RAIRO
INFORMATIQUE THÉORIQUE

S. A. GREIBACH

A note on NSPACE ($\log_2 n$) and substitution

RAIRO – Informatique théorique, tome 11, n° 2 (1977), p. 127-132.
<http://www.numdam.org/item?id=ITA_1977__11_2_127_0>

R.A.I.R.O. Informatique théorique/Theoretical Computer Science
(vol. 11, n° 2, 1977, p. 127 à 132).

A NOTE ON NSPACE($\log_2 n$) AND SUBSTITUTION (1)

par S. A. GREIBACH (2)

Communicated by R. V. BOOK

Abstract. – NSPACE($\log_2 n$) *is closed under substitution into linear context-free languages and under nonerasing substitution into almost simple multihead finite state acceptor languages. Consequently it contains the substitution closure of the on-line one counter and of the linear context-free languages.* NSPACE($\log_2 n$) *is closed under nonerasing substitution into 2-head one-way finite state acceptor languages if and only if it is closed under nonerasing substitution if and only if* NSPACE($\log_2 n$) = $P = NP$.

It is not known whether the class of languages accepted by log space bounded Turing machines is closed under nonerasing substitution – or even under nonerasing homomorphism. However, we observe that we can prove closure in two very limited cases.

In this note we assume that our Turing machines have a two-way read only input tape with endmarkers and one working tape. A machine M *accepts in time* $T(n)$ (*space* $S(n)$) if for each string w accepted by M there is an accepting computation of M on input w taking at most $T(\text{Max } (1, |w|))$ steps (using at most $S(|w|)$ squares of the working tape) (3). We write the language accepted by M as $L(M)$. All machines are assumed nondeterministic unless otherwise specified.

DEFINITION 1 : Let

$\text{NSPACE}(\log_2 n) = \{ L(M) \mid M$ is a Turing machine accepting in space $\log_2 n \}$.

$P = \{ L(M) \mid \exists k \geqq 1$, M is a deterministic Turing machine accepting in time $n^k \}$, and

$NP = \{ L(M) \mid \exists k \geqq 1$, M is a Turing machine accepting in time $n^k \}$.

(1) Received 8-9-76 and in final form 24-9-76. This paper was supported in part by the National Science Foundation under Grant DCR 74-15091.

(2) Department of System Science, University of California, Los Angeles

(3) For a word w, $|w|$ is the length of w.

A multihead finite state acceptor is a device with a finite state control, no working tape, and an input tape with endmarkers and some number of two-way read only heads. We shall use freely the fact that NSPACE($\log_2 n$) is equal to the class of languages accepted by multihead finite state acceptors (*cf.* [2], [7]).

Now we give our formal definitions of substitution and of linear context-free grammars and languages.

DEFINITION 2 : Let Σ be a finite alphabet. A *substitution* τ on Σ^* associates to each a in Σ a language $\tau(a)$ and is extended to Σ^* by $\tau(e) = \{ e \}$, for the empty string e, and $\tau(ax) = \tau(a)\tau(x)$ for a in Σ and x in Σ^*, and to a language L by $\tau(L) = \{ y \mid \text{for some } w \text{ in } L, y \text{ is in } \tau(w) \}$. We call τ *nonerasing* if $\tau(a)$ does not contain e for every a in Σ. For a class of languages L, we call τ an L-substitution if $\tau(a)$ is in L for each a in Σ. For classes of languages L_1 and L_2, we say that L_2 is *closed under substitution (nonerasing substitution) into* L_1, if $\tau(L)$ is in L_2 whenever L is in L_1 and τ is an L_2-substitution (a nonerasing L_2-substitution).

Our first theorem concerns substitution into linear context-free languages.

DEFINITION 3 : A *context-free grammar* is a quadruple $G = (V, \Sigma, P, S)$ where V is a finite vocabulary, Σ is contained in V, S is in $V - \Sigma$ and P is a finite set of productions or rules of the form $Z \to y$ for Z in $V - \Sigma$ and y in V^*. If for each rule $Z \to y$ in P, y contains at most one member of $V - \Sigma$, then G is *linear context-free*. For $Z \to y$ in P, u and v in V^*, we write $uZv \Rightarrow uyv$ and let $\overset{*}{\Rightarrow}$ be the transitive reflexive extension of $\Rightarrow$. The *language generated by* G is

$$L(G) = \{ w \text{ in } \Sigma^* \mid S \overset{*}{\Rightarrow} w \}.$$

We call $L(G)$ a *context-free* language; if G is linear context-free, then $L(G)$ is a *linear context-free language*.

THEOREM 1 : NSPACE($\log_2 n$) *is closed under substitution into linear context-free languages.*

Proof : Let $L \subseteq \Sigma^*$ be a linear context-free language and τ a substitution on Σ such that $\tau(a)$ is in NSPACE($\log_2 n$) for each a in Σ. We can clearly assume that $L = L(G)$ for a grammar $G = (V, \Sigma, P, S)$ all of whose rules are of the forms $Z \to aY$, $Z \to Ya$, $Z \to e$, for a in Σ and Z and Y in $V - \Sigma$. Also each $\tau(a) = L(M_a)$ for a nondeterministic k_a – head finite state acceptor M_a.

We sketch the construction of a multihead finite state acceptor M for $\tau(L)$. This machine will have $5 + \text{Max}(\{ k_a \mid a \text{ in } \Sigma \})$ heads. We distinguish the first 5 heads as E, L_1, L_2, R_1 and R_2.

First notice that using head E whose position will be unimportant, M can perform simple tasks such as : determine whether heads i and j coincide,

determine whether i lies to the left of j, move head i to coincide with j, move head i somewhere right of j. The idea is that head E can be put on the left endmarker and heads i and j moved left and E right in synchronism. In this way the relative locations of i and j can be determined while E holds their places.

If e is in L, M guesses initially whether or not it is looking for a word in $\tau(w) = \{ e \}$; if M is, M can determine ad hoc whether the input tape contains e.

Otherwise, M assumes that it is looking for a word in $\tau(w)$ for some $w \neq e$. Initially, M has heads L_1 and L_2 on the left endmarker and moves R_1 and R_2 to the right endmarker. It has a production register in its finite state control into which it initially puts an arbitrary production of P with left had side S. Then the first cycle starts.

Now suppose that we are at the start of a cycle. First M examines the production in its register. There are three cases; the cases $Z \to aY$ and $Z \to Ya$ are obviously symmetric, so let us discuss $Z \to aY$. If e is in $\tau(a)$, M can nondeterministically replace $Z \to aY$ by any production with left hand side Y and go to the next cycle. If M does not or cannot elect this alternative, then M moves L_1 to coincide with L_2 and then moves L_2 a number of spaces right, checking that L_2 remains left of R_1. Then M moves k_a heads to coincide with L_1 and starts simulating M_a. In this simulation, head L_1 represents the left endmarker of the input tape of M_a, while the square on which L_2 sits is imagined to contain both the rightmost input symbol of M_a and the right endmarker. Each time one of the k_a heads simulating the heads of M_a is moved, head E is used to check whether the head moved now coincides with either L_1 or L_2; this also prevents motion left of L_1 or right of L_2. When and if M_a reaches an accepting configuration, M places in its production register any production with left hand side Y and proceeds to the next cycle. The case of a production $Z \to Ya$ is similar except that here R_2 will mark the right endmarker of M_a and R_1 both the left endmarker and the leftmost input symbol.

A production $Z \to e$ is handled differently. Now M determines whether L_2 and R_1 occupy immediately adjacent positions, R_1 to the right of L_2. If they do, then M has verified that the whole input lies in $\tau(w)$ for some w in $L(G)$, so M accepts. □

The key to the proof of Theorem 1 lies in the fact that a linear context-free language can be accepted by a 2-head nondeterministic finite state acceptor such that no head reads — i.e., observes the contents, not just the relative location — the same square twice and the two heads never read the same square. A similar construction will work for any class of multihead machines such that no input square is read twice and this condition is imposed in some regular way. One class of such machines is the class of simple multihead finite state machines introduced by Ibarra [6], [8]. A *simple* multihead machine has one-way (left-to-right) input heads such that only one head (the *sensing*

head) can distinguish the input symbols; the other heads (the *counting* heads) can only distinguish whether or not they are on an endmarker. We relax the condition slightly to get the family of *almost simple* machines.

DEFINITION 4 : An *almost simple* multihead finite state acceptor has one *sensing* head which moves left-to-right on the input tape and can distinguish the input symbols from each other and from the endmarkers, and any number of *counting* heads, which can move in both directions on the input tape and can distinguish input symbols from endmarkers but not from each other.

The class of almost simple multihead finite state acceptor languages is readily seen to be equal to the class of languages accepted in $\log_2 n$ space by Turing machines with a one-way input tape; however the finite state acceptor formulation is more convenient for our purposes.

THEOREM 2 : NSPACE($\log_2 n$) *is closed under nonerasing substitution into the class of almost simple multihead finite state acceptor languages.*

Proof : This time we consider a language of the form $\tau(L_1)$, where $L_1 \subseteq \Sigma^*$ is the language accepted by an almost simple k-head finite state acceptor M_1 and τ is a nonerasing substitution on Σ^* such that for each a in Σ, $\tau(a) = L(M_a)$ for some k_a head finite state acceptor M_a.

The construction of a multihead finite state acceptor M for $\tau(L_1)$ is similar to the construction in Theorem 1. This time M has $5 + k + \text{Max}(\{ k_a \mid a \text{ in } \Sigma \})$ heads. The first 5 heads are called E, C_1, C_2, L and R.

Again, if e is in $\tau(L_1)$, M accepts e in an ad hoc fashion and otherwise assumes it is verifying that the input is in $\tau(w)$ for some nonempty w in L_1. The sensing head of M_1 moves left-to-right across w. To find M_1's input symbols, M moves heads L and R across the input, marking out substrings guessed to belong to some $\tau(a)$. Machine M simulates M_a as in the proof of Theorem 1, regarding L as the left endmarker of M_a and R as the right. If the simulation is successful (ending in an accepting configuration of M_a), then the head simulating the sensing head of M_1 can behave as if it has just read a. The heads simulating the $k - 1$ counting heads of M_1 need only have $|w|$ squares marked off for them; since τ is nonerasing, $|w|$ is no longer than the input of M. So M first moves head C_1 right to some arbitrary square; if this square is, say, i squares right of the left endmarker it represents a guess that $|w| = i - 1$. Then the heads simulating the counting heads of M_1 will move between the left endmarker and C_1, regarded as the right endmarker for M_1; after one of these heads is moved M uses head E to check that the head has not moved right of C_1. To verify the guess for $|w|$, M moves head C_2 right after each simulation of a machine M_a and at the end compares its position with the location of C_1. □

An on-line one counter acceptor has one-way input head and a counter for working tape. The class of on-line one counter languages is closed under

erasing finite (and in fact regular) substitution and each such language can be accepted by an almost simple finite state acceptor with one sensing and one counting head [3], [4]. An erasing substitution can be effected by an erasing finite substitution followed by a nonerasing substitution. So NSPACE($\log_2 n$) is closed under substitution into on-line one counter languages.

COROLLARY 1 : NSPACE($\log_2 n$) *is closed under substitution into the substitution closure of the linear context-free and the on-line one counter languages.*

COROLLARY 2 : NSPACE($\log_2 n$) *contains the closure under substitution of the linear context-free languages and the on-line one counter languages.*

It is not known how Corollary 2 can be strengthened and in particular whether NSPACE($\log_2 n$) contains all context-free languages. Sudborough showed that NSPACE($\log_2 n$) contains all context-free languages if and only if it contains all languages accepted by nondeterministic polynomially time bounded, $\log_2 n$ space bounded auxiliary pushdown store acceptors [9].

We can see that it is not very likely that Theorem 2 can be significantly strengthened. If we try to consider multihead finite state machines with either two one-way sensing heads or one two-way sensing head and one two-way counting head, we run into the usual P vs NP problem.

THEOREM 3 : *The following statements are equivalent.*

(1) NSPACE($\log_2 n$) = P = NP.

(2) NSPACE($\log_2 n$) *is closed under nonerasing substitution.*

(3) NSPACE($\log_2 n$) *is closed under nonerasing substitution into the class of languages accepted by deterministic 2-head finite state acceptors with two one-way sensing heads.*

(4) NSPACE($\log_2 n$) *is closed under nonerasing substitution into the class of languages accepted by deterministic 2-head finite state acceptors with one two-way sensing head and one two-way counting head.*

Proof : Since NP is closed under nonerasing substitution (as can be seen using the methods of [1]), (1) implies (2). Obviously (2) implies (3) and (4). On the other hand, suppose that either (3) or (4) holds. For any polynomially time bounded Turing machine M, consider the language L_M consisting of all strings of the form $wcw_1c \ldots w_tc$ where $w_1, \ldots, w_t$ are the successive instantaneous descriptions of an accepting computation of M on w, written in an alphabet distinct from the input alphabet of w and c is a new symbol. It is not hard to see that L_M can be accepted by machines of the types described in (3) and (4). Now a nonerasing homomorphism is a particular type of nonerasing finite substitution. Thus if either (3) or (4) hold, NSPACE($\log_2 n$) must contain L_M^c, which is obtained from L_M by turning everything after the first c into c's. If $L_1 \subseteq \Sigma^*$, c is a symbol not in Σ, and f is a nondecreasing function, then L_2

is an *f-representative* of L_1 if $L_2 \subseteq L_1 c^*$ and for each w in L_1, there is an integer $m \leqslant f(|w|)$ such that wc^m is in L_2. If f is a polynomial, we call an f-representative a *polynomial representative*. If NSPACE($\log_2 n$) contains any polynomial representative of L_1, it contains L_1 [5]. Clearly L_M^c is a polynomial representative of $L(M)$. Hence if L_M^c is in NSAPCE($\log_2 n$), then $L(M)$ is also in NSPACE($\log_2 n$). Thus either (3) or (4) implies that *NP* is contained in NSPACE($\log_2 n$) and so (1) holds. □

We can also remark that Theorems 1 and 2 although based on the same proof strategy are independent, since the classes of linear context-free languages and of almost simple multihead finite state acceptor languages are incomparable [5].

REFERENCES

1. R. V. BOOK, S. A. GREIBACH and B. WEGBREIT, *Time- and Tape-Bounded Turing Acceptors and AFLs*, J. Computer System Sciences, **4**, 1970, p. 606-621.
2. S. A. COOK, *Characterizations of Pushdown Machines in Terms of Time-Bounded Computers*, J. Association Computing Machinery, **18**, 1971, p. 4-18.
3. S. A. GREIBACH, *Erasable Context-Free Languages*, Information and Control, **29**, 1975, p. 301-326.
4. S. A. GREIBACH, *A Note on the Recognition of One Counter Languages*, Revue Française d'Automatique, Informatique et Recherche Opérationnelle, R-2, **9**, 1975, p. 5-12.
5. S. A. GREIBACH, *Remarks on the complexity of nondeterministic counter languages*, Theoretical Computer Science, **1**, 1976, 269-289.
6. O. H. IBARRA, *A Note on Semilinear Sets and Bounded-reversal Multihead Pushdown Automata*, Information Processing Letters, **3**, 1974, 25-28.
7. O. H. IBARRA, *On two-way multihead automata*, J. Computer System Sciences, **7**, 1973, p. 28-36.
8. O. H. IBARRA and C. E. KIM, *A Useful Device for Showing the Solvability of Some Decision Problems*, Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, Hershey, Pennsylvania, May, 1976, p. 135-140.
9. I. H. SUDBOROUGH, *On Deterministic Context-Free Languages, Multihead Automata, and the Power of an Auxiliary Pushdown Store*, Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, Hershey, Pennsylvania, May, 1976, p. 141-148.