a) = q$, $t(s_0, s_0, a) = q$. Then in M $(s, \bar{\tau}, \varepsilon) \vdash_M^* (s_0, \tau', \varepsilon) \vdash_M (s, \tau', 2) \vdash_M (s_0, \tau', \varepsilon)$ and $d(s_0, \tau'(\varepsilon)) = (q, a, -1)$ where $\tau'(\varepsilon) = \overset{a}{s_0}$.

If $||\tau|| > 1$ then $\tau = a\,[\Pi_1, \Pi_2]$ and by definition of $\bar{t}$, $\bar{t}(\tau) = t(\bar{t}(\Pi_1), \bar{t}(\Pi_2), a) = q$ so if $\bar{t}(\Pi_1) = q_1$ and $\bar{t}(\Pi_2) = q_2$ then $t(q_1, q_2, a) = q$.

Since $||\Pi_1|| < ||\tau||$ and $\bar{t}(\Pi_1) = q_1$ we have by the induction hypothesis $(s, \bar{\Pi}_1, \varepsilon) \vdash_M^* (q_4, \Pi_1', \varepsilon)$ where $\Pi_1'(\varepsilon) = \overset{\Pi_1(\varepsilon)}{q_3}$ and $d(q_4, \Pi_1'(\varepsilon)) = (q_1, \Pi_1(\varepsilon), -1)$.

Also since $||\Pi_2|| < ||\tau||$ $(s, \bar{\Pi}_2, \varepsilon) \vdash_M^* (q_6, \Pi_2', \varepsilon)$ where $\Pi_2'(\varepsilon) = \overset{\Pi_2(\varepsilon)}{q_5}$ and $d(q_6, \Pi_2'(\varepsilon)) = (q_2, \Pi_2(\varepsilon), -1)$.

Since $\tau \upharpoonright 1 = \Pi_1$ and $\tau \upharpoonright 2 = \Pi_2$ Lemma 2 can be applied. Together with the above statements this gives

$$(s, \bar{\tau}, \varepsilon) \vdash_M (s, \bar{\tau}, 1) \vdash_M^* (q_4, a[\Pi_1', \bar{\Pi}_2], 1)$$
$$\vdash_M (q_1, \bar{\tau}, \varepsilon) \vdash_M^* (s, \tau', 2) \quad \text{where } \tau'(\varepsilon) = \overset{a}{q_1},\ \tau'(w) = \bar{\tau}(w) \text{ for } w \neq \varepsilon$$
$$\vdash_M^* (q_6, \tau'(\varepsilon)\,[\bar{\Pi}_1, \Pi_2'], 2)$$
$$\vdash_M (q_2, \tau', \varepsilon)$$

So $(s, \bar{\tau}, \varepsilon) \vdash_M^* (q_2, \tau', \varepsilon)$ with $\tau'(\varepsilon) = \overset{a}{q_1}$ and $d(q_2, \overset{a}{q_1}) = (q, a, -1)$ since $t(q_1, q_2, a) = q$.

This completes half the induction part of the proof. The other half is similar but uses the second part of Lemma 2. Since $\mathcal{U}$ and M have the same accepting states this is sufficient to prove the theorem.

So we know that every recognizable set is accepted by a DLBTA. Next we will look more closely at the number of steps in the derivations of LBTA's.

It is possible to find a bound for the number of steps needed in an accepting derivation of a LBTA M on a tree τ in terms of the number of states and tape symbols in M and the size of the tree τ. This tells us that given a tree τ and a LBTA M it is decidable whether τ is accepted by M. This fact then enables us to show that there is a recursive set of trees which is not accepted by any LBTA. So we have the next two theorems.

Theorem 3: Given a LBTA $M = \langle K, \Sigma, \Gamma, d, Q, F\rangle$ and a tree $\tau \in \Sigma\#$, it is decidable whether $\tau \in T(M)$.

Proof: Let n = number of nodes in τ

m = number of nodes added to τ as endmarkers in $\bar{\tau}$

g = number of symbols in Γ minus 1

k = number of states in K

We will show that if $\tau \in T(M)$ there is an accepting derivation in M with less than or equal to $k(n+m)\ (g^n)$ steps.

Since the endmarkers stay the same in any derivation there are g^n possible trees that can appear in M derivations starting with $\bar{\tau}$. The number of different possible configurations (q, Π, w) of M when inputed $\bar{\tau}$ will then be the number of states in M times g^n times the number of nodes in $\bar{\tau}$ or $k(g^n)(n+m)$. Let $p = k(g^n)(n+m)$. Suppose τ has an M derivation of more than p steps, i.e., $(q_0, \bar{\tau}, \epsilon) \vdash_r (q, \Pi, w)$ with $q \in F$, $q_0 \in Q$ and $r > p$. Let $\alpha_1, \ldots, \alpha_r$ be the sequence of configurations in this derivation. Since $r > p$ two α_i's must be the same, say $\alpha_j = \alpha_s$ with $j < s$. But then $\alpha_1 = (q_0, \bar{\tau}, \epsilon) \vdash^* \alpha_j \vdash \alpha_{s+1} \vdash^* (q, \Pi, w)$ is a shorter derivation for τ. Since the derivation gets shorter each time, this process can be repeated until a derivation of length $\leq p$ is obtained.

For any tree τ and LBTA M each of the numbers n, m, g, and k is easily obtained so p can be calculated. Also all derivations for τ in M can be checked since there are only a finite number with less or equal p steps. One of these derivations reaches a final state if and only if $\tau \in T(M)$.

In Hopcroft and Ullman (1969) the existence of recursive sets that are not context-sensitive is proved by a standard diagonalization argument. The corresponding theorem about tree sets is the following theorem.

Theorem 4: There is a recursive set of trees over any finite alphabet Σ which is not accepted by any LBTA.

Proof: Since Σ is finite, $\Sigma\#$ can be enumerated $\tau_1, \tau_2, \ldots$ Since each of the components of a LBTA is finite (the names of states and names of symbols in $\Gamma - \Sigma$ are irrelevant) and in particular the number of transitions are finite, all LBTA with input alphabet Σ can be enumerated $M_1, M_2, \ldots$

Define the set of trees $A = \{\tau_i : \tau_i \notin T(M_i)\}$. For each i, τ_i and M_i can be obtained and by Lemma 4 it can be decided if $\tau_i \in T(M_i)$ so A is a recursive set of trees.

Suppose A is accepted by some LBTA, say $A = T(M_j)$. If $\tau_j \in A$, then since $A = T(M_j)$ $\tau_j \in T(M_j)$ but then by definition of A, $\tau_j \notin A$. If $\tau_j \notin A$ since $A = T(M_j)$ then $\tau_j \notin T(M_j)$ but then by definition of A $\tau_j \in A$. So we have a contradiction and A is not accepted by any LBTA.

Let Rec_Σ denote the recognizable sets over Σ and $LB_\Sigma (DLB_\Sigma)$ denote the sets of trees over Σ accepted by LBTA (DLBTA). Then Theorems 1, 2 and 4 can be summarized

$$Rec_\Sigma \subsetneq DLB_\Sigma \subset LB_\Sigma \subsetneq \Sigma\# \quad \text{for any alphabet } \Sigma .$$

Section II: Closure Properties

Next we consider closure properties of the sets of trees defined by LBTA's and DLBTA's. Let LB (DLB) denote the class of all sets of trees accepted by some LBTA (DLBTA). In this section we will demonstrate that LB and DLB are in fact closed under the operations of union, intersection, projection and inverse projection. Also we show that DLB is closed under complement but can say nothing about the class LB and complementation. So the situation with closure properties is exactly the same as with linear bounded automata or context-sensitive languages.

That LB is closed under union is easy to see. The construction goes as follows:

Theorem 5: LB is closed under $\cup$.

Proof: Suppose $A, B \in LB$. Then $A = T(M)$, $B = T(N)$ where $M = \langle K, \Sigma, \Gamma, d, Q, F \rangle$, $N = \langle L, \Sigma', \Gamma', d', R, G \rangle$. Assume $K \cap L = \phi$ and M and N have the same manner of acceptance. Then $M' = \langle K \cup L, \Sigma \cup \Sigma', \Gamma \cup \Gamma', d \cup d', Q \cup R, F \cup G \rangle$ is a LBTA such that $T(M') = T(M) \cup T(N)$.

Often it will be convenient to have extra storage space or multiple tracks at each node of the tree. The same technique that Myhill (1960) used for linear bounded automaton can be used for these tree automaton to show that multi-track LBTA are essentially the same as LBTA. This technique is to make the alphabet consist of new symbols which are columns of the old symbols. The transitions of a LBTA can then depend only on the symbol present on one track or on any combination of the symbols on the tracks at that node. Since multi-track LBTA are equivalent to LBTA we will use multiple tracks when convenient. Note that in the proof of Theorem 1 we could have used a 2 track DLBTA but instead modified the alphabet.

In the next theorem we will show that LB is closed under intersection. To do this, we create a LBTA which first traverses the tree in endorder fashion making a copy of the input tree on a second track. Then it simulates a LBTA M on the first track and if M would accept the tree returns to the top of the tree and starts over on the second track with a LBTA N. If N also accepts the tree then the new LBTA does. This LBTA will accept $T(M) \cap T(N)$.

Theorem 6: LB (DLB) is closed under intersection.

Proof: Suppose $A = T(M)$, $B = T(N)$ with $M = \langle K, \Sigma_1, \Gamma_1, d_1, Q, P\rangle$ and $N = \langle L, \Sigma_2, \Gamma_2, d_2, R, G\rangle$. Assume $K \cap L = \phi$ and both M and N accept trees by moving past an endmarker in a final state.

Let $\Gamma_1' = \{a' : a \in \Gamma_1\}$, $\Gamma_2' = \{a' : a \in \Gamma_2\}$, $\Gamma' = \Gamma_1' \cup \Gamma_2' \cup \Gamma_1 \cup \Gamma_2$. The new symbols will be used to mark the top node of the tree. Let $p, q_0, q_1, q_2 \notin K \cup L$. Let $M' = \langle K \cup L \cup \{p, q_0, q_1, q_2\}, \Sigma_1 \cup \Sigma_2, \Gamma_1 \cup \Gamma_2 \cup \Gamma_1' \cup \Gamma_2' \cup \{1,2\}, d, \{q_0\}, G\rangle$ with d defined as follows:

$d(q_0, a) = (q_1, a', 0)$ $\quad a \in \Gamma_1 \cup \Gamma_2$

$d(q_1, a) = (q_1, \overset{a}{1}, 1)$ $\quad a \in \Gamma'$

$d(q_1, \$) = (q_2, \$, -1)$

$d(q_2, \overset{a}{1}) = (q_1, \overset{a}{2}, 2)$ $\quad a \in \Gamma'$

$d(q_2, \overset{a}{2}) = (q_2, \overset{a}{a}, -1)$ $\quad a \in \Gamma_1 \cup \Gamma_2$

$d(q_2, \overset{a'}{2}) \ni (q, \overset{a'}{a'}, 0)$ $\quad$ each $q \in Q$

These states do the endorder traversal to copy the input onto track 2

for $q \in K$, $a \neq \$$ on track 1

$d(q, a) = d_1(q, a)$ leaving track 2 untouched

$d(q, a') = \{(q', b', n) : (q', b, n) \in d_1(q, a)\}$ track 2 untouched

for $q \in K$

$d(q, \$) \ni (p, \$, -1)$ if $d_1(q, \$) \ni (q', \$, 1 \text{ or } 2)$ $q' \in F$

$\ni (q', \$, n)$ if $d_1(q, \$) \ni (q', \$, n)$ $n = 0, -1$

$d(p, a) = (p, a, -1)$ $a \in \Gamma_1 \cup \Gamma_2$ looking at track 1

$d(p, a') \ni (q', a', 0)$ each $q' \in R$

for $q \in L$, a on track 2

$d(q, a) = d_2(q, a)$ leave track 1 untouched

$d(q, a') = \{(q', b', n) : (q', b, n) \in d_2(q, a)\}$

Then $T(M') = T(M) \cap T(N)$ since the accepting states of M' are those of N and the only time M' simulates N is if the tree would be accepted by M.

In the above proof if both M and N are DLBTA then M' is also deterministic so DLB is closed under intersection.

Corollary 1: LB (DLB) is closed under intersection with a recognizable set.

Given two alphabets Σ_1 and Σ_2 we call a mapping $g: \Sigma_1^{\#} \to \Sigma_2^{\#}$ a projection if $g(\Lambda) = \Lambda$ and $(g(\tau))(w) = g(\tau(w))$ for all w. In other words a projection is just an extension of a mapping from Σ_1 into Σ_2 to trees over these alphabets. So every projection can be defined in terms of this map from Σ_1 into Σ_2.

Theorem 7: LB (DLB) is closed under inverse projection.

Proof: Suppose $U \subseteq \Sigma_2^{\#}$ is in LB and $g: \Sigma_1 \to \Sigma_2$ is a projection then $g^{-1}(U)$ is in LB. Let $U = T(M)$, $M = \langle K, \Sigma_2, \Gamma_2, d, Q, F \rangle$. Assume $\Sigma_1 \cap \Gamma_2 = \phi$. Let $N = \langle K, \Sigma_1, \Gamma_2 \cup \Sigma_1, d', Q, F \rangle$

where $d'(q, a) = d(q, b)$ if $a \in \Sigma_1$ and $g(a) = b \in \Gamma_2$

$= d(q, a)$ if $a \in \Gamma_2$

Then $\tau \in T(N)$ iff $g(\tau) \in T(M)$ so $g(T(N)) = T(M) = U$ or $T(N) = g^{-1}(U)$.

Once again if M is deterministic, N will be deterministic.

Theorem 8: LB is closed under projection.

Proof: Suppose $g:\Sigma_1 \to \Sigma_2$ and $U \subseteq \Sigma_1^{\#}$ is in LB. Let $U = T(M)$ with M a LBTA. Define a LBTA N which first traverses an input tree $\tau \in \Sigma_2^{\#}$ replacing the label at each node by a possible preimage (nondeterministically chosen) under the projection g. N finishes this changing of labels at the top of the tree and then runs the automaton M. Any tree $\Pi \in \Sigma_1^{\#}$ such that $g(\Pi) = \tau$ can be obtained by N by a suitable choice of preimages and if M would accept this tree Π then N accepts τ so $T(N) = g(T(M)) = g(U)$.

Before we can prove that DLBTA's are closed under $\cup$ and projection we need to know that we can eliminate looping in DLBTA's and so force each DLBTA to halt on every input tree. This will also enable us to prove that DLBTA's are closed under complement. To do this we first show that every LBTA has an equivalent LBTA which halts on every input tree by moving past an endmarker. In Theorem 3 we established a bound on the number of possible steps that could be needed in a derivation to determine if an input tree is to be accepted by a particular LBTA. For DLBTA's this means that if a derivation has exceeded this bound then it is in a loop and so will never halt. Our current objective is to show that a LBTA can be modified to keep count of the number of steps or configurations it goes through so that when it reaches the bound it can stop and so halt on every input tree. The following theorem will give us simpler LBTA's with which to work.

Theorem 9: If M is an LBTA (DLBTA) there is an equivalent LBTA (DLBTA) N such that for all derivations of N on an input tree τ, N either doesn't halt or leaves the tree by moving past an endmarker.

Proof: Let $M = \langle K, \Sigma, \Gamma, d, Q, F\rangle$. Assume that M accepts by moving past an endmarker. When inputed τ there are two ways that M could fail to satisfy the condition stated above in a derivation for τ. Either M might halt in the middle of the tree because $d(q,a)$ is undefined for some q and a or M could back up off the top of the tree. These possibilities can be eliminated if transitions are defined for all states and symbols of Γ and if the top node is specially marked and the automaton not permitted to back up when at that node. This can be done as follows:

Let $N = \langle K \cup \{p,r\}, \Sigma, \Gamma', d', \{r\}, F\rangle$ with $\Gamma' = \Gamma \cup \{a' : a \in \Gamma\}$ and d' defined by

$$d'(r,a) \ni (q,a',0) \qquad \text{each } q \in Q,\ a \in \Sigma$$

$$d'(q,a) = \begin{cases} d(q,a) \text{ if } d(q,a) \neq \phi \\ (p,a,2) \text{ if } d(q,a) = \phi \end{cases} \qquad q \in K,\ a \in \Gamma$$

$$d'(p,a) = (p,a,2) \quad a \in \Gamma'$$

$$d'(q,a') \ni \begin{cases} (q',b',n) \text{ if } (q',b,n) \in d(q,a) \text{ and } n \neq -1 \\ (p,a',2) \text{ if } (q',b'-1) \in d(q,a) \text{ any } b,\ q' \end{cases}$$

$T(M) = T(N)$ since the accepting states are the same and all changes result in the state p which is not accepted but causes N to move past an endmarker. So N and M are equivalent and N has the desired condition.

In the next theorem we will show how given a LBTA M, to construct an equivalent LBTA M' which will simulate M and keep count of the number of configurations M goes through and if the count gets too big for an input tree, M' will halt in a nonaccepting state by moving past an endmarker. M' will then be a LBTA which halts on every input by moving past an endmarker. M' will have basic routines which will be executed as follows:

1) M' will make a move of M

2) M' will store the current state and trace a path back

to the top of the tree

3) M' will add 1 to the count and check to see that the count is smaller than the bound

4) M' will follow the path back to its place in the tree and continue with step 1

Each of these routines except possibly step 1 will be deterministic.

This construction is essentially the same as the one used by Kuroda (1964) to prove the same result about lba's. It is possible to do the same sort of thing for LBTA's because the nodes of the input tree can be ordered in such a way that they can be used for counting the number of configurations of the original LBTA.

The number of possible configurations for M is $k(g^n)(n+m)$ where k = number of states in M, g = number of symbols in Γ minus 1, n = number of nodes in the input tree, and m = number of nodes added as endmarkers. Since we haven't allowed LBTA's to change the endmarkers we only have n nodes available to use for the counting and so we need the bound in terms of n and not involving m. Since $m \leq n+1$, $kg^n(n+m) \leq kg^n(2n+1) < 3kng^n$. This number is easier to use as a bound than the more exact $(2n+1)\cdot kg^n$. If $\tau \in T(M)$ it has a derivation of less than $3kng^n$ steps so we will have M' stop all derivations which use $3kng^n$ steps of M. Because the nodes of the input tree can be linearly ordered they can be used to count to $(g)^n$ in base g. An extra track at each node will be used for this counting. Also an extra track at each node will be used to count the number of times $(g)^n$ is reached. When this count reaches 3kn, M' will halt in a non-accepting state. The nodes of the input tree will be used in reverse preorder fashion for this counting procedure.

Theorem 10: For every LBTA (DLBTA) M there is an equivalent LBTA (DLBTA) M'

which halts on every input tree by moving past an endmarker.

Proof: Assume $M = \langle K, \Sigma, \Gamma, d, Q, F\rangle$ and that M satisfies the conditions of the previous lemma. Let $K = q_1, \ldots, q_n$. M' will use 5 tracks for the following purposes

track 1: symbols of input tree

track 2: used by counting routine to order nodes of tree

track 3: first counting track for g^n

track 4: used to count number of times g^n reached on track 3

count up to 3kn times

track 5: store state when go to counting routine

Let $\overline{\Gamma} = \{a' : a \in \Gamma\}$, $\Gamma'' = a'' : a \in \Gamma\}$. Symbols in $\overline{\Gamma}$ are used to mark the top node and those in Γ'' are used to mark paths in the tree as needed by M' to find the place of M in the tree after attending to the counting procedure.

Let $\Gamma' = \Gamma \cup \overline{\Gamma} \cup \Gamma'' \cup K \cup \{0, 1, \ldots, g-1\} \cup \{1, \ldots, 3k\}$, and

$K' = K \cup \{p, p_1, \ldots, p_n, r, s, s_0, s_1, \ldots, s_8\}$.

Now let $M' = \langle K', \Sigma, \Gamma', d', \{s_0\}, F\rangle$ where d' defines the basic routines of M'. These transitions are listed below. Since 5 tracks are involved assume that $d'(q,x)$ means that x is on track 1 unless otherwise stated.

for each $q \in Q$, and $a \in \Sigma$ $d'(s_0, a) \ni (q, a', 0)$

M' makes a move like M by

for each $q \in K$ $d'(q,x) \ni (p_i, y, n)$ if $d(q,x) \ni (q_1, y, n)$

$d'(q,x') \ni (p_i, y', n)$ if $d(q,x) \ni (q_1, y, n)$

M' stores the state

for $p_1, \ldots, p_n$ $d'(p_i, x) = (s_1, \overset{x}{q_i}, -1)$ with q_i on track 5 and leaves a path back to the top of the tree by

$d'(s_1, x) = (s_1, x'', -1)$

$d'(s_1, x') = (s_2, x', 2)$ with 2 on track 2

The counting routine for M' is

$d'(s_2, x) = (s_2, x, 2)$ $x \neq \$$ 2 on track 2

$d'(s_2, \$) = (s_3, \$, -1)$

$d'(s_3, x) = (s_2, x, 1)$ tracks 1 and 2

$d'(s_3, x) = (s_4, x, 0)$ node found to add to count

$\emptyset$ means blank on track 2

add 1 on track 3

$d'(s_4, i) = (r, i+1, -1)$ if i on track 3 is less than g-1

$d'(s_4, x') = (s_5, x', 0)$ numbers on track 3, overflow of $(g)^n$

$d'(s_4, x) = (s_3, x, -1)$ normal carry in count

overflow of track 3, add 1 to track 4

$d'(s_5, x) = (s_5, x, 2)$ $x \neq \$$, 2 on track 2

$d'(s_5, \$) = (s_8, \$, -1)$

$d'(s_8, 2) = (s_5, 1, 1)$ 2, 1 on track 2

$d'(s_8, 1) = (r, \emptyset, 0)$ not at top node and i<3k; track 2 and 4 count ok

$d'(s_8, 1) = (s_8, \emptyset, -1)$ not at top node tracks 2 and 4

$d'(s_8, x') = (p, x', 2)$ count too high tracks 1, 2 and 4

$d'(s_8, x') = (r, x', 0)$ i<3k-1 count okay

$d'(p, x) = (p, x, 2)$ move past endmarker

When count is okay M' traces the path back to place in tree

$d'(r, x) = (r, x, -1)$

$d'(r, x') = (s_6, x', 1)$

$d'(s_6, x'') = (s_6, x, 1)$

$d'(s_6, x) = (s_7, x, -1)$ track 5 blank

$d'(s_6, x) = (q_j, x, 0)$ q_j on track 5
q_j

$d'(s_7, x) = (s_6, x, 2)$

Since no new final states are added to M' $T(M') \subseteq T(M)$. Also if a tree $\tau \in T(M)$ then M accepts τ by a derivation of less than $3kng^n$ steps so $T(M) \subseteq T(M')$ and M and M' are equivalent. Furthermore M' halts on every input tree by moving past as endmarker. If M is determinisitic so is M'.

This theorem enables us to prove the following:

Corollary 2: DLB is closed under complement.

Proof: By Theorem 10 we may assume that $M = \langle K, \Sigma, \Gamma, d, \{q\}, F\rangle$ is a DLBTA which halts on every input tree by moving past an endmarker in some state. Since M has only one possible derivation for any tree inputed, any tree $\tau \in \Sigma\#$ is either accepted by M or rejected by M because M leaves the tree in a non-accepting state. So $M' = \langle K, \Sigma, \Gamma, d, q, K-F\rangle$ will accept $\Sigma\# - T(M)$.

Corollary 3: DLB is closed under union.

Proof: Assume M and N are DLBTA which halt on every input tree by moving past an endmarker in an accepting or rejecting state. Let P be a DLBTA which copies the input tree onto a second track (as in Theorem 6) and then starts M on track 1. If M accepts the tree than have P accept the tree. If M would reject the tree have P return to the top node and start N on track 2 accepting the tree if N would. Then $T(P) = T(M) \cup T(N)$ and P is a DLBTA.

We will next show that DLB is closed under projection. Suppose $A = T(M) \subseteq \Sigma_1^{\#}$ with M a DLBTA and $g: \Sigma_1^{\#} \to \Sigma_2^{\#}$ a projection. The intuitive idea of an automaton N to accept g(A) is to have N list at each node of an input tree the possible elements of Σ_1 which are projected onto the label of that node, and then in an orderly fashion try each possible preimage tree to see if M would accept it.

If M would accept any preimage tree of the input tree then have N accept the tree. All possible preimage trees can be tried before N rejects a tree since DLBTA's always either accept or reject an input tree.

Theorem 11: Suppose $A = T(M) \subseteq \Sigma_1^{\#}$, $M = \langle K, \Sigma, \Gamma, d, \{q_0\}, F\rangle$ a DLBTA and $g: \Sigma_1^{\#} \to \Sigma_2^{\#}$ a projection. Then $g(A) \subseteq \Sigma_2^{\#}$ is accepted by a DLBTA.
Proof: Assume M halts on every input by moving past an endmarker in an accepting or rejecting state. Let m equal the largest number of elements of Σ_1 mapped onto a single element in Σ_2.

Construct a DLBTA $N = \langle K' \cup K, \Sigma_2, \Sigma_2 \cup \Gamma_1, d', \{s_0\}, F\rangle$ using m+2 tracks which works as follows:

1) N traverses the input tree in endorder using tracks 2 to m+1 to list $g^{-1}(a)$ at each node labeled by $a \in \Sigma_2$. Use a marker like ' above a symbol in the list to indicate the current preimage for that node. Initially mark the symbol on track 2 and place this symbol on track 1 so that when the top of the tree is reached a first possible preimage tree is on track 1.
2) N runs M on track 1
3) if M would accept this tree then N does so
4) if M would reject this tree then N adjusts the markers at each node to the next possible preimage tree. If all

possibilities have been tried then N rejects the tree, if not then N finishes traversing the tree in endorder placing the marked symbol at each node on track 1. When N reaches the top node it continues with step 2.

Track m+2 is needed to store temporary information when N traverses the tree. The way in which all possible preimage trees are tried before a tree is rejected by N can use some explanation. The fact that the nodes of the tree can be ordered by N (here in endorder) permits us to think of the tree as just n places with up to m choices for each place. If we number the places 1 to n we can systematically try all combintaions of the symbols listed. When the symbol in the i th place is changed then all possible choices for each place before i must be tried again.

Step 4 is formalized below where p, p', r, r', and $s \notin K$ for $q \in K$ $d'(q,\$) = (p,\$,-1)$ if $d(q,\$) = (q',\$,1 \text{ or } 2)$ with $q' \in K-F$

$d'(p, x) = (p, x, -1)$

$d'(p, top) = (p', top, 1)$ tracks 1 and m+2
$\qquad\qquad\qquad\quad 1$

$d'(p', x) = (p', x, 1)$ tracks 1 and m+2, $x \neq \$$
$\qquad\qquad\quad 1$

$d'(p', \$) = (r, \$, -1)$

$d'(r, x) = (p', x, 2)$ $x \neq$ top, tracks 1 and m+2
$\quad 1 \qquad\quad 2$

$d'(r, top) =$ reject tree

$d'(r, x) = (r', a_{i+1}, -1)$ move marker down one
$\quad a_1 \qquad\quad a_1$ rest of markers on tree
$\qquad\quad\;\; \vdots \qquad\quad\;\; \vdots$ fixed so continue
$\qquad\quad\;\; \vdots \qquad\quad\;\; \vdots$ putting symbol on track 1
$\qquad\quad\;\; a_i' \qquad\quad a_{i+1}'$ for each node
$\qquad\quad\;\; a_j \qquad\quad a_j$
$\qquad\quad\;\; 2 \qquad\quad \emptyset$

$d'(r, x) = (r, a_1, -1)$ marker was on last symbol at that node so start over

$$\begin{matrix} a_1 & a_1' \\ \vdots & \vdots \\ a_j' & a_j \\ 2 & \not\emptyset \end{matrix}$$

$d'(r', top) = (q_0, top, 0)$ start M on this preimage tree

$d'(r', x) = (s, x, 2)$ tracks 1 and m+2
$1 \quad 2$

$d'(r', x) = (r', a_1, -1)$ 2 on track m+2

$$\begin{matrix} a_1' & a_1' \\ \vdots & \vdots \\ 2 & \not\emptyset \end{matrix}$$

$d'(s, \$) = (r', \$, -1)$ $d'(s, x) = (s, x, 1)$ tracks 1, m+2
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad 1$

If $\tau \in T(M)$ then $g(\tau) \in T(N)$ since τ is one of the possible preimage trees for $g(\tau)$ and the final states for M and N are the same. So $g(A) \subseteq T(N)$. If $\tau' \in T(N)$ then there is $\overline{\tau} \in T(M)$ such that $g(\overline{\tau}) = \tau'$ so $\tau' \in g(A)$ and $T(N) \subseteq g(A)$ and so the proof is complete.

We could now give an alternate proof for Theorem 1. Doner (1970) gave a characterization of the recognizable sets as the least class of sets containing some special sets $E_\Sigma(\sigma, \sigma', \sigma'')$ and closed under the Boolean operations $\cup$, $\cap$, complement and arbitrary projection. Since we have just shown that DLB is closed under $\cup$, $\cap$, complement and projection we just have to show that $E_\Sigma(\sigma, \sigma', \sigma'')$ is in DLB. For any $\sigma, \sigma', \sigma'' \in \Sigma$ $E_\Sigma(\sigma, \sigma', \sigma'')$ is the set of all trees τ over Σ such that for some $w \in \text{dom } \tau$ $\tau(w) = \sigma$, $\tau(w1) = \sigma'$, and $\tau(w2) = \sigma''$. It is easy to construct a DLBTA which traverses the tree looking for a node labeled by σ and when it finds one checks to see if the successor

nodes are labeled σ' and σ''.

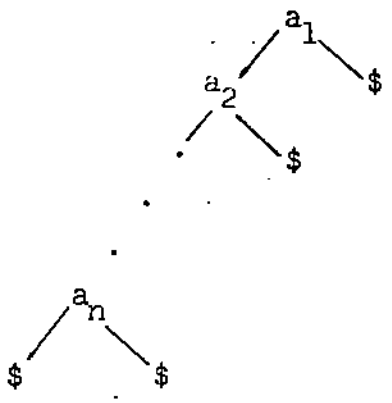
We can summarize the closure properties of the classes of tree languages LB and DLB in the following table.

Operation	LB	DLB
union	yes	yes
intersection	yes	yes
$\cap$ with recognizable set	yes	yes
complement	?	yes
projection	yes	yes
inverse projection	yes	yes

Section III: Relation to lba's

So far we have not discussed the question of whether or not nondeterministic and deterministic linear bounded tree automata are equivalent. Since this same question for linear bounded automata is an open problem and other results about LBTA's follow so closely the results about lba's we do not expect an easy answer. It is not surprising then to find that this question reduces to the same question about lba's. We will show that nondeterministic and deterministic LBTA's are equivalent iff nondeterministic and deterministic lba's are equivalent. In order to prove this we will first show that given an LBTA M we can construct an lba N which processes trees written in Polish notation and simulates the action of M on the trees so that N only accepts strings which are the Polish notation of trees $\overline{\tau}$ where τ is accepted by M. Also we will show that if L is a language over Σ accepted by an lba and $\$ \notin \Sigma$ then there is an LBTA N which accepts the set of trees whose Polish notation is $\{ \alpha \underbrace{\$ \ldots \$}_{n+1} : \alpha \in L, |\alpha| = n \}$.

That is, if $\alpha = a_1 \ldots a_n$ then the "comb" tree



is accepted by N.

The Polish notation referred to here is Polish prefix notation which really just amounts to writing the nodes of a tree in preorder. Finally we show that if M is a dlba such that every α in T(M) is the Polish prefix notation of a tree over Σ with endmarker \$ then there is a DLBTA N which simulates M and accepts just the trees whose Polish notation M accepts.

Theorem 12: Let $L \subset \Sigma^*$ be accepted by an lba M and $\$ \notin \Sigma$ then

$\{\tau$: Polish notation of $\overline{\tau}$ is $\alpha \underbrace{\$\ldots\$}_{n+1}$ where $\alpha \in L$ and $|\alpha| = n\}$

is accepted by a LBTA N.

Proof: Assume $M = \langle K, \Sigma, \Gamma, d, Q, F\rangle$ and $d: K \times \Gamma \to P(K \times \Gamma \times \{L,R,C\})$.

Let $\Sigma' = \{x' : x \in \Sigma\}$ be a marked copy of Σ.

Let $N = \langle K \cup \{p, r, s, t, z\}, \Sigma, \Gamma \cup \Sigma', d', \{p\}, F\rangle$ where d' is defined by

$d'(p, x) = (r, x', 2)$ for $x \in \Sigma$

$d'(r, \$) = (s, \$, -1)$

$d'(s, x) = (t, x, 1)$ for $x \in \Sigma$

$d'(t, \$) = (z, \$, -1)$

$d'(z, x) = (z, x, -1)$ for $x \in \Sigma$

$d'(z, x') \ni (q, x, 0)$ for each $q \in Q$

$d'(t, x) = (s, x, 2)$ for each $x \in \Sigma$

} tree in correct form; go back to top of tree and start M

and for each $q \in K$

$$d'(q, x) = \{(q', y, n) : (q', y, m) \in d(q, x) \text{ and for } m = L\ n = -1,\ m = C\ n = 0, \text{ and } m = R\ n = 1\}$$

N first checks to see that the input tree is a comb tree of the proper form and then simulates M on the leftmost branch of the tree. So N accepts the set of trees described above. If M is a deterministic lba (dlba) then N is a DLBTA.

Theorem 13: Given a LBTA (DLBTA) $M = \langle K, \Sigma, \Gamma, d, Q, F\rangle$ there is a lba (dlba) N such that $T(N) = \{\alpha : \alpha$ is Polish notation for a tree $\overline{\tau}$ with $\tau \in T(M)\}$.

Proof: The idea is to equip N with routines to process the Polish notation strings in order to determine where the right subtree of a node begins and which symbol is the proper antecedent of a node. This will ensure that the proper node is found when the lba imitates a move of the LBTA that was a back-up or move into right subtree.

Assume $K = \{q_1, \ldots, q_n\}$. Let $K' = K \cup \{p_1, \ldots, p_n, t_1, \ldots, t_n, p, f, r, r_1, \ldots, r_6, s, s', s_1, \ldots, s_6\}$, and $N = \langle K', \Sigma, \Gamma \cup \{*, 1, 2\} \cup K, d', Q, F \cup \{p\}\rangle$. N will be a 2 track lba with d' defined as follows:

for each $(q', \sigma', 1) \in d(q, \sigma)$ $\sigma \neq \$$ let $(q', \sigma', R) \in d'(q, \sigma)$ in N

for $(q', \sigma', 0) \in d(q, \sigma)$ $\sigma \in \Gamma$ let $(q', \sigma', C) \in d'(q, \sigma)$ in N

for $(q', \$, n) \in d(q, \$)$ $n = 1$ or 2 in the LBTA M

let $d'(q, \$) \ni \begin{cases} (p, \$, R) & \text{if } q' \in F \\ (f, \$, R) & \text{if } q' \notin F \end{cases}$ this takes care of case where M would move past an endmarker

and $d'(p, x) = (p, x, R)$ all x

$d'(f, x) = (f, x, R)$ all x

for $(q_i, \sigma', 2) \in d(q, \sigma)$ with $\sigma \neq \$$ in M

let $(r, \sigma', R) \in d'(q, \sigma)$ tracks 1,2
q_i

$d'(r, x) = (r_1, \underset{1}{x}, R)$ $x \neq \$$ tracks 1 and 2

$d'(r, \$) = (r_3, \underset{*}{\$}, L)$ tracks 1 and 2

$d'(r_1, x) = (r_1, \underset{1}{x}, R)$ $x \neq \$$ on track 1

$d'(r_1, \$) = (r_2, \$, R)$

$d'(r_2, x) = (r_1, \underset{2}{x}, R)$ tracks 1 and 2 $x \neq \$$

$d'(r_2, \$) = (r_3, \underset{*}{\$}, L)$

$d'(r_3, 2) = (r_4, \emptyset, L)$ track 2 $\emptyset$ stands for blank

$d'(r_3, 1) = (r_5, \emptyset, L)$ track 2

$d'(r_3, \emptyset) = (r_3, \emptyset, L)$ track 2

$d'(r_3, q_i) = (p_i, \emptyset, R)$ track 2

$d'(r_4, 1) = (r_5, \emptyset, L)$ track 2

$d'(r_4, x) = (r_4, \emptyset, L)$ $x = 2$ or $\emptyset$ track 2

$d'(r_5, n) = (r_6, n, R)$ $n = 1$ or 2 track 2

$d'(r_5, q_i) = (p_i, \emptyset, R)$ track 2

$d'(r_6, \emptyset) = (r_6, \emptyset, R)$ track 2

$d'(r_6, *) = (r_2, \emptyset, R)$ track 2

$d'(r_5, \emptyset) = (r_5, \emptyset, L)$ track 2

and $d'(p_i, \emptyset) = (p_i, \emptyset, R)$ $\emptyset$ on track 2

$d'(p_i, *) = (q_1, \emptyset, R)$ $\emptyset$ on track 2

As can be seen a move into the right subtree by the LBTA is rather complicated for the lba.

For $(q_i, \sigma', -1)$ $d(q, \sigma)$ a routine similar to that given above is needed. It would also use 2 tracks, the states s, s', s_1, ..., s_6 to locate the proper antecedent node in the tree and the states t_1, ..., t_n to carry the next state from the node σ left to the node located by the s_j's.

So for $(q_1, \sigma', -1) \in d(q, \sigma)$ in M

let $(s, \sigma', L) \in d'(q, \sigma)$ in N using tracks 1 and 2
q_1

and $d'(s, \$) = (s', \$, L)$

$d'(s', \$) = (s_1, \$, L)$

$d'(s', \sigma) = (s_4, \underset{1}{\sigma}, R)$

$d'(s, \sigma) = (s_4, \underset{1}{\sigma}, R)$

$d'(s_1, \$) = (s_1, \$, L)$

$d'(s_1, \sigma) = (s_2, \sigma, L) \quad \sigma \neq \$$

$d'(s_2, \$) = (s_1, \$, L)$

$d'(s_2, \sigma) = (s_3, \underset{1}{\sigma}, R) \quad \sigma \neq \$$

$d'(s_3, \$) = (s_4, \underset{*}{\$}, R)$

$d'(s_3, \underset{\not 1}{\sigma}) = (s_3, \underset{*}{\sigma}, R)$

$d'(s_3, \underset{1}{\sigma}) = (s_4, \underset{*}{\sigma}, R)$

$d'(s_4, \underset{\not 1}{\$}) = (s_5, \underset{*}{\$}, R)$

$d'(s_4, \underset{\not 1}{\sigma}) = (s_3, \underset{*}{\sigma}, R) \quad \sigma \neq \$$

$d'(s_4, \underset{1}{\sigma}) = (s_5, \underset{*}{\sigma}, R)$

$d'(s_4, \underset{*}{\sigma}) = (s_4, \underset{*}{\sigma}, R)$

$d'(s_5, \underset{\not 1}{\sigma}) = (s_4, \underset{*}{\sigma}, R) \quad \sigma \neq \$$

$d'(s_5, \underset{\not 1}{\$}) = (s_6, \underset{*}{\$}, L)$

$d'(s_5, \underset{*}{\sigma}) = (s_5, \underset{*}{\sigma}, R)$

$d'(s_5, \underset{1}{\sigma}) = (s_5, \underset{*}{\sigma}, R)$

$$d'(s_5, \underset{q_1}{\sigma}) = (t_1, \underset{\not b}{\sigma}, L)$$

$$d'(s_6, \underset{*}{\sigma}) = (s_6, \underset{*}{\sigma}, L)$$

$$d'(s_6, \underset{1}{\sigma}) = (s_2, \underset{1}{\sigma}, L)$$

and finally

$$d'(t_1, \underset{*}{\sigma}) = (t_1, \underset{\not b}{\sigma}, L)$$

$$d'(t_1, \underset{1}{\sigma}) = (q_1, \underset{\not b}{\sigma}, L)$$

The way the lba N locates the proper antecedent of a node in a tree is to start at the node and move left until two consecutive symbols ($\neq\$$) are encountered, say x_1x_2. The lba stops at x_1 and moves right until the end of the subtree begun by x_1 is encountered, then goes left again until the next two symbols ($\neq\$$) are reached. It continues in this way until when moving to the right to complete a subtree begun by x the q_1 on track 2 is found. Then the lba uses the states t_i to go back to x which is the proper antecedent node and so go on to the next move of the LBTA.

With d' so defined N is forced to process each string α just as M would process the tree $\bar{\tau}$ which has Polish notation α. The state p is the only new final state and it is only used when M would have moved past an endmarker in an accepting state so that the lba N does not continue processing the string but instead moves off the right end of the string in the state p. So the lba N properly simulates the LBTA M.

Corollary 4: If an LBTA M processes a tree $\bar{\tau}$ with n nodes in $f(n)$ moves then the lba N described in the theorem will process the string α which is the Polish notation of $\bar{\tau}$ within $f(n)\cdot n^2$ moves.

Proof: A move into the right subtree by the LBTA M causes the lba N to

reverse directions at most n times. For it starts moving to the right and reverses each time it hits the right end of a subtree (2 consecutive \$'s), then goes to the left until it reaches the top node of this subtree, goes one more square to the left to see if this node is the one started from, if not then goes right to the next right end of a subtree, continuing until the whole left subtree of a node is found. Each time the direction is reversed the lba switches at a different square than previously so there are at most n changes in direction. Going in either direction the lba can take at most n steps since the length of the input is n so it takes at most n^2 moves of the lba to simulate a move of the LBTA into the right subtree of a node.

Similarly, on a back-up move the lba N

1) goes left until two ($\neq$\$) symbols together, say xy, ignoring symbols with * on track 2
2) goes right until the end of the subtree begun by x, if don't encounter q_1 on track 2 go to step 1
3) if encounter q_1 on track 2 then return to symbol marked by 1 on track 2 — this is antecedent node

Again, each square where N will reverse direction is different, so at most n^2 moves of N are needed for this one move of M.

All other moves of M are simulated by N in one move.

This corollary shows that the simulation of LBTA's by lba's is reasonably efficient, although the details of the simulation look messy when written down.

Since Polish prefix notation just amounts to preordering the nodes of a tree it is possible to simulate a dlba by a DLBTA.

Theorem 14: If $M = \langle K, \Sigma \cup \{\$\}, \Gamma, d, \{q_1\}, F\rangle$ is a lba (dlba) such that α in $T(M)$ is the Polish notation of a tree over Σ with endmarker

\$ then there is a LBTA (DLBTA) N such that T(N) = {τ : Polish notation of $\bar{\tau}$ is in T(M)}.

Proof: Suppose $K = \{q_1, \ldots, q_n\}$.

Let $N = \langle K \cup \{p_1, \ldots, p_n, r_1, \ldots, r_n, s_1, \ldots, s_n\}, \Sigma, \Gamma \cup \{1,2\}, d', \{q_1\}, F\rangle$ where N uses 2 tracks and d' is defined in the following manner.

For each $q \in K$, $x \in \Gamma$ such that $(q_i, x', R) \in d(q, x)$

let $d'(q, x) \ni \begin{cases} (p_1, \$, -1) & \text{if } x = \$ \\ (q_i, \underset{1}{x'}, 1) & \text{if } x \neq \$ \\ & 1 \text{ on track } 2 \end{cases}$

For each $p_1, \ldots, p_n$

let $d'(p_i, \underset{1}{x}) = (q_i, \underset{2}{x}, 2)$ tracks 1 and 2

$d'(p_i, \underset{2}{x}) = (p_i, \underset{\not\emptyset}{x}, -1)$ tracks 1 and 2

} find next node in preorder or Polish arrangement of nodes of tree

For each $q \in K$, $x \in \Gamma$ such that $(q_i, x', L) \in d(q, x)$

let $(r_i, \underset{\not\emptyset}{x'}, -1) \in d'(q, \underset{n}{x})$, n = 1, 2 or blank

For each $r_1, \ldots, r_n, s_1, \ldots, s_n$

let $d'(r_i, x) = (q_i, \underset{\not\emptyset}{x}, 0)$

$d'(r_i, \underset{2}{x}) = (s_i, \underset{1}{x}, 1)$

$d'(s_i, \$) = (q_i, \$, 0)$

$d'(s_i, \underset{n}{x}) = (s_i, \underset{2}{x}, 2)$ $x \neq \$$ n = 1, 2 or blank

} find previous node in Polish arrangement of nodes of tree

For each $q \in K$, $x \in \Gamma$ such that $(q_i, x', C) \in d(q, x)$

let $(q_i, \underset{\not\emptyset}{x'}, 0) \in d'(q, x)$

d' forces N to copy faithfully the action of M and the extra states $p_1, \ldots, p_n, r_1, \ldots, r_n, s_1, \ldots, s_n$ and the second track are used only to locate the correct previous and next node in a preorder or Polish arrangement of the nodes of the tree. Since the final states for M and N are the same N will accept a tree τ iff M would accept the Polish

notation of the tree $\bar{\tau}$. Note that N will be deterministic if M was.

Corollary 5: In theorem 14 if the lba (dlba) M on input α with $|\alpha| = n$ takes f(n) steps then the simulation by the LBTA (DLBTA) N takes at most $f(n)\cdot n$ steps.

Proof: One move of the lba requires at most the depth of the tree number of steps to locate the proper previous or next node and the depth of the tree is $\leq n$.

The previous theorems now enable us to prove the following theorem.

Theorem 15: LBTA's and DLBTA's are equivalent iff lba's and dlba's are equivalent.

Proof: Assume that for every LBTA M there is an equivalent DLBTA M'. Suppose N is an lba. By Theorem 12 there is a LBTA N_1 such that $T(N_1) = \{\tau$: Polish notation of $\bar{\tau}$ is $\alpha \underbrace{\$\ldots\$}_{n+1}$ where $\alpha \in T(N)$ and $|\alpha| = n\}$. By assumption LBTA and DLBTA are equivalent so there is a DLBTA N_2 such that $T(N_2) = T(N_1)$. By theorem 13 there is a dlba N_3 which accepts the set $\{\beta : \beta = \alpha \$\ldots\$$ is the Polish notation of $\bar{\tau}$ with $\tau \in T(N_2)\}$. A dlba N_4 to accept T(N) is obtained by having N_4 add a \$ to the end of the input string for each symbol in the input, add one extra \$ and then start N_3 on the beginning of the resulting string. Then $T(N_4) = \{\alpha : \alpha \underbrace{\$\ldots\$}_{n+1} \in T(N_3)$ with $|\alpha| = n\} = T(N)$. Since every dlba is also a lba this proves one half of the theorem.

Conversely, assume that lba and dlba are equivalent and let N be a LBTA. By theorem 13 there is a lba N_1 such that $T(N_1) = \{\alpha : \alpha$ is Polish notation for a tree $\bar{\tau}$ with τ T(N)$\}$. By assumption lba's and dlba's are equivalent so there is a dlba N_2 such that $T(N_1) = T(N_2)$. Then by theorem 14 there is a DLBTA N_3 such that $T(N_3) = \{\tau$: Polish notation of

$\bar{\tau}$ is in $T(N_2)\} = T(N)$.

Since also every DLBTA is a LBTA the theorem is proved.

Since the languages defined by lba's are just the context-sensitive languages and dlba's are closed under complement, this theorem provides a new viewpoint for the complementation problem for context-sensitive languages. Perhaps thinking in terms of trees will simplify the problem. Before we can hope to make any progress more examples of interesting sets of trees are needed since all the tree sets studied so far can be accepted by DLBTA's.

So far we have not considered what types of languages can be obtained as the yields of sets of trees accepted by LBTA. Doner (1970) showed that the yield of a recognizable set is a context-free language. A simple example will show that even sets accepted by DLBTA do not necessarily yield context-free languages. Consider the set $A = \{a[\Pi, \Pi] : \Pi \in \{a, b\}\#, \Pi \neq \Lambda\}$. Let $L = \{yld(\tau) : \tau \in A\}$. Then $L = \{ww$: w is nonempty word over $\{a, b\}\}$ is not context-free (see Chomsky (1959)). It is tedious but routine to construct a DLBTA to accept A. The idea is to have the DLBTA traverse the two subtrees in endorder fashion comparing one node at a time. A second track will be needed by the automaton to leave paths to the current node in each subtree as it moves back and forth each time.

Is every context-sensitive language the yield of a set accepted by a LBTA? We know every context-sensitive language is defined by a linear bounded automaton. We can construct an LBTA which when imputed the tree

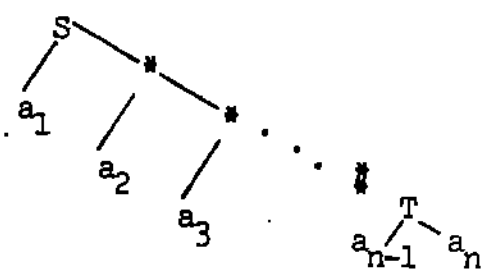
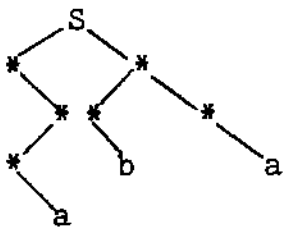


Figure 1

mimics the action of the linear bounded automaton (lba) on the string $a_1a_2 \ldots a_n$. A tree of this form is called a comb tree.

For a string $\alpha \in \Sigma^*$ and a number $n > 1$ we can find a tree with at least n nodes that has yield α. For example, for α = aba and n = 10 one such tree is



This ability to get as much space as wanted in the interior of the tree will enable us to simulate Turing Machines acting on strings. So we can show

Theorem 16: Every phrase structure language is the yield of some tree set accepted by a DLBTA.

Proof: Every phrase structure language $\subset \Sigma^*$ is the set accepted by a deterministic Turing Machine. Given such a Turing Machine M we will construct a DLBTA N which uses the nodes of the tree to simulate the action of M on the string that is the yield of the tree. N will be constructed to work on trees that have elements of Σ only on the frontier of the input tree, a special symbol $S \notin \Sigma$ on the root of the tree and the symbol $* \notin \Sigma$ on all other nodes. For a particular Turing Machine M and a string $\alpha \in \Sigma^*$ there is such a tree that has as many nodes as the number of tape squares used by M with input α.

The DLBTA N will linearize the nodes of an input tree by using endorder and will start processing each tree by moving the symbols making up the yield onto the beginning nodes of the tree (in endorder). Once the yield is placed on the beginning nodes N will move to the first node and begin to imitate M. Each transition of M will require several tran-

sitions in N to locate the proper previous or next node before changing into the new state of M.

The fact that trees can have long paths without contributing a corresponding number of characters to the yield of a tree resulted in our being able to simulate turing machines. If we restrict the trees in such a way that the number of nodes in a tree is related to the size of the yield then only context-sensitive languages are obtained as yields. One such restriction is to require that all nodes must have either 0 or 2 successors. A set of trees will be called 0-2 branching if every tree in the set is such that each node has either 0 or 2 branches. With these restricted sets of trees we prove:

Theorem 17: L is a context-sensitive language iff L is the set of yields of a set of 0-2 brahcning trees accepted by some LBTA.

Proof: If L is a context-sensitive language then $L = T(M)$ where M is an lba. Let S, T, and $*$ be symbols not in alphabet of L. It is now easy to construct a LBTA N which when inputed a comb tree with yield $a_1a_2\ldots a_{n-1}a_n$ mimics the lba's moves on this string. The LBTA N when inputed a tree would first check to see that it has the form of a comb tree pictured in Figure 1 (which is a 0-2 branching tree). N will accept a tree only if the lba would accept the yield of the tree. So $\{\text{yield}(\tau) : \tau \in T(N)\} = L$ since $L = T(M)$.

To prove the converse, first note that if a tree τ is 0-2 branching and $|\text{yield}(\tau)| = n$ then the number of nodes in τ (no endmarkers) is $= 2n-1$. Must add $2n$ endmarkers to get $\bar{\tau}$, so the maximum number of nodes in $\bar{\tau}$ is $4n-1$.

Now given an LBTA M which accepts a set of 0-2 branching trees an lba N which will accept just those strings which are yields of trees accepted by M can be constructed. The lba N on input α, $|\alpha| = n$,

will nondeterministically generate a sequence of length = 4n-1 of symbols in $\Sigma \cup \{\$\}$ where Σ is the alphabet of M. Then N will check that the sequence so generated is Polish notation for a tree $\bar{\tau}$ where τ is a 0-2 branching tree with yield α (note α is not destroyed when this sequence was generated). N will then simulate M on this generated sequence and accept α if M would accept the tree generated. Theorem 13 described this simulation of a LBTA by a lba on the Polish notation of a tree.

Section IV: Tag's and DLBTA's

Joshi, Takahashi, and Levy (1973) studied an interesting type of tree grammar called tree adjunct grammars. In these grammars trees are formed by inserting special adjunct trees into the middle of another tree. We will next show that given one of these grammars a DLBTA which accepts just the trees produced by this grammar can be constructed. These Tag sets are of interest because they are the most complicated form of tree set that can be shown to be accepted by DLBTA's (or by LBTA's). More complicated tree grammars are the context-free dendrogrammars of Rounds (1970) which include Tag's as a special case but there does not seem to be any way to show that LBTA's could accept these tree sets.

We need to define these tree adjunct grammars. Let V be a finite alphabet and $W \subset V$. We call W the terminal alphabet and V - W the nonterminal alphabet.

Def. 5: A tree adjunct grammar (Tag) is a pair G = (C,A) where C and A are finite sets of trees over V satisfying

1) if $\gamma \in C$ then $yld(\gamma) \in W^*$ and $\gamma(\varepsilon) = S$ where S is a distinguished symbol in V - W
2) if $\gamma \in A$ then $\gamma(\varepsilon) = X$ then $X \in V - W$ and $yld(\gamma) \in W^*XW^*$.

C is called the set of center trees and A the set of adjunct trees.

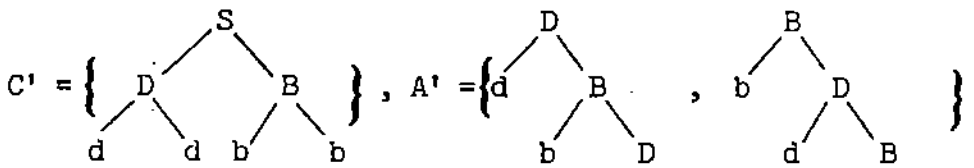
Suppose $\gamma \in A$ and $\tau \in V\#$ with $\tau(w) = \gamma(\varepsilon) = \gamma(v)$, v on the frontier of γ. Then γ is <u>adjoinable</u> at w in τ and $\tau(w, \gamma)$ is the tree obtained from τ by adjoining γ at w. $\tau(w, \gamma)$ is defined as $\tau[w/\gamma]\ [wv/\tau/w]$.

For a given Tag $G = (C, A)$ we write $\tau \mid\!\vdash_G \tau'$ iff for some γ and some w, γ is adjoinable at w to τ and $\tau' = \tau(w, \gamma)$. $\mid\!\vdash_G^*$ is the reflexive, transitive closure of $\mid\!\vdash_G$.

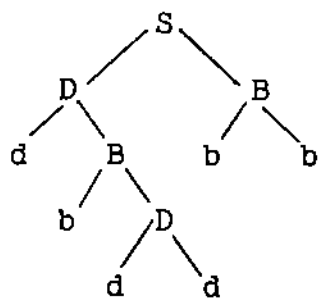
The <u>tree set</u> of a Tag G, $t(G)$ is defined as

$t(G) = \{\tau' \in V\# :$ for some $\tau \in C$, $\tau \mid\!\vdash_G^* \tau'\}$.

We still assume that all trees are binary although Tag's are usually not so restricted. An example of a Tag is $G' = (C', A')$ where



An example of a tree in $t(G')$ is



For more examples and a further discussion of Tag's see Joshi, Takahashi, and Levy (1973).

We now want to show that every set of trees generated by a Tag is accepted by a DLBTA. The idea is that at each node of the input tree an adjunct tree for that label or a special symbol ** will be selected. Then the machine will traverse the tree in a bottom to top fashion, try-

ing at each node to remove a copy of the adjunct tree indicated there. The symbol ** will be used to mean either that there are no adjunct trees for this symbol or that this time no attempt to remove an adjunct tree at this node will be made. After as much shrinking as possible of the tree by removing adjunct trees, the DLBTA will check to see if a center tree remains. If so, it will accept the tree. By systematically trying all combinations of the choices of adjunct trees or do nothing symbol at each node of the tree, a correct derivation (if there is one) of this input tree by the given Tag will be found. The provisions of shrinking the tree at a lower node before trying at a higher node and of allowing the do nothing symbol as one of the choices when picking adjunct trees to try to remove take care of any problems that would be caused by any overlapping of parts of trees or copies of adjunct trees appearing in other adjunct trees or the center trees.

Theorem 18: Every tree set of a Tag G = (C, A) is accepted by some DLBTA.

Proof: The DLBTA which will be constructed will use four tracks for the following purposes.

track 1: labels and *'s when remove an adjunct tree

track 2: used for ordering nodes when traversing the tree

track 3: to put selection of adjunct tree to try to remove

track 4: to store the original labels of the tree

The general routine of the DLBTA will be

1. Traverse the tree copying the input labels onto track 4 and puting the first choice of adjunct tree on track 3 at each node.
2. Traverse the tree in reverse preorder (right subtree, left subtree, node) trying to remove the adjunct tree listed at each node.

When the machine reaches the root of the tree it will have shrunk

the tree as much as possible for this choice of adjunct trees so it will try to recognize one of the center trees. If successful, the process ends.

3. Otherwise, traversing the tree in reverse preorder again, the machine will restore the original labels to track 1 and fix the next set of choices for adjunct trees. Go to step 2.

If the most adjunct trees for any symbol is k and the input tree has n nodes (not counting endmarkers) then the above procedure will be repeated at most $(k+1)^n$ times before the tree is finally rejected.

Suppose G is such that $C = \{\tau_1, \ldots, \tau_n\}$ and $A = \{\Pi_1, \ldots, \Pi_m\}$, and the trees are over the alphabet Σ. Let Σ' be the set of symbols in Σ which have adjunct trees. $\Sigma' = \{\sigma_1, \ldots, \sigma_k\}$, $k \leq m$. Let $i_1, \ldots, i_{end}$ enumerate the adjunct trees for the symbol σ_1.

Let $M' = \langle K, \Sigma, \Gamma', d', \{r\}, F\rangle$ be a DLBTA which runs through the center trees in some order and accepts the input tree if it is one of the center trees. When M' reads a label not right for the center tree it is currently checking, it backtracks to the top of the tree and starts to check for the next tree. If all the center trees have been tried, M' goes into the state u.

For each adjunct tree let $N_j = \langle L_j, \Sigma, \Gamma_j, f_j, r_j\rangle$ be a DLBTA which tries to recognize the adjunct tree Π_j but does not check for endmarkers on the branches from the one occurrence of $X_j = \Pi_j(\epsilon)$ on the frontier of Π_j. If the tree Π_j is recognized each label on track 1 of this adjunct tree is replaced by a $*$, $*_1$, or $*_2$ so that these nodes will be ignored in the future. The $*_1$ and $*_2$ are used to indicate a path from the root of this adjunct tree to the occurrence of X_j on its frontier (this X_j is not replaced). This procedure ends with N_j at the root of the adjunct tree in the state s_j. If

the adjunct tree is not present, N_j returns to the node it started at and goes into the state s_j.

Assume M' and each N_j doesn't change the labels on track 1 except to remove an adjunct tree. Let $L = \cup L_j$ and $P = \{p_i : q_i \in K \cup L - \{u, s_1, \ldots, s_m\}\}$. Let $M = \langle K \cup L \cup P \cup \{q,s,t,v,w\}, \Sigma, \Gamma, d, \{q\}, F\rangle$ be a DLBTA combining M' and $N_1, \ldots, N_m$. Γ contains Γ' and each Γ_j and d is defined as follows:

for $q_i \in K - \{u\}$ and $x \neq *_1, *_2, *$, $d(q_i, x) = (q', x, n)$ if $d'(q_i, x) = (q', x, n)$ $n = 0,1,2$

$= (p', x, -1)$ if $d'(q_i, x) = (q', x, -1)$

for $q_i \in L_j - \{s_j\}$ and $x \neq *_1, *_2, *$, $d(q_i, x) = (q', x, n)$ if $f_j(q_i, x) = (q', x, n)$ $n = 0,1,2$

$= (p', x, -1)$ if $f_j(q_i, x) = (q', x, -1)$

for $p_i \in P$, $d(p_i, x) = (p_i, x, -1)$ for $x = *, *_1, *_2$

$d(p_i, \sigma) = (q_i, \sigma, 0)$ for $\sigma \neq *, *_1, *_2$

for $q_i \in K \cup L - \{u, s_1, \ldots, s_m\}$

$d(q_i, *_1) = (q_i, *_1, 1)$

$d(q_i, *_2) = (q_i, *_2, 2)$

the transitions for step 1 in the general routine are

$d(q, \overset{\not\emptyset}{x}) = (q, \overset{2}{x}, 2)$; $d(q, \$) = (q, \$, -1)$

$d(q, \overset{2}{x}) = (q, \overset{1}{x}, 1)$; $d(q, \overset{1}{x}) = (q, \overset{\not\emptyset}{x}, -1)$ $\sigma_i \in \Sigma'$, $x \neq$ top

$\overset{\not\emptyset}{\sigma_i}$ $\overset{i_1}{\sigma_i}$ tracks 1,2,3,4

$$d(q, x) = \begin{matrix} 1 \\ \not\beta \\ \sigma \end{matrix} (q, x, -1), \quad \sigma \in \Sigma - \Sigma', \; x \neq top \qquad \begin{matrix} \not\beta \\ ** \\ \sigma \end{matrix}$$

$$d(q, top) = \begin{matrix} 1 \\ \not\beta \\ \sigma \end{matrix} (s, top, 0) \quad \text{with} \quad \begin{matrix} \not\beta \\ \alpha \\ \sigma \end{matrix}$$

$\alpha = **$ if $\sigma \in \Sigma - \Sigma'$

$\alpha = i_1$, if $\sigma = \sigma_i$

the transitions for step 2 are

$$d(s, x) = \begin{matrix} \not\beta \end{matrix} (s, x, 2) \qquad d(s, \$) = (s, \$, -1)$$

$$d(s, x) = \begin{matrix} 2 \end{matrix} (s, x, 1) \qquad d(s, x) = \begin{matrix} 1 \end{matrix} (t, x, 0)$$

$$d(t, x) = \begin{matrix} ** \end{matrix} (s, x, -1) \text{ tracks } 1,3; \; x \neq top \qquad d(t, top) = \begin{matrix} ** \end{matrix} (r, top, 0) \text{ tracks } 1,3$$

$$d(t, x) = \begin{matrix} j \end{matrix} (r_j, x, 0) \text{ tracks } 1,3; \; i \leq j \leq m \qquad d(s_j, top) = (r, top, 0)$$

$$d(s_j, x) = (s, x, -1) \quad x \neq top$$

the transitions for step 3 are

$$d(u, x) = (u, x, -1) \; x \neq top$$

$$d(u, top) = \begin{matrix} \not\beta \end{matrix} (v, top, 2)$$

$$d(v, x) = \begin{matrix} \not\beta \end{matrix} (v, x, 2) \; ; \qquad d(v, x) = \begin{matrix} 2 \end{matrix} (v, x, 1)$$

$$d(v, \$) = (v, \$, -1)$$

$$d(v, x) = \begin{matrix} 1 \\ ** \\ \sigma \end{matrix} (v, \sigma, -1) \qquad \begin{matrix} \not\beta \\ \alpha \\ \sigma \end{matrix}$$

$\alpha = **$ if $\sigma \in \Sigma - \Sigma'$

$\alpha = i_1$ if $\sigma = \sigma_i$

tree will be rejected if $x = top$ because moves out of tree

$$d(v, x) = \begin{matrix} 1 \\ i_j \\ \sigma^{i_j} \end{matrix} (w, \sigma, -1) \quad x \neq top \; ; \qquad \begin{matrix} \not\beta \\ i_{j+1} \\ \sigma^{i_{j+1}} \end{matrix} \quad i_j < i_{end}$$

$$= \begin{matrix} \not\beta \\ i_{j+1} \\ \sigma^{i_{j+1}} \end{matrix} (s, \sigma^{top}, 0) \quad x = top$$

$$d(v, x) = \begin{matrix} 1 \\ i_{end} \\ \sigma^{i_{end}} \end{matrix} (w, \sigma, -1) \quad x \neq top \; ; \qquad \begin{matrix} \not\beta \\ ** \\ \sigma \end{matrix}$$

$$= \begin{matrix} \not\beta \\ ** \\ \sigma \end{matrix} (s, \sigma^{top}, 0) \quad x = top$$

$$d(w, x) = \begin{matrix} \not\beta \end{matrix} (w, x, 2)$$

$$d(w, \$) = (w, \$, -1)$$

$$d(w, x) = \begin{matrix} 2 \\ \end{matrix} (w, x, \begin{matrix} 1 \\ \end{matrix})$$

$$d(w, x) = \begin{matrix} 1 \\ \sigma \end{matrix} (w, \sigma, \begin{matrix} \not{\emptyset} \\ \sigma \end{matrix}) \quad \text{tracks } 1,2,4 \quad x \neq \text{top}$$

$$d(w, \text{top}) = \begin{matrix} 1 \\ \sigma \end{matrix} (s, \sigma \text{ top}, \begin{matrix} \not{\emptyset} \\ \sigma \end{matrix}) \quad \text{tracks } 1,2,4$$

This completely defines M.

If the original Tag is a nice one and so the adjunct trees are not interconnected a more efficient machine can be constructed.

Bibliography

Brainerd, Walter S. (1969), "Tree Generating Regular Systems", Information and Control 14, 217-231.

Chomsky, N. (1959), "On Certain Formal Properties of Grammars", Information and Control 2, 137-167.

Doner, John E. (1966), "Decidability of the Weak Second Order Theory of Two Successors", Notices of Amer. Math Soc. 13, 634-635.

Doner, John E. (1970), "Tree Acceptors and Some of their Applications", Journal of Computer and System Sciences 4, 406-451.

Hopcroft, John and Ullman, Jeffrey (1969), Formal Languages and Their Relation to Automata, Addison-Wesley Publishing Company.

Joshi, Aravind; Levy, Leon, and Takashashi, Masako (1973), "A Tree Generating System", Automata, Languages and Programming, M. Nivat ed., North Holland Publishing Company, 453-466.

Knuth, Donald (1968), The Art of Computer Programming, vol. 1 - Fundamental Algorithms, Addison-Wesley Publishing Company.

Kuroda, S. Y. (1964), "Classes of Languages and Linear-Bounded Automata", Information and Control 7, 207-223.

Landweber, P. S. (1963), "Three Theorems on Phrase Structure Grammars of Type 1", Information and Control 6, 131-136.

Myhill, John (1960), "Linear Bounded Automata", Wadd Technical Note 60-165, Wright Patterson Air Force Base, Ohio.

Rounds, William (1970), "Mappings and Grammars on Trees", Math. Systems Theory 4, 257-287.

Thatcher, J. W. and Wright, J. B. (1968), "Generalized Finite Automata with an Application to a Decision Problem of Second-Order Logic", Math. Systems Theory 2, 57-81.