INFORMATION AND CONTROL **67**, 12–22 (1985)

Fast Recognition of Pushdown Automaton and Context-free Languages

WOJCIECH RYTTER*

Institute of Informatics, Warsaw University, 00-901 Warsaw, Poland

We prove: (1) every language accepted by a two-way nondeterministic pushdown automaton can be recognized on a random access machine in $O(n^3/\log n)$ time; (2) every language accepted by a loop-free two-way nondeterministic pushdown automaton can be recognized in $O(n^3/\log^2 n)$ time; (3) every context-free language can be recognized on-line in $O(n^3/\log^2 n)$ time. We improve the results of Aho, Hopcroft, and Ullman (*Inform. Contr.* **13**, 1968, 186–206), Rytter (*Inform. Process. Lett.* **16**, 1983, 127–129), and Graham, Harrison, and Ruzzo (*ACM Trans. Programm. Lang. Systems* **2**, No. 3, 1980, 415–462). © 1985 Academic Press, Inc.

1. INTRODUCTION

The languages accepted by two-way nondeterministic pushdown automata (2npda's, for short) are much harder to recognize than context-free (1npda) languages. The main reason for this situation is the ability of the input head of the automaton to move in two directions. Aho, Hopcroft, and Ullman (1968) proved that every 2npda language can be recognized in $O(n^3)$ time on a random access machine (RAM). Recently this bound was improved for some subclasses of 2npda languages (see Rytter, 1982, 1983). In this paper we prove that every 2npda language can be recognized in $O(n^3/\log n)$ time and every language accepted by a loop-free 2npda can be recognized in $O(n^3/\log^2 n)$ time on a RAM. These results can be generalized to multihead 2npdas. It was shown by Rytter (1984) that context-free trace languages can be accepted by loop-free multihead 2npdas. Hence our results can be used to improve slightly the algorithms for the recognition of context-free trace languages, see Rytter (1984).

We use a simple idea of compressing the subsets of a given set and precomputing set operations on some small subsets. A similar method works for on-line recognition of context-free languages (cfls, for short). We modify the algorithm of Graham, Harrison, and Ruzzo (1980) and apply

*Preliminary version of this paper has appeared in the "Proceedings of the Conference of Math. Foundations of Computer Science, 1984.

the compressed representation of the columns of the parsing matrix. This reduces the cost by the factor $\log n$.

Our model of the computation is the random access machine (RAM) with the uniform cost criterion. We restrict ourselves to RAMs with word lengths of $O(\log n)$ bits.

This model is similar to what is called the "RAC" model, introduced in Angluin and Valiant (1979). Weicker (1976) has shown that cfl recognition on a unit cost RAM in time $O(n^2 \log n)$ is possible by encoding an entire column of the parsing matrix into one integer of $O(n \log n)$ bits, and by assuming two such integers can be added in unit time. To avoid such "abuses" of the RAM model we restrict ourselves to RAMs with words of the length $O(\log n)$.

2. An Implementation of Set Operations

Denote $U = [0...m-1]$. We want to execute quickly operations DIF and INSERT, where DIF computes the difference of two subsets of U, and INSERT inserts an element into a given subset. The subsets can be represented by characteristic vectors, giving straightforward $O(m)$ time implementation of DIF. This cost can be reduced using the following method. Let $p = \lceil \log_2(m)/4 \rceil$, assume for simplicity that m is divisible by p and let $s = m/p$. We maintain together with the characteristic vector X of a given set (of the boolean type array $[0...m-1]$) a compressed version X' of X, where X' is of the small integer type array $[0...s-1]$.

We divide X into sections of length p. The ith element of X "appears in" the tth section, where $t = i \operatorname{div} p$. Let section $(r, X) = (X(p \cdot r),$ $X(p \cdot r + 1), ..., X(p \cdot r + p - 1))$. For a binary sequence $a = (a_0, ..., a_{p-1})$ denote by code (a) the binary number represented by a. Now define

$$X'(r) = \text{code}\ (\text{section}\ (r, X)).$$

If $q = \text{code}\ (a_0, ..., a_{p-1})$ then denote $\text{set}\ (q) = \{k \mid a_k = 1, 0 \leqslant k < p\}$. Hence the codes of sections can be treated as sets and we can define some set operations. Let $P = [0...2^{p-1})$. The key observation is that the number of such objects (elements of P treated as sets) is very small because of the chosen value of p. For $q_1, q_2 \in P$ define

$$\text{dif}(q_1, q_2) = \text{set}\ (q_1) - \text{set}\ (q_2);$$

$$\text{add}\ (q1, k) = \text{set}^{-1}\ (\text{set}\ (q_1) \cup \{k\}) \qquad \text{for}\ 0 \leqslant k \leqslant p-1.$$

We precompute and store in a table the values of dif (q_1, q_2), add(q_1, k) for all possible arguments. This costs only $O(m)$ time due to the small size of p.

Now we assume (disregarding the cost of preprocessing) that the costs of operations dif and add are constant. Assume that we have two sets whose characteristic vectors are X, Y (identify sets with their characteristic vectors), and that we know X' and Y'. We implement operations DIF and INSERT as follows:

```
function DIF (X, Y)      {we know X', Y'}
begin
DIF: = ∅;
for r: = 0 to s − 1 do
   for each q ∈ dif (X'(r), Y'(r)) do DIF: = DIF ∪ {q + r · p};
end function.
procedure INSERT (i, X);
begin X(i): = 1; X'(i div p): = add (X'(i div p), i mod p) end.
```

The following lemma follows directly from our implementation.

LEMMA 1. Assume that values of dif, add are precomputed. Then

(1) The cost of computing DIF (X, Y) is $O(m/\log(m) + v)$, where v is the size of the result;

(2) The cost of computing INSERT (i, X) is $O(1)$.

3. Fast Recognition of 2npda Languages

We fix a 2npda A and input w of length n. The surface configuration (configuration, for short) is a triple (state, top symbol, input head position). K denotes the set of configurations. Assume for simplicity that $K = [0...m-1]$ (configurations are numbered).

Let 0 be the number of the initial and $m-1$ of the accepting configuration. Assume that initially and in the moment of acceptance the stack is one element and each move is a pop or a push. The pair $(k, l) \in K^2$ is *below* (i, j) iff A can go from k to i in one push move and from j to l in a pop move, and top symbols in k, l are the same. A pair (i, j) is said to be *realizable*, if A starting in the configuration i with a one-element stack can go after some number (including zero) of moves to the configuration j ending also with a one-element stack. Let R denote the set of all realizable pairs and let below(i, j) denote the set of pairs which are below(i, j). Observe that the set below(i, j) has $O(1)$ size and can be computed in constant time. The next lemma is analogous to Lemma 3.2(a) in (Rytter, 1982). The resulting algorithm is similar to the algorithm MAIN in (Rytter, 1982). However the whole construction goes back at least to (Aho, Hopcroft, and Ullman, 1968).

LEMMA 2. (a) A accepts w iff $(0, m-1) \in R$;

(b) R is the minimal set satisfying the conditions:

(1) $(i, i) \in R$ for every $i \in K$;

(2) if $(i, j) \in R$ then $\text{below}(i, j) \subseteq R$,

(3) if $(i, k) \in R$ and $(k, j) \in R$ then $(i, j) \in R$.

Now the simulation of A can be reduced to the computation of R. We start with the set R consisting initially only of the pairs (i, i) and successively augment R according to Lemma 2(b). Let Q be a queue. The operation insert(x, S) inserts x into the set S. The operation delete(Q) deletes an element from Q, the deleted element is the value of this operation.

ALGORITHM 1.

```
begin
1: $Q := R := \{(i, i) | i \in K\}$;
while $Q \neq \varnothing$ do
  begin
   $(i, j) :=$ delete$(Q)$;     {the pair $(i, j)$ is to be processed}
2: for each $(k, l) \in$ below$(i, j)$ do
   if $(k, l) \notin R$ then insert $((k, l), R)$ and insert$((k, l), Q)$;
3: for each $(k, i) \in R$ such that $(k, j) \notin R$ do
     insert$((k, j), R)$ and insert$((k, j), Q)$;
4: for each $(j, k) \in R$ such that $(i, k) \notin R$ do
     insert$((i, k), R)$ and insert$((i, k), Q)$
  end;
if $(0, m-1) \in R$ then ACCEPT
end.
```

In Algorithm 1 instructions 1, 2 correspond to conditions 1, 2, respectively, in Lemma 2, and instructions 3, 4 correspond to condition (3).

We represent R by the boolean matrix B such that $B(i, j) = 1$ iff $(i, j) \in R$. This gives a direct implementation of Algorithm 1 in $O(n^3)$ time. The most costly are instructions 3, 4. Using Lemma 1 we reduce the complexity.

THEOREM 1. Every 2npda language can be recognized in $O(n^3/\log n)$ time.

Proof. We maintain, together with the matrix B, the sets

$$\text{COL}\,(j) = \{k | B(k, j) = 1\},$$
$$\text{ROW}\,(j) = \{k | B(j, k) = 1\} \qquad \text{for } 0 \leqslant j \leqslant m-1.$$

The instruction 3 can be now replaced by:

for each $k \in \text{DIF}(\text{COL}(i), \text{COL}(j))$ *do*
INSERT $(k, \text{COL}(j))$ and INSERT $(j, \text{ROW}(k))$ and insert $((k, j), Q)$.

Using compressed representation of sets COL (j) and ROW (j) the cost of this instruction is $O(m/\log(m) + v)$, where v is the number of pairs inserted into Q (according to Lemma 1). We proceed analogously with instruction 4. We execute $O(m^2)$ such instructions. Hence the total cost is $O(m^3/\log(m))$ plus the total number of inserted pairs. Each pair is inserted into Q at most once, and there are $O(m^2)$ pairs. The thesis follows now from the fact that $m = O(n)$. This completes the proof.

We say that a 2npda A is loop-free iff there is no possible infinite computation of the automaton on any (finite) input word. For example, each cfl can be recognized by a loop-free 1npda.

THEOREM 2. *Every language accepted by a loop-free 2npda A can be recognized in $O(n^3/\log^2 n)$ time.*

Proof. We refer the reader to the algorithm in (Rytter, 1983) simulating loop-free 2npda's in $O(n^3/\log n)$ time. The algorithm computes recursively the set of terminators of a given configuration. The dominating instruction in this algorithm is of the form

$$S := S \cup S_1$$

where S, S_1 are sets of size $O(n)$ of configurations. $O(n^2/\log n)$ such instructions are executed. We can rewrite this instruction in the form

$$S := S \cup \text{DIF}(S_1, S).$$

Next we can apply Lemma 1. The cost of each instruction is now $O(n/\log(n) + v)$, where v is the number of newly added elements into S, this gives total cost $O(n^3/\log^2 n)$ if we disregard numbers v. There are $O(n)$ sets S and we never delete elements, hence the numbers v give in total $O(n^2)$ cost. This completes the proof.

The theorem can be generalized to multihead 2npdas. It can be proved that each loop-free 2npda with k input heads can be simulated in $O(n^{3k}/\log^2 n)$ time. We show an application.

Let C be a symmetric binary relation on the input alphabet. C can be thought as a concurrency relation and input symbols as atomic processes. If L is a language then by closure(L, C) denote the least language L' containing L and such that $uabv \in L'$ and aCb implies $ubav \in L'$ for each pair of input symbols a, b and each pair of strings u, v. It was shown by Rytter

(1984) that if L is a cfl then closure(L, C) can be accepted by a loop-free 2npda A with k input heads, where k is the size of the maximal clique of the relation C (the maximal number of processes which can work simultaneously).

Applying our simulation of multihead 2npda's we obtain

THEOREM 3. *If L is a cfl and C a symmetric binary relation over the input alphabet then the language* closure(L, C) *can be recognized in $O(n^{3k}/\log^2 n)$ time, where k is the size of the maximal clique of C.*

4. A Fast On-Line Recognition of cfls

The fastest known algorithm for the on-line recognition of cfls had time complexity $O(n^3/\log n)$ on RAM, see Graham et al. (1980).

Remark. There is a simple alternative proof of the existence of such an algorithm. For each cfl there is a on-line Turing machine recognizing it in $O(n^3)$ time. Galil (1976) has proved that if there is a Turing machine recognizing a given language on-line in $O(n^s)$ time, then an algorithm can be constructed which recognizes the same language on-line on RAM in $O(n^s/\log n)$ time (assuming $s > 1$). This implies the existence of an on-line algorithm recognizing cfls in $O(n^3/\log n)$ time.

We show that using a compressed representation of the parsing matrix the algorithm of Graham, Harrison, and Ruzzo (1980) can be sped up by the factor log (n), giving an $O(n^3/\log^2 n)$ time on-line recognizer.

Let $G = (V_N, V_T, P, S)$ be a context-free grammar in the Chomsky normal form, where

V_N is the set of nonterminal symbols,
V_T is the set of terminal symbols,
P is the set of productions, and
S is the starting nonterminal symbol.

Whenever we write $A \rightarrow BC$ or $A \rightarrow a$, then this means that $A \rightarrow BC$, $A \rightarrow a$, respectively, is a production from P. We write $A \overset{*}{\rightarrow} v$ iff the string v can be derived from A in the grammar. Let $P(V_N)$ denote the set of all subsets of V_N.

Let us fix G and let $w = a_1 a_2 \ldots a_n$ be a given input word of length n. For $0 \leqslant i \leqslant j \leqslant n$ and $A \in V_N$ we say that the triple (A, i, j) is *realizable* iff $A \overset{*}{\rightarrow} a_{i+1} \ldots a_j$. Let R be the set of all realizable triples.

It can be proved that R is the minimal set satisfying the conditions:

(a) $(A, i, i+1) \in R$ for every $0 \leqslant i \leqslant n$, $A \rightarrow a_{i+1}$, and

(b) if (B, i, k), $(C, k, j) \in R$, and $A \rightarrow BC$ then $(A, i, j) \in R$.

We represent the set R by the matrix T (called the parsing matrix) of the type array $[0...n, 0...n]$ of $P(V_N)$, such that

$$T(i,j) = \{A \mid (A, i, j) \in R\}.$$

We use an algebraic approach. For $N_1, N_2 \in P(V_N)$ define

$$N_1 * N_2 = \{A \mid A \to BC \text{ and } B \in N_1, C \in N_2\}.$$

If $(N_0,..., N_n)$, $(M_0,..., M_n)$ are vectors (rows or columns) with entries in $P(V_N)$ and $N \in P(V_N)$ then define

$$(N_0,..., N_n) * (M_0,..., M_n) = (N_0 * M_0,..., N_n * M_n),$$
$$(N_0,..., N_n) \cup (M_0,..., M_n) = (N_0 \cup M_0,..., N_n \cup M_n),$$
$$(N_0,..., N_n) * N = (N_0 * N,..., N_n * N).$$

Let COL (j) denote the jth column of T, and assume that initially the value of each entry ot T is the empty set. We start with the following algorithm:

ALGORITHM 2.

```
begin {$j$th output symbol is 1 iff $a_1...a_j \in L(G)$}
for $j := 1$ to $n$ do {compute COL $(j)$}
    begin
    $a_j :=$ next input symbol;
    $T(j-1, j) := \{A \mid A \to a_j\}$;
1:  for $k := j-1$ downto 0 do
        COL $(j)$ := COL $(j) \cup$ COL$(k) * T(k, j)$;
    if $S \in T(0, j)$ then write(1) else write(0)
    end
end.
```

Time complexity of this algorithm is $O(n^3)$ and it is dominated by the total cost of instruction 1. This cost was improved by dividing columns into sections.

Let $c = |P(V_N)|$ and $p = \lceil \log_c (n)/2 \rceil$. Assume for simplicity that $n+1$ is divisible by p. There are $O(n^{1/2})$ vectors of length p whose entries are subsets of V_N. We write column-vectors and column-sections horizontally.

Let colsection(r, j) = section$(r$, COL$(j))$ = $(T(p \cdot r, j),..., T(p \cdot r + p - 1, j))$. For a column-vector $\bar{X} = (X_0,..., X_{p-1})$ whose entries are subsets of V_N, and for $0 \leqslant i \leqslant n$ define

$$\text{PRODUCT}(i, \bar{X}) = \bigcup_{0 \leqslant k \leqslant p-1} \text{COL}\,(i+k) * X_k$$

(if all X_k's are empty then the result is $\varnothing$). Whenever we refer to COL(j) or colsection(r, j) we refer to the corresponding part of T.

ALGORITHM 3.

```
begin
    for $j := 1$ to $n$ do
    begin
  $aj :=$ next input symbol;
  $T(j-1, j) := \{A \mid A \to a_j\}$;
  {compute $j$th column section by section starting with the section containing entry $T(j-1, j)$}
  for $r := \lceil (j-1)/p \rceil$ downto 0 do
  {compute and process colsection($r, j$)}
  begin
  1: for $i := rp + p - 1$ downto $rp$ do
         colsection($r, j$) := colsection($r, j$) $\cup$ colsection($r, i$) $*$ $T(i, j)$;
  2: H := PRODUCT($i$, colsection($r, j$)); $\{i = r \cdot p\}$
  3: COL($j$) := COL($j$) $\cup$ $H$
  end;
if $S \in T(0, j)$ then write(1) else write(0)
end
end.
```

Observe that immediately after executing instruction 1 we have $i = r \cdot p$ (assume that the last value of the variable used in the instruction *for* is preserved). Algorithm 3 is a refinement of Algorithm 2. Instruction 1 in Algorithm 2 is replaced by the instructions computing COL(j) section by section. We refer the reader to Graham, Harrison, Ruzzo (1976); however, we eliminated here the operation predict used there. After executing instruction 1 in Algorithm 3, colsection(r, j) is completely computed. Now instruction 3 is equivalent to

$$\text{COL}(j) := \text{COL}(j) \cup \text{COL}(rp) * T(rp, j) \cup \text{COL}(rp+1) * T(rp+1, j) \cup \ldots$$
$$\cup\, \text{COL}(rp+p-1) * T(rp+p-1, j).$$

Algorithm 3 is designed in such a way that the compressed representation can be easily applied. Let us analyse the algorithm.

Instructions (1–3) are executed $O(n^2/\log n)$ times. The cost of instruction 1 is $O(\log^2 n)$; this gives in total $O(n^2 \log n)$ cost. The total cost of instruction 2 can be reduced by applying the tabulation method. We store the computed values of PRODUCT($i, \bar{X}$) in the table TAB($i, \bar{X}$). Each entry of this table initially contains the special value "undefined." Replace instruction 2 by

```
2':  $\{i=rp\}$ $\bar{X} := \mathrm{colsection}(r, j)$;
     if $\mathrm{TAB}(i, \bar{X})$ = "undefined" then $\mathrm{TAB}(i, \bar{X}) := \mathrm{PRODUCT}(i, \bar{X})$;
     $H := \mathrm{TAB}(i, \bar{X})$;
```

In this manner the total cost of executed instructions 2 (after replacement) is $O(n^{2.5} \log n)$, because there are only $O(n^{1.5})$ possible pairs (i, X) and for each such pair we compute the operation PRODUCT at most once.

Now the cost of instruction 3 is dominating. The total cost of executed instructions 3 is $O(n^3/\log n)$. The key to the improvement of the whole algorithm is a fast implementation of this instruction.

THEOREM 4. Every context-free language can be recognized on-line in $O(n^3/\log^2 n)$ time.

Proof. We describe an implementation of instruction 3. We need to compute quickly instructions $X := X \cup H$, where X and H are vectors whose elements are subsets of V_N (X plays the role of $\mathrm{COL}(j)$). It is enough to implement efficiently the operation

$$\mathrm{DIF}_1(H, X) = \{(A, k) | A \in H_k \text{ and } A \notin X_k\},$$

where $X = (X_0, ..., X_n)$ and $H = (H_0, ..., H_n)$. We will maintain together with vectors X, H the functions $\tilde{X}$, $\tilde{H}$, whose domain is V_N and whose ranges are boolean vectors, such that

$$(\tilde{X}(A))_i = 1 \quad \text{iff } A \in X_i \quad (\tilde{H} \text{ is defined analogously}).$$

We have

$$\mathrm{DIF}_1(H, X) = \bigcup_{A \in V_N} \{(A, k) | k \in \mathrm{DIF}(\tilde{H}(A), \tilde{X}(A))$$

where DIF is the operation from Section 2.

Now we can apply the compressed representation of boolean vectors. Observe that the size of V_N is constant. Assume that together with H, X we maintain the collections of compressed versions of $\tilde{H}(A)$, $\tilde{X}(A)$ for $A \in V_N$. Lemma 1 implies that now we can compute the operation $\mathrm{DIF}_1(H, X)$, and consequently the instruction $X := X \cup H$ in time $O(n/\log(n) + v)$, where v is the size of $\mathrm{DIF}_1(H, X)$. Hence we can compute instruction $\mathrm{COL}(j) := \mathrm{COL}(j) \cup H$ in time $O(n/\log(n) + v)$, where v is the number of elements which are newly added to the entries of $\mathrm{COL}(j)$ by this instruction. We never delete elements, hence numbers v give in total in Algorithm 3, $O(n^2)$ cost, because of the size of the parsing table. Hence the total cost is $O(n^3/\log^2 n)$, because we execute $O(n^2/\log n)$ instructions 3. However we have to maintain the compressed representations of boolean vectors $\widetilde{\mathrm{COL}(j)}(A)$, $\tilde{H}(A)$ for $A \in V_N$. This can be done by recording in the table TAB the compressed representations related to vectors recorded in TAB originally in Algorithm 3. The instruction 2′ should be slightly modified (we leave the details to the reader). The extra cost is small because there are only $O(n^{1.5})$ entries of TAB. Hence the total cost is $O(n^3/\log^2 n)$.

Now we overcome the fact that n is not known in advance (the algorithm is on-line). Initially the algorithm assumes that $n = 2$. In general, it assumes that $n = 2^k$ and if the $(2^k + 1)$th symbol is read the computation starts over with $n = 2^{k+1}$ suppressing the initial outputs (see Galil, 1976). This does not change the order of time complexity. This completes the proof.

Recently the author has learned that a method of matrix compression (similar to our technique of compressing the sets) was used by Bird (see Bird, 1977). The method of compressing the subsets can be also applied to the algorithm called the "Four Russians" algorithm in Aho, Hopcroft and Ullman (1974) reducing the cost of boolean matrix multiplication from $n^3/\log(n)$ to $n^3/\log^2 n$.

RECEIVED July 23, 1985

REFERENCES

AHO, A., HOPCROFT, J., AND ULLMAN, J. (1968), Time and tape complexity of pushdown automaton languages, *Inform. Contr.* **13**, 186–206.

AHO, A., HOPCROFT, J., AND ULLMAN, J. (1974), "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass.

ANGLUIN, D., AND VALIANT, L. (1979), Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. System Sci.* **18**, 155–193.

BIRD, R. (1980), Tabulation techniques for recursive programs, *Comput. Surveys* **12**, No. 4.

BIRD, R. (1977), "On the Tabulation of Recursive Programs," Technical Report, University of Reading.

GALIL, Z. (1976), Two fast simulations which imply some fast string matching and palindrom recognition algorithms, *Inform. Process. Lett.* **4**, 85–87.

GRAHAM, S., HARRISON, M., AND RUZZO, W. (1976), On-line context-free recognition in less than cubic time, in "Proceedings, 8th ACM Sympos. Theory of Computing," pp. 112–120.

GRAHAM, S., HARRISON, M., AND RUZZO, W. (1980), An improved context-free recognizer, *ACM Trans. Programm. Lang. Systems* **2**, No. 3, 415–462.

RYTTER, W. (1982), A note on two-way nondeterministic pushdown automata, *Inform. Process. Lett.* **15**, 5–9.

RYTTER, W. (1982a), Time complexity of unambiguous path systems, *Inform. Process. Lett.* **15**, 102–104.

RYTTER, W. (1983), Time complexity of loop-free two-way pushdown automata, *Inform. Process. Lett.* **16**, 127–129.

RYTTER, W. (1984a), Some properties of trace languages, *Fund. Inform.* **7**, No. 1, 117–127.

RYTTER, W. (1984b), Time complexity of two way pushdown automata and recursive programs, in "Combinatorial Algorithms on Words" (A. Apostolico and Z. Galil, Ed.), Springer-Verlag, in press.

WEICKER, R. (1976), "Context-free Recognition by a RAM with Uniform Cost Criterion in Time $n^2 \log n$," Technical Report No. 182, Pennsylvania State University.