INFORMATION AND COMPUTATION 73, 75–86 (1987)

# Parallel Time $O(\log n)$ Recognition of Unambiguous Context-free Languages

WOJCIECH RYTTER*

*Institute of Informatics, Warsaw University, 00-901 Warszawa, PKiN VIII p, P.O. Box 1210, Poland*

We prove that every unambiguous context-free language can be recognized in $O(\log n)$ time on a parallel random access machine without write conflicts (P-RAM) using a polynomial number of processors. This strengthens the result of Reif (1982, *in* "Proceedings, 23rd IEEE Symp. Found. Comput.," pp. 114–118), who proved that every deterministic context-free language can be recognized on a P-RAM in $O(\log n)$ time. 

## 1. INTRODUCTION

The parallel random access machine (P-RAM) was defined by Fortune and Wyllie (1978). It consists of a collection of synchronous deterministic unit-cost RAMs with shared memory locations indexed by natural numbers (or tuples of numbers). Simultaneous reads are allowed: on each step, any given memory location can be read simultaneously by any number of processors. However, simultaneous writes are not allowed; no two distinct processors can attempt to write into the same memory location on the same step. We denote by W-RAM the parallel random access machine which allows resolution of both read and write conflicts. However, we assume that write conflicts are weak: if two distinct processors attempt to write simultaneously into the same location then they attempt to write the same value. W-RAM is our auxiliary model. Ruzzo (1980) gave an alternating machine algorithm for the recognition of cfl's in time $O(\log^2 n)$ and polynomial tree size. This algorithm can be simulated in logarithmic time on a W-RAM. Ruzzo's construction was simplified by Rytter (1985). However, the resulting algorithms require $\Omega(\log^2 n)$ time on the P-RAM. It is a hard open question whether general context-free recognition can be done in logarithmic time on a P-RAM. Reif (1982) has proved that such recognition can be done for deterministic cfl's, using an algorithm whose correctness is hard to prove. We extend this result to unambiguous cfl's.

* A Preliminary version of this paper was presented at the Conference of Fund. of Comput. Theory, Cottbus, 1985.



0890-5401/87 $3.00

The action of the parallel instruction: *for each* $a \in S$ *parallel do* instruction($a$) consists of:

(1) assigning a processor to each $a \in S$; each assigned processor has access to its value of $a$;

(2) executing simultaneously each instruction(a).

(The set $S$ can be characterized by a condition.)

We demonstrate the use of this instruction in the following example:

$$\textit{for each } i \in [1 \cdots n] \textit{ parallel do if } x_i \text{ then result} := \text{true};$$

assume that initially the value of result is false. Then the instruction in one step computes the boolean or of $n$ variables on W-RAM.

Assume now that the input has the special property: at most one variable has value true. Now our instruction computes the same function on a P-RAM. The special property of the input guarantees that there are no write conflicts.

We use a similar approach in the recognition of cfl's. We start with a preliminary algorithm working in $O(\log n)$ time on a W-RAM. The special property of the input is in this case the unambiguity of the grammar. The algorithm makes $O(\log n)$ boolean matrix multiplications. Each of these multiplications can be made in $O(1)$ time on a P-RAM because of the structure of the graphs corresponding to multiplied matrices (implied by the unambiguity of the grammar).

## 2. Preliminaries

Each unambiguous context-free grammar (which does not generate the empty word) can be transformed into an equivalent unambiguous grammar in Chomsky normal form without useless nonterminals (each nonterminal can be reached from the starting nonterminal and each nonterminal generates some terminal string). Hence we can assume that all considered grammars are in Chomsky normal form and have no useless nonterminals.

Let $G = (V_N, V_T, P, S)$, where

$V_N$ is the set of nonterminals,

$V_T$ is the set of terminal symbols,

$P$ is the set of productions, and

$S$ is the starting nonterminal.

We write $A \to u$ iff $A \to u$ is a production, and we write $v \to^* u$ iff $u$ can be derived from $v$ in the grammar $G$.

Let $w = a_1 a_2 \cdots a_n$ be a given input string of length $n$. Let $w[i:j]$ denote the substring $a_{i+1} \cdots a_j$ for $0 \leqslant i < j \leqslant n$, and $w[i:i]$ denote the empty word. Define the set $N = \{(A, i, j) : A \in V_N,\ 0 \leqslant i < j \leqslant n\}$.

The elements of $N$ are called nodes. Each node $(A, i, j)$ can be interpreted as a syntactic tree (disregarding internal labels) with the root $A$ and leaves $w[i:j]$.

We say that $(A, i, j)$ is realizable iff there is a derivation $A \to {}^*w[i:j]$. We say that a pair of nodes $((A, i, j), (B, k, l))$ is realiable iff there is a derivation $A \to {}^*w[i:k]\, Bw[l:j]$, $i \leqslant k < l \leqslant j$ and $(A, i, j) \neq (B, k, l)$.

(The pair of nodes $(x, y)$ can be interpreted as a syntactic tree $x$ with the gap $y$.)

We write $y, z \vdash x$ and $z, y \vdash x$ iff $x, y, z$ are of the form $x = (A, i, j)$, $y = (B, i, k)$, $z = (C, k, j)$ and $A \to BC$.

Define a directed graph $\bigcup(G, w) = (V, E)$, where $V = N$ and $(x, y) \in E$ iff for some realizable node $z$ we have $y, z \vdash x$ (or equivalently $z, y \vdash x$).

*Remark.* A pair of nodes $(x, y)$ is realizable iff there is a path in $\bigcup(G, w)$ of nonzero length from $x$ to $y$.

EXAMPLE. Let us consider the following grammar $G$:

$$S \to SS, \qquad S \to AB, \qquad B \to SB, \qquad B \to b, \qquad A \to a.$$

The nonterminal symbols are $S$, $A$, $B$, and terminal symbols are $a$, $b$. Let $w = aabbb = a_1 a_2 a_3 a_4 a_5$.

Realizable nodes are: $(A, 0, 1)$, $(A, 1, 2)$, $(B, 2, 3)$, $(B, 3, 4)$, $(B, 4, 5)$, $(S, 1, 3)$, $(S, 0, 4)$, $(B, 1, 4)$, $(B, 0, 5)$.

Figure 1 presents the subgraph of $\bigcup(G, w)$ spanned by the set of nodes reachable from $(S, 0, 5)$. The input string $w$ can be derived from $S$ iff the node $(S, 0, 5)$ is realizable. In the graph there is an edge from $(S, 0, 5)$ to $(B, 1, 5)$ because $(A, 0, 1)$, $(B, 1, 5) \vdash (S, 0, 5)$, and $(A, 0, 1)$ is realizable.

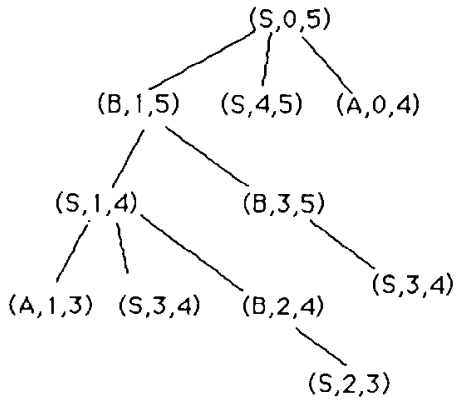


FIGURE 1

There is an edge from $(B, 1, 5)$ to $(B, 3, 5)$ because $(S, 1, 3)$, $(B, 3, 5) \vdash (B, 1, 5)$, and $(S, 1, 3)$ is realizable.

We can see that there is a path from $(S, 0, 5)$ to $(S, 2, 3)$. This implies that the pair $((S, 0, 5), (S, 2, 3))$ is realizable. This means that there is a derivation $S \to {}^*w[0:2]\, Sw[3:5]$. The pair $((S, 0, 5), (S, 2, 3))$ can be interpreted as a tree with a gap, see Fig. 2. The gap corresponds to the tree deriving $a_3 = w[2:3]$ from $S$.

LEMMA 1 (key lemma). *If the grammar $G$ is unambiguous then for every two nodes $x$, $y$ there is at most one path from $x$ to $y$ in the graph $\bigcup(G, w)$.*

*Proof.* If there are two different paths in the graph $\bigcup(G, w)$ from the node $(A, i, j)$ to the node $(B, k, l)$ then there are two different derivations of $w[i:k]\, Bw[l:j]$ from $A$ in the grammar. However, this is impossible in the unambiguous grammar without useless nonterminals, because $B$ can be replaced by some terminal string derived from $B$ (not necessarily a substring of $w$). This completes the proof. ∎

Let $R$ be the set of all realizable nodes and realizable pairs of nodes. The next lemma follows from the definitions.

LEMMA 2. *$R$ is the least set satisfying the conditions*:

(0) *for each $A \in V_N$, $0 \leqslant i \leqslant n$, if $A \to a_{i+1}$ then $(A, i, i+1) \in R$;*

(1) *if $z \in R$ and $y, z \vdash x$ then $(x, y) \in R$;*

(2) *if $(x, y), (y, z) \in R$ then $(x, z) \in R$;*

(3) *if $(x, y) \in R$ and $y \in R$ then $x \in R$;*

*for each $x, y, z \in R$.*

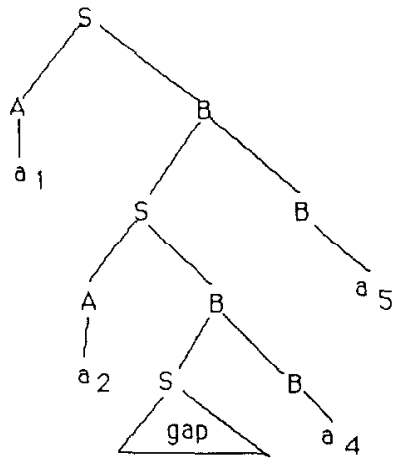


FIGURE 2

*Remark.* We could add another rule: if $y, z \in R$ and $y, z \vdash x$ then $x \in R$. However, such a rule is redundant. If $y, z \in R$ and $y, z \vdash x$ then $(x, y) \in R$ because of rule (1) from Lemma 2. Now we have $(x, y) \in R$ and $y \in R$ and rule (3) implies that $x \in R$.

We say that an element $x \in R$ is initial iff it belongs to $R$ because of rule (0). Every other element of $R$ can be derived (generated) from initial elements using rules (1)–(3) some number of times. For each $u \in R$ we define level($u$) to be the minimal height of a tree deriving $u$ according to rules (1)–(3) from the initial elements (proving that $u$ is realizable). More formally:

$$\text{if } u \text{ is initial then } \text{level}(u) = 0.$$

Assume now that $u$ is not initial; $u$ can be a node or a pair of nodes. If $u = x \in N$ and $x \in R$ then the last applied rule to prove that $x$ is realizable was rule (3); $u$ was derived from some realizable elements of the form $(x, y)$ and $y$. We define $\text{level}(x) = \min(\max(\text{level}(x, y), \text{level}(y)) + 1)$, where the minimum is over all such $(x, y)$, $y$.

If $u$ is a pair $(x, y)$ of nodes and $u$ is realizable then the last applied rule was (1) or (2). (Observe that only one of them could be applied, due to the unambiguity.)

If $(x, y)$ was derived because of (the last applied) rule (1) then there was some realizable node $z$ such that $y, z \vdash x$. We define $\text{level}(x, y) = \min(\text{level}(z) + 1)$, where the minimum is over all such $z$.

If $(x, y)$ was derived because of rule (2), then there were some realizable pairs $(x, z)$, $(z, y)$. We define in this case $\text{level}(x, y) = \min(\max(\text{level}(x, z), \text{level}(z, y)) + 1)$, where the minimum is taken here over all such $(x, z)$, $(z, y)$.

EXAMPLE (continued). The element $((S, 0, 5), (S, 2, 3))$ corresponds to the syntactic tree with all leaves terminal but one (see Fig. 2). The height of this tree is 5. However the height of the tree deriving this element from initial elements is 3 (see Fig. 3). Hence $\text{level}((S, 0, 5), (S, 2, 3)) \leqslant 3$. In fact, there is no tree with a smaller height in this case.

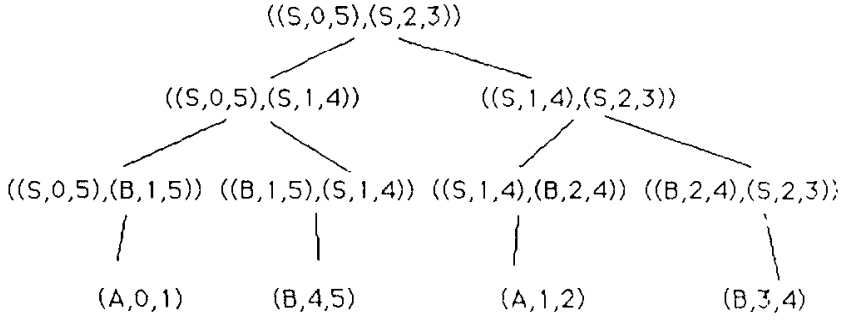


FIGURE 3

LEMMA 3. *There exists a constant $c$ such that for all $u \in R$* $\text{level}(u) \leqslant c \log(n)$.

*Proof.* We define the function $\text{size}(u)$, such that $\text{size}(u) = 1$ if $u$ is an initial element, and $\text{size}(u) \leqslant n$ for every $u$. It is enough to prove that each noninitial realizable element $u$ can be derived from realizable elements of the sizes bounded by $c_1\ \text{size}(u)$, using a constant number of rules (1)–(3), where $c_1 < 1$.

Eeach element of $R$ corresponds to a syntactic tree. We define the size of the element to be the number of terminal leaves in this tree:

If $u = (A, i, j)$ then $\text{size}(u) = j - i$,

if $u = ((A, i, j), (B, k, l))$ then $\text{size}(u) = j - i - (l - k)$.

We use tree cutting techniques. We describe below how syntactic trees can be decomposed into much smaller subtrees. There are two cases:

(1) Syntactic trees without gaps. Consider a realizable node $(A, i, j)$ and a syntactic tree $T$ of the derivation $A \to {}^*w[i : j]$. Let $m = j - i$. There is a node $y$ of $T$ (corresponding to some realizable node $(B, k, l)$) such that: the subtree $T_1$ of $T$ rooted at $y$ has at most $\lceil \frac{2}{3} m \rceil$ leaves, and the subtree $T_2$ resulting from $T$ by replacing $T_1$ by a nonterminal leaf has also at most $\lceil \frac{2}{3} m \rceil$ leaves.

Hence $(A, i, j)$ can be derived from $(B, k, l)$ and $((A, i, j), (B, k, l))$ and both these elements have their size bounded by $\lceil \frac{2}{3}\ \text{size}(A, i, j) \rceil$.

(2) Syntactic trees with gaps. Consider a realizable pair of nodes $(x, y)$, where $x = (A, i, j)$ and $y = (B, k, l)$. Let $T$ be the tree corresponding to the derivation $A \to {}^*w[i : k]\ Bw[l : j]$. Each vertex of this tree can be treated as an element of $N$, for example, the root can be identified with $(A, i, j)$.

We consider the path from the leaf corresponding to $(B, k, l) = y$, to the root $x$ (see Fig. 4). Let $y1$ be the first node on this path (going bottom-up) of a size bigger than $\text{size}(x, y)/2$. Hence the size of the subtree rooted at $y1$

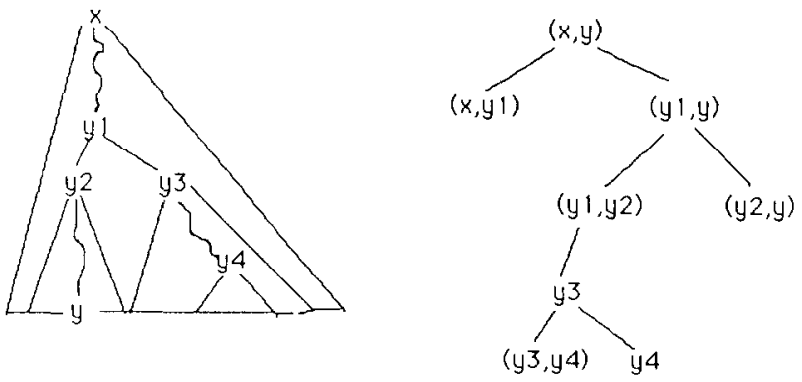


FIGURE 4

is bigger than size($T$)/2, where by the size of the tree we mean the number of its terminal leaves.

Consider the immediate successors $y2$, $y3$ of $y1$, where $y2$ lies on the path from $y1$ to $y$. Let $T_1$, $T_2$, $T_3$ be subtrees rooted at $y1$, $y2$, $y3$, respectively. Let $T'$ be the tree rooted at the root of $T$ obtained as a result of treating $y1$ as a nonterminal leaf.

We have size($T_2$), size($T'$) $\leqslant$ size($T$)/2. The subtree $T_3$ can be big, however, all its leaves are terminal (it is without a gap) and case (1) applies. A node $y4$ can be found dividing this tree into smaller trees (one of them with the gap $y4$).

All the elements $(x, y1)$, $(y2, y)$, $(y3, y4)$, $y4$ are of a size bounded by $c_1$ size$(x, y)$, where $c_1 < 1$ for sufficiently big $(x, y)$. The element $(x, y)$ can be derived from these elements by applying the rules (1)–(3) 0(1) times (see Fig. 4). This completes the proof. ∎

*Remark.* A similar tree-cutting technique was used by Rytter (1985) in analyzing the operation bush, acting on path systems; bush($P$) was defined there as a bushy version of the path system $P$.

## 3. The Algorithm

The computation of $R$ can be described informally as follows:

compute the set of initial nodes using rule (0);
*repeat* $c \log(n)$ *times* {$c$ is a constant from Lemma 3}
  apply in one step rules (1)–(3) wherever it is possible;
*if* $(S, 0, n)$ is generated *then* accept.

We assume the following terminology. Inserting a node $x \in N$ into $R$ is equivalent to pebbling this node. We introduce the logical tables COND, pebbled, and EDGE. Pebbling a node $x$ consists of executing pebbled $(x) :=$ true. The node is pebbled if we already know that it is realizable. At this moment the table EDGE is redundant, later it will play a crucial role to eliminate write conflicts. Assume that initially all tables contain the values false; $c$ is a constant from Lemma 3.

ALGORITHM {preliminary version}

*begin*
0: *for each* $0 \leqslant i < n$, $A \in V_N$ such that $A \to a_{i+1}$ *parallel do*
  pebble node $(A, i, i+1)$;
*repeat* $c \log(n)$ *times*
*begin* {invariants}

1: *for each* $x, y, z$ *such that* $y, z \vdash x$ and pebbled($z$) *parallel do*
*begin* EDGE($x, y$) := true; COND($x, y$) := true *end*;
{invariants}
2: *for each* $x, z, y$ such that COND($x, z$) *and* COND($z, y$) *parallel do*
COND($x, y$) := true;
{invariants}
3: *for each* $x, y$ such that COND($x, y$) and pebbled($y$) *parallel do*
pebble node $x$
*end*;
if the node $(S, 0, n)$ is pebbled then ACCEPT
*end.*

Instructions 0–3 correspond to the rules in Lemma 2. The input string is generated by the grammar iff $(S, 0, n)$ is realizable. The correctness of the algorithm follows from Lemma 2 and 3. However, the algorithm works on a W-RAM and we have to eliminate write conflicts. We shall achieve this using some invariants of the algorithm.

There are no write conflicts in instruction 0. We claim that also there are no write conflicts in instruction 1. This follows from the following invariant of the algorithm:

(A1) if EDGE($x, y$) then there is exactly one pebbled node $z \neq y$ such that

$$y, z \vdash x.$$

This follows directly from the unambiguity of the grammar and the definition of the relation.

Observe that if COND is viewed as an $n$-by-$n$ boolean matrix, then instruction 2 is equivalent to COND := COND $\vee$ COND $\cdot$ COND. However, in general we cannot multiply boolean matrices in $O(1)$ time without write conflicts.

Let $H = (V, E)$ be the directed graph such that $V = N$ and $E = \{(x, y) : \text{EDGE}(x, y)\}$.

Observe that $H$ is a subgraph of $\bigcup(G, w)$. We write $x \Rightarrow y$ iff EDGE $(x, y)$, and let $\Rightarrow^*$ be the transitive closure of $\Rightarrow$.

If $W \subseteq N$ then we say that a node $y \in W$ is maximal in $W$ iff there is no $x \in W$ such that EDGE($x, y$).

LEMMA 4. *The following are invariants of the algorithm*:

(A2) (*key invariant*) *for every two nodes* $x, y$ *there is at most one path from* $x$ *to* $y$ *in the graph* $H$;

(A3) *if* COND($x, y$) *then* $x \Rightarrow^* y$;

(A4) *if* $x \Rightarrow^* z \Rightarrow^* y$ *and* COND$(x, y)$ *then* COND$(x, z)$ *and* COND$(z, y)$;

(A5) *if* COND$(x, z)$ *and* COND$(z, y)$ *and not* COND$(x, y)$ *then there is exactly one pair of nodes* $y1, y2$ *satisfying the condition*:

$c1(x, y1, y2, y)$: COND$(x, y1)$ *and* COND$(y1, y)$ *and* (*not* COND$(x, y2)$) *and* (COND$(y2, y)$ *or* $y2 = y$) *and* EDGE$(y1, y2)$;

(A6) *if* $x \Rightarrow y1$, $x \Rightarrow y2$, $y1 \neq y2$, *and there are paths from* $y1$, $y2$ *to pebbled nodes then* $y1$, $y2$ *are pebbled*;

(A7) *for every* $x \in N$ *the set* $\{y: x \Rightarrow^* y$ *and pebbled*$(y)\}$ *has at most two maximal elements. If it has two maximal elements then they are brothers* (*sons of the same father*).

*Proof.* (A2) $H$ is a subgraph of $\bigcup(G, w)$, hence (A2) follows from Lemma 1.

(A3) The table COND is changed in instructions 1 and 2. Whenever we set COND$(x, y) :=$ true in instruction 1 then we set also EDGE$(x, y) :=$ true, hence $x \Rightarrow^* y$. If the invariant holds before instruction 2 then it holds also after executing this instruction, because COND$(x, y)$ and COND$(y, z)$ imply $x \Rightarrow^* y \Rightarrow^* z$ *and* $x \Rightarrow^* z$.

(A4) The proof is by induction on the number of iterations. When COND$(x, y)$ is set to true in instruction 2 then there must be a $u$ with COND$(x, u)$ and COND$(u, y)$. Since there is only one path from $x$ to $y$, $z$ and $u$ both lie on this path, say $x \Rightarrow^* z \Rightarrow^* u \Rightarrow^* y$.

By the induction hypothesis we therefore had COND$(x, z)$ and COND$(z, u)$ at the end of the previous iteration. Hence COND$(x, z)$ and COND$(z, y)$ are at the end of this iteration.

(A5) We have $x \Rightarrow^* z \Rightarrow^* y$. From (A4) we know that all nodes $u$ before $z$ satisfy COND$(x, u)$, and all nodes $u$ after $z$ (on the path from $x$ to $y$) satisfy COND$(u, y)$. We also know from (A4) that all the nodes $u$ with COND$(x, u)$ form a contiguous segment of the path. Hence $y1$ and $y2$ as defined exist and are the unique pair of nodes with this property (see Fig. 5).

(A6) If $x \Rightarrow y1$ then there is a pebbled node $y1'$ with $y1, y1' \vdash x$. If $x \Rightarrow y2$ then there is a pebbled node $y2'$ with $y2, y2' \vdash x$. Since there is a path from $y1$ to a pebbled node then $y1$ is realizable, similarly for $y2$. Hence $y2 = y1'$ and $y1 = y2'$ by unambiguity. Thus $y1$, $y2$ are pebbled.

(A7) (1) First we prove that if $x \Rightarrow^* y$ and $x$, $y$ are pebbled then all the nodes on the path from $x$ to $y$ are also pebbled.

Suppose that this is not true. Then there are two pebbled nodes $x$, $y$ and

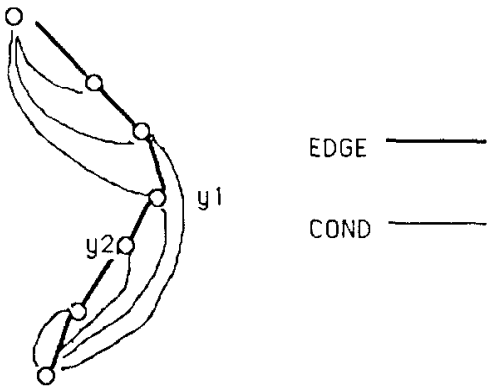


FIG. 5. $c1(x, y^1, y2, y)$.

the path $x \Rightarrow y_1 \Rightarrow y_2 \Rightarrow \cdots \Rightarrow y_k \Rightarrow y$ such that $y_1, y_2, \ldots, y_k$ are not pebbled. We derive a contradiction.

Consider the first moment when the node $x$ was pebbled. It was pebbled in instruction 3 because for some pebbled node $z$ we had COND$(x, z)$. There are two cases:

*Case* 1. One of $y_i$ lies on the path from $x$ to $z$. Then COND$(y_i, z)$ also held because of (A4) and $y_i$ was pebbled when $x$ was pebbled.

*Case* 2. No $y_i$ lies on the path from $x$ to $z$. Take the immediate successor $v$ of $x$ on the path from $x$ to $z$. Now $v$ and $y_1$ are sons of $x$ lying on the paths leading to pebbled nodes. The invariant (A6) implies that both of them are pebbled. Hence $y_1$ is pebbled. This proves our claim.

(2) Let $W = \{y : x \Rightarrow^* y$ and $y$ is pebbled$\}$ and $Y = \{y \in W : y$ is maximal in $W\}$, and let $T$ be the minimal subtree of $H$ containing $x$ and all $y \in Y$.

Assume that $Y$ has more than two elements and let $x1$ be the lowest common ancestor in $T$ of the nodes in $Y$. It follows from what was proved in part (1) that no path connects two nodes in $Y$. Hence $x1$ is not an element of $Y$.

The node $x1$ has at least two sons, so by (A6) all sons of $x1$ are pebbled. It then follows from what was proved in part (1) and from the definition of maximality that every $y \in Y$ must be a son of $x1$. But by the reasoning of the proof of (A6), $x1$ can have only two pebbled sons. These sons are the only elements of $Y$ and they are brothers. This completes the proof. ∎

THEOREM. *Every unambiguous context-free language can be recognized on a P-RAM in* $O(\log n)$ *time using a polynomial number of processors.*

*Proof.* It is enough to eliminate write conflicts from instructions 2, 3 in the algorithm. We apply the invariant (A5) and the condition $c1$ from Lemma 4.

Replace instruction 2 by the following instruction:

2′: *for each* $x$, $y1$, $y2$, $y$ *such that* $c1(x, y1, y2, y)$ *parallel do* COND$(x, y) :=$ true;

The invariant (A5) implies that this instruction is equivalent to instruction 2 and there are no write conflicts. For each $x$, $y$ there is at most one pair $y1$, $y2$ satisfying $c1(x, y1, y2, y)$. Observe that if COND$(x, y)$ already holds then there is no such pair; however, in this case we can omit the execution of COND$(x, y) :=$ true.

Define the condition:

$$c2(x, y1, y) = (\text{(COND}(x, y) \text{ and (COND}(x, y1) \text{ or } y1 = x)$$
$$\text{and EDGE}(y1, y) \text{ and pebbled}(y)$$
$$\text{and (not pebbled}(y1) \text{ and (not pebbled}(x)).$$

If $x$ is not pebbled then $c2$ expresses the fact that $y$ is maximal in the set $\{y\text{: COND}(x, y) \text{ and pebbled}(y)\}$. Moreover, for each $x$, $y$ there is at most one $y1$ with $c2(x, y1, y)$; $y1$ should be the immediate predecessor of $y$ on the path from $x$ to $y$.

Using $c2$ we could eliminate write conflicts such that no more than two processors attempt to write into the same location on the same step, because of (A7). We introduce an auxiliary table ALLOWED to eliminate write conflicts completely.

Let $\ll$ be any linar order on $N$ computable on a RAM in $O(1)$ time. Replace instruction 3 by the following instruction:

3′: *for each* $x$, $y$ *parallel do* ALLOWED$(x, y) :=$ true;
*for each* $x$, $y1$, $y$, $z$ such that $y \ll z$ and $c2(x, y1, y)$ and $c2(x, y1, z)$ *parallel do* {$y$, $z$ are maximal nodes reachable from $x$ and pebbled} ALLOWED$(x, z) :=$ false;
*for each* $x$, $y1$, $y$ such that $c2(x, y1, y)$ and ALLOWED$(x, y)$ *parallel do* pebble node $x$;

The invariant (A7) implies that there are at most two nodes $y$ and $z$ such that for some $y1$, the conditions $c2(x, y1, y)$, $c2(x, y1, z)$, $y \ll z$ hold, for a given $x$. Such nodes should be sons of a uniquely determined node $y1$. The table ALLOWED "eliminates" the second maximal element corresponding to a given node $x$. This proves that there are no write conflicts in instruction 3′ and this instruction is equivalent to instruction 3 (disregarding the table ALLOWED).

After replacing instructions 2, 3 by instructions 2′, 3′, respectively, we obtain the required algorithm. Obviously we are using a polynomial number of processors. This completes the proof. ∎

*Remark.* The number of processors is polynomial; however, the degree of the corresponding polynomial is very high. There are $O(n^2)$ nodes. Hence in instructions 2', 3' we are using $O(n^8)$ processors, implementing these instructions directly. We can lower this number using the following observation: there are $O(n)$ nodes, which can be a son of a fixed node. Observe also that if $c1\ (x, y1, y2, y)$ holds then $y2$ is a son of $y1$, and if $c2(x, y_1, y)$ holds then $y$ is a son of $y1$. Hence only tuples with such properties can be considered in instructions 2' and 3'. Using these two observations, the number of processors can be lowered to $n^7$.

## ACKNOWLEDGMENTS

K. Mehlhorn has made many helpful comments on this paper. We are also grateful to A. M. Gibbons for linguistic help.

RECEIVED October 8, 1985; ACCEPTED September 20, 1986

## REFERENCES

FORTUNE, S., AND WYLLIE, J. (1978), Parallelism in random access machines, *in* "Proceedings, 10th ACM Symp. Theory of Comput.," pp. 114–118.

REIF, J. (1982), Parallel $O(\log n)$ acceptance of deterministic cfl's, *in* "Proceedings, 23rd IEEE Symp. Found. Comput. Sci.," pp. 290–296.

RUZZO, W. L. (1980), Tree-size bounded alternation, *J. Comput. System Sci.* **21**, 218–235.

RYTTER, W. (1985), 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.

*Printed in Belgium*