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.

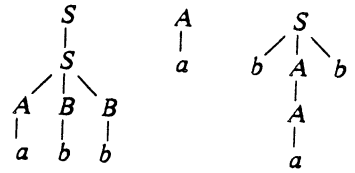

Fig. 1. A derivation forest that is not a 3-derivation forest.

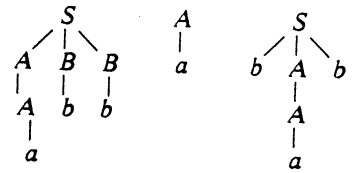

Fig. 2. A derivation forest that is a 3-derivation forest.

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.