Theoretical Computer Science 37 (1985) 217-243
North-Holland

APPLICATIONS OF SCHEDULING THEORY TO FORMAL LANGUAGE THEORY

Jakob GONCZAROWSKI

Institute of Mathematics and Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel

Manfred K. WARMUTH*

Computer Science Department, University of California, Santa Cruz, CA 95064, U.S.A.

Communicated by E. Shamir
Received March 1985

Abstract. Context-free grammars are extended to the case where it is required that at each derivation step, a fixed number k of nonterminals must be rewritten in parallel. This way of rewriting constitutes a 'missing link' between context-free rewriting, where only one nonterminal is rewritten in each step ($k = 1$), and E0L rewriting, where always all nonterminals are rewritten. We approach the study of these families by investigating their computational complexity. In both the E0L and CF case, as well as for the case $k = 2$, simple dynamic programming membership algorithms exist (see [13, 22, 29]). We solve the general problem using results from Scheduling Theory.

Rewriting k symbols at each derivation step corresponds to scheduling the corresponding derivation forest on k processors. Using Scheduling Theory techniques, we present dynamic programming membership algorithms that run in polynomial time, for constant k. On the other hand, it is shown that membership is NP-hard if k is a variable of the problem, even when the grammar is fixed. An analogous NP-hardness result is shown for the case where the k symbols to be rewritten are required to be adjacent.

Contents

* This research has been done while the second author visited the Hebrew University of Jerusalem. This research was supported by the United States-Israel Binational Science Foundation, Grant No. 2439/82.

1. Introduction

In Formal Language Theory, one of the major points of investigation into language families is their membership and parsing complexity. Some well-known language families are parsable in polynomial time, such as the context-free languages [8, 29] and E0L languages [22]. Context-free languages are obtained by sequential rewriting, i.e., applying a production to a single symbol, whereas E0L languages (see e.g. [24]) are obtained by rewriting all the symbols in parallel. (Moreover, in E0L systems, also terminal symbols are rewritten.) On the other hand, even modest attempts to extend the grammars that generate these languages escalate the membership complexity to NP-completeness. An example for this is the ET0L family (see (2, 27]), where one allows more than one set of productions. A rewriting step consists of selecting one of these sets, and rewriting all the symbols in parallel, using productions from that set only.

As we have pointed out above, context-free languages are produced by rewriting one symbol at each derivation step, whereas for E0L languages all symbols are rewritten. We look at a 'missing link' between these two language families, where one applies productions to a fixed number, k, of symbols at each step. Two variations of this model have been treated [13]. In one case, we choose any k symbols; in the other case, we insist on these k symbols being adjacent. These grammar families can be viewed as special cases of regular pattern grammars [18, 19] with the patterns $0^*1^k0^*$ and $0^*(10^*)^k$, respectively. For $k \geq 2$, the unrestricted case is well known to generate languages that are not context-free (see e.g. [13]). For the 2-adjacent case, this has been shown recently [4].

In [13], polynomial time or $\log^2$ space parsing algorithms were specified for the case where the grammar is propagating, and $k = 2$, for both variations. The membership problems for arbitrary k (and propagating grammars), however, were only proven to be in NP, by establishing a polynomial upper bound on the length of the shortest derivation for each word in the language.

The algorithms for $k = 2$ presented in [13] are not extendible to the general case. They all do context-free rewriting, while guessing in addition, how many 'partners' a derivation subtree needs from its neighbour subtrees to the left and to the right. For arbitrary k, this is not sufficient; one needs also information about the order of rewritings across subtree boundaries. In the case of unrestricted k-rewriting (not necessarily adjacent) we overcome this difficulty by using results from Scheduling Theory.

We call a derivation a 'k-derivation' if k nonterminals are rewritten at each step (i.e., the unrestricted variation). Now, a derivation forest corresponds to a k-derivation if and only if there exists a k-processor schedule for all internal nodes of F, and this schedule does not have any idle periods. The k nodes rewritten in the i^{th} step of the derivation are exactly the nodes scheduled in the i^{th} slot of the schedule. Thus, the internal nodes of F become the unit-length tasks of the corresponding scheduling problem, and the edges of the forest specify the precedence constraints between the tasks.

The problem of finding optimal k-processor schedules for a system of unit-length tasks subjected to precedence constraints has been studied extensively. For certain very restricted families of precedence graphs and variable k [16, 23], as well as for arbitrary precedence graphs and $k=2$ [3, 9, 10], an optimal schedule can be found in polynomial time. On the other hand, if we allow k to be a variable of the problem instance, then the corresponding decision problem is NP-complete [26], even for special families of precedence graphs [12, 20, 21, 28]. The complexity of the problem for fixed k and arbitrary precedence graphs remains open, even for $k=3$.

Recently, polynomial algorithms have been presented [6, 7, 12] for finding optimal schedules for certain families of graphs and fixed k, where the corresponding problems for arbitrary k are NP-complete [12, 20, 21, 28]. In [5, 6] the notion of *median* was used to find optimal schedules for various kinds of forests and other families of precedence graphs, if k is constant. The median partitions a graph with at least k components into a 'hard portion' and an 'easy portion'. The hard portion consists roughly of the $k-1$ highest (weakly connected) components of the precedence graph, and the easy portion conists of the remaining components. Note that the hard portion contains only a constant number of components if k is constant. The precedence constraints of the easy portion are of no relevance; only its size needs to be considered. Intuitively, finding an optimal schedule reduces thus to finding an optimal schedule for the hard portion.

We use the notion of median, together with the fact that scheduling forests according to height is optimal (see [1, 5, 16]), to develop polynomial algorithms for the membership problem of the (unrestricted) k-rewrite languages if k is constant. As in [13], we assume here that the grammar is propagating. The parameter k appears in the exponent of the running time of our algorithms. This is not surprising, because we show that, if k is a variable of the problem, then the membership problem is NP-hard. An analogous result is given for the case of adjacent rewriting. Our algorithm decides that membership is decidable in polynomial time if k is fixed, even if the grammar is variable. On the other hand, we present fixed grammars,[1] for which the membership problem is NP-hard[2] in the adjacent and unrestricted case, if k is variable.

As a corollary to our reductions, we show that the non-emptiness problem is NP-hard, where k and the grammar are variable, for adjacent as well as unrestricted k-rewriting. It has been shown in [14] that the emptiness problem is polynomial if the grammar is constant and only k is variable.

The plan of this paper is as follows. In Section 2, we define basic notions from Formal Language Theory and Scheduling Theory. In Section 3, we relate scheduling on trees to parse trees of our grammars. This is followed by the polynomial membership algorithm for (unrestricted) k-rewriting, if k and G are constant and

[1] To start off the process of rewriting k symbols at each step, the first sentential form must have at least k nonterminals. To avoid trivial results, we extend our grammars in that case by always starting with an axiom which depends on k.

[2] In our reductions only two terminal symbols are needed. The case of one terminal symbol has been shown to be in polynomial time [14].

G is propagating. This algorithm is then extended in the Earley style [8] to yield a polynomial algorithm even if G is variable. Our algorithms actually solve the parsing problem, because they can be used to construct the derivation for the given word, in polynomial time. In Section 4, we show that the membership problem is NP-hard if k is part of the input and the grammar is fixed, for both versions of rewriting. As corollaries we show that the non-emptiness problem is NP-hard if both k and G are variable. We conclude this paper with a summary of all the results and some open problems.

2. Basic notions and definitions

We assume the reader to be familiar with basic Formal Language Theory, as, e.g., in the scope of [15, 24, 25]. Some notions need, perhaps, an additional explanation. An *alphabet* Σ is a finite set of symbols. A *word* w is a finite sequence of symbols, and $|w|$ stands for its length. The empty word is denoted by λ. A *language* is a set of words. The reflexive and transitive closure of a language L is denoted by the *Kleene Star* and is written as L^*. We will identify a singleton set $\{a\}$ with its element a whenever this does not cause confusion. The cardinality of a set X is denoted by $\#X$.

A *context-free grammar* (CFG) G is a quadruple $\langle \Sigma, P, S, \Delta \rangle$, where Σ is the *alphabet* of G, Δ is the *terminal alphabet* of G, $\Sigma - \Delta$ is the *nonterminal alphabet* of G, $P \subseteq (\Sigma - \Delta) \times \Sigma^*$ is the set of *productions* of G, and $S \in \Sigma - \Delta$ is the *start symbol* of G.

Using standard Formal Language notation, we will write $A \to x$ if $(A, x) \in P$. The length of the longest right-hand side of a production in G is denoted by $\mathrm{Maxr}(G)$.

Let $u_0, \ldots, u_k \in \Sigma^*$, and let $(A_1, w_1), \ldots, (A_k, w_k) \in P$. We then say that $u_0 A_1 u_1 A_2 \ldots A_k u_k$ *directly derives* $u_0 w_1 u_1 w_2 \ldots w_k u_k$ *in* $\langle G, k \rangle$, and we write

$$u_0 A_1 u_1 A_2 \ldots A_k u_k \Rightarrow_{G,k} u_0 w_1 u_1 w_2 \ldots w_k u_k.$$

If $u_1 \ldots u_{k-1} = \lambda$, then we say that $u_0 A_1 A_2 \ldots A_k u_k$ *directly adjacent-derives* $u_0 w_1 w_2 \ldots w_k u_k$ *in* $\langle G, k \rangle$, and we write

$$u_0 A_1 A_2 \ldots A_k u_k \Rrightarrow_{G,k} u_0 w_1 w_2 \ldots w_k u_k.$$

(We shall omit k if $k = 1$, and we write then $\Rightarrow_G$; we shall omit G if it is obvious from the context.)

We denote by $\Rightarrow^*_{G,k}$ and $\Rrightarrow^*_{G,k}$ the reflexive and transitive closure of $\Rightarrow_{G,k}$ and $\Rrightarrow_{G,k}$ respectively. If $u \Rightarrow^*_{G,k} v$ ($u \Rrightarrow^*_{G,k} v$), we say that u *derives* v (u *adjacent-derives* v, respectively) *in* $\langle G, k \rangle$.

A *k-derivation* (*k-adjacent-derivation*, respectively) in G is a sequence of words $w_1, \ldots, w_{l+1}$, such that, for all $1 \leq i \leq l$,

$$w_i \Rightarrow_{G,k} w_{i+1} \quad (w_i \Rrightarrow_{G,k} w_{i+1}, \text{ respectively})$$

together with a mechanism that keeps track of the individual productions applied at each step. (Such a mechanism is necessary, as there might be more than one way to obtain w_{i+1} from w_i.) We shall not specify this mechanism explicitly, in order to keep the definitions concise. The *length* of a k-derivation $w_1, \ldots, w_{l+1}$ is l, and the i^{th} *step* of this derivation is the process of deriving w_{i+1} from w_i.

A *sentential form* (of G) is a word w, such that $S \Rightarrow_G^* w$.

The (*unrestricted*) *k-language of* G is the set

$$U_k(G) = \{w \in \Delta^* : \exists u \in \Sigma^* \text{ such that } S \Rightarrow_G u \text{ and } u \Rightarrow_{G,k}^* w\}.$$

The *adjacent k-language of* G is the set

$$A_k(G) = \{w \in \Delta^* : \exists u \in \Sigma^* \text{ such that } S \Rightarrow_G u \text{ and } u \Rightarrow_{G,k}^* w\}.$$

Note that, in particular, all words directly derived from S are in $U_k(G)$ and $A_k(G)$. We observe that $U_1(G)$ and $A_1(G)$ are both the context-free language defined by the grammar G.

We shall use trees and forests in the usual manner of Formal Language Theory. The reader is assumed to be familiar with the basic notions, such as root, parent, child, ancestor, descendant, internal node, and leaf.

The *depth* of a node v is its distance from the root of its tree, plus 1. The *height* of v is the distance to its furthest descendant. Leaves have thus height zero, and roots have depth one. The *height* of a forest is the maximal height of its roots. $|F|$ denotes the number of nodes in F, whereas $\#F$ denotes the number of trees in F.

If F is a forest, then the *bare forest* of F is the forest obtained by deleting all the leaves in F; it is denoted by Bare(F).

The *child forest* of F is the forest obtained by deleting all the roots from F.

A *derivation forest* is a forest where each internal node is labelled with a nonterminal of the grammar. Furthermore, if a node is labelled with the nonterminal A, and its children's labellings (from left to right) form the word w, then $A \to w$ must be a production of the grammar. In particular, a k-derivation forest is a derivation forest that corresponds to a k-derivation.

2.1. Remark. One has to distinguish the height of a k-derivation forest and the length of a k-derivation; the latter is usually much larger than the former.

Some of these notions are illustrated in the following example.

2.2. Example. Let G be the CFG $\langle\{S, A, B, a, b\}, P, S, \{a, b\}\rangle$, where P consists of the following productions:

$$S \to bAb, \qquad S \to ABB, \qquad S \to S,$$

$$A \to A, \qquad A \to a, \qquad B \to b.$$

Figs. 1 and 2 each show a derivation forest for $SAS \Rightarrow_G^* abbabab$. The forest is not a 3-derivation forest, whereas that in Fig. 2 is a 3-derivation forest.

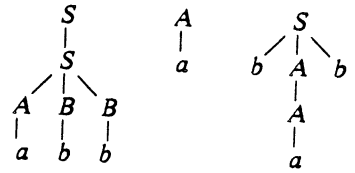

Fig. 1. A derivation forest that is not a 3-derivation forest.

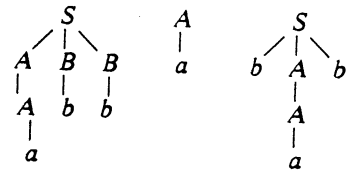

Fig. 2. A derivation forest that is a 3-derivation forest.

A 3-derivation that corresponds to Fig. 2 is

$$\underline{SAS} \Rightarrow_{G,3} \underline{AB}BabAb \Rightarrow_{G,3} A\underline{bB}abAb \Rightarrow_{G,3} abbabab$$

(symbols rewritten are underlined).

We proceed now to define schedules on forests. We assume that there are k processors (corresponding to rewriting k symbols at each derivation step). Every node in a forest is considered to be a unit-length task, where the parent-child relation in the forest specifies the precedence constraints. A k-schedule is then a sequence of slots, where each slot contains up to k tasks, each slot indicating what tasks are to be scheduled in the corresponding unit of time. This is formalized in the following definition.

2.3. Definition. Let F be a forest and let $k \geq 1$. A *k-schedule* of F is a function σ mapping the nodes of F onto the set $\{1, \ldots, l\}$, for some $l \leq |F|$, such that

(i) $1 \leq \#\sigma^{-1}(i) \leq k$ for all $1 \leq k \leq l$,

(ii) for each pair of nodes v_1, v_2 in F, if v_2 is a successor of v_1, then $\sigma(v_2) > \sigma(v_1)$.

The nodes of F are also called *tasks*. l is called the *length* of σ, and $\sigma^{-1}(i)$ is called the ith *slot* of σ.

The tasks of slot i are scheduled at *time* i (i.e., $\#\sigma^{-1}(i)$ out of the k processors are assigned a task at that time). There are $k - \#\sigma^{-1}(i)$ *idle periods* in slot i.

A schedule σ has $p(\sigma)$ idle periods, where

$$p(\sigma) = \sum_{i=1}^{l} (k - \#\sigma^{-1}(i)) = l \cdot k - |F|.$$

A schedule σ is *optimal* for F if there is no schedule σ' of F with $p(\sigma') \leq p(\sigma)$. (Note that optimal schedules have minimal length.) The number of *idle periods* of F, $p(F)$, is the number of idle periods in an optimal schedule for F.

A schedule σ is *perfect* if $p(\sigma) = 0$.

2.4. Example. The following is a 3-schedule with idle periods for the bare forest of the derivation forest of Fig. 2 (rather than showing the nodes, we show the symbols that they are labelled with):

Time slot			
1	2	3	4
S	*B*	*A*	*A*
A	*B*	*A*	
S	*A*		

The following 3-schedule is perfect for the same forest; it correlates with the derivation in Example 3.6:

Time slot		
1	2	3
S	*A*	*A*
A	*B*	*B*
S	*A*	*A*

It is easy to see that there is a natural correspondence between k-derivation forests and perfect k-schedules of their bare forests; if $\nu_1, \ldots, \nu_k$ are the nodes labelled with the symbols that are rewritten in step i, then $\nu_1, \ldots, \nu_k$ appear in slot i of the corresponding schedule.

Using the above notions of Scheduling Theory, we can now give alternate definitions of k-derivation forests and k-languages.

2.5. Lemma. (i) *A derivation forest is a k-derivation forest if and only if its bare forest has a perfect schedule.*

(ii) *Let* $G = \langle \Sigma, P, S, \Delta \rangle$ *be a CFG. A word w is in* $U_k(G)$ *if and only if there exists a derivation tree T of w from S in G, such that* $p(\text{Bare}(T)) = k-1$.

Proof. Part (i) follows from the said above. Part (ii) follows from the observation that all $k-1$ idle periods must be in the first slot of the schedule. Hence, all the other slots do not have any idle periods, i.e. all the derivation steps but the first one must be k-derivation steps. □

Thus, to determine whether $w \in U_k(G)$, we are only interested in the number of idle periods for an optimal schedule for the derivation tree, and not in the schedule itself, particularly not in its length. The computation of p will depend on the heights of the derivation subtrees.

A Highest Level First (HLF) k-schedule for a forest F is obtained as follows:

(1) If F consists of at least k trees, then $\sigma^{-1}(1)$ contains the roots of the k highest trees (for trees of equal height, the choice is arbitrary).

(2) Otherwise, $\sigma^{-1}(1)$ is the set of all the roots.

(3) The tail of the schedule is constructed similarly, with the nodes in $\sigma^{-1}(1)$ deleted from F.

Note that the first schedule of Example 2.4 is not optimal and not HLF, whereas the second one is HLF.

In one of the first papers of Scheduling Theory [16], it was shown that scheduling upside-down forests according to HLF produces optimal schedules. More recently, the same was shown for ordinary forests.

2.6. Theorem ([1, 5]). *Any HLF schedule for a forest is optimal.*

3. Membership in $U_k(G)$ is polynomial

In this section we shall show that UM, the membership problem for $U_k(G)$, is solvable in polynomial time, where G is a constant propagating CFG and k is a constant positive integer. A dynamic programming algorithm (see e.g. [6, 13, 29]) for UM will be developed that is based on results from Scheduling Theory, in particular the notion of median (see [5, 6]). This section is concluded by showing that a variant of the algorithm solves the membership problem in polynomial time even when the grammar is not constant, i.e., if it is part of the input.

We shall first develop the needed scheduling theory results. In [6] it was shown how to schedule a forest on a system of processors where the number of processes is allowed to vary with time. In our case, the number of processes is always k. If there were a unique derivation forest for every word to be parsed, then we could have used the HLF method (Theorem 2.6) to decide whether that forest is also a k-derivation forest. There may, however, be a family of possible derivation forests for a given word, as defined by the grammar. We will proceed bottom-up, keeping track of all those derivation trees that derive subwords of the input word. Even though the number of these trees may be exponential in the size of the input word, we can 'parametrize' these trees, obtaining a polynomial size characterization. These parametrized trees will be called *frames*. A simpler version of this technique is used in the Younger algorithm for context-free membership [29] of a word w, where each derivation subtree is parametrized by the start position of the subword of w that it generates, by the length of the subword, and by the root symbol of this subtree. In our case, we will parametrize the root symbol, the start and end position of the

subword within w, the height of the subtree, and the number of its nodes. In [13] it was shown that for propagating grammars G, the height, and thus also the size, of the subtrees can be polynomially (in fact linearly) bounded in the length of the word to be tested for membership.

The number of idle periods for a collection of frames will be computed using the *median*, which was introduced and used in [5, 6] to present a polynomial time scheduling algorithm for various kinds of forests and other graphs, assuming that the number of processes is constant. Intuitively, all those trees in a forest which are higher than the median are 'hard' to schedule; all the other trees are 'easily' schedulable.

We shall use the median to show that UM is polynomial. There can be only up to $k-1$ trees that are higher than the median. Thus, the portion of a forest that is hard to schedule consists of at most a constant number of trees. As will be seen later, the total number of frames for the relevant trees is polynomially bounded. There will thus be only a polynomial number of collections of frames representing the 'hard' portions that we need to consider. This will allow us to use a dynamic programming scheme, by growing height, which leads to a polynomial time algorithm for UM.

The following definition is a restriction to forests of the definition given in [5, 6].

3.1. Definition. The *k-median* of a forest F is one plus the height of the k^{th} highest tree of F. If F contains less than k trees, then the median is zero.

The *k-high forest* of F is the set of all those trees in F which are strictly higher than the median. The *k-low forest* is the set of the remaining trees.

Whenever k is understood from the context, we shall drop k and write *schedule*, *median*, *high forest*, and *low forest*. The high forest and low forest of a forest F are denoted by $\text{High}(F)$ and $\text{Low}(F)$, respectively.

The following theorem is a restatement of [6, Theorem 3.1], restricted to forests.

3.2. Theorem. *Let F be a forest and σ be a schedule for* $\text{High}(F)$ *with q idle periods. Then there is a schedule σ' for the whole forest F, such that:*

(i) *if $q \geq |\text{Low}(F)|$, then σ' is at most as long as σ;*

(ii) *if $q < |\text{Low}(F)|$, then σ' has idle periods only in its last slot.*

The proof presented in [6] is constructive; in fact, the running time of the algorithm that constructs σ' from σ and $\text{Low}(F)$ is linear in the size of F. Observe that, if σ is optimal for $\text{High}(F)$, then σ' is optimal for all of F. In case (ii), this is immediate because any schedule with idle periods in only one slot is optimal. In case (i), both schedules σ and σ' are of the same length; otherwise, if σ' were shorter than σ, then this would give a schedule for $\text{High}(F)$ that were shorter than σ. Thus, σ' must be a minimum length schedule for F, because σ was such a schedule for a smaller forest, namely $\text{High}(F)$.

Combining Theorems 2.6 and 3.2, we get the following lemma.

3.3. Lemma. *Assume that the HLF schedules for* High(F) *have q idle periods. Then the HLF schedules for F have*

(i) $q - |\text{Low}(F)|$ *idle periods if* $q \geq |\text{Low}(F)|$, *and*

(ii) $-|F| \bmod k$ *idle periods otherwise.*

Proof. We observe first that an HLF schedule for High(F) is at most as long as an HLF schedule for F, because High(F) is a subforest of F, and because HLF schedules are optimal.

For case (i), let σ be an HLF schedule for High(F). By Theorem 3.2(i), it follows that there is a schedule of F which is at most as long as σ. But HLF schedules are optimal. Thus, in particular, any HLF schedule σ' of F satisfies this property. It follows thus that σ and σ' are of equal length. The number of idle periods of σ' is therefore equal to that of σ, minus those idle periods 'filled' with the nodes from Low(F), i.e., (i) holds.

To prove (ii), observe that, by Theorem 3.2 (ii), there is a schedule σ' for F that has idle periods only in its last slot. Such a schedule is optimal; hence, any HLF schedule of F has the same length as σ' and the same number of idle periods. Since the total number of tasks to be scheduled is $|F|$, the last slot of σ' must contain $-|F| \bmod k$ idle periods. □

Assume now that, given a CFG G and a constant k, we want to test whether $w \in U_k(G)$, $|w| = n$.

The following lemma bounds the size of derivation trees, that are relevant to us, polynomially in n. G is required to be *propagating*, i.e., without productions of the form (A, λ). Even though it has not been shown that this is actually a normal form, we will limit ourselves to propagating grammars in the rest of this section.

3.4. Lemma. *Let $G = \langle \Sigma, P, S, \Delta \rangle$ be a CFG, and let $w \in \Delta^*$, $|w| = n$. Then $w \in U_k(G)$ if and only if there exists a k-derivation tree T of w from S in G, such that the height of T is at most $\chi(n) = n \times k^{2k} \# \Sigma^{k(k+1)/2}$,[3] and $|T| \leq n\chi(n)$.*

Proof. The bound on the height was shown in [13]. The bound on the total number of nodes follows now from the propagating property of G; at each tree level, there can be at most n nodes. □

We parametrize now our trees, as outlined before, into *frames*, and start operating on frames rather than trees.

[3] The authors of this paper have proven that $\chi(n)$ can be reduced to $n \times (k + \# \Sigma) \times \# \Sigma^k$ for $U_k(G)$, and to $k \times \Sigma^k$ for $A_k(G)$.

3.5. Definition. Let G be the CFG $\langle \Sigma, P, S, \Delta \rangle$, and let $w = a_1 \dots a_n$, where $a_1, \dots, a_n \in \Delta$.

A *frame* R *(of* w*)* is a quintuple $\langle A, l, r, h, c \rangle$, such that $A \in \Sigma$ is the *root* of R, $1 \leq l \leq r \leq n$, and there is a derivation tree T of $a_l \dots a_r$ from A in G, such that its bare tree has height h and c nodes. If the derivation tree is of height zero, i.e. A is a terminal symbol, then $c = 0$ and $h = -1$.

A tree T as above is called a *frame tree* for R.

The *height* of R is h; the *size* of R, denoted $|R|$, is c. An ordered set $\mathcal{R}$ of frames is called a *frame collection*. The *height* of $\mathcal{R}$ is the maximum of the frame heights in $\mathcal{R}$. The *size* of $\mathcal{R}$ is the sum of the sizes of the frames in $\mathcal{R}$.

If F is a forest, such that the ith tree in F is a frame tree for the ith frame in $\mathcal{R}$, for $1 \leq i \leq \# F = \# \mathcal{R}$, then F is called a *frame forest of* $\mathcal{R}$.

3.6. Example. The forest of Fig. 1 is a frame forest for the frame collection

$$\{\langle S, 1, 3, 2, 5 \rangle, \langle A, 4, 4, 0, 1 \rangle, \langle S, 5, 7, 2, 3 \rangle\}.$$

The notions of median, high collection (high forest) and low collection (low forest) carry over from forests to frame collections in the obvious way. In particular,

$$p(\mathcal{R}) = \min\{p(\text{Bare}(F)) : F \text{ is a frame forest of } \mathcal{R}\}.$$

The following lemma is a restatement of Lemma 2.5, for frames.

3.7. Lemma. *Let* $G = \langle \Sigma, P, S, \Delta \rangle$ *be a CFG. A word* w *is in* $U_k(G)$ *if and only if there exists a frame* $R = \langle S, 1, |w|, h, c \rangle$*, for some* h *and* c*, such that* $p(R) = k - 1$.

We can now redefine $U_k(G)$ in terms of frames, after introducing the analog of child forests.

3.8. Definition. Let $R = \langle A, l, r, h, c \rangle$ be a frame of a word w, and let $\mathcal{R} = \{R_1, \dots, R_j\}$ be a frame collection of w, where $R_i = \langle A_i, l_i, r_i, h_i, c_i \rangle$ for all $1 < i \leq j$. We say that $\mathcal{R}$ is a *child collection* of R if: $A \to A_1 \dots A_j \in P$, $l = l_1$, $r_j = r$, and $l_i = r_{i-1} + 1$ for all $2 \leq i \leq j$, $h = 1 + \max\{h_1, \dots, h_j\}$, and $c = 1 + \sum_{i=1}^{j} c_i$.

A *child collection* of a frame collection $\mathcal{R}$ is obtained by choosing a child collection for each of the frames in $\mathcal{R}$ and by taking their union.

By Lemma 3.4, we have only to consider those frames $R = \langle A, l, r, h, c \rangle$ with $h \leq \chi(n)$ and $c \leq n\chi(n)$, as G is propagating. Clearly, there are at most $\# \Sigma$ choices for A, and n choices for each l and r. Since χ is a linear function, we get the following bound.

3.9. Corollary. *There are only* $O(n^5)$ *frames which have to be computed while parsing a word of length* n.

Our main goal is now to show that the number of idle periods of each frame of a word w is computable in polynomial time. Using Lemma 3.7 and Corollary 3.9, we get thus a membership algorithm for $U_k(G)$, that operates in polynomial time, if k and G are constant.

To compute the number of idle periods of a frame, we need to compute the number of idle periods of various frame collections. For this purpose we restate Lemma 3.3, using the frame notation.

3.10. Corollary. *Let $\mathcal{R}$ be a frame collection. Then*

$$p(\mathcal{R}) = \begin{cases} p(\text{High}(\mathcal{R})) - |\text{Low}(\mathcal{R})| & \text{if } p(\text{High}(\mathcal{R})) \geq |\text{Low}(\mathcal{R})|, \\ -|\mathcal{R}| \bmod k & \text{otherwise.} \end{cases}$$

Note that $|\text{Low}(\mathcal{R})|$ can easily be computed from $\mathcal{R}$; we just have to sum up the last components of the frames in $\text{Low}(\mathcal{R})$. This operation is linear in $\#\mathcal{R}$. But $\#\mathcal{R}$ is bounded by the length of the word, w, that we want to test for membership; the propagating property promises that each frame derives at least one symbol in w. Thus, Corollary 3.10 implies that computing $p(\mathcal{R})$ reduces to computing $p(\text{High}(\mathcal{R}))$, as long as $\mathcal{R} \neq \text{High}(\mathcal{R})$.

By its definition, a high collection consists of at most $k-1$ frames. It follows thus from Corollary 3.9 that, in order to parse a word of length n, one has to compute the idle periods of at most $O(n^{5(k-1)})$ high collections (we recall that k is constant).

The algorithm that we are developing here constructs all the frames using a dynamic programming schema. Then, the number of idle periods for each possible high collection is computed, again by dynamic programming. This is done by recurring on the frame height, as shown below, and by then applying Corollary 3.10 to reduce the resulting frame collection to width $\leq k-1$.

3.11. Lemma. *Let j be the number of frames in a frame collection $\mathcal{Q}$, where $\mathcal{Q} = \text{High}(\mathcal{Q})$. Then*

$$P(\mathcal{Q}) = \begin{cases} 0 & \text{if } \mathcal{Q} \text{ is empty,} \\ k - j + \min\{p(\mathcal{R}') : \mathcal{R}' \text{ is a child collection of } \mathcal{Q}\} & \text{otherwise.} \end{cases}$$

Proof. Obviously, the lemma holds for the empty frame collections. Let thus $\mathcal{Q}$ be nonempty. By the definition of a high collection, $j \leq k-1$. Hence, the first slot of an HLF schedule for $\mathcal{Q}$ clearly contains all the 'roots' of $\mathcal{Q}$, i.e., the first slot has $k-j$ idle periods. We recall that, by Theorem 2.6, HLF schedules are optimal. In the remaining slots of the schedule, there are thus $p(\mathcal{R}')$ idle periods, where $\mathcal{R}'$ is the child collection of $\mathcal{Q}$ with the minimum number of idle periods. □

Observe that $p(\mathcal{R}')$ can itself be computed as outlined in Corollary 3.10.

We can now present the membership algorithm for $U_k(G)$. The algorithm first constructs, via dynamic programming, the set of all the frames for the input word, w. Then it computes the number of idle periods, looping on all the candidates $\mathcal{Q}$ for high collections (i.e., all collections of up to $k-1$ frames), by increasing height, using the recurrences stated in Corollary 3.10 and Lemma 3.11. Finally, the algorithm tests whether there is a frame that *covers* all of w, i.e., that is of the form $\langle A_1, 1, |w|, h, c\rangle$, where A_1 is the start symbol, and that has exactly $k-1$ idle periods; by Lemma 3.7, $w \in U_k(G)$ if and only if there is such a frame.

Algorithm UM

Given: A propagating CFG $G = \langle \Sigma, P, A_1, \Delta\rangle$ and an integer $k > 0$, where $\Sigma = \{A_1, \ldots, A_m\}$.

Input: A word $w \in \Delta^*$, $w = A_{p_1} \ldots A_{p_n}$.

Output: **accept** if $w \in U_k(G)$, otherwise **reject**.

begin
comment test all words directly derived from S;
if $A_1 \to w$ **then accept**;
comment construct all the frames of w, of height h;
for $i := 1$ **to** n **do** $\langle A_{p_i}, i, i, -1, 0\rangle$ is a frame;
for $h := 0$ **to** $\chi(n)$ **do**
for all $(A, B_1 \ldots B_j) \in P$ **do**
for all $1 \leq l_0 \leq \ldots \leq l_j \leq n$ **do**
for all $h_1, \ldots, h_j$ **with** $\max\{h_1, \ldots, h_j\} = h-1$ **do**
for all $0 \leq c_1, \ldots, c_j \leq n\chi(n)$ **do**
if $\langle B_i, l_{i-1}, l_i, h_i, c_i\rangle$ is a frame or $1 \leq i \leq j$
then $\langle A, l_0, l_j, h, c_1 + c_2 + \cdots + c_j + 1\rangle$ is a frame;
comment compute the number of idle periods for all collections of up to $k-1$ frames;
$p(\{\}) := 0$;
for $h := -1$ **to** $\chi(n)$ **do**
for all frame collections $\mathcal{R}$ of height h, consisting of up to $k-1$ frames, each of positive height **do**
begin
$q := \infty$;
for all child collections $\mathcal{R}'$ of $\mathcal{R}$ **do**
begin
$\mathcal{Q} := \text{High}(\mathcal{R}')$;
comment Since the height of $\mathcal{Q}'$ is $h-1$, we can recur on $p(\mathcal{Q}')$;
if $p(\mathcal{Q}') \geq |\text{Low}(\mathcal{R}')|$
then $p(\mathcal{R}') := p(\mathcal{Q}') - |\text{Low}(\mathcal{R}')|$
else $p(\mathcal{R}') := -|\mathcal{R}'| \bmod k$;
$q := \min(q, p(\mathcal{R}'))$;
end;

$p(\mathcal{R}) := k - \#\mathcal{R} + q$;

end;

comment This is the membership test;

for $h := 1$ **to** $\chi(n)$ **do**

for $c := 2$ **to** $n \times \chi(n)$ **do**

if $\langle A_1, 1, n, h, c\rangle$ is a frame **and** $p(\langle A_1, 1, n, h, c\rangle) = k - 1$

then accept

else reject

end;

3.12. Theorem. *The above algorithm for* UM *runs in time polynomial in* n, $O(n^{5(k-1)(\mathrm{Maxr}(G)+1)+1})$, *if both* k *and* G *are constant.*

Proof. Determining the frame takes time $O(n^{4\mathrm{Maxr}(G)+2})$; 1 in the exponent for h; $\mathrm{Maxr}(G)+1$ for $l_0, \ldots, l_j$; $\mathrm{Maxr}(G)$ for $h_1, \ldots, h_j$; $2\,\mathrm{Maxr}(G)$ for $c_1, \ldots, c_j$ (see Lemma 3.4). Enumerating the tuples takes time $O(n^{5(k-1)(\mathrm{Maxr}(G)+1)+1})$: 1 in the exponent for h, $5(k-1)$ for the tuples specifying $\mathcal{R}$ (this is the total number of high collections that have to be considered); there are up to $n^{(k-1)\mathrm{Maxr}(G)}$ frames in a child collection of a collection of $(k-1)$ frames, hence $n^{5(k-1)\mathrm{Maxr}(G)}$ possible child collections; i.e., the total is $n^{5(k-1)(\mathrm{Maxr}(G)+1)+1}$. This is the dominant term, and thus the theorem holds. □

We now proceed to prove that the membership problem for $U_k(G)$ is still polynomial when G is part of the input.

3.13. Theorem. *Let* k *be a fixed positive integer. Then it can be decided in polynomial time whether* $w \in U_k(G)$, *for any propagating CFG* G *and word* w.

Proof. Observe first that the function $\chi(n)$ (see Lemma 3.4) is polynomial in the size of the grammar (actually in the size of its nonterminal alphabet). Hence, the number of frames is polynomial in the size of w and G. We notice that there are two flaws in the UM algorithm that cause it not to be polynomial in the size of G. One is the fact that we have $\mathrm{Maxr}(G)$ nested loops while computing all the frames. The other is the computation of all the child collections (for any collection of up to $k-1$ frames, there exist up to $n^{5(k-1)\mathrm{Maxr}(G)}$ child collections). We can, however, also overcome these difficulties by using dynamic programming, similarly as in Earley's algorithm (see [8]). This is done by first computing the set of extended frames of w in polynomial time. These extended frames have two additional components, a high collection and a median, and they are defined below. Using extended frames, we can easily compute the number of idle periods of the candidates for high collections. The acceptance test is analogous to that for constant G.

An *extended frame* E is a 7-tuple $\langle A, l, r, h, c, \mathcal{H}, \mu\rangle$, such that: $R = \langle A, l, r, h, c\rangle$ is a frame, and there is a child collection of R with median μ and high collection $\mathcal{H}$. (Note that $\mathcal{H}$ consists of ordinary frames.)

E is also called an *extended frame* of the frame R.

Extended frame collections are defined accordingly.

Note that the total number of extended frames is polynomial in the size of G and w.

Let $w = A_{p_1} \dots A_{p_n}$. To compute the set of extended frames for w, we make use of tuples $[A \to B_1 \dots B_m, i, l, r, h, c, \mathcal{H}, \mu]$, where $A \to B_1 \dots B_m$ is the production used at the 'root' of the frame, i represents the dot in Earley's tuples [8]; it indicates that we have already processed frames for $B_1, \dots, B_i$; i.e., there exists a frame collection $\mathcal{R}$ with $\#\mathcal{R} = i$, the j^{th} frame in $\mathcal{R}$ has B_j as its root, for $1 \leq j \leq i$, $\mathcal{R}$ covers $A_{p_l} \dots A_{p_r}$, the height of $\mathcal{R}$ is h, $|\mathcal{R}| = c$, $\text{High}(\mathcal{R}) = \mathcal{H}$, and the median of $\mathcal{R}$ is μ.

We now show how all the tuples, frames, and extended frames can be found in polynomial time. The frames of height -1 are determined as before. For all productions $A \to B_1 \dots B_m$,

(1) if $R = \langle B_1, l, r', h', c'\rangle$ is a frame, then $[A \to B_1 \dots B_m, 1, l, r', h', c', \mathcal{H}, 0]$ is a tuple, where

$$\mathcal{H} = \begin{cases} R & \text{if } h' > 0, \\ \{\} & \text{otherwise.} \end{cases}$$

(2) For all $2 \leq i \leq m$, if $[A \to B_1 \dots B_m, i-1, l, r', h', c', \mathcal{H}', \mu']$ is a tuple, and $\langle B_i, r'+1, r, h'', c''\rangle$ is a frame, then $[A \to B_1 \dots B_m, i, l, r, \max(h', h''), c'+c'', \mathcal{H}, \mu]$ is a tuple, where μ is the maximum of μ' and the median of $\mathcal{H}' \cup R$, and $\mathcal{H} = \{Q \in \mathcal{H}' \cup R : \text{the height of } Q \text{ is greater than } \mu\}$.

(3) If $[A \to B_1 \dots B_m, m, l, r, h, c, \mathcal{H}, \mu]$ is a tuple, then $\langle A, l, r, h+1, c+1, \mathcal{H}, \mu\rangle$ is an extended frame, and $\langle A, l, r, h+1, c+1\rangle$ is a frame.

Assume now that all frames of height h have been found. Then, using the above computation, all frames and extended frames of height $h+1$ can be found (see (3)). Note that the number of tuples is polynomial in the size of G and w. Hence, the above computation is polynomial for a fixed h. Since the height of those frames that we have to consider is bounded by $\chi(n)$ (see Lemma 3.4), it takes polynomial time to find all frames and extended frames.

We proceed now to compute the number of idle periods for each possible extended high collection. The UM algorithm achieved this by recurring from the high collection $\mathcal{R}$ to each possible child collection $\mathcal{R}'$, and from there to the high collection $\mathcal{Q}$ of $\mathcal{R}'$. We shall attempt to recur from the high collection candidate, $\mathcal{R}$, directly to each $\mathcal{Q}$ (each possible high collection of a child collection).

Note that all the child collections of an extended frame $\langle A, l, r, h, c, \mathcal{H}, \mu\rangle$ have the same high collection, $\mathcal{H}$. Hence, all the child collections of an extended frame collection have also the same high collection, which can be obtained from the individual high collections and the medians, as follows.

Let $\mathscr{E} = \{E_1, \ldots, E_m\}$ be an extended frame collection, where $E_i = \langle B_i, l_i, r_i, h_i, c_i, \mathscr{H}_i, \mu_i \rangle$, for $1 \leq i \leq m$, and let $\mathscr{H} = \bigcup_{i=1}^{m} \mathscr{H}_i$. Let μ_0 be the median of $\mathscr{H}$. Then the median of the high collection of all the child collections of $\mathscr{E}$ is

$$\mu = \max\{\mu_i : 0 \leq i \leq m\}.$$

The desired high collection, $\mathscr{Q}$, is the set of all those frames in $\mathscr{H}$ that are higher than μ.

In order to compute the number of idle periods we operate on collections of up to $k-1$ extended frames. The high collection component of each extended frame contains up to $k-1$ frames. Hence, $\#\mathscr{H} < k^2$. It follows that the median, μ, and the high collection, $\mathscr{Q}$, can be found in constant time. The size of the low collection can be computed in constant time after we have found $\mathscr{Q}$; it is

$$r = |\mathscr{E}| - \#\mathscr{E} - |\mathscr{Q}|,$$

because $\#\mathscr{E}$ roots were removed from $\mathscr{E}$ to arrive at the child collection.

We are now ready to rewrite the recurrence of Corollary 3.10. Let $\mathscr{E}$, $\mathscr{Q}$, and r be as above.

$$p(\mathscr{E}) = \begin{cases} p(\mathscr{Q}) - r & \text{if } p(\mathscr{Q}) > r, \\ -(|\mathscr{Q}| + r) \bmod k & \text{otherwise.} \end{cases}$$

Let $\mathscr{R}$ be a frame collection. Then

$$p(\mathscr{R}) = \min\{p(\mathscr{E}) : \mathscr{E} \text{ is an extended frame collection of } \mathscr{R}\}.$$

It is now easy to see that the above recurrence can be used in a dynamic programming scheme to compute the number of idle periods for all collections of up to $k-1$ frames and extended frames. We do this computation according to increasing height. Since the number of frames and extended frames, as well as the height, is polynomially bounded in the size of G and w, the resulting algorithm runs in polynomial time. Hence, the theorem holds. □

3.14. Remark. Like the Earley [8] and the Younger [29] algorithm, Algorithm UM does not only solve the membership problem; the information collected for an input word w can be used to construct an appropriate k-derivation $x_1, \ldots, x_l = w$, where $S \to x_1 \in P$. It is easy to see that the time to do this is bounded by the running time of Algorithm UM. Similarly, for variable grammars, the algorithm outlined in Theorem 3.13 also yields an appropriate k-derivation.

To see how to retrieve a schedule (i.e. k-derivation), the reader is also referred to [6]. It is shown there how to schedule a forest when the number of processors may vary with time.

4. NP-hard problems

It was shown in Section 3 that the membership problem for $U_k(G)$ is polynomial for constant k, even if G is part of the input. We shall investigate the case where k is variable and G is constant. Let $G=\langle \Sigma, P, S, \Delta\rangle$. We first observe that, if G is fixed, then the membership problem for $U_k(G)$ is polynomial. This is true because for all $k > \text{Maxr}(G)$,

$$U_k(G) = \{w \in \Delta^* : S \to w \in P\}.$$

Membership is thus trivial in these cases. If $k \leq \text{Maxr}(G)$, then k is bounded by a constant, since G is constant. Hence, membership is polynomial, because we can apply the UM algorithm. In order to make full use of the fact that k is variable, we look at k-derivations that start with S^k. Note that the results from Section 3 hold also if we start rewriting from an axiom, ω; we just add a production $S \to \omega$ to the grammar, where S is the start symbol.

As we shall see in this section, the complexity is changed drastically if k is variable, even if G is constant; both the unrestricted and the adjacent problems are NP-hard.

Let $G=\langle \Sigma, P, S, \Delta\rangle$ be a CFG. The language

$$\text{UX}_k(G) = \{w \in \Delta^* : S^k \Rightarrow^*_{G,k} w\}$$

is called the *extended k-language of G*.

The *Unrestricted Extended Membership* problem (UXM) is now stated as follows: Given a positive integer k and a word w, is w in $\text{UX}_k(G)$, for a fixed CFG G?

We shall prove the NP-hardness of UXM by reducing to it the problem of 1-1-1 *Scheduling* (1S) [11].

An instance $\langle \mathcal{J}, t\rangle$ of 1S is a set $\mathcal{J}$ of q triples and an integer t such that $1 \leq t \leq q$. A triplet consists of three tasks with release times in the range $[1, 3t]$. Then the following question is NP-complete (see [11]): Is it possible to schedule, on one processor, the tasks from t triplets in slots 1 through $3t$, such that no task is scheduled before its release time?[4]

4.1. Example. Let $\mathcal{J}$ have tree triplets of tasks

$$(\tau, \tau_2, \tau_3), \quad (\tau_4, \tau_5, \tau_6), \quad (\tau_7, \tau_8, \tau_9)$$

with respective release times

$$(3, 4, 6), \quad (1, 2, 4), \quad (1, 5, 6).$$

Then the following one-processor schedule is a solution to $\langle \mathcal{J}, 2\rangle$: $\tau_4, \tau_5, \tau_1, \tau_2, \tau_6, \tau_3$.

We shall now present the grammar G to be used in the NP-hardness construction. Let

$$G = \langle \{S, B, D, N, W, a, b, c, d, e, f\}, P, S, \{a, b, c, d, e, f\}\rangle,$$

[4] A task with release time τ must be scheduled in slot τ or later.

where P consists of the following productions:

$$S \to aN, \quad S \to bN, \quad S \to c, \quad N \to aN, \quad N \to bN, \quad N \to c,$$
$$S \to aW, \quad S \to BW, \quad B \to b, \quad W \to aW, \quad W \to BW, \quad W \to c,$$
$$S \to dD, \quad S \to e, \quad D \to dD, \quad D \to e,$$
$$S \to f.$$

G generates four kinds of trees. One derives f from S, and another one derives a word in d^*e from S. There are, however, two kinds of trees deriving words in $\{a, b\}^*c$, depending on the first production chosen. If the first production was $S \to aN$, then the derivation tree is a *narrow* tree; otherwise, if $S \to aW$ was chosen, it is a *wide* tree. Every word in $\{a, b\}^*c$ has exactly one wide and one narrow tree. For an example of a narrow and a wide tree, see Fig. 3.

We shall now describe the reduction. Let $\langle \mathcal{J}, t \rangle$ be an instance of $1|S|$, where $\mathcal{J} = \{J_1, \ldots, J_q\}$. Without loss of generality we may assume that for each triplet in $\mathcal{J}$, all the release times are different. Otherwise, if there are two tasks τ_1 and τ_2 with identical release times, we may assume that τ_1 is always scheduled before τ_2. We may thus increase the release time of τ_2 by 1.

Let now J_i have release times (r_1, r_2, r_3), in increasing order. Then J_i is encoded in the word

$$w_i = a^{r_1 - 1} b a^{r_2 - r_1 - 1} b a^{r_3 - r_2 - 1} b a^{3t - r_3} c.$$

The corresponding instance of UXM is now the pair $\langle w, k \rangle$, where

$$k = 3q + 1 \quad \text{and} \quad w = f(d^{3t}e)^{2q} w_1 \ldots w_q.$$

Obviously, $\langle w, k \rangle$ can be constructed in polynomial time from $\langle \mathcal{J}, t \rangle$, and $\langle w, k \rangle$ is polynomial in the size of $\langle \mathcal{J}, t \rangle$. The following example shows the instance of $\langle w, k \rangle$ that corresponds to $\langle \mathcal{J}, t \rangle$ of Example 4.1, and then a k-derivation forest of w.

4.2. Example. Let $\langle \mathcal{J}, t \rangle$ be as in Example 4.1. Then $k = 10$ and

$$w = f(d^6e)^6 aabbabcbbabaacbaaabbc.$$

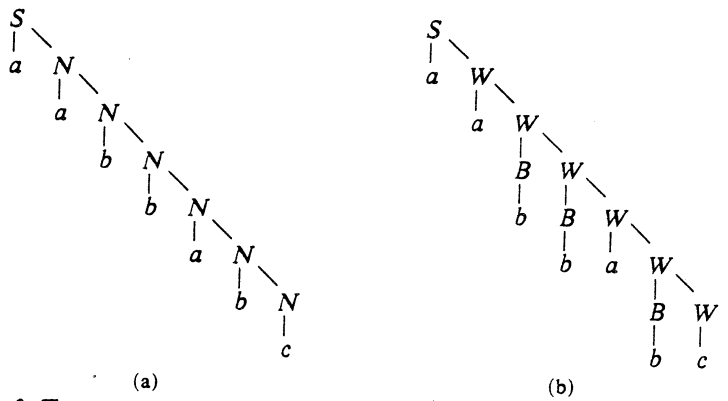

Fig. 3. The two trees for the word a^2b^2abc. (a) The narrow tree. (b) The wide tree.

A k-derivation forest is shown in Fig. 4. The corresponding k-derivation is then

$(\underline{S}^{10}, f(d\underline{D})^6 a\underline{WBW}b\underline{N},$

$f(d^2\underline{D})^6 aa\underline{W}b\underline{BW}ba\underline{N},$

$f(d^3\underline{D})^6 aa\underline{BW}bba\underline{W}baa\underline{N},$

$f(d^4\underline{D})^6 aab\underline{BW}bbaB\underline{W}baaa\underline{N},$

$f(d^5\underline{D})^6 aabba\underline{W}bba\underline{B}a\underline{W}baaab\underline{N},$

$f(d^6D)^6 aabba\underline{BW}bbabaa\underline{W}baaabb\underline{W},$

$f(d^6e)^6 aabbabcbbabaacbaaabbc).$

(The symbols to be rewritten are underlined.)

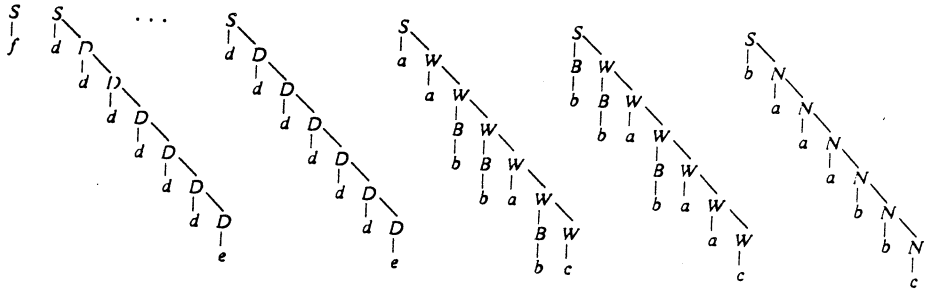

Fig. 4. A k-derivation for w.

We now proceed to prove the NP-hardness result for the unrestricted case.

4.3. Theorem. UXM *is* NP-*hard.*

Proof. Given an instance $\langle \mathcal{J}, t\rangle$ of 1S, we need to show that $\langle \mathcal{J}, t\rangle$ is a solution to 1S if and only if $\langle w, k\rangle$ is a solution to UXM, where w and k are obtained from $\langle \mathcal{J}, t\rangle$ as described above.

Let now $\mathcal{J}'$ be a solution to $\langle \mathcal{J}, t\rangle$. We shall first present a derivation forest F of w from S^k. Then we will see that there is a k-derivation for F.

We let the first start symbol in S^k derive f. The next $2q$ start symbols derive $d^{3t}e$ each, and the last $(2q+1+i)$th start symbol derives w_i, for $1 \leq i \leq q$.

As pointed out before, each w_i can be derived by either of two trees; a wide one or a narrow one. If $J_i \in \mathcal{J}'$, then we let w_i be derived by a wide tree, otherwise we let it be derived by a narrow tree.

It remains to be shown that this derivation forest is a k-derivation forest. Note first that F has $k(3t+1)$ internal nodes. This is true because the trees deriving $d^{3t}e$, as well as the narrow trees, have $3t+1$ internal nodes each, and wide trees have

three additional nodes. Together with the first start symbol, we obtain thus

$$1+2q(3t+1)+q(3t+1)+3t=(3q+1)(3t+1)=k(3t+1)$$

nodes.

We call those nodes of F that are labeled with S, N, or W, *chain nodes* and the ones labeled with B *branch nodes*. We construct a k-derivation for F. All trees in F, except the leftmost one, have $3t+1$ chain nodes each. We rewrite the ith chain node, $1 \leq i \leq 3t+1$, in the ith derivation step. Since there are $3q$ such trees, we are missing one more nonterminal in each step. The first step will rewrite the leftmost start symbol; each of the remaining derivation steps will rewrite one of the $3t$ occurrences of B from the wide trees. (Recall that $\#\mathcal{J}' = t$.)

To determine where each B is rewritten, we use the schedule σ, which is the solution of $\langle \mathcal{J}, t \rangle$. Let τ_j be the task scheduled at time j, for $1 \leq j \leq 3t$, such that τ_j appears in the triplet J_i. Let the release time of τ_j be r. Then τ_j corresponds to that occurence of B in the tree of w_i which is obtained at step r. We will rewrite this occurence of B in the $(j+1)$st derivation step. This is valid because, for each task, the time it is scheduled in σ is at least as large as the release time; i.e., $j \geq r$, and thus $j+1 > r$.

All the B's are thus rewritten in different derivation steps (recall that all the scheduling times are different). Hence, F is a k-derivation forest of w, i.e. $w \in \mathbf{UX}_k(G)$.

We shall now prove the converse direction. Let $\langle \mathcal{J}, t \rangle$ be an instance of 1S, and let k and w be obtained from $\langle \mathcal{J}, t \rangle$ as described above. It will be shown that $w \in \mathbf{UX}_k(G)$ implies that $\langle \mathcal{J}, t \rangle$ has a one-processor schedule of the required form. Let F be a k-derivation forest of w. Clearly, F consists of $k = 3q+1$ trees, one for each start symbol. It follows from the structure of the grammar that the first tree derives f from S, the next $2q$ symbols derive $d^{3t}e$ each, and the $(2q+1+i)$th tree derives w_{i_1} for $1 \leq i \leq q$.

We shall now see that there are exactly $3t$ wide trees in F. The tree that derives w_i is either wide or narrow; it contains $3t+1$ chain node in both cases. Moreover, each of the trees deriving $d^{3t}e$ has $3t+1$ internal nodes, and the tree deriving f has a single one. Each wide tree has three additional internal nodes; the branch nodes. The total number of internal nodes, except for the branch nodes, is thus $s = 3q(3t+1)+1$.

There are $3q$ occurrences of b in w. It follows thus that there are $s' \in \{0, \ldots, 3q\}$ branch nodes in F. Note now that the number of internal nodes must be a multiple of $k = 3q+1$, i.e.,

$$s + s' = 0 \bmod k.$$

But the only s' that satisfies the above equality is $s' = 3t$. There are thus $3t$ branch nodes in F, i.e., F contains exactly t wide trees. The total number of internal nodes of F is thus $(3q+1)(3t+1) = k(3t+1)$, and, therefore, the k-derivations of F must have $3t+1$ steps.

We observe that each of the trees, except for the first one, has height $3t+1$, the height of the first tree is one. Thus, each derivation step, except for the first one, has to rewrite all the chain nodes, and exactly one of the branch nodes.

We proceed to construct the solution to $\langle \mathcal{J}, t\rangle$ from F. Let $\mathcal{J}'$ be the set of all those triplets J_i that correspond to words w_i with wide derivation trees. We shall now construct the one-processor schedule σ of the tasks in $\mathcal{J}'$.

A branch node at depth i corresponds to a task with release time $i-1$. The step at which it is rewritten into b, say j, will indicate that the corresponding task is scheduled at time $j-1$. Note that $j \geq i$, hence $j-1 \geq i$. Since each branch node is written in a different derivation step, between 2 and $3t+1$, we obtain thus a valid one-processor schedule; hence, $\langle \mathcal{J}', t\rangle$ is a solution to 1S. □

We now prove a similar NP-hardness result for adjacent rewriting.

Let $G = \langle \Sigma, P, S, \Delta\rangle$ be a CFG. The language

$$\mathbf{AX}_k(G) = \{\omega \in \Delta^* : S^k \Rightarrow^*_{G,k} w\}$$

is called the *extended adjacent k-language* of G.

The *Adjacent Extended Membership* problem (AXM) is stated as follows: Let G be a fixed CFG. Given a positive integer k and a word w, is w in $\mathbf{AX}_k(G)$?

We shall prove the NP-hardness of AXM by reducing to it the *Exact Three Cover* problem (X3C).

An instance $\langle \mathcal{J}, t\rangle$ of X3C consists of a nonnegative integer t and a set $\{J_1, \ldots, J_q\}$ of integer triplets in the interval $[1, 3t]$. Then it is NP-complete to ask whether there is a subset $\mathcal{J}'$ of $\mathcal{J}$, such that $\#\mathcal{J}' = t$, and each integer in $[1, 3t]$ occurs exactly once in $\mathcal{J}'$. The above problem was first proved to be NP-complete in [17].

4.4. Example. Let $\mathcal{J} = \{(3, 4, 6), (1, 2, 5), (1, 5, 6)\}$. Then the first two triplets form a solution to $\langle \mathcal{J}, 2\rangle$.

Note the similarity between X3C and 1S. X3C may be regarded as a one-processor scheduling problem, where each task has to be scheduled at its release time.

Without loss of generality we may assume that $q > 3t$, otherwise, we may pad the problem as follows.

Let $\hat{t} = t+1$, and let

$$\hat{\mathcal{J}} = \mathcal{J} \cup \{(3t+1, 3t+2, 3t+3)\}$$
$$\cup \{J : J \text{ contains one element from } \{1, \ldots, 3t\} \text{ and two elements from } \{3t, 3t+1, 3t+2\}\}.$$

Obviously, $\langle \hat{\mathcal{J}}, \hat{t}\rangle$ has a solution if and only if $\langle \mathcal{J}, t\rangle$ has a solution, and $\#\hat{\mathcal{J}} > 3(t+1)$.

The grammar

$$G = \langle \{A, B, D, E, N, S, W, W', a, b, c, d, e\}, P, S, \{a, b, c, d, e\}\rangle,$$

that will be used in the NP-hardness construction, has the following set of productions:

$$S \to N, \quad N \to AN, \quad N \to BN, \quad N \to c, \quad A \to A, \quad B \to B, \quad A \to a, \quad B \to b,$$
$$S \to W, \quad S \to BW', \quad W \to AW, \quad W \to ABW', \quad W \to c,$$
$$W'W, \quad W' \to BW',$$
$$S \to D, \quad S \to d, \quad D \to E, \quad D \to e, \quad E \to D, \quad E \to d.$$

G generates three kinds of trees, *dummy* trees generating strings over $\{d, e\}$, and *wide* and *narrow* trees, similarly as in the UXM reduction. Examples of a wide and a narrow tree for a^2b^2abc are shown in Fig. 5; contrast threse with the trees in Example 4.1.

We shall now describe the reduction. Let $\mathcal{J} = \{J_1, \ldots, J_q\}$. Let the ith triplet, J_i, be (r_1, r_2, r_3), such that the integers are listed in increasing order. Now J_i is encoded as in the UXM reduction, into

$$w_i = a^{r_1-1}ba^{r_2-r_1-1}ba^{r_3-r_2-1}ba^{3t-r_3}c.$$

The instance of AXM corresponding to $\langle \mathcal{J}, t \rangle$ is the pair $\langle w, k \rangle$, such that $k = (3t+1)q$, and $w = vw_1 \ldots w_q$, where

$$v = \begin{cases} d(e^q d^q)^{3t/2-1} e^q d^{q-1} & \text{if } 3t \text{ is even,} \\ d(e^q d^q)^{(3t-1)/2} e^{q-1} & \text{otherwise.} \end{cases}$$

Obviously, $\langle w, k \rangle$ is constructible from $\langle \mathcal{J}, t \rangle$ in polynomial time and space.

The derivation forest of Fig. 6 corresponds then to the instance of X3C from Example 4.4.

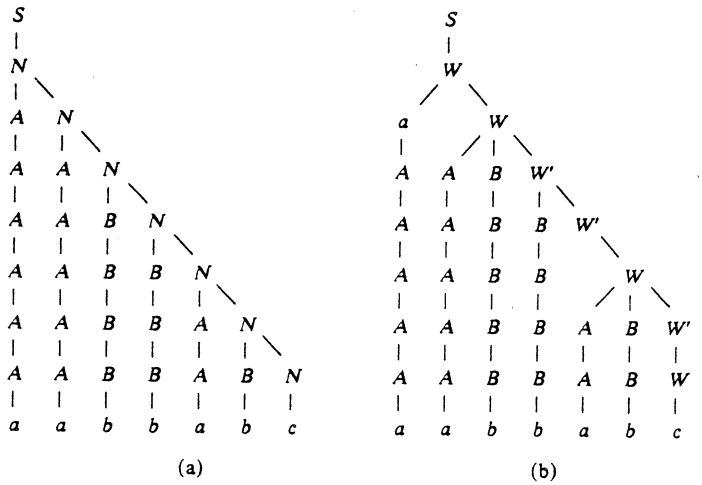

Fig. 5. Two trees for the word a^2b^2abc. (a) The narrow tree (b) The wide tree.

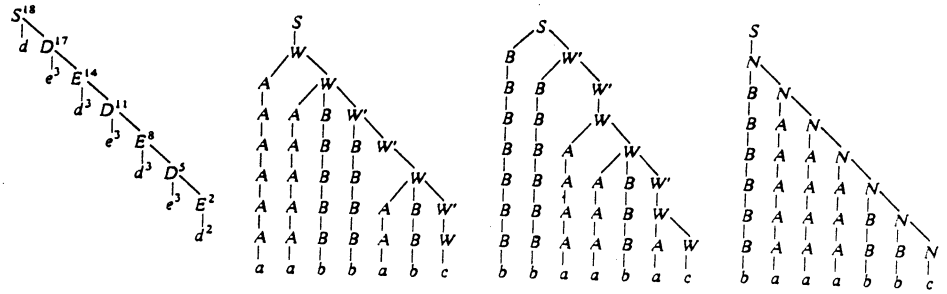

Fig. 6. A k-derivation for w.

4.5. Example. The derivation forest for

$$\langle \mathcal{J}, t\rangle = \langle \{(3, 4, 6), (1, 2, 5), (1, 5, 6)\}, 2\rangle$$

is shown in Fig. 6. This forest is an adjacent k-derivation forest for $k = 21$; at each step, all the nonterminal symbols in a sentential form are rewritten.

4.6. Theorem. AXM *is* NP-*hard.*

Proof. Using similar arguments as in the first part of the proof of Theorem 4.3, it can easily be seen that $w \in \mathbf{AX}_k(G)$ if the given instance of X3C, $\langle \mathcal{J}, t\rangle$, has a solution, where $\langle w, k\rangle$ is obtained from $\langle \mathcal{J}, t\rangle$ as described above; we use wide trees for triplets participating in the solution, and narrow trees for the other triplets.

For the converse direction, let $w \in \mathbf{AX}_k(G)$. We shall prove the existence of an exact 3-cover $\mathcal{J}' \subseteq \mathcal{J}$.

We first show that each block of symbols rewritten at any one step starts at the right end of the sentential form. Observe that $|w_1 \dots w_q| = k$, and $|vw_1| = k - q + (3t+1)$. But, by our assumption, $(3t+1) \leq q$, and, hence, $|vw_1| \leq k$.

Since our grammar G is non-erasing, it follows that every symbol in w_1 must have been obtained at the last derivation step. Similarly, all 'parents' of these symbols must have been obtained at the previous step, and so forth. Recall now that $|w_1| = 3t+1$. From the structure of the grammar and from the said above it follows now that the derivation of w_1 from S, and therefore the whole derivation of w, consists of $3t+2$ steps. It can now easily be seen that the derivation of w_q from S must be of the same length. In particular, the rightmost symbol must be rewritten $3t+2$ times; otherwise, w_q could not be of length $3t+1$. Since rewriting is adjacent, the k rightmost symbols in every sentential form are rewritten.

We observe now that every sentential form must contain exactly k nonterminals. Let us assume on the contrary that there is a nonterminal, X, with k nonterminals to its right. The number of symbols to the right of X can, however, not decrease, because G is propagating. X can thus never be rewritten, since it is not one of the k rightmost symbols, and we arrive at a contradiction.

We shall determine the number of those nonterminals in each sentential form that participate in the derivation of $w_1 \dots w_q$. Call these (occurrences of) nonterminals *proper* and all others *dummy*. Let $x_0, \dots, x_{3t+1}$, w be the sentential forms of our adjacent k-derivation. Then, obviously, x_0 contains q proper nonterminals, all of them start symbols. We prove that there are $iq+1$ proper nonterminals in x_i, for $1 \leq i \leq 3t$, and $(3t+1)q$ proper nonterminals in x_{3t+1}.

We recall that all nonterminals in each sentential form $x_0, \dots, x_{3t+1}$ are rewritten simultaneously. Thus, all the dummy nonterminals in a sentential form must be the same. Each derivation step can, therefore, result in either a block of d's or a block of e's. The word w contains $3t+1$ such blocks. There could not have been any dummy nonterminal in x_{3t+1}, because $|w_1 \dots w_q| = k$. It follows thus that the ith block first appears in x_i, for $1 \leq i \leq 3t+1$. Hence, the number of proper nonterminals increases by the number of d's an e's obtained at a derivation step. We conclude that $iq+1$ proper nonterminals occur in x_i, $1 \leq i \leq 3t$, and $(3t+1)q$ proper nonterminals occur in x_{3t+1}.

Note now that a narrow tree contributes i symbols to x_i. A wide tree usually contributes i; however, it contributes $i+1$, if it is a tree for w_j, and i is a member of the triplet J_j. The only possibility to obtain $iq+1$ proper nonterminals in x_i is having $q-1$ trees contribute i symbols each and one tree contribute $i+1$ symbols. Since $1 \leq i \leq 3t$, there are thus t wide trees in the derivation forest. Moreover, for each i, $1 \leq i \leq 3t$, exactly one of these trees contributes an $(i+1)$st symbol to x_i. Hence, the t triplets encoded in the wide trees cover the set $\{1, \dots, 3t\}$. □

4.7. Remark. In the grammars used in the reductions of Theorems 4.3 and 4.6, we have made use of six terminal symbols. Note that we could have encoded these symbols in binary notation without changing the correctness of the reductions. Hence, UXM and AXM are NP-hard even if we restrict ourselves to a two-letter alphabet. For the case of UXM, this is best possible, because in [14] it is shown that for one-letter alphabets, UXM is polynomial. On the other hand, if both k and the grammar are variable, then the non-emptiness problem is already NP-hard, both for adjacent and unrestricted rewriting, as shown below. We contrast this result with the fact that the emptiness problem for $U_k(G)$ is polynomial if k is variable but G is constant (see [14]).

4.8. Corollary. *The nonemptiness problem is* NP*-hard for* $U_k(G)$ *and* $A_k(G)$, *if* k *and* G *are variable.*

Proof. We shall outline the proof for $U_k(G)$. The proof for $A_k(G)$ will be analogous. In Theorem 4.3, we have encoded each instance of 1S into a word, w. Given the same instance of 1S, we construct now a grammar that derives only the word w. First, we introduce the production $S \to S_1 \dots S_k$. We recall that $k = 3q+1$, and that w is of the form $f(d^{3t}e)^{2q}w_1 \dots w_q$.

Let now $TD_1, \ldots, TD_{2q+1}$ be the (unique) derivation trees of $f(d^{3t}e)^{2q}$, and let TN_{2q+1+i} and TW_{2q+1+i} be the narrow and wide trees for w_i, respectively. We replace S at the root of TD_i by S_i, for $1 \leq i \leq 2q+1$, and we replace S by S_i in TN_i and TW_i, for $2q+2 \leq i \leq 3q+1$. Note that TN_i and TW_i have the same root symbol. Then, we index all other nonterminal symbol in these trees by a unique index, and we add to the grammar all those productions that derive these trees

It is easy to see that the size of the resulting grammar is polynomial in the size of the instance of 1S, because each tree contains at most $3t+4$ nonterminals. Obviously, w is in the k-language of this grammar if and only if there is a solution to the given instance to 1S. Hence, the corollary holds. □

5. Summary

We have investigated the complexity of the membership problem for a variety of cases. Table 1 gives us an overview of the results obtained. There is one entry missing in this table, where only k is variable, and both G and w are constant. In [14] it is shown that, for this case, the unrestricted k-language membership problem is in P, also for nonpropagating grammars (starting with S^k).

Table 1.

Constant	Variable	Result	Remarks
k	G, w	in P	Propagating G, for the unrestricted case
G	k, w	NP-hard	Starting with S^k, both cases
w	k, G	NP-hard	Both cases

Note that the problem for fixed w and variable G and k yields a nonemptiness problem; given a grammar, we can always replace every terminal symbol by the empty word, λ. Then the fixed word λ is in the resulting language if and only if the language is non-empty. Hence, deciding nonemptiness is NP-hard in this case. On the other hand, nonemptiness is in P if only k is variable (see [14]).

The main open problem is the complexity of the adjacent membership problem for constant k. This problem is also unsolved for the case where only k is variable. Similarly as for unrestricted rewriting, one can construct a scheduling problem, where perfect schedules correspond to k-adjacent derivations. This scheduling problem does, however, not correspond to a natural situation of resource allocation and has thus not been investigated previously. In particular, the notion of median is not applicable to this 'adjacent scheduling' problem.

Our polynomial membership algorithms work only for propagating grammars. The membership complexity for non-propagating grammars is still open. The

propagating property is essential for Lemma 3.4, which bounds the size of a derivation forest, and thus the number of frames, polynomially in the size of the grammar and of the input word.

Also, it would be interesting to know which of the above NP-hard problems are in NP, and hence NP-complete. This has been shown in [14] for the UXM problem, i.e., where the grammar is fixed. The corresponding problem, where the grammar is part of the input, remains open.

References

[1] J. Bruno, Deterministic and stochastic scheduling problems with tree-like precedence constraints, *NATO Conference*, Durham, England, 1981.

[2] C. Beeri and E. Shamir, Checking stacks and context-free programmed grammars accept P-complete languages, *Proc. 2nd Coll. on Automata, Languages and Programming*, Lecture Notes in Computer Science 14 (Springer, Berlin, 1974) 27-33.

[3] E.G. Coffman, Jr. and R.L. Graham, Optimal scheduling for two-processors systems, *Acta Inform.* 1 (1972) 200-213.

[4] E. Dahlhaus and H. Gaifman, Concerning two-adjacent context-free languages, Submitted for publication, 1985.

[5] D. Dolev and M.K. Warmuth, Scheduling flat graphs, *SICOMP*, to appear.

[6] D. Dolev and M.K. Warmuth, Profile scheduling of opposing forests and level orders, *SIAM J. Algebraic Discrete Methods*, to appear.

[7] D. Dolev and M.K. Warmuth, Scheduling precedence graphs of bounded height, *J. Algorithms* 5 (1984) 48-59.

[8] J. Earley, An efficient context-free parsing algorithm, *Comm. ACM* 13 (1970) 94-102.

[9] M. Fujii, T. Kasami and K. Ninomiya, Optimal sequencing of two equivalent processors, *SIAM J. Appl. Math.* 17 (4) (1969) 784-789; Erratum, *SIAM J. Appl. Math.* 20 (1971) 141.

[10] H.N. Gabow, An almost linear algorithm for two processor scheduling, *J. ACM* 29(3) (1982) 766-780.

[11] H.N. Gabow, University of Colorado at Boulder, Private communication, 1981.

[12] M.R. Garey, D.S. Johnson, R.E. Tarjan and M. Yannakakis, Scheduling opposing forests, *SIAM J. Algebraic Discrete Methods* 4 (1) (1983) 72-93.

[13] J. Gonczarowski and E. Shamir, Pattern selector grammars and several parsing algorithms in the context-free style, *J. Comput. System Sci.*, to appear.

[14] J. Gonczarowski and M.K. Warmuth, Manipulating derivation forests by scheduling techniques, Tech. Rept. 85-3, Hebrew Univ., 1985.

[15] M.A. Harrison, *Introduction to Formal Language Theory* (Addison-Wesley, Reading, MA, 1978).

[16] N.C. Hu, Parallel sequencing and assembly line problems, *Oper. Res.* 9 (6) (1961) 841-848.

[17] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972).

[18] H.C.M. Kleijn and G. Rozenberg, Context-free like restrictions on selective rewriting, *Theoret. Comput. Sci.* 16 (1981) 237-269.

[19] H.C.M. Kleijn and G. Rozenberg, On the generative power of regular pattern grammars, *Acta Inform.* 20 (1983) 391-411.

[20] J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity of scheduling under precedence constraints, *Oper. Res.* 26 (1978) 22-35.

[21] E.W. Mayr, Well structured parallel programs are not easier to schedule, Tech. Rept. STAN-CS-81-880, Dept. of Computer Science, Stanford Univ., 1981.

[22] J. Opatrny and K. Culik, II, Time complexity of recognition and parsing of E0L languages, in: A. Lindenmayer and G. Rozenberg, eds., *Automata, Languages, Development* (North-Holland, Amsterdam, 1976) 243-250.

[23] C.H. Papadimitriou and M. Yannakakis, Scheduling interval-ordered tasks, *SIAM J. Comput.* **10** (3) (1979) 405–409.
[24] G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems* (Academic Press, New York, 1980).
[25] A. Salomaa, *Formal Languages* (Academic Press, New York, 1973).
[26] J.D. Ullman, NP-complete scheduling problems, *J. Comput. System Sci.* **10** (1976) 384–393.
[27] J. van Leeuwen, The membership question for ETOL langauges is polynomially complete, *Inform. Process. Lett.* **3** (1975) 138–143.
[28] M.K. Warmuth, Scheduling on profiles of constant breadth, Ph.D. Thesis, Dept. of Computer Science, Univ. of Colorado, Boulder, 1981.
[29] D.H. Younger, Recognition and parsing of context-free languages in time n^3, *Inform. Control* **10** (1967) 189–208.