IA

AFDELING INFORMATICA IN 9/75 JUNE

J. VAN LEEUWEN

DETERMINISTICALLY RECOGNIZING EOL-LANGUAGES IN TIME $O(n^{3.81})$

IA

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

AMS(MOS) subject classification scheme (1970): 68A20, 68M05

ACM -Computing Reviews- categories: 5.23, 5.25

Deterministically recognizing EOL-languages in time $\mathcal{O}(n^{3.81})$

by

J. van Leeuwen

ABSTRACT

We show that when appropriate data-structures are defined, Valiant's fast algorithm for the recognition of context-free languages in time $\mathcal{O}(n^{2.81})$ can be modified into an $\mathcal{O}(n^{3.81})$ algorithm for deterministically recognizing EOL-languages, a powerful proper generalization of the context-free languages.

KEY WORDS & PHRASES: *Lindenmayer languages, deterministic recognition, Valiant's algorithm.*

1. INTRODUCTION

About seven years ago A. LINDENMAYER [3] proposed a number of easy mechanisms which could model the parallel behaviour of cellular growth in one-dimensional organisms (filaments). His "type 0" (interaction-less) systems have since led to a study of EOL-grammars which generate languages very much like ordinary context-free grammars do except that in derivations we require that in each step all symbols of an intermediate string are rewritten simultaneously according to productions listed in the grammar. The algorithmic structure of EOL-languages and various generalizations has been actively investigated in the past few years, but instead of documenting this paper with a long list of references we kindly refer the reader to HERMAN & ROZENBERG [1] for further information. EOL-languages are a powerful, proper generalization of the context-free languages strictly included in the indexed languages, containing for example languages like $\{a^{2^n} \mid n \geq 0\}$ and $\{a_1^n \dots a_k^n \mid n \geq 0\}$ (any k).

One of the main structural differences between context-free and EOL-grammars is that in derivation-trees of the latter all paths from the root to contributing leaves must be of equal length. It is no surprise therefore that efficient recognition-algorithms for EOL-languages are somewhat like similar procedures for context-free languages to which an appropriate counting-mechanism has been added.

VAN LEEUWEN [8] showed that EOL-languages can be deterministically recognized in space $\mathcal{O}(\log^3 n)$ (but rather inefficiently in time) and with a different organization in polynomial time (using more space in that case). OPATRNY [4] has analyzed this further and showed that the dynamic programming method underlying Younger's algorithm ([9]) can be modified for EOL-languages and implemented on a multitape Turing machine in time $\mathcal{O}(n^4)$.

Although Opatrny's algorithm is clearly efficient, we shall prove that one can do asymptotically faster on a random-access computer, very much like Valiant observed for context-free languages, thus providing a better bound on the computational complexity of EOL-languages.

We show that appropriate data-structures ("ladders") can be defined and manipulated at a reasonably low cost such that Valiant's fast algorithm for computing the transitive closure of matrices over non-associative domains ([7]) can be applied with only a limited overhead to give an $\mathcal{O}(n^{3.81})$ algorithm for EOL-recognition.

2. SOME PRELIMINARIES.

For standard language-theoretic concepts and notations we refer to HOPCROFT & ULLMAN [2] and to SALOMAA [5].

An EOL-grammar (see[1]) is a 4-tuple $G = \langle V,\Sigma,S,\delta\rangle$ where V,Σ, and S are as usual and δ is a finite substitution from V into V^*. When $\lambda \notin \delta(A)$ for all $A \in V$ then G is called λ-free. $L \subseteq \Sigma^*$ is called an EOL-language when there exists an EOL-grammar $G = \langle V,\Sigma,S,\delta\rangle$ such that $L = \delta^*(S) \cap \Sigma^*$.

Finding efficient recognition-procedures for EOL-languages is complicated by the lack of suitable normal forms that one can use. About the simplest form to which all EOL-grammars can be effectively reduced is expressed in the following result (see e.g. VAN LEEUWEN[8]).

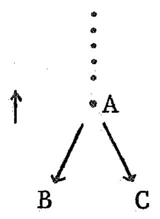
LEMMA 2.1. *Each (λ-free) EOL-language can be generated by an EOL-grammar* $G = \langle V,\Sigma,S,\delta\rangle$ *in which* δ *is* λ*-free and for all* $A \in V$ *and* $w \in V^*$: $w \in \delta(A) \Rightarrow |w| = 1$ *or* $|w| = 2$.

In such an EOL-grammar in normal form one can distinguish symbols which always divide ($w\in\delta(A)\Rightarrow|w|=2$), symbols which never divide ($w\in\delta(A)\Rightarrow|w|=1$), and symbols which may or may not divide ($\delta(A)$ contains words of length 1 and length 2).

In the reduction-algorithm that is given later we shall always do length-preserving reductions

$$\begin{tikzcd}[row sep=small]
 & \vdots \\
\uparrow & A_2 \arrow[d] \\
 & A_1 \arrow[d] \\
 & B
\end{tikzcd}
\qquad B \in \delta(A_1),\quad A_1 \in \delta(A_2),\ldots\ldots$$

"immediately", leaving contracting reductions



BC $\in \delta(A)$,...

as steps of more explicit concern.

In making reductions all the way up to the root of the tree, how far "up" should one expect to go? Although derivations in an EOL-grammar could very well extend arbitrarily far, eliminating periodicities shows that they need never be more than linear in the length of the resulting word (VAN LEEUWEN[8]).

<u>LEMMA 2.2</u>. *For each* EOL-*grammar* G *one can find an (integral) constant* c *such that all non-empty words in the language of* G *at least have one derivation of length* $\leq c \cdot |w|$.

A trivial modification of the EOL-grammar $G = \langle V,\Sigma,S,\delta\rangle$ generating a language L makes that all non-empty words $w \in L$ have a derivation of precisely length $c \cdot |w|$: add a new start-symbol T ($\notin V$ originally) to G and define $\delta(T) = \{T\} \cup \{\delta(S)\}$

3. PREPARING REPRESENTATIONS

Let L be a (λ-free) EOL-language, $G = \langle V,\Sigma,S,\delta\rangle$ an EOL-grammar in normal form generating L such that all $w \in L$ have at least one derivation of length $c \cdot |w|$ in G. Let n be some natural number (to be interpreted as the length of a non-empty word).

A *semi-ladder* (of length cn) is an array-like data-structure

$$F = \begin{array}{|c|}\hline F_{cn} \\ \hline F_{cn-1} \\ \hline \vdots \\ \hline F_1 \\ \hline F_0 \\ \hline \end{array}$$

of directly addressible fields containing pointers to the ordered representation of subsets of V. Defining $E \leq F$ when for all i $E_i \subseteq F_i$ makes the set of semi-ladders into a lattice.

A semi-ladder F (of length cn as before) is called a *ladder* if the following condition on the field of F holds:

$$\textit{for all } i < c.n, \textit{ all } A \in F_i, B$$
$$\textit{if } A \in \delta(B) \textit{ then } B \in F_{i+1}$$

Thus, in a ladder all possible reductions using letter-to-letter rewritings have been made.

As an easy, but useful result we observe:

LEMMA 3.1. *For each semi-ladder* E *there is a unique, smallest ladder* F *containing* E. *Given* E *one can compute* F *in time* $O(n)$.

PROOF. The first assertion follows by observing that the set of ladders (of length cn) is a lattice too. Given E, the following algorithm will compute the smallest ladder F containing it

```
$F_0 \leftarrow E_0$
for i to cn - 1 do ($X \leftarrow \{B \notin F_{i+1} \mid \delta(B) \cap F_i \neq \emptyset\}$;
                    $F_{i+1} \leftarrow$ merge $(F_{i+1}, X)$
                   )                                    □
```

For semi-ladders E and F (of the same length cn) let E + F be the semi-ladder H determined by $H_i = E_i \cup F_i$. Clearly when E and F are ladders, then so is E + F.

LEMMA 3.2. *The sum of two (semi-)ladders is computable in time* $O(n)$.

PROOF. The time needed to merge two fields is bounded by a constant. □

It is more interesting that one can also meaningfully define a product-operation for semi-ladders (although later it will only be applied for ladders).

For semi-ladders E and F let $E \circ F$ be the smallest ladder containing the semi-ladder H defined by $H_0 = \emptyset$, $H_{i+1} = \{A \mid \exists_{B \in E_i, C \in F_i} BC \in \delta(A)\}$.

LEMMA 3.3. *The product of two (semi-)ladders is computable in time* $O(n)$.

PROOF. The following algorithm combines computing H and the method of 3.1. to determine the smallest ladder containing it:

```
$(E \circ F)_0 \leftarrow \emptyset$
$X \leftarrow \emptyset$
for i to cn - 1 do
    ($(E \circ F)_{i+1} \leftarrow$ merge $(\{A \mid \exists_{B \in E_i, C \in F_i} BC \in \delta(A)\}, X)$;
     $X \leftarrow \{A \mid \exists_{B \in (E \circ F)_{i+1}} B \in \delta(A)\}$
    )
```

The time needed for basic instructions is again bounded by a constant. □

The product is non-associative (but distributive over +). The bracketed product of several semi-ladders together is guaranteed to have various empty fields, but explicit use of this information does not pay off in the worst case analysis of the recognition-algorithm that we shall give.

LEMMA 3.4. *The sum of two* $n+1$*-by-*$n+1$ *matrices whose elements are pointers to (semi-)ladders of length* cn *is computable in time* $O(n^3)$, *the product in time* $O(n^{3.81})$.

PROOF. STRASSEN [6] shows that sum and product can be computed in $O(n^2)$ and $O(n^{2.81})$ basic operations over a domain where laws as for ladder-manipulation are satisfied. In 3.2 and 3.3 was shown that basic operations require $O(n)$ time each. □

4. AN EOL-RECOGNITION PROCEDURE

Let L be a (λ-free) EOL-language, $G = \langle V,\Sigma,S,\delta \rangle$ an EOL-grammar in normal form generating L as before. We assume that all $w \in L$ have at least one derivation of length $c \cdot |w|$.

Suppose we wish to determine whether or not $w = a_1 \ldots a_n$ belongs to L. Define a matrix

$$M = \begin{bmatrix} \phi & M_{12} & \phi & \phi & \cdots & \phi \\ \phi & \phi & M_{23} & \phi & & \vdots \\ \phi & \phi & \phi & M_{34} & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & M_{nn+1} \\ \phi & \cdots & \cdots & \cdots & \cdots & \phi \end{bmatrix}$$

in which M_{ii+1} is a pointer to the smallest ladder containing

$$\begin{array}{|c|l} \hline \phi & \text{—— cn} \\ \hline \vdots & \\ \hline \phi & \text{—— 1} \\ \hline a_i & \text{—— 0} \\ \hline \end{array}$$

Preparing the matrix therefore requires $O(n^2)$ time.

Define the transitive closure of M over a non-associative domain in the usual fashion:

$$M^+ = M^{(1)} \cup M^{(2)} \cup M^{(3)} \cup \ldots\ldots$$

where

$$M^{(1)} = M$$

$$M^{(j)} = \sum_{i=1}^{j-1} M^{(j-i)} \circ M^{(i)} .$$

Observe that the entries of M^+ can be computed diagonal-wise just as in Younger's algorithm [9].

<u>THEOREM 4.1.</u> $w \in L$ *if and only if* $S \in (M^+_{1,n+1})_{cn}$.

PROOF. We show by induction on k that for $j \leq cn$: $A \in (M^+_{i\ i+k})_j$ if and only if $A \overset{j}{\Rightarrow} a_i \ldots a_{i+k-1}$.

This is clear for $k = 1$ (by construction). For $k > 1$ we observe that

$$M^+_{i\ i+k} = M^+_{i\ i+k-1} \circ M^+_{i+k-1\ i+k} + \ldots + M^+_{i\ i+j-1} \circ M^+_{i+j-1\ i+k} + $$
$$+ \ldots + M^+_{i\ i+1} \circ M^+_{i+1\ i+k} .$$

If A $(M^+_{i\ i+k})_j$ then $A \in (M^+_{i\ i+j-1} \circ M^+_{i+j-1\ i+k})_j$ for some j. By definition of the product there must be a $j' \leq j$, $B \in (M^+_{i\ i+j-1} \circ M^+_{i+j-1\ i+k})_{j'}$, and $C \in (M^+_{i\ i+j-1})_{j'-1}$, $D \in (M^+_{i+j-1\ i+k})_{j'-1}$ such that

$$A \overset{j-j'}{\Rightarrow} B \text{ (by construction)} \Rightarrow CD \overset{j'-1}{\Rightarrow} (a_i \ldots a_{i+j-2})(a_{i+j-1} \ldots a_{i+k-1})$$

using the induction-hypothesis for C and D.

If $A \overset{j}{\Rightarrow} a_i \ldots a_{i+k-1}$ then revert the same argument to show that necessarily $A \in (M^+_{i\ i+k})_j$. □

With 4.1 the recognition of w is essentially reduced to computing the transitive closure of M.

VALIANT [7] has recently shown that for upper-triangular matrices (like M) the "non-associative" transitive closure can still be computed in a number of basic operations in the order of Strassen's fast matrix-multiplication algorithm (ignoring the usual overhead of recursive programming). Valiant's observation carries over in the present situation, although one has to go through the entire analysis of Valiant's algorithm once again to see exactly where the extra time-factors must be included accounting for the more expensive manipulations of matrix-sums and -products in the various steps.

One easily obtains then

THEOREM 4.2. EOL-*languages can be recognized in time* $O(n^{3.81})$.

It is hard to conclude a practical rather than structural relevance of 4.2, an implementation on a random-access computer-model would invite the same criticisms as Strassen's matrix-multiplication algorithm as a numerical procedure. It shows, nevertheless, that the smallest number of arithmetic operations on elements of the system is bounded by $O(n^{3.81})$, in that way establishing a tight bound on the asymptotic growth of the complexity of EOL-recognition.

REFERENCES

[1] HERMAN, G.T. & G. ROZENBERG, *Developmental systems and languages*, North-Holland Publ. Company, Amsterdam (1975).

[2] HOPCROFT, J.E. & J.D. ULLMAN, *Formal languages and their relation to automata*, Addison-Wesley, Reading, Mass. (1969).

[3] LINDENMAYER, A., *Mathematical models for cellular interactions in development*, J. Theor. Biol. 18(1968) 280-299,300-315.

[4] OPATRNY, J., *The time-complexity of EOL-languages*, presented at the Conference on Formal Languages, Automata, and Development, Noordwijkerhout, April 1-6 (1975).

[5] SALOMAA, A., *Formal languages*, Acad. Press, New York (1973).

[6] STRASSEN, V., *Gaussian elimination is not optimal*, Num. Math 13(1969) 354-356.

[7] VALIANT, L., *General context-free recognition in less than cubic time*, Techn. Rep., Dept. of Computer Science, Carnegie-Mellon University (1974), also: J. Comp. Syst. Sci. 10(1975) 308-315.

[8] VAN LEEUWEN, J., *The tape-complexity of context-independent developmental languages*, J. Comp. Syst. Sci. (to appear).

[9] YOUNGER, D.H., *Recognition and parsing of context-free languages in time* $O(n^3)$, Inf. Control 10(1967) 189-208.