SUBLINEAR PARALLEL TIME RECOGNITION OF TREE ADJOINING LANGUAGES

Michael A. Palis
and Sunil Shende

MS-CIS-88-66
LINC LAB 127

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

August 1988

Acknowledgements: This research was supported in part by DARPA grant NOOO14-85-K-0018, NSF grants MCS-82-07294, DCR-84-10413, MCS-83-05221, MCS-8219196-CER, IRI84-10413-AO2 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Sublinear Parallel Time Recognition of Tree Adjoining Languages*

Michael A. Palis and *Sunil Shende*
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Abstract

A parallel algorithm is presented for recognizing the class of languages generated by tree adjoining grammars, a tree rewriting system which has applications in computational linguistics. This class of languages is known to properly include all context-free languages; for example, the non-context-free sets $\{a^n b^n c^n\}$ and $\{ww\}$ are in this class. It is shown that the recognition problem for tree adjoining languages can be solved by a concurrent-read, exclusive-write parallel random-access machine (CREW PRAM) in $O(log^2(n))$ time using polynomially many processors. This extends a previous result for context-free languages.

* Research supported in part by ARO grant DAA29-84-9-0027, NSF grants MCS-8219116-CER, MCS-82-07294, DCR-84-10413, MCS-83-05221, and DARPA grant N00014-85--K-0018.

1. Introduction

Tree adjoining grammars (TAG's) were introduced in 1975 by Joshi, Levy and Takahashi [JOSH75] as a generalization of context-free grammars (CFG's). Unlike a CFG, a TAG is a tree rewriting system: the basic elements are trees and the basic operation is *adjunction* which allows new trees to be created from a finite collection of basic trees. In [JOSH75], it was shown that the class of languages generated by TAG's (called tree adjoining languages or TAL's) properly includes all context-free languages (CFL's). For example, the sets $\{a^n b^n c^n\}$ and $\{ww\}$ are TAL's but not CFL's.

In the last few years, TAG's have received renewed attention in the area of computational linguistics because they have been found to be a useful grammatical model for natural language [KROC85]. What makes them more appealing is the fact that the important decidable and closure properties of CFL's also hold for TAL's [VIJA85]. In particular, TAL's are also polynomial-time recognizable. This was first shown by Vijay-Shanker and Joshi [VIJA85] who gave an $O(n^6)$-time recognition algorithm.

In [PALI87] we gave a parallel implementation of the TAL recognition algorithm that runs in linear time on a systolic array with n^5 processors. The systolic algorithm is optimal: the speed-up is linear in the number of processors. In a sense, our work extends the previous work by [KOSA75, CHIA84, CHAN87] on systolic CFL recognition/parsing.

In this paper, we present a faster parallel recognition algorithm for TAL's. In particular, we show that TAL recognition can be solved by a concurrent-read, exclusive-write parallel random-access machine (CREW PRAM) in $O(\log^2(n))$ time using polynomially many processors. Thus, TAL recognition is in *NC* (= class of problems solvable by PRAM's in polylogarithmic time and a polynomial number of processors). Our result extends that of Rytter [RYTT85], where he showed that CFL recognition can be solved in $O(\log^2(n))$ time by a CREW PRAM with n^6 processors (see also [RYTT87, RUZZ80]).

2. Tree Adjoining Grammars

Tree adjoining grammars were first defined in [JOSH75]. The original definition has been modified since; the one used here is from [VIJA85, WEIR87].

Definition 2.1. A *tree adjoining grammar* (*TAG*) is a 4-tuple $G = (V_N, V_T, I, A)$ where V_N is a finite set of nonterminal symbols, V_T is a finite set of terminal symbols, I is a finite set of initial trees and A is a finite set of auxiliary trees.

An initial tree has the following properties: internal nodes are labeled by nonterminal symbols; leaf nodes are labeled by either terminal symbols or the empty string ε.

An auxiliary tree has the following properties: internal nodes are labeled by nonterminal symbols; there is exactly one leaf node (called the foot node) which is labeled by the same nonterminal symbol that labels the root node; all other leaf nodes are labeled by either terminal symbols or ε.

A tree is called *elementary* if it is either an initial tree or an auxiliary tree. A node of an elementary tree is called a *nonterminal* node if it is labeled by a nonterminal symbol. We assume that every node of every elementary tree has a unique index; this can be done in one of several ways, say by a tuple (tree number, position within the tree).

There is an operation called *adjunction* with which trees can be composed. Let Γ be a tree containing some nonterminal node x and let Γ_x be the subtree rooted at x. Let Δ be an auxiliary tree whose root node is labeled by the same nonterminal symbol that labels x. (See Figure 2.1, but ignore the C_i's for the moment.) *Adjoining* Δ *into* Γ *at* node x results in a new tree Θ obtained as follows: replace the subtree Γ_x by Δ, then replace the foot node of Δ by Γ_x. We say that $\Theta = ADJ(\Gamma, x, \Delta)$.

Adjunction can be generalized by associating a *constraint* with each nonterminal node of an elementary tree. Constraints may be of two types - *selective* adjunction (SA) or *obligatory* adjunction (OA) - and are represented as tuples of the form (*type*, *subset*) where the *type* $\in$ {SA,OA} and *subset* is a subset of auxiliary trees. For a node with an SA constraint, any tree that is adjoined at the node should belong to the specified subset (the subset itself should contain only auxiliary trees whose roots have the same label as the node). The same is true for a node with an OA constraint except that, in addition, at least one adjunction should take place at the node. We say that an auxiliary tree is *adjoinable* at a node if this tree belongs to the subset given by the node's constraint. Note that if a node has an SA constraint and the specified subset is empty, then no adjunction can take place at the node. This special case is sometimes

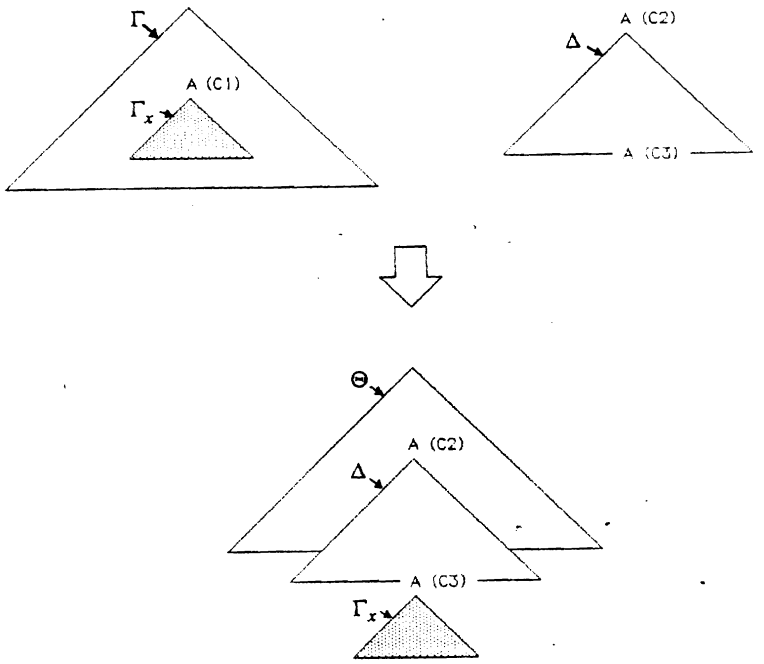

Figure 2.1. The adjunction operation.

referred to as the *null* adjunction (NA) constraint.

In constrained adjunction, the nodes of the resulting tree Θ would have the constraints shown in Figure 2.1 (indicated by the C_i's). More precisely, the portion of Θ copied from Δ have the same constraints as Δ; the tree copied from Γ_x have the same constraints as Γ_x, except for the root which instead gets the constraint of the foot node of Δ.

A tree Γ *derives* a tree Θ *in one step* (written $\Gamma \rightarrow \Theta$) iff Θ results from Γ by adjoining an auxiliary tree at some node of Γ. Γ *derives* Θ (written $\Gamma \rightarrow^* \Theta$) iff there is a sequence of zero or more trees starting with Γ and ending in Θ such that every tree in the sequence derives its successor in one step.

Let x be a node of some elementary tree Γ and let Γ_x be the subtree rooted at x. The *tree set* of x is defined as $T(x) = \{\Theta \mid \Gamma_x \rightarrow^* \Theta$ and Θ has no nodes with OA constraints$\}$. If x is the root of an initial tree, then every tree in $T(x)$ has the property that its frontier (the left-to-right sequence of its leaf labels) is a string in V_T^*. For a TAG G, we define the *language generated by* G as $L(G) = \{w \mid w$ is the frontier of some tree in $T(x_0)$ and x_0 is the root of some initial tree$\}$. A language L is called a *tree adjoining*

language (*TAL*) iff $L = L(G)$ for some TAG G.

TAG's are known to be strictly more powerful than CFG's [JOSH75]. For example, the sets $\{a^n b^n c^n\}$ and $\{ww\}$ are TAL's but not CFL's. Figure 2.2 gives an example of a TAG G for which $L(G) = \{a^n b^n c^n \mid n > 0\}$.

$$G = (\ \{S\},\ \{a,b,c\},\ \{t_0\},\ \{t_1,t_2\}\ ).$$

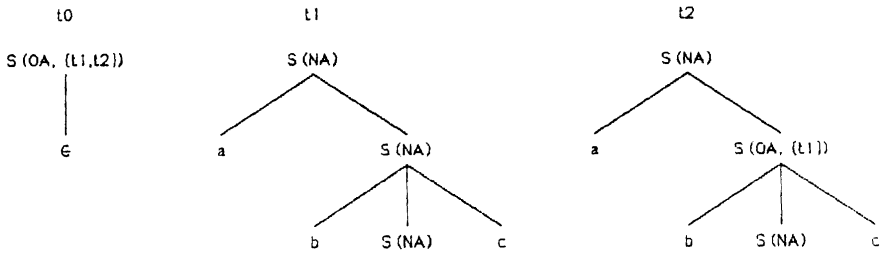

Figure 2.2. A TAG G generating the language $\{a^n b^n c^n \mid n > 0\}$.

If a TAL L does not contain the empty string ε, then L is generated by a TAG G in *normal form* [WEIR87], which is analogous to the Chomsky normal form for CFG's [AHO72]. A TAG G is in normal form iff it has a single initial tree of the form depicted in Figure 2.3-(a), and auxiliary trees each having one of the forms depicted in Figure 2.3-(b)[1]. In particular, note that: (1) every tree is binary; (2) every node with exactly one child has an OA constaint; and (3) every node with two children, and every foot node, has an NA constraint. Figure 2.4 gives a TAG G_1 in normal form which is equivalent to the TAG G of Figure 2.2.

For the remainder of the paper, we consider only TAL's which does not contain ε and assume that the corresponding TAG's are in normal form.

3. Derivation Trees and Realizable Items

Let x be a node of some elementary tree Γ. Call x an *open* node if Γ is an auxiliary tree and the

1 The normal form given here is slightly different, but equivalent (in terms of the languages generated) to the one given in [WEIR87].

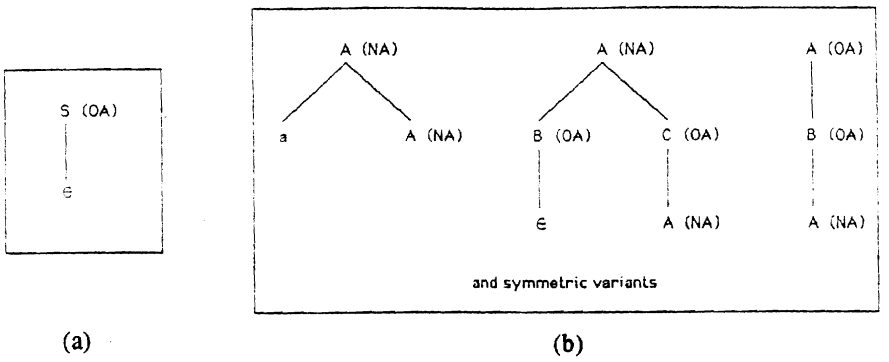

Figure 2.3. A TAG in normal form.

G_1 = ({S,T,A,B,C}, {a,b,c}, {t_0}, {$t_1,t_2,t_3,t_4,t_5,t_6,t_7$}).

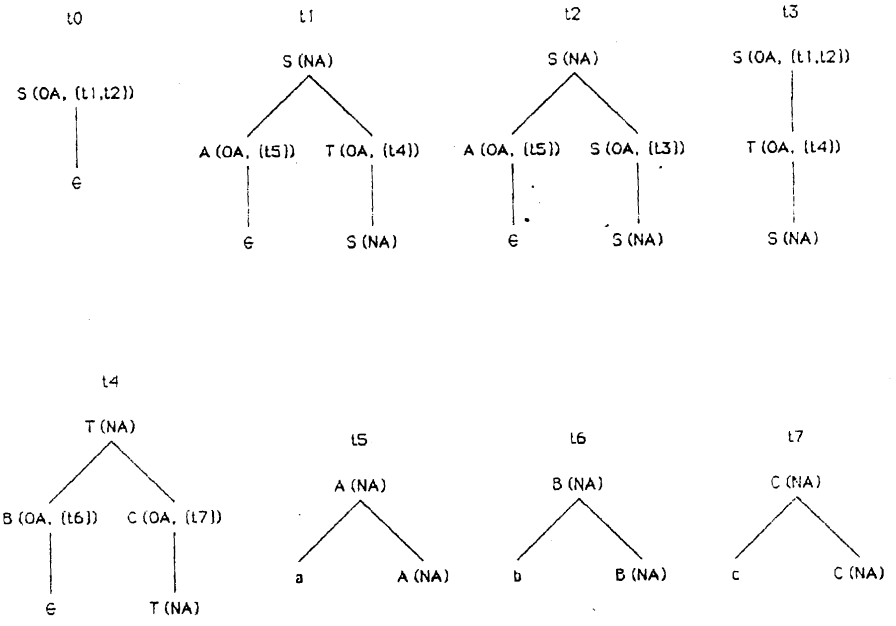

Figure 2.4. A TAG G_1 in normal form equivalent to the TAG G of Figure 2.2.

foot node of Γ is a descendant of x. Otherwise, call x a *closed* node. It is easy to verify that every tree Θ in $T(x)$ has a frontier of the form $u_1 A u_2$, where u_1, $u_2 \in V_T^*$ and either: $A \in V_N$ is the label of the foot node of Γ (if x is an open node), or $A = \varepsilon$ (if x is a closed node). In both cases, we define the *size* of Θ as $|u_1| + |u_2|$.

Let (x,y) be an ordered pair of nodes. Let Θ be a tree in $T(x)$ containing a subtree Δ in $T(y)$, as shown in Figure 3.1 -(a) to -(c). In the figure, x and y are assumed to be open nodes; the three cases correspond to the three possible positions, relative to the frontier of Δ, of the foot node (labeled A) of the elementary tree containing x. As special cases: if x is a closed node then A is absent, and if y is a closed node then the subtree below Δ does not exist. Now, consider the tree Π that results when Δ is replaced by the single node y (see Figure 3.1-(d) to -(e)). We call the set of all such trees Π as the *tree set* of (x,y), or $T(x,y)$. In general, the frontier of Π consists of 4 terminal substrings, labeled u_1 to u_4 in the figure. The *size* of Π is defined as $\sum_{i=1}^{4} |u_i|$.

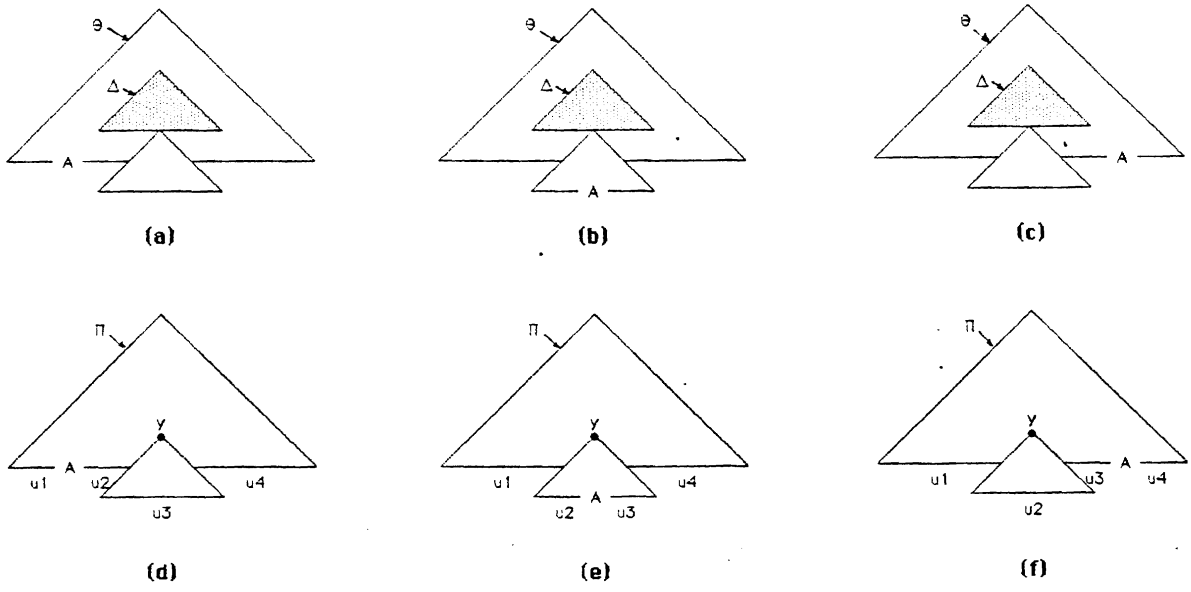

Figure 3.1. Forms of trees in the tree set of (x,y).

Recall that every tree Θ in $T(x)$ is derived via one or more sequence of adjunctions starting with the subtree rooted at x. We now introduce the notion of a *derivation tree* which allows us to represent a sequence of adjunctions corresponding to a tree in $T(x)$.

Definition 3.1. Let x be a node of some elementary tree Γ. A *derivation tree for* x is defined inductively as follows:

- If x is a leaf node (of Γ), then the tree D consisting of the single node x is a derivation tree for x.
- If x is an internal node with an OA constraint, x_1 is its only child, and x_2 is the root of an auxiliary tree adjoinable at x, let D_1 and D_2 be derivation trees for x_1 and x_2, respectively. Then the tree D with root x, left subtree D_1 and right subtree D_2 is a derivation tree for x.
- If x is an internal node with an NA constraint, left child x_1, and right child x_2, let D_1 and D_2 be derivation trees for x_1 and x_2, respectively. Then the tree D with root x, left subtree D_1 and right subtree D_2 is a derivation tree for x.

We let $D(x)$ denote the set of all derivation trees for x.

A derivation tree for x specifies a sequence of adjunctions that results in some tree Θ in $T(x)$. Procedure *CONVERT* below returns this tree Θ:

```
procedure CONVERT(D);
/* Returns the tree Θ corresponding to derivation tree D. */
begin
    if D is a single node tree then
        Θ ← D
    else
        let x be the root of D;
        Θ1 ← CONVERT(left subtree of x);
        Θ2 ← CONVERT(right subtree of x);
        if x has an OA constraint then
            let Δ be the tree with root x and the single subtree Θ1 below it;
            Θ ← ADJ(Δ,x,Θ2)
        else /* x has an NA constraint */
            Θ ← tree with root x, left subtree Θ1 and right subtree Θ2;
        endif;
    endif;
    return(Θ);
end CONVERT.
```

We leave it to the reader to verify that if $D \in D(x)$ then $CONVERT(D) \in T(x)$, and if $\Theta \in T(x)$ then there is at least one $D \in D(x)$ such that $\Theta = CONVERT(D)$. The *size* of D, denoted $|D|$, is defined as the size of the corresponding tree $CONVERT(D)$.

Definition 3.2. Let (x,y) be an ordered pair of nodes. Let $D_1 \in D(x)$ be a derivation tree for x containing a subtree D_2 with root node y (hence, $D_2 \in D(y)$). Then the tree obtained by replacing D_2 in D_1 by the single node y is called a *derivation tree for* (x,y). The set of all such derivation trees is denoted

$D(x,y)$.

As before, one can verify that if $D' \in D(x,y)$ then $CONVERT(D') \in T(x,y)$, and if $\Theta' \in T(x,y)$ then there is at least one $D' \in D(x,y)$ such that $\Theta' = CONVERT(D')$. $|D'|$ is defined as the size of the corresponding tree $CONVERT(D')$.

In the following discussion, we consider a fixed terminal string $\alpha = a_1 a_2 \cdots a_n$, $a_i \in V_T$ and $n > 0$, which is to be recognized. For convenience, we use α_{ij} to denote the substring $a_{i+1} \cdots a_j$. In particular, $\alpha_{ii} = \varepsilon$.

Definition 3.3. Let α be as above and let x and y be nodes.

(1) A derivation tree $D \in D(x)$ is *valid* for the pair of substrings $(\alpha_{ij},\alpha_{kl})$ iff $CONVERT(D)$ has a frontier $\alpha_{ij} A \alpha_{kl}$.

(2) A derivation tree D' in $D(x,y)$ is *valid* for the 4-tuple of substrings $(\alpha_{ij},\alpha_{kl},\alpha_{pq},\alpha_{rs})$ iff $CONVERT(D')$ has the frontier shown in Figure 3.2 (depending on the form of $CONVERT(D')$).

Part (2) of the definition is intended to capture the fact that if the node y in $CONVERT(D') \in T(x,y)$ is replaced by a tree in $T(y)$ whose frontier is $\alpha_{pq} B \alpha_{rs}$, the result is a tree in $T(x)$ whose frontier is $\alpha_{ij} A \alpha_{kl}$. Put another way, if the leaf node y in $D' \in D(x,y)$ is replaced by a derivation tree in $D(y)$ which is valid for $(\alpha_{pq},\alpha_{rs})$, then the result is a tree in $D(x)$ which is valid for $(\alpha_{ij},\alpha_{kl})$.

Definition 3.4. Let $n > 0$ be an integer. An *item* is a tuple of the form (x,i,j,k,l) where x is a node and $0 \le i \le j \le k \le l \le n$. The *size* of the item is $(j-i) + (l-k)$. A *pair of items* is a tuple of the form (I_1,I_2), where I_1 and I_2 are items. The *size* of (I_1,I_2) is size(I_1) - size(I_2).

Definition 3.5. Let α be the terminal string to be recognized.

(1) An item (x,i,j,k,l) is *realizable* iff there is a derivation tree in $D(x)$ which is valid for $(\alpha_{ij},\alpha_{kl})$.

(2) A pair of items $((x,i,j,k,l),(y,p,q,r,s))$ is *realizable* iff there is a derivation tree in $D(x,y)$ which is valid for $(\alpha_{ij},\alpha_{kl},\alpha_{pq},\alpha_{rs})$.

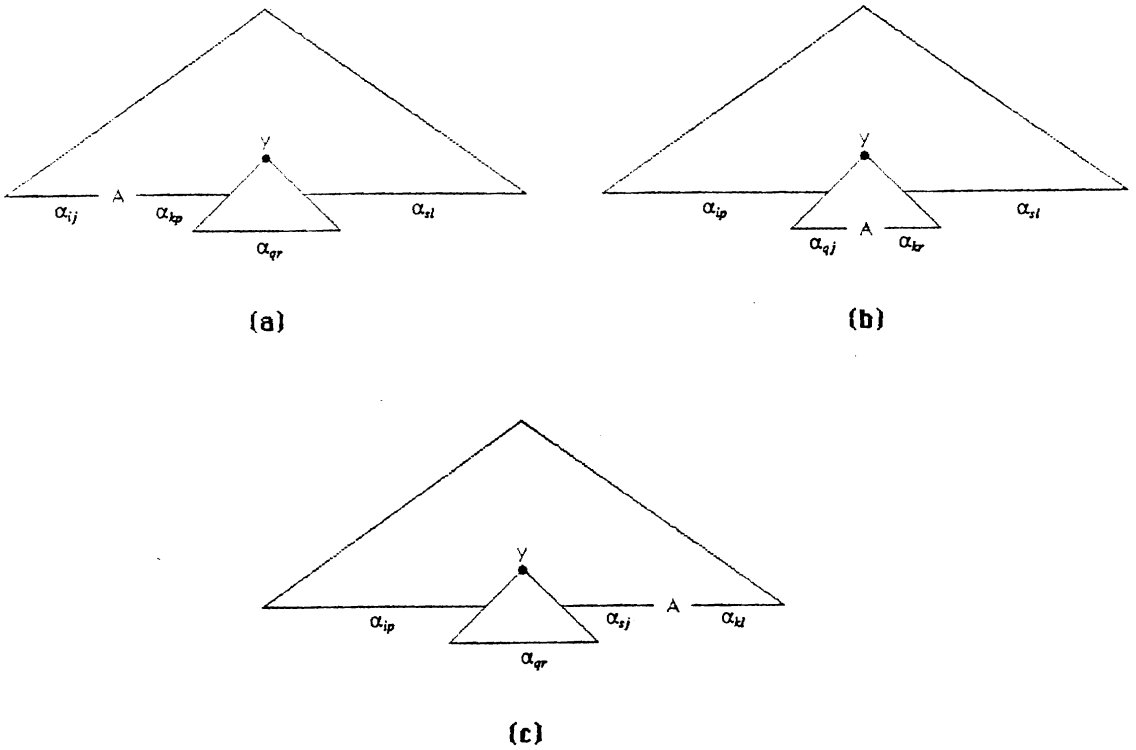

Figure 3.2. The frontier of $CONVERT(D')$, where $D' \in D(x,y)$ is valid for $(\alpha_{ij},\alpha_{kl},\alpha_{pq},\alpha_{rs})$.

Thus, if $|\alpha| = n$, then $\alpha \in L(G)$ iff for some $0 \leq j \leq n$, $(x_0,0,j,j,n)$ is realizable, where x_0 is the root node of the initial tree of G.

Definition 3.6. Let I_x, I_y and I_z be items. We write I_y, I_z :- I_x and I_z, I_y :- I_x iff one of the following conditions holds:

- $I_x = (x,i,j,k,l)$, $I_y = (y,m,j,k,p)$, $I_z = (z,i,m,p,l)$, $i \leq m \leq j \leq k \leq p \leq l$, x is a node with an OA constraint, y is its only child, and z is the root of an auxiliary tree adjoinable at x.
- $I_x = (x,i,j,k,l)$, $I_y = (y,i,m,m,p)$, $I_z = (z,p,j,k,l)$, $i \leq m \leq p \leq j \leq k \leq l$, x is a node with an NA constraint, y and z are the left and right children of x, respectively, and y is a closed node.
- $I_x = (x,i,j,k,l)$, $I_y = (y,i,j,k,m)$, $I_z = (z,m,p,p,l)$, $i \leq j \leq k \leq m \leq p \leq l$, x is a node with an NA constraint, y and z are the left and right children of x, respectively, and z is a closed node.

Let R be the set of all realizable items and realizable pairs of items. The next theorem follows from the definitions.

Theorem 3.1. R is the least set satisfying the following conditions:

(0) For each leaf node x of an elementary tree,

(a) if $label(x) \in (V_N \cup \varepsilon)$ then $(x,i,i,i,i) \in R$ for $0 \leq i \leq n$;

(b) if $label(x) = \alpha_{i,i+1}$ then $(x,i,i,i,i+1) \in R$ and $(x,i,i+1,i+1,i+1) \in R$;

(1) If $I_z \in R$ and I_y, I_z :- I_x then $(I_x,I_y) \in R$;

(2) If $(I_x,I_y) \in R$ and $(I_y,I_z) \in R$, then $(I_x,I_z) \in R$;

(3) If $(I_x,I_y) \in R$ and $I_y \in R$, then $I_x \in R$.

Thus, we can decide if $\alpha \in L(G)$ using the algorithm below:

Algorithm A:

```
begin
    R ← set obtained by applying rule (0) of Theorem 3.1;
    repeat
        R' ← R;
        apply rules (1) - (3) of Theorem 3.1 to R;
    until (R' = R);
    if (x0,0,j,j,n) ∈ R for some 0 ≤ j ≤ n then
        accept
    else
        reject
    endif;
end.
```

We now show that Algorithm A terminates after $O(\log(n))$ iterations of the **repeat** loop. The proof employs a "tree-cutting" technique similar to those used in [RYTT85, RYTT87, RUZZ80]. For a realizable item I_x, define level(I_x) = k iff I_x first becomes a member of R at the k-th iteration of the **repeat** loop. Similarly, for level((I_x,I_y)), where (I_x,I_y) is a realizable pair of items. Let $L_1(n) = \max\{\text{level}(I_x) \mid \text{size}(I_x) \leq n\}$ and $L_2(n) = \max\{\text{level}((I_x,I_y)) \mid \text{size}((I_x,I_y)) \leq n\}$.

Theorem 3.2. For $n \geq 2$,

(1) $L_1(n) \leq \max\{L_1(2n/3), L_2(2n/3)\} + 1$;

(2) $L_2(n) \leq \max\{L_1(2n/3), L_2(2n/3)\} + 3$.

Proof. Consider a realizable item $I_x = (x,-,-,-,-)$ of size $n \geq 2$. Then, there is a derivation tree D_x for x such that $|D_x| = n$. Moreover, it is easy to prove that there is an internal node $y \neq x$ in D_x such that $n/3 \leq |D_y|$, $|D_{x,y}| \leq 2n/3$, where D_y is the subtree rooted at y and $D_{x,y}$ is D_x with D_y replaced by the single node y. Clearly, D_y and $D_{x,y}$ represent some realizable I_y and (I_x,I_y) with levels at most $L_1(2n/3)$ and $L_2(2n/3)$, respectively. Part (1) of the theorem then follows from the fact that I_x becomes a member of R by applying rule (3) of Theorem 3.1 to (I_x,I_y) and I_y.

Consider a realizable pair of items (I_x,I_y) of size $n \geq 2$, where $I_x = (x,-,-,-,-)$ and $I_y = (y,-,-,-,-)$. Then, there is a derivation tree $D_{x,y}$ for (x,y) such that $|D_{x,y}| = n$. Consider the path from node y to node x in $D_{x,y}$ and let y_1 be the first node in this path such that $|D_{y_1,y}| > n/2$, where $D_{y_1,y}$ is the subtree rooted at y_1. Thus, $|D_{x,y_1}| \leq n/2$, where D_{x,y_1} is $D_{x,y}$ with $D_{y_1,y}$ replaced by the single node y_1. (See Figure 3.3).

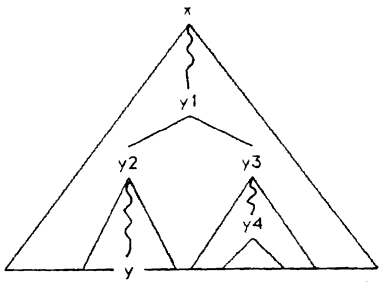

Figure 3.3. Derivation tree $D_{x,y}$.

Let y_2 and y_3 be the children of y_1, where y_2 is the child having y as a descendant. Let $D_{y_2,y}$ and D_{y_3} be the subtrees rooted at y_2 and y_3, respectively. Clearly, $|D_{y_2,y}| \leq n/2$. D_{y_3}, on the other hand, can have size $m > 2n/3$; however, from the first part of the proof, a node y_4 can be found such that $|D_{y_4}|$, $|D_{y_3,y_4}| \leq 2m/3 \leq 2n/3$, where D_{y_4} is the subtree rooted at y_4 and D_{y_3,y_4} is D_{y_3} with D_{y_4} replaced by the

single node y_4.

Clearly, the trees D_{x,y_1}, $D_{y_2,y}$, D_{y_3,y_4} and D_{y_3} represent some (I_x,I_{y_1}), (I_{y_2},I_y), (I_{y_3},I_{y_4}) and $I_{y_3} \in R$, respectively, all of size $\leq 2n/3$. Hence, all have levels at most $k = \max\{L_1(2n/3), L_2(2n/3)\}$. Moreover, (I_x,I_y) can be derived from these elements by applying the following rules of Theorem 3.1:

(a) Apply rule (2) to (I_{y_3},I_{y_4}) and I_{y_4} to obtain I_{y_3};

(b) Apply rule (1) to I_{y_3} and I_{y_2}, I_{y_3} :- I_{y_1} to obtain (I_{y_1},I_{y_2});

(c) Apply rule (2) to (I_{y_1},I_{y_2}) and (I_{y_2},y) to obtain (I_{y_1},I_y);

(d) Apply rule (2) to (I_x,I_{y_1}) and (I_{y_1},I_y) to obtain (I_x,I_y).

Thus, (I_x,I_y) would be in R after the $(k + 3)$-th iteration (steps (b) and (c) can be completed during the same iteration). Part (2) of the theorem then follows. □

Corollary 3.1. For $n \geq 1$, $L_1(n), L_2(n) \leq c \log(n) + 4$, where $c = 3/\log(3/2)$.

Proof. One can show that $L_1(1), L_2(1) \leq 4$. The rest of the proof is straightforward induction on n and is left to the reader. □

Since all realizable items and realizable pairs of items have size at most n, it follows that Algorithm A terminates after $c \log(n) + 4$ iterations of the **repeat** loop.

4. The Parallel Algorithm

Algorithm A can be easily implemented in parallel on a PRAM that allows both concurrent reads and concurrent writes. Every item (x,i,j,k,l) and every pair of items $((x,i,j,k,l),(y,p,q,r,s))$ is assigned a cell in global memory. Initially, all cells are set to 0; membership in R would be indicated by writing 1 on the cell. The PRAM executes as in Algortihm A, except that a rule is applied in parallel to all items and pairs of items.

The initialization step (application of rule (0)) can be executed in constant time using $O(n)$ processors. For one iteration of the **repeat** loop, rules (1) - (3) can each be applied in constant time using $O(n^{12})$

processors by considering all possible triples (I_x, I_y, I_z) and assigning one processor to each triple. Finally, acceptance can be checked in constant time using $O(n)$ processors. Thus, the PRAM runs in time $O(\log(n))$ and uses $O(n^{12})$ processors.

Note that concurrent writes (as well as concurrent reads) may happen since an item or a pair of items can be in R by applying the same rule during the same iteration to different sets of elements. Using well-known techniques (see, e.g., [KUCE82]), the parallel algorithm can be made to run on a CREW PRAM in $O(\log^2(n))$ time and the same number of processors.

References

[AHO72] Aho, A. V. and J. D. Ullman, *Theory of Parsing, Translation and Compiling, Volume 1: Parsing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[CHAN87] Chang, J. H., O. H. Ibarra and M. A. Palis, Parallel parsing on a one-way array of finite-state machines, *IEEE Trans. Comput.*, C36:1 (Jan. 1987), pp. 64-75; extended abstract also appeared in *Proc. 1986 Intl. Conf. Parallel Proc.*, St. Charles, Illinois, pp. 887-894.

[CHIA84] Chiang, Y. and K. Fu, Parallel parsing and VLSI implementations for syntactic pattern recognition, *IEEE Trans. Patt. Anal. and Mach. Int.* 6:3 (1984), pp. 302-313.

[JOSH75] Joshi, A. K., L. S. Levy and M. Takahashi, Tree adjunct grammars, *J. Comp. Syst. Sci.*, 10 (1975), pp. 136-163.

[KOSA75] Kosaraju, S. R., Speed of recognition of context-free languages by array automata, *SIAM J. Comput.*, 4:3 (1975), pp. 331-340.

[KROC85] Kroch, A. S. and A. K. Joshi, The linguistic relevance of tree adjoining grammar, *Tech. Rpt. MS-CIS-85-16, Dept. of Comp. and Info. Sci.*, University of Pennsylvania, June 1985.

[KUCE82] Kucera, L., Parallel computation and conflicts in memory access, *Inf. Proc. Lett.*, 14:2 (1982), pp. 93-96.

[PALI87] Palis, M. A., S. Shende and D. Wei, An optimal linear-time parallel parser for tree adjoining languages, *Tech. Rpt. MS-CIS-87-50, Dept. of Comp. and Info. Sci., University of Pennsylvania*, June 1987.

[RUZZ80] Ruzzo, W. L., Tree-size bounded alternation, *J. Comput. System Sci.*, 21, pp. 218-235.

[RYTT85] Rytter, W., The complexity of two-way pushdown automata and recursive programs, in *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil (eds.), NATO ASI Series F:12, Springer-Verlag, New York/Berlin.

[RYTT87] Rytter, W., Parallel $O(\log n)$ recognition of unambiguous context-free languages, *Info. and Comp.*, 73 (1987), pp. 75-86.

[VIJA85] Vijayashanker, K. and A. K. Joshi, Some computational properties of tree adjoining grammars, *Proc. 23rd Ann. Meet. Assoc. Comp. Ling.*, Chicago, Illinois, July 1985, pp. 82-93.

[WEIR87] Weir, D. J., From context-free grammars to tree adjoining grammars and beyond, *Tech. Rpt. MS-CIS-87-42, Dept. of Comp. and Info. Sci., University of Pennsylvania*, May 1987.