Bracket-Languages are Recognizable in Logarithmic Space

Kurt Mehlhorn

Fachbereich 10
Angewandte Mathematik und
Informatik der Universität
des Saarlandes
D-6600 Saarbrücken
Im Stadtwald

A 75/12
September 1975

Summary:

In this paper, we prove lower bounds for the space requirement of the membershipproblem of context-free languages. A context-free language $L \subseteq \Sigma^*$ is called strongly non-regular if there exist words $u,v,w,x,y \in \Sigma^*$ with $L \cap uv^*wx^*y$ non-regular.

Main Theorem: To decide the membershipproblem of strongly non-regular context-free languages requires $O(\log n)$ space infinitely often.

As corollaries we obtain lower bounds for several families of deterministic languages ,e.g. realtime languages, and about the minimal growth rate of tape constructable functions.

Context-free languages are an important topic in practical as well as in theoretical computer science. Much effort was devoted to the construction of time - and (or) space - efficient recognition algorithms.

Early results were obtained by Lewis, Hartmanis and Stearns [5] : Every context-free language can be recognized on an off-line Turing machine in space $O(\log^2 n)$, where n is the length of the input. Recent results by Sudborough [7] and Monien [6] indicate that it is probably very hard to beat this bound. They showed that the existence of a general context-free recognition algorithm working in logarithmic space would imply the equality of deterministic of nondeterministic context-sensitive languages, i.e. a solution to Myhill's LBA-problem.

However, it is conceivable that large subclasses of the context-free languages (e.g. the deterministic languages) are recognizable in less than $O(\log^2 n)$ units of space. In [1] Alt and Mehlhorn attacked this problem by proving that the word-problem for many classes of deterministic context-free languages requires at least $O(\log n)$ units of space infinitely often. Hotz and Messerschmidt [4] attacked the problem from the other end: Dyck-languages can be recognized in $O(\log n)$ units of space. We extend their results and show that every bracket-language can be recognized in $O(\log n)$ units of space.

For the notation we refer the reader to [3] . A context-free grammar is a 4-tuple $G = (V, \Sigma, P, S)$, where V is the set of non-terminals (variables), Σ is the set of terminals, P is the set of productions and S is the startsymbol. (,) are two special symbols in Σ. A production $A \to (\alpha)$ is called bracketed iff $\alpha \in ((\Sigma \cup V) - \{(,)\})^*$. A context-free language L is a bracket-language if it can be generated by a context-free grammar G all of which rules are bracketed. The bracketing is a linear encoding of the syntax tree. In the sequel, L is always a bracket-language and $w \in \Sigma^*$ is a word of length n. We want to decide if $w \in L$.

First, we find out if there is a correct number of brackets in w. To do so, we scan w from left to right once always keeping a count (in binary) of the number of unmatched open brackets. This number should always be positive except when scanning the very last symbol of w.

If w fails this test then it is rejected. Otherwise, the brackets in w encode a tree. This tree must be the parse-tree. So, we only have to find out if there is a consistent labelling of the interior nodes of the tree by the variables of the grammar. This can be done by labelling the tree bottom-up according to the following rules:

I) The leaves of the tree are labelled by the symbols of w from left to right.

II) Let v_0 be a node with sons $v_1,\ldots,v_k$. If v_i is labelled by $V_i \subseteq V$, $1 \leq i \leq k$, then v_0 can be labelled by

$$V_0 = \{A \; ; \; A \to A_1A_2\ldots A_k \in P \text{ and } A_i \in V_i \text{ for } 1 \leq i \leq k\}$$

A tree is a parse of w according to G if the label of the root contains the start symbol S. Note the similarity of this labelling procedure with Younger's recognition algorithm [8] .

The label of a node is used only once: when its father is labelled. Thereafter its label is never used again. Therefore we can discard it.

Definition: An interior node (non-leaf) v is called essential if it is labelled, but its father is not labelled.

We want to label the nodes of the tree in such a judicious order, so to keep the maximal number of nodes which are simultaneously essential as small as possible. There is a well known strategy

for achieving this: Label larger subtrees first [2,5] .
It is incorporated in the following marking procedure.

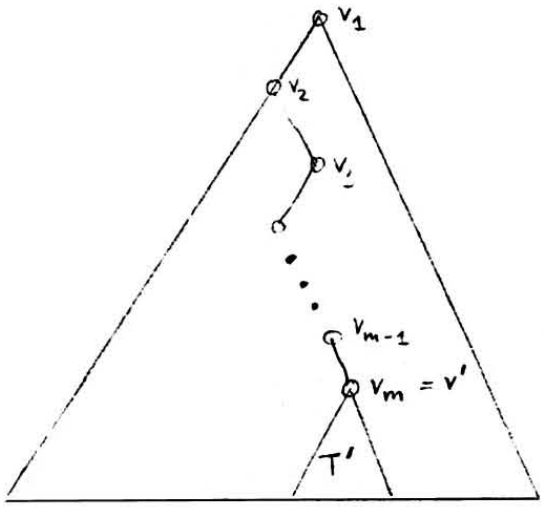
```
procedure label (tree T) ;

begin let v be the root of T ;
      if all of v's sons are leaves
      then
           begin
                let the i-th son of v be labelled
                by a_i ∈ ∑ , 1 ≤ i ≤ k;
                label v with {A; A → a_1...a_k ∈ P};
                comment v is now an essential node;
           end
      else
           begin
                let T_1,T_2,...,T_k be the direct subtrees of T;
                let Π be that permutation of {1,...,k}
                satisfying
                (1) ||T_Π(1)|| ≥ ||T_Π(2)|| ≥...≥ ||T_Π(k)||
                (2) ||T_Π(l)|| = ||T_Π(l+1)|| implies Π(l) < Π(l+1)
                (Here ||T|| is the number of leaves of tree T)
                for l=1 to k do label (T);
                let V_i , 1 ≤ i ≤ k , be the label of the root
                of subtree T_i;
                label v by
                {A; A → A_1...A_k ∈ P and A_i ∈ V_i}
                commont v is now an essential node;
                the roots of the subtrees T_i are not essential
                anymore;
           end;
end;
```

The following facts are easily proved by induction on the data structure.

Fact 1: ([3]) At most $r \cdot (\log |w| + 1)$ nodes are simultaneously essential, where $r = \max \{ |\alpha| ; A \to \alpha \in P \}$.

Consider any call of the recursive marking procedure. Let T' be the argument of the active call and let v' be its root. Let further $v_m, v_{m-1}, \ldots, v_1$ with $v' = v_m$ be the path from v' to the root.



Fact 2: Let v be any essential node. Then v is the son of some v_i for $1 \leq i \leq m-1$.

Fact 3: If none of v_i's sons ($1 \leq i \leq m-1$) is essential then v_{i+1} is the root of the leftmost subtree (of the tree with root v_i) having a maximal number of leaves.

We may summarize these three facts as follows: The essential information is concentrated along the path from the root of the argument of the active call to the root of the total tree. The amount of essential information is logarithmically bounded in the length of the input.

Definition: A node v is called partially analyzed if at least one, but not all of its sons are labelled.

Note that the partially analyzed nodes are exactly the fathers of essential nodes. Therefore we may reformulate the three facts listed above.

Fact 1': At most $r \cdot (\log |w| + 1)$ nodes are simultaneously partially analyzed.

Fact 2': Let v be any partially analyzed node.
Then v is some v_i, $1 \leq i \leq m-1$.

Fact 3': If v_i, $1 \leq i \leq m-1$, is not partially analyzed, then v_{i+1} is the root of the leftmost subtree (of the tree with root v_i) having a maximal number of leaves.

We associate with every partially analyzed node an r-tuple over $2^V \cup \{*\}$. The l-th component of this r-tuple is

$V' \subseteq V$ if the root of the l-th subtree is (already) labelled by $V' \subseteq V$,

$*$ otherwise.

Using these concepts we can rewrite our algorithm. We keep the r-tuples assigned to the partially analyzed nodes in a stack. As before, we label larger subtrees first. Whenever we return from a call of *label*, with argument T say, we recompute the rank (leftmost largest, rightmost smallest, other) of the argument tree among its brother trees. If T is the leftmost largest subtree then we create a new r-tuple and push it on the stack. Then we proceed with the second largest subtree. If T is the rightmost smallest subtree then all its brother trees are already labelled and their labels are stored in the top element of the stack. We use this element to compute the label of the root of the supertree of T and pop the stack. Then we return one level in the recursion. Otherwise we update the top stack element and proceed with the next smaller subtree. This leads to the following non-recursive program.

```
( 1)  tree: T ; subset of V: H ;
( 2)  let T have k subtrees ;
( 3)  if  k > r  then reject ;
( 4)  if all of T's subtrees are leaves
( 5)  then begin  let a_i be the label of the i-th subtree;
( 6)               H<- {A; A -> a_1...a_k ∈ P}
           end
( 7)  else begin T<- leftmost maximal subtree of T;
( 8)               goto (4)
           end
( 9)  compute the position of T among its brother trees;
(10)  case  position  of
(11)  leftmost largest: create a new r-tuple;
(12)                    if T is the l-th subtree then let its
                        l-th component be H;
(13)                    push the  r-tuple on the stack;
(14)  rightmost smallest: if T is the l-th subtree then set the
                          l-th component of the top stack element
                          to H;
(15)                      let k be the number of non * components
                          of the stack top;
(16)                      H<- {A; A -> A_1...A_k ∈ P and A_l ∈ l-th
                          component of stack top};
(17)                      pop the stack;
(18)                      T<- super-tree  of T;
(19)                      goto (9)
```

(20) other : if T is the l-th subtree then set the l-th component of the stack-top to H;

esac;

(21) T<-- next smaller brother of T;

(22) goto (2);

The correctness of this algorithm follows from the preceding discussion. Furthermore, fact 1 ensures us that the length of the stack will never exceed r (log |w| + 1). It is obvious that lines (2), (3), (7), (9), (18) and (21) can be computed in logarithmic space. Therefore we obtain.

Theorem: Every bracket language can be recognized by a O(log n) space-bounded Turing machine.

Alt and Mehlhorn show in [1] that bracket languages also require O(log n) space infinitely often.

Corollary: The space complexity of the word problem for bracket languages is exactly O(log n).

We want to close with a remark on the time complexity of our recognition procedure. Since the machine is O(log n) space-bounded it is certainly polynomially time bounded. In fact, it is easily seen that the time complexity is $O(n^2)$.

Bibliography

[1] H. Alt, K. Mehlhorn : Lower Bounds for the Space Requirement of Some Families of Context-Free Languages, Techn. Bericht der Univ. des Saarlandes 1975.

[2] Cook, S.A.: An Observation on time-storage trade-off, ICSS, 9, 308-316 (1974)

[3] Hopcroft, J.E and Ullman J.D.: Formal languages and their relation to automata, Addison-Wesley, Reading, Mass., 1969

[4] Hotz, G. and Messerschmidt, J.: Dyck-Sprachen sind in Bandkomplexität log n analysierbar, Techn. Bericht der Univ. des Saarlandes, 1974

[5] Lewis, P.M., Hartmanis, J., and Stearns, R.E.: Memory bounds for the recognition of context-free and context-sensitive languages, IEEE Conference Record on Switching Circuit Theory and Logical Design, 191-202, 1965

[6] Monien, B.: Transformational methods and their application to complexity problems, 2-te GI-Fachtagung, Kaiserslautern, 1975

[7] Sudborough, I.H.: On tape-bounded complexity classes and multihead finite automata, 14 th IEEE SWAT Conference, 138-144, 1973

[8] Younger, D.H.: Recognition and parsing of context-free languages in time n^3, Inf. and Control, 10, 189-208, 1967