OBSERVATIONS ON LOG(N) TIME PARALLEL RECOGNITION OF UNAMBIGUOUS CFL'S

PETER ROSSMANITH AND WOJCIECH RYTTER

ABSTRACT. We present two simpler versions of the $\log(n)$ time parallel recognition of unambiguous context-free languages on CREW PRAM's. The first algorithm combines the ideas of the algorithms given in [3] and [7]. The result related to that algorithm is an unambiguous decomposition of trees into smaller ones. The analysis of this algorithm is quite simple. The second algorithm is based on unique paths decomposing trees into smaller subtrees. It has a shorter description and it is simpler to explain how it works. However, its correctness is nontrivial. Both algorithms reduce the numbers of processors by a linear factor in the case of general unambiguous cfl's. For linear unambiguous cfl's the number of processors is reduced to $O(n^2)$ by a very simple algorithm. The paper shows how to remove write conflicts in parallel computations on trees by using two kinds of unique decompositions of trees. Finally we show that our results can be generalized for finitely ambiguous cfl's.

KEYWORDS: Parallel algorithms, unambiguous context free languages

1. INTRODUCTION

Our model of parallel computations is the CREW PRAM, see for example [2] for the definition. Logarithmic time computations in such a model are hard, especially for the recognition of nontrivial context-free languages (cfl's, in short). It was shown in [6] that deterministic cfl's are recognizable in logarithmic time by a very complicated algorithm. This result was strengthened in [7] to the class of unambiguous cfl's, which properly contains deterministic cfl's. However, the corresponding algorithm was also too complicated. Our aim is to give a simpler algorithm. Then we reduce the number of processors by a linear factor. A similar reduction was done for $\log(n)$ recognition of deterministic cfl's in [4], where the number of processors was reduced almost by a linear factor (from n^3 to $n^{2+\varepsilon}$).

This research was supported by DFG-SFB 324 A4.

Authors' addresses: Peter Rossmanith, Institut für Informatik, Technische Universität München, Arcisstr. 21, 8000 München 2, Fed. Rep. of Germany. Wojciech Rytter, Institute of Informatics, Warsaw University, PKiN 8 p. 00901 Warsaw, Poland.

Assume we have a context-free grammar (cfg, in short) in Chomsky normal form generating the language L and an input string w of size n. The size of the grammar is considered to be constant and the total size of the problem is n. We consider the word problem:

check if $x \in L$

Let N be the set of nonterminal and terminal symbols, respectively. Denote by V the set $V = \{ (A,i,j) \mid A \in N, 0 \leq i < j \leq n \}$. The elements of V will be nodes of certain graphs and we refer to them as nodes. The element $x = (A,i,j)$ also corresponds to a possible derivation tree rooted at A and with leaves forming w_{ij}, where $w_{ij} = x_i x_{i+1} \dots x_{j-1}$, if $w = x_1 x_2 \dots x_n$, see Fig. 1.

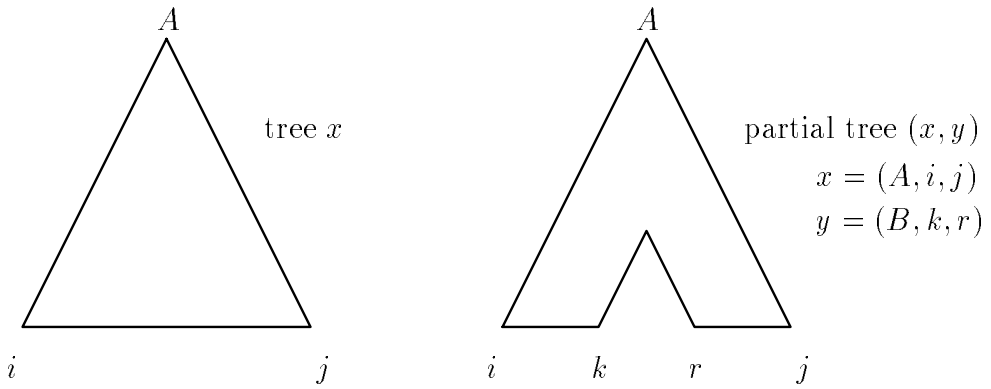


FIGURE 1.

The elements x and (x,y) can be treated as trees and partial trees, respectively, see Fig. 1. In fact they are potential trees, since we do not know at the beginning if the interior of the tree can be filled according to the grammar. We say that (A,i,j) is realizable iff there is a derivation $A \stackrel{*}{\Rightarrow} w_{ij}$. The pair (partial tree) $((A,i,j),(B,k,r))$ is realizable, iff there is a derivation $A \stackrel{*}{\Rightarrow} w_{ik} B w_{rj}$.

The size of x, denoted by $|x|$, is defined to be $j - i$. The size of the partial tree (x,y) is $|x| - |y|$. Let $x = (A,i,j)$. Then we write $y, z \vdash x$ iff $(y = (B,i,k), z = (C,k,j))$ or $(z = (B,i,k), y = (C,k,j))$ and $A \to BC$ is a production. In other words the tree corresponding to (A,i,j) can be composed of trees corresponding to y and z (in any order).

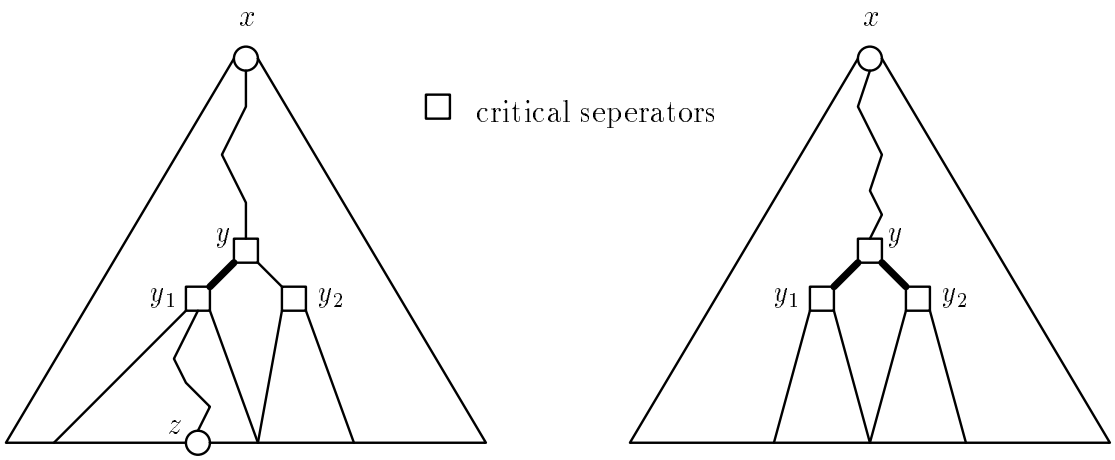
Let $T = \{ (A,i,i+1) \mid A \to w_{i,i+1}$ is a production $\}$. T is the set of realizable elements of size 1. All their elements can be generated from T using the composition $\vdash$. In the algorithm we keep two boolean tables $Tree(x)$ and $PTree(x,y)$. $Tree(x)$ will hold, if the algorithm "knows" that x is realizable (there is really a tree corresponding to x). Similarly, $PTree(x,y)$ will hold, iff the algorithm knows that there is derivation tree corresponding to

(x, y). The membership problem is reducible to the computation of all realizable elements. The input string is generated by the grammar, iff the element $(S, 0, n)$ is realizable, where S is the start symbol of the grammar.

2. Algorithm based on unique critical separators

Our first algorithm is based on unique decompositions of trees into smaller ones. We have the following obvious fact which is based on a simple observation: When climbing down a fixed path in a binary tree, you will reach a unique node which is not smaller than half the size of the whole tree, but its son is. When climbing down a binary tree going always to the bigger son, you will reach a unique node where both sons are smaller or have equal size as half the size of the whole tree, but the node itself is bigger than half the size of the whole tree.

Fact 1. For each subtree rooted at an internal node x and a descendant z of x there are unique nodes y, y_1, y_2 which satisfy the condition $\Phi_a(x, y, z, y_1, y_2)$ and there are (possibly different) unique nodes y, y_1, y_2 which satisfy the condition $\Phi_b(x, y, y_1, y_2)$ from Fig. 2. (We call such nodes critical separators.)



(Φ_a)

$\Phi_a(x, y, z, y_1, y_2) = (y_1, y_2 \vdash y)$ & $Tree(y_2)$ & $PTree(y_1, z)$ & $PTree(x, y)$ & $|(x, y)|, |(y_1, z)| < |(x, z)|/2$

(Φ_b)

$\Phi_b(x, y, y_1, y_2) = (y_1, y_2 \vdash y)$ & $y_1 \ll y_2$ & $Tree(y_1)$ & $Tree(y_2)$ & $PTree(x, y)$ & $|y_1|, |y_2| \leq |x|/2$ & $|y| > |x|/2$

Figure 2. The decomposition of trees into smaller ones

Let $\ll$ be the relation of the lexicographic order. Then the left son is always smaller than the right one. The condition Φ_b expresses the fact that y is of size at least $|x|/2$, but sons

of y of sizes smaller than $|x|/2$. There are other properties expressed in Φ_b, which cause the whole condition to look ugly big. They just describe that the situation presented in Fig. 2 (Φ_b) is "known" to the algorithm: There is a partial tree (x, y), where y_1, y_2 are sons of y and y_1 is the left son.

Similarly, the condition Φ_a expresses the simple property that y_1 is a son (left or right) of y on the path from x to z such that size of (y, z) is at least $|(x, z)|/2$ and size of (y_1, y) is strictly less than $|(x, z)|/2$. Other properties express that the situation in Fig. 2 (Φ_a) is known to the algorithm.

The algorithm which results from Fact 1 is written below. It is the simplest algorithm (known to us) for the recognition of unambiguous cfl's in logarithmic time. The intuitive idea is to find all parts of derivation trees by composing small parts into larger ones. This has to be done uniquely to avoid write conflicts and roughly equally sized parts have to be composed to get a logarithmic running time. Fact 1 provides such a decomposition.

Algorithm 1
Compute in parallel the set of all trees and partial trees of size one;
repeat $\log n$ **times begin**
  for each x, y, y_1, y_2, z **such that** $\Phi_b(x, y, z, y_1, y_2)$
    do in parallel $PTree(x, z) := true$;
  for each x, y, y_1, y_2 **such that** $\Phi_a(x, y, y_1, y_2)$
    do in parallel $Tree(x) := true$
end.

The *correctness condition* is the following.

> After the computation of the algorithm we have: $Tree(x) = true$ iff x is realizable.

We say that x and (x, y) are generated iff $Tree(x)$, $PTree(x, y)$, respectively, equals true. It is easily seen from the properties of the decompositions, see Fig. 2, that after k iterations all realizable trees and partial trees of size up to 2^k are generated. Hence after $\log(n)$ iterations we generate all elements. This proves the correctness.

However we are interested in the number of processors. We have to assign processors to tuples of elements satisfying some conditions. It can be easily seen that n^7 are enough: There are $O(n^{10})$ tuples (x, y, y_1, y_2, z). Since $\Phi_a(x, y, y_1, y_2, z)$ includes $y_1, y_2 \vdash y$ and there

is only a linear number of pairs (y_1, y_2) for each y that satisfy $y_1, y_2 \vdash y$, it is suffices to use n^7 processors. Our aim is to reduce this by a linear factor. Then the total work of our algorithm will be similar to the total work of the polylogarithmic time recognition of general cfl's. We reduce the number of processors by extending our data structure.

For each generated partial tree (x, y) the algorithm knows the value $Son(x, y)$ of the son of x on the path from x to y. Assume that this son initially equals nil. Hence we have the invariant:

$PTree(x, y)$ iff $Son(x, y) \neq nil$.

We have to add an additional instruction (instruction 1). Now the instruction "**for each** x, y, y_1, y_2, z **such that** $\Phi_b(x, y, z, y_1, y_2)$ **do in parallel** $PTree(x, z) := true$" is equivalent to the composition of two instructions (1) and (2) in the new algorithm. If the condition $\Phi_b(x, y, z, y_1, y_2)$ holds, y_1 is the first node on the path from y to z. As soon as $PTree(y, z) = true$, it will be guaranteed that $y_1 = Son(y, z)$. In algorithm 1 this needs not to be true.

We obtain the next algorithm as a version of Algorithm 1 by adding a new instruction which computes $Son(y, z)$ in the situation of Fig. 2 (Φ_b). Since the computation of $Son(y, z)$ is independent of x it is sufficient to use $O(n^5)$ processors.

Algorithm 1'

Compute in parallel the set of all trees and partial trees of size one;

repeat $\log n$ **times**

begin

1: **for each** y, y_1, y_2, z **such that** $(y_1, y_2 \vdash y)$ & $Ptree(y_1, z)$ & $Tree(y_2)$
do in parallel begin $PTree(y, z) := true$; $Son(y, z) := y_1$ **end;**

2: **for each** x, y, y_2, z **such that** $\Phi_b(x, y, z, Son(y, z), y_2)$
do in parallel begin $PTree(x, z) := true$; $Son(x, z) := Son(x, y)$ **end**;

3: **for each** x, y, y_1, y_2 **such that** $\Phi_a(x, y, y_1, y_2)$
do in parallel $Tree(x) := true$

end.

The algorithm proves the following theorem.

Theorem 2. *The recognition of unambiguous cfl's can be done in* $O(\log(n))$ *time with* n^6 *processors on a CREW PRAM.*

Proof. There are no write conflicts in the algorithm due to Fact 1. We analyse separately the number of processors needed in each of the instructions 1, 2, and 3. In instruction 1 we need a processor for each tuple (y, y_1, y_2, z). However, for a given y there is only a linear number of pairs y_1, y_2, hence there are only $O(n^5)$ such tuples (y, y_1, y_2, z). A similar argument shows that the same number of processors suffices in instruction 3. In instruction 2 we have a constant number of possiblities for y_2, if y and z are given, since $Son(y, z)$ is determined by y and z. Then $Son(y, z)$ plays the role of node y_1 in Fig. 2. Observe that when given y_1, then there is only a constant number of y_2 which are possible candidates to satisfy $y_1, y_2 \vdash y$. There are $O(n^6)$ tuples x, y, z in instruction 2. For each such tuple we have to consider only a constant number of possible values of y_2. Hence $O(n^6)$ processors are enough. This completes the proof. □

It follows from the proof that the hardest part is instruction 2. The other ones require only $O(n^5)$ processors. In the next section we give a very similar algorithm in which counterparts of instructions 1 and 3 require only $O(n^4)$ processors.

3. Algorithm based on heaviest paths decompositions

In Algorithm 1 we had two complicated conditions for tuples of nodes. Now we get rid of the second condition (the more complicated one). The resulting algorithm looks simpler, but its correctness is less visible. In the algorithm we use ideas of the algorithm in [7]. We refer the reader for details to [2] and [7]. Define the following relation $x < y$ iff $|x| < |y|$ or $(|x| = |y|$ and $x \ll y)$.

Assume we have a tree and for each node we take the edge to the bigger son (with respect to $<$). Then for each node v we obtain a unique "heaviest" path from v to a leaf, see Fig. 3. Such a path has the property that the nodes in the subtree of v which are not on this path have sizes at least twice smaller than v.

In our next algorithms we shall create edges between two elements in instruction 1. Assume the element x is realizable and we look only at its derivation tree. Then each edge created in instruction 1 which goes down from x belongs to the heaviest path starting at x. The boolean matrix $R(x, y)$ contains at the beginning *false* and the relation "x is connected to y by a heaviest path" at the end of the computation. If $R(x, y) = true$, then $Son(x, y)$ contains the son of x on the heaviest path to y.

Algorithm 2

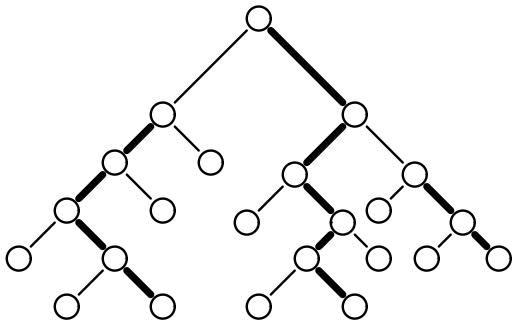


FIGURE 3. The heaviest path decomposition of a unique tree realizing a given element

repeat $2\log_2(n)$ **times**

begin

1: **for each** nodes x, y, z such that $y, z \vdash x$ & $y < z$ & y marked **do in parallel**
   begin $R(x,z) := true$; $Son(x,z) := z$ **end**

2: **for each** nodes x, y, z **do in parallel**
   if $R(x,y)$ & $R(y,z)$ & (not $R(x, Son(y,z))$ **then**
   begin $R(x,z) := true$; $Son(x,z) := Son(x,y)$ **end**;

3: **for each** nodes x, y **do in parallel if** $R(x,y)$ & $y \in T$ **then** mark x;

end.

Observe that in Algorithm 2 the conditions Φ_a, Φ_b are much simplified, especially the second one. Let T be the set of initially realizable elements x of size one; initially mark all elements of T.

Let us restrict our attention to the behaviour of the algorithm on the supposed tree deriving the given input string w and on the subtrees deriving subwords of w. For simplicity we can assume that there is a derivation tree for w, if there is no such tree then it is easily seen that the goal element $(S, 0, n)$ will not be generated. Take the heaviest path starting at x. In a given moment of the algorithm denote by $Cond(x) = y$ the lowest element on this path such that $R(x,y)$. Then the algorithm on a given tree works essentially like a tree contraction method called the parallel pebble game on trees in [2]. It was shown that there is a constant c such that $c\log(n)$ iterations in this pebble game suffice. This implies that we can take a constant c here. In fact $c = 2$ is sufficient. Recently the exact minimal value of this constant was found (by a complicated calculation) to be $c = 1/\log_2(\phi)$ in [5], where ϕ is the golden ratio. Hence Algorithm 2 is correct with respect to the generation of all realizable objects.

However, the most interesting property of the algorithm is that there are no write conflicts

in instruction 3.

Fact 3. Assume that in Algorithm 2 just before executing instruction 3 we have $R(x, y_1), R(x, y_2)$ and both y_1 and y_2 are in T. Then $y_1 = y_2$.

Proof. If there is a path to a marked node then x is realizable. Then due to unambiguity there is a unique tree generating x. All elements y such that $R(x, y)$ are on the heaviest path started at x. Hence y_1, y_2 are on this top-down path. They are leaves, hence they should be equal, as there is a unique leaf ending a single top-down path. This completes the proof of the theorem. □

Another nice property of the algorithm is that it shows where the hardest part of the algorithm is located. It is easy to see that a cubic number of processor is sufficient to execute instructions 1 and 3. However, we need a processor for each triple (x, y, z) in instruction 2. This gives $O(n^6)$ processors. In fact the main difficulty lies in computing the multiplication of two m by m boolean matrices A, B with the following unique computation property: If $A(x, y)$ and $B(y, z)$ then y is uniquely determined by x, z. Our approach to compute this was as follows:

for each x, y, z **do in parallel if** $A(x, y)$ & $B(y, z)$ **then** $Result(x, z) := true$.

For each (x, y, z) we need one processor. Hence we need m^3 processors in constant time compuation. However, it is easy to see that we can multiply such matrices in logarithmic time with only m^2 processors of CREW PRAM. If we could use the same number of processors in the constant time multiplication of such matrices then we could have $\log(n)$ time recognition of unambiguous cfl's with only n^4 processors, as in our case $m = n^2$.

4. LOG(N) TIME RECOGNITION OF UNAMBIGUOUS LINEAR LANGUAGES

We can assume that G is a linear context free grammar in a Chomsky-like normal form. Each production is of the form $A \to Ba, A \to aB$, or $A \to a$, where capital letters denote non-terminals and small letter denote terminals. Let w be an input string of length n.

Theorem 4. *Each unambiguous linear cfl can be recognized in* $O(\log n)$ *time with* n^2 *processors.*

Proof. Let the set T of initial elements consist of triples of the form $(a, i, i+1)$, where a is the $(i+1)$-th letter of the input w. We take the graph G whose set of nodes is again V and edges are of the form (x, y), where $y, z \vdash x$ for some z in T. Define $X = \{\ (A, i, i+1) \mid A \to w_{i,i+1}\ \}$.

We use the same approach as in the efficient $\log^2 n$ recognition of unambiguous cfl's in [1]. The element x is realizable iff there is a path from x to an element of X in the graph G. The graph G has the unique path property (at most one path from a given node to another) due to unambiguity. The number of edges of G is quadratic. The computation of all nodes from which an element of X is accessible can be made via an extended tree contraction method. We refer the reader to [1]. The time is logarithmic and the number of processors is proportional to the size of G, hence it is quadratic. This completes the proof. □

Remark 5. If the underlying grammar is finitely ambiguous rather than unambiguous then all our algorithms may contain write conflicts. However, if the grammar is k-ambiguous, then at most k processors write into one memory cell at the same time. By a straightforward technique this write conflicts can be solved by increasing the number of processors by a power of k. In the case of general k-ambiguous cfl's this gives us an $O(n^{6k})$ processors algorithm and in the case of linear cfl's an $O(n^{2k})$ processors $\log(n)$ time algorithm by a naive method. Can it be done better?

References

1. M. Crochemore, M. Chytil, M. Monien, and W. Rytter. Efficient recognition of unambiguous context free languages. *Theoretical Computer Science*, 81(2):311–316, 1991.
2. A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.
3. K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In *Proc. of 15th Symposium on Mathematical Foundations of Computer Science*, number 452 in Lecture Notes in Computer Science, pages 399–406. Springer, 1990.
4. B. Monien, H. Schäpers, and W. Rytter. Fast recognition of cfl's with smaller number of processors. 1991.
5. W. Plandowski, W. Rytter, and T. Szymacha. Exact analysis of three tree contraction algorithms. In *Proc. of 8th Conference on Fundamentals of Computer Science*, number 529 in Lecture Notes in Computer Science, pages 370–379. Springer, 1991.
6. J. Reif. Parallel $O(\log n)$ acceptance of deterministic CFLs. In *Proc. of 23rd IEEE Symposium on Foundations of Computer Science*, pages 290–296, 1982.
7. W. Rytter. Parallel time $O(\log n)$ recognition of unambiguous context-free languages. *Information and Computation*, 73:75–86, 1987.