Complexity Arguments in Algebraic Language Theory

by

Helmut Alt
Kurt Mehlhorn

Bericht A 78/10

July 1978

Fachbereich 10 -
Universität des Saarlandes
D-6600 Saarbrücken
W-Germany

Bericht A 78/10

July 1978

Abstract: Each algorithm recognizing any generator of the class of context-free languages requires space $\Omega(\log n)$ and time × space $\Omega(n^2)$.

Resumé: "Chaque algorithme reconnaissant un quelconque générateur du cône rationnel des langages algébriues necessite l'espace $\Omega(\log n)$ et un produit espace x temps en $\Omega(n^2)$."

0. Introduction

A frequently used method in complexity theory is reduction. The idea is the following: Assume, that there are given two formal languages $L_1 \subset \Delta^*$ and $L_2 \subset \Sigma^*$ (Δ, Σ finite alphabets) and a recursive partial function $g : \Delta^* \to \Sigma^*$ with the property:

$$w \in L_1 \iff g(w) \in L_2.$$

We denote this property by $L_1 \underset{g}{\leq} L_2$.

Assume further, that there is an algorithm A to recognize L_2. Then the following algorithm B recognizes L_1 (input w):

1. compute g(w)
2. apply A

So the time (space) requirements of B are those of A plus those to compute g.

Thus upper bounds for the complexity of L_2 turn into upper bounds for L_1 and lower bounds for the complexity of L_1 turn into lower bounds for L_2.

The most well-known application of reduction is to show that certain languages are NP-complete [1].

The importance of reductions between languages in complexity theory delivers a connection to algebraic language theory. One result of the latter theory is that for every generator G of the rational cone of context-free languages there is a certain fixed language E_1 and a simple reduction g with $E_1 \underset{g}{\leq} G$. For this language we will show (again by reduction) lower bounds for space-complexity and the product of time and space complexity. So we have these lower bounds for all generators of the class of context-free languages.

By this we have a complexity criterion that certain languages are non-generators by giving recognition algorithms whose complexities are below the lower bounds mentioned above.

1. The product of time- and space complexity of a multitape turing machine

In the following we mean by 'multitape turing machine' an offline turing machine with one read-only input tape and several work tapes.
We say that it has time complexity $T(n)$ if it performs at most $T(n)$ steps for each input of length n. Space complexity $S(n)$ means that for each input of length n at most $S(n)$ cells of the work tapes are used. For two functions $f,g : N \to N$, where N is the set of natural number, $f = O(g)$ means that there exists a constant c, such that $f(n) \leq c \cdot g(n)$ for almost all $n \in N$. $f = \Omega(g)$ means that there exists a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$ for infinitely many $n \in N$.
For reasons of simplicity we assume that the Turing-machines are deterministic but the proofs and results apply to nondeterministic machines as well. There exists a close connection between the product of time and space complexity of multitape turing machines and the time complexity of one-tape-Turing-machines. In fact the following holds (cf. [3]):

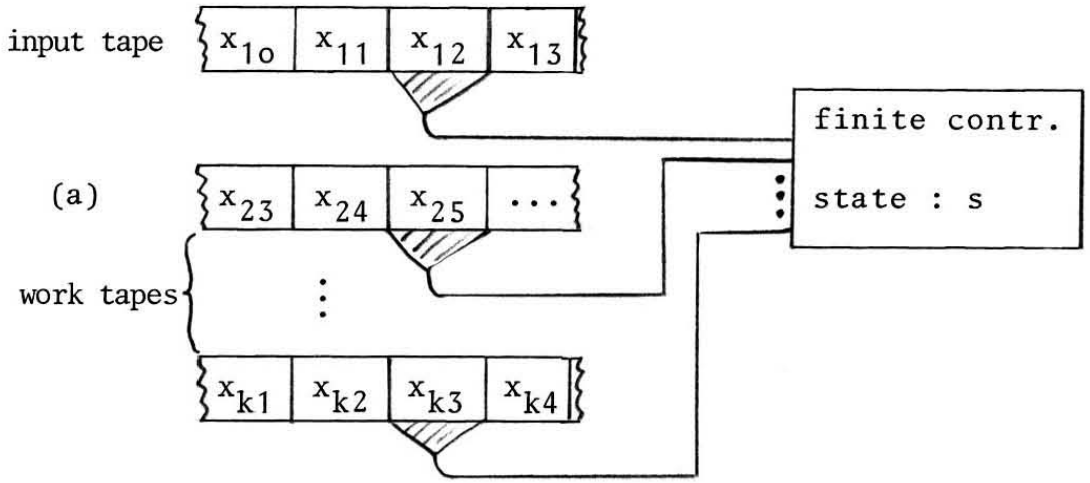
Theorem 1:

To every multitape-Turing machine M with time complexity T and space complexity S one can construct an equivalent one-tape turing machine with time complexity $T \cdot S$.

Proof (for a more detailed version see [3]):

If M has k-1 work tapes, M' has k tracks on its tape.

We simulate an arbitrary configuration of M



by the following configuration of M':

(b)

$x_{1,-1}$	x_{10}	x_{11}	#	x_{12}	x_{13}	x_{14}	x_{15}
x_{22}	x_{23}	x_{24}	#	x_{25}	x_{26}	x_{27}	x_{28}
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
x_{k0}	x_{k1}	x_{k2}	#	x_{k3}	x_{k4}	x_{k5}	x_{k6}

finite contr.
state : s

The special symbol # marks in each track the head position on the corresponding tape of M.
So M' reads exactly the tuple which is read by the k heads of M and can simulate one step of M by overprinting the symbols on the tracks and moving the #'s according to the movement of M's heads.
This operation obviously requires constantly many steps.
After this M' has to bring back all #'s in one cell again according to (b).

This is done by shifting each of the tracks 2 to k. Since on these tracks there are at most S(n) non blank cells, the

time complexity for this is $O(S(n))$. So the time complexity of M' to simulate one step of M is $O(S(n))$ i.e. the whole time complexity is $O(S(n)\cdot T(n))$.
By a well known result in complexity theory (cf.[8]) the constant in the O-term can be reduced to 1.

Since one knows lower bounds on the time complexity of one-tape turing machines ([8],[7]) we have herewith lower bounds on the time-space-product of arbitrary turing machines:

Let be $PAL \subset \{0,1,\not\in\}^*$ the language of palindromes over $\{0,1\}$, i.e.

$$PAL = \{w \not\in w^R \mid w \in \{0,1\}^*\}$$

where $(x_1 \ldots x_n)^R := x_n \ldots x_1$ for all $x_1,\ldots,x_n \in \{0,1\}$

It is known (cf. Th. 1o.7 in [8]) that any one tape Tm. recognizing PAL has time complexity $\Omega(n^2)$.

So we get

Corollary 1:

Let M be a multitape turing machine with time complexity $T(n)$ and space complexity $S(n)$ recognizing PAL. Then

$$T(n)\cdot S(n) = \Omega(n^2).$$

2. Complexity Lower Bounds for the Recognition of Generators

In this chapter we will establish lower bounds on the complexity of algorithms recognizing generators of the class of context-free languages. (i.e. languages G with the property that every context-free language can be obtained by applying a rational transducer (= nondeterministic finite automaton with output) on them. For more details see [5]).

We get the lower bounds by the fact that every generator is "easily" reducible to some fixed language E_1, which is itself "easily" reducible to PAL.

Let $E_1 \subset \{a,b,c,d\}^*$ be the language generated by the grammar

$$S \to aSbSc \mid d$$

Then in [4] the following is shown:

Theorem: For any generator G of the class of context-free languages there exists a regular language K and a homomorphism ϕ such that

$$E_1 = \phi^{-1}(G) \cap K .$$

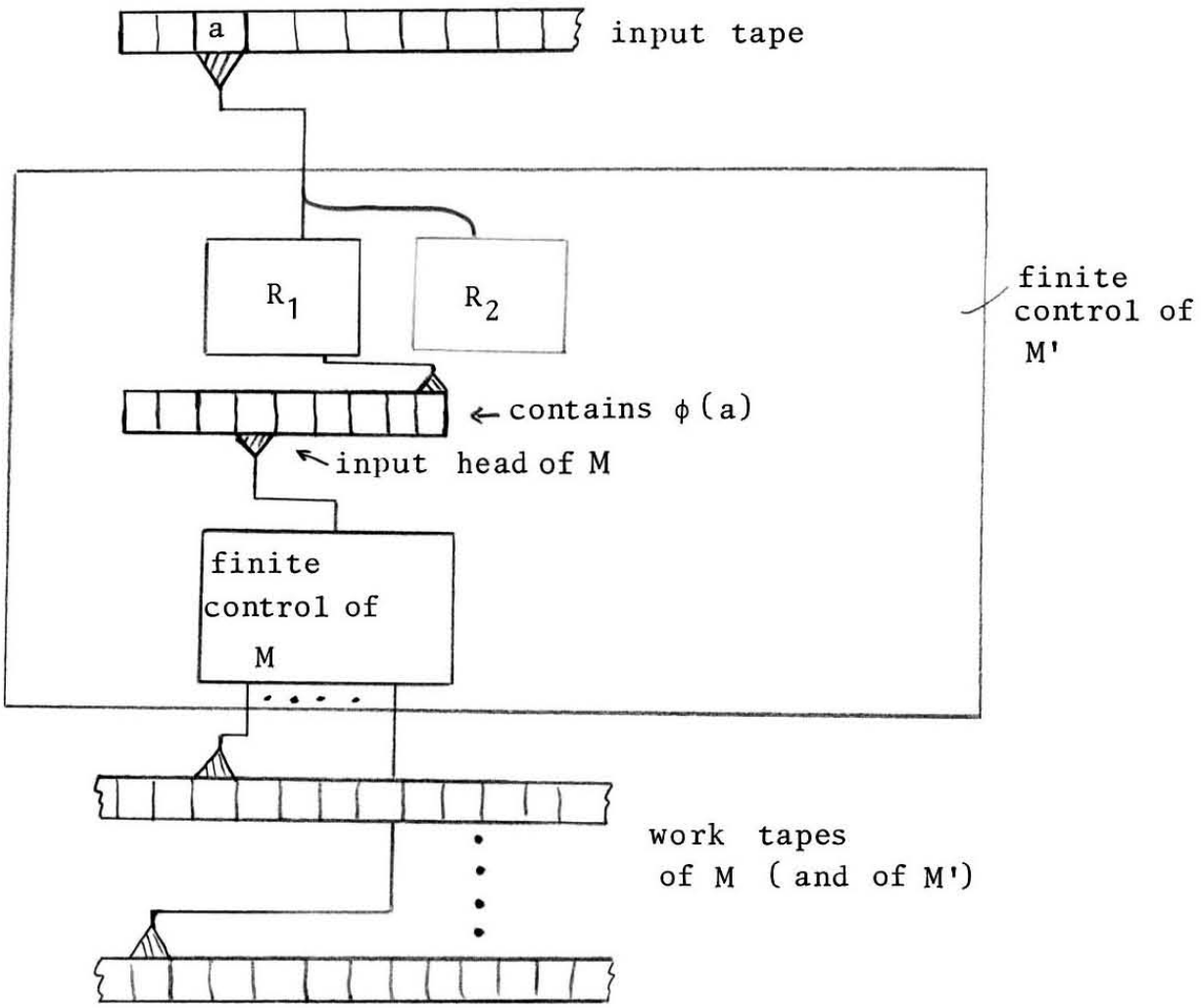
This result delivers us an "easily" computable reduction from any generator G to E_1.

In fact let G be a generator and let ϕ , K be chosen as in the theorem above. Then obviously

$$w \in E_1 \iff \phi(w) \in G \text{ and } w \in K$$

Let R_1 be a finite automaton with output computing ϕ and R_2 a finite automaton recognizing K.

If now M is some (multitape-) turing-machine recognizing G then the following turing machine M' will recognize E_1:



If R_2 has determined that the input is in K the input head is reset to the beginning of the input and M' works in the following way: The finite tape in the finite control will contain $\phi(a)$ if a is the symbol just read by M'. If M''s input head is going to leave this tape at its right (left) border R will produce $\phi(b)$ where b is the symbol right (left) from a in the input.

So the amount of time (space) of M' in addition to the requirement of M is linear (constant) in the length of the input. Since obviously linear time is necessary to recognize any generator G, it holds:

Corollary 2:

If some generator G is recognizable by some Turing machine with time complexity T and space complexity S then E_1 is recognizable with time T and space S too.

With this corollary it is possible to transfer lower bounds for the recognition of E_1 to the recognition of any generator.

In order to establish these lower bounds, we consider the language PAL.

PAL is reducible to E_1, i.e. it holds

Lemma 1: There exists a function

$$g: \{0,1,\not c\}^* \to \{a,b,c,d\}^* \text{ with:}$$

a) $PAL \underset{g}{\leq} E_1$

b) g is computable in linear time and constant space.

Proof: We define g by

i) $g(w) = \varepsilon$ if $w \notin \{0,1\}^* \cdot \{\not c\} \cdot \{0,1\}^*$

ii) $g(w_1 \not c w_2) = \phi_1(w_1 \not c) \cdot \phi_2(w_2)$ where

ϕ_1, ϕ_2 are the homomorphismus defined by

x	0	1	$\not\in$
$\phi_1(x)$	adb	a	d
$\phi_2(x)$	c	bdc	ε (arbitrary)

and we claim: $w \in PAL \iff g(w) \in E_1$ for all $w \in \{0,1,\not\in\}^*$.

Proof:

$\Rightarrow$: Induction on $k = (|w|-1)/2$, where $|w|$ denotes the length of w.

If $k = 0$ then $w = \not\in$

$$\Rightarrow \quad g(w) = d \in E_1.$$

$k \to k+1$ $w \in PAL$ with $|w| > 1$. Then obviously

$$w = \alpha w' \alpha \text{ with } \alpha \in \{0,1\}, w' \in PAL.$$

By induction hypothesis $g(w') \in E_1$.

If $\alpha = 0$ then

$$g(w) = adb\, g(w')c$$

and hence

$$S \Rightarrow aSbSc \overset{*}{\Rightarrow} adbg(w')c = g(w).$$

An analogous statement holds, if $\alpha = 1$.

$\Leftarrow$: Let w be such that $g(w) \in E_1$. We showby induction on $|g(w)|$ that $w \in PAL$. If $|g(w)| \leq 3$ then $g(w) = d$ and hence $w = \not\in$. Assume now $|g(w)| > 3$. Then by definition of g

$$w = w_1 \not\in w_2$$

for some $w_1, w_2 \in \{0,1\}^*$

Since $g(w) \in E_1$ $g(w)$ cannot begin or end with a 'd' and hence $w_1, w_2 \neq \varepsilon$.

Assume that

$$w = 0w_1' \not\in w_2' 1 \qquad \text{for some } w_1', w_2' \in \{0,1\}^*$$

Then

$$g(w) = adb\ g(w')\ bdc \quad , \quad \text{where } w' = w_1' \not\in w_2'$$

Since the first three letters of $g(w)$ can only be generated by

$$S \rightarrow aSbSc \rightarrow adbSc$$

it follows

$$adbSc \overset{*}{\rightarrow} g(w)$$

i.e.

$$S \overset{*}{\rightarrow} g(w')bd$$

This is not possible since obviously no word in E_1 with lenght ≥ 2 ends with a 'd'. In the same way the assumption that $w = 1w_1' \not\in w_2' 0$ leads to a contradiction, such that

$$w = \alpha w_1' \not\in w_2' \alpha \qquad \text{for some } \alpha \in \{0,1\}\ w_1', w_2' \in \{0,1\}^*$$

Assume that $\alpha = 0$.
So $g(w) = adb\ g(w')c$.

Like above it follows that

$$S \overset{*}{\rightarrow} g(w')$$

i.e. $g(w') \in E_1$. By induction hypothesis we get: $w' \in PAL$ and thus

$$w = 0w'0 \in PAL.$$

Analogously one can show that $w \in PAL$ if $\alpha = 1$.

The fact, that g is computable in linear time, is obvious.
Using the same construction as in corollary 2, we get:

Lemma 2:

If E_1 is recognizable with time T and space S the same holds for PAL.
Moreover, if any generator G is recognizable with time T and space S the same holds for PAL.

Now the lower bounds on the recognition of PAL can be transferred to any generator G. By corollary 1 we get:

Theorem 2:

Let G be any generator recognizable by some turing machine with time complexity T and space complexity S. Then

$$T(n) \cdot S(n) = \Omega(n^2).$$

The second lower bound on recognition of generators we get by a result in [2]:

If some non-regular deterministic language is recognizable with space complexity S then

$$S(n) = \Omega(\log n).$$

Especially this result holds for PAL, such that we have:

Theorem 3: If some generator G is recognizable with space complexity S, then

$$S = \Omega(\log n)$$

These lower bounds has been known before for some special generators e.g. for the Dyck-language with two types of brackets D_2 ([2] and [3]).

They provideus with complexity criterions in algebraic languages theory:

In order to show that a language is no generator, it suffices to give a recognition algorithm using less than $\Omega(\log n)$ space or less than $\Omega(n^2)$ time x space.

For example the context-free language

$$\{0,1,\#\}^* \setminus \{\text{bin}(1)\ \#\ \text{bin}(2)\ \#\ldots\#\ \text{bin}(n) \mid n \in N\}$$

where bin(i) is the binary representation of the number i, cannot be a generator since it is recognizable with space $O(\log \log n)$. (cf. [9]).

Furthermore the languages $\{a^n b^n;\ n \in N\}$ and $D_1'^*$ (Dyck-language with one bracket type) cannot be generator, since they are recognizable in space x time $O(n(\log n)^2)$. The ordinary counter automaton with its counter represented in binary does the job. We leave it as an exercise to the reader that both languages are recognizable in space x time $O(n \log n)$. In the first case it is only necessary to observe that one half of the carries propagate only one position, an additional one quarter propagates only two positions,... . In the second case this observation has to be combined with a redundant number representation.

None of the applications above is new. Moreover, having space x time complexity $\Omega(n^2)$ is only a necessary condition for being generator. The criterion seems to fail for the language L_o in [6].

The relevance of this contribution certainly lies in giving a connection between complexity theory and algebraic language theory.

References
==========

[1] Aho, A.V., Hopcroft, J.E., Ullman, J.D.:
The Design and Analysis of Computer Algorithms,
Addison-Wesley, Publ. Comp., 1974

[2] Alt, H., Mehlhorn, K.:
Lower Bounds for the Space Complexity of Context-free
Recognition, 3rd Coll. on Automata, Languages and Programming,
Edinburgh, 1976

[3] Alt, H.:
Einige Beobachtungen zum Produkt von Zeit und Platzbedarf
bei Turingmaschinen,
Universität des Saarlandes, Fachbereich Informatik,
Report A76/09

[4] Beauquier, J.:
Générateurs algébriques non-ambigus,
3rd Colloquium on Automata, Languages and Programming,
Edinburgh 1976

[5] Boasson, L.:
Classification of the Context-Free Languages, MFCS 1977,
Lecture Notes in Computer Science, Vol. 53, pp. 34-43

[6] Boasson, L.:
Un Langage Algebrique Particulier,
to appear in R.A.I.R.O. Informatique Théorique

[7] Hartmanis, J.:
One-Tape Turing Machine Computations, JACM 15, 1968,
pp. 325-339

[8] Hopcroft, J.W., Ullman, J.D.: Formal Languages and their
Relation to Automata, Addison-Wesley, Reading,Mass. 1969

[9] Lewis, P.M., Hartmanis, J., 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, 179-2o2,
1965