Splittability of Bilexical Context-Free Grammars is Undecidable

Mark-Jan Nederhof*
University of St. Andrews

Giorgio Satta**
University of Padua

Bilexical context-free grammars (2-LCFGs) have proved to be accurate models for statistical natural language parsing. Existing dynamic programming algorithms used to parse sentences under these models have running time of $\mathcal{O}(|w|^4)$, where w is the input string.

A 2-LCFG is splittable if the left arguments of a lexical head are always independent of the right arguments, and vice versa. When a 2-LCFGs is splittable, parsing time can be asymptotically improved to $\mathcal{O}(|w|^3)$. Testing this property is therefore of central interest to parsing efficiency. In this article, however, we show the negative result that splittability of 2-LCFGs is undecidable.

1. Introduction

Bilexical context-free grammars, or 2-LCFGs for short, are specialized context-free grammars where each nonterminal is associated with a terminal symbol representing a lexical element of the language of interest. Furthermore, no more than two symbols can be used in the right-hand side of a rule, and the terminal symbol associated with the left-hand side of a rule must also occur on the right-hand side. In this way, 2-LCFGs are able to specify syntactic constraints as well as lexically specific preferences that influence the combination of predicates with their arguments or modifiers. Models based on 2-LCFGs have therefore been of central interest in statistical natural language parsing, as they allow selection of high-quality parse trees. One can in fact see 2-LCFGs as abstract models of the head automata of Alshawi (1996), the probabilistic projective dependency grammars of Eisner (1996), and the head-driven statistical models of Charniak (2001) and Collins (2003).

Parsing algorithms based on 2-LCFGs can be very efficient. In the general case, existing dynamic programming algorithms have running time of $\mathcal{O}(|w|^4)$, where w is the input string. (In this article we disregard complexity factors that depend on the input grammar, which we will consider to be constants.) In cases in which, for each (lexical) head, the two processes of generating its left and right arguments are, to some

* School of Computer Science, University of St. Andrews, North Haugh, St. Andrews, Fife, KY16 9SX, Scotland. E-mail: markjan.nederhof@gmail.com.

** Department of Information Engineering, University of Padua, via Gradenigo 6/A, I-35131 Padova, Italy. E-mail: satta@dei.unipd.it.

Submission received: 8 September 2010; revised submission received: 21 January 2011; accepted for publication: 17 April 2011.

extent, independent one of the other, parsing based on 2-LCFGs can be asymptotically improved to $\mathcal{O}(|w|^3)$. The reader is referred to Eisner and Satta (1999) for a detailed presentation of these computational results.

In the literature, this condition on independence between left and right arguments of each head has been called **splittability** (Eisner 1997; Eisner and Satta 1999). Testing for splittability on an input 2-LCFG is therefore of central interest to parsing efficiency. The computability of this test has never been investigated, however.

In this article splittability is defined for 2-LCFGs in terms of equivalence to another grammar in which independence between left and right arguments of each head is ensured by a simple syntactic restriction. This restriction is called **split form**. Informally, a 2-LCFG is in split form if it can be factorized into individual subgrammars, one for each head, and each subgrammar produces the left and the right arguments of its head through two derivation processes, one happening strictly after the other.

One may believe that a necessary and sufficient condition for splittability is that a 2-LCFG does not allow recursive structures that alternately generate left (L) and right (R) arguments of some head a. These structures could well result in subderivations producing sequences of the form $L^n a R^n$, for any $n \geq 0$, which would appear to preclude the application of the $\mathcal{O}(|w|^3)$ algorithm.

This situation, however, does not mean in general that the grammar is not splittable. Although a subset of the rules in a grammar may generate structures such as those just discussed, there may be additional rules that make the observed dependencies between left and right arguments irrelevant. In fact, splittability of a 2-LCFG is undecidable, as will be shown in this article.

Our result is based on the fact that it is undecidable whether a linear context-free grammar with a center marker (to be defined later) generates a regular language. This fact is originally proved in this article, and does not follow from the weaker result stating the undecidability of regularity of (general) linear context-free languages, which is well known in the formal language literature (Hopcroft and Ullman 1979, exercise 8.10a, page 214) and which is originally due to Hunt III and Rosenkrantz (1974).

The remaining part of this article is organized as follows. In Section 2 we present some preliminary background, and in Section 3 we define 2-LCFGs and the notion of splittability. In Section 4 we prove the main result of this article, and draw some conclusions in Section 5.

2. Preliminaries

In this article we assume the reader is familiar with the notions of context-free grammar, Turing machine, and undecidability (see, e.g., Hopcroft and Ullman 1979). We briefly summarize the adopted notation now.

A **context-free grammar** (CFG) is a 4-tuple $G = (\Sigma, N, S, R)$, where Σ is a finite set of terminals, N is a finite set of nonterminals, disjoint from Σ and including the start symbol S, and R is a finite set of rules. Each rule has the form $A \to \alpha$, where $A \in N$ and $\alpha \in (\Sigma \cup N)^*$. A CFG is called **linear** if each rule $A \to \alpha$ has at most one nonterminal in the right-hand side α.

We associate with G a binary relation called rewrite and denoted by the symbol $\Rightarrow_G$, defined such that $\gamma A \gamma' \Rightarrow_G \gamma \alpha \gamma'$ if $A \to \alpha$ is a rule in R and $\gamma, \gamma' \in (\Sigma \cup N)^*$. We drop subscript G from $\Rightarrow_G$ whenever it is understood from the context. The reflexive and transitive closure of $\Rightarrow$ is denoted as $\Rightarrow^*$. The language **generated** by G is defined as $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

We now introduce a special class of CFGs that will play a central role in the proofs in this article. Assume a distinguished symbol $\# \in \Sigma$, which we will refer to as a **center marker**. The class of **linear CFGs with center marker** #, or lin-CFG(#) for short, is the class of CFGs in which each rule either has the form $A \to \#$ or the form $A \to uBv$, where $A, B \in N$ and $u, v \in (\Sigma \setminus \{\#\})^*$, where symbol "$\setminus$" denotes set difference. We say a grammar in lin-CFG(#) is in **binary form** if the total length of u and v in rules $A \to uBv$ is at most 1. It is not difficult to see that there is a language-preserving transformation of grammars in lin-CFG(#) to binary form, as suggested by the following example.

Example 1
Let $\Sigma = \{a, b, \#\}$. One member of lin-CFG(#) is the CFG G defined by the rules:

$$\begin{aligned} S &\to a\ S\ a, \\ S &\to a\ S\ b, \\ S &\to b\ S\ a, \\ S &\to b\ S\ b, \\ S &\to \#. \end{aligned}$$

For this grammar, strings in $L(G)$ have the form $u\#v$, with $u \in (\Sigma \setminus \{\#\})^*$ and $v \in (\Sigma \setminus \{\#\})^*$ having the same length.

A rule such as $S \to aSb$ can be replaced by a pair of rules in binary form, for example $S \to aA$ and $A \to Sb$, where A is a new nonterminal.

3. Bilexical CFGs and Splittability

We start this section with the definition of 2-LCFG, which is based on Eisner and Satta (1999). Let N_D be a finite alphabet whose symbols will be called **delexicalized** nonterminals. We combine N_D with a set Σ of terminal symbols as follows:

$$N_D(\Sigma) = \{A[a] \mid A \in N_D,\ a \in \Sigma\}.$$

A **bilexical context-free grammar** is a CFG $G = (\Sigma, N_D(\Sigma) \cup \{S\}, S, R)$. Every rule in R has one of the following five forms:

- $S \to A[a]$;
- $A[a] \to B[b]\ C[a]$;
- $A[a] \to B[a]\ C[c]$;
- $A[a] \to B[a]$;
- $A[a] \to a$.

The nonterminal occurrences $C[a]$ and $B[a]$ in the second and third rules, respectively, and the nonterminal occurrence $B[a]$ in the fourth rule will be referred to as **head child** occurrences. Notice that the head child of a rule is always associated with the same terminal as the left-hand-side nonterminal. The nonterminal occurrence $A[a]$ in the first rule, and the nonterminal occurrences $B[b]$ and $C[c]$ in the second and third rules, respectively, will be referred to as **maximal projection** occurrences. Observe how in a

parse tree generated by a 2-LCFG, each occurrence of a lexical element (represented as a terminal symbol) is also part of several head child occurrences of nonterminals above it, up to some unique maximal projection occurrence. We assume that the head child occurrence in a rule is always marked within the rule itself, in order to disambiguate cases in which the head child and the maximal projection share the same terminal ($a = b$ in the second rule or $a = c$ in the third).[1]

Let $G = (\Sigma, N, S, R)$ be a 2-LCFG and choose some $A[a] \in N$ that occurs as maximal projection in some rule in R. We now define a grammar $G^{(A[a])} = (\Sigma^{(A[a])}, N^{(A[a])}, A[a], R^{(A[a])})$ in lin-CFG(a). The main idea here is that $G^{(A[a])}$ captures all descending paths in parse trees of G from any maximal projection occurrence of $A[a]$ down to a, treating the maximal projection occurrences of G's nonterminals to the left and right of these paths as terminal symbols of $G^{(A[a])}$. Each such maximal projection nonterminal $B[b]$, when treated as a terminal, will be denoted as $\overline{B[b]}$. The start symbol is $A[a]$ and the rule set $R^{(A[a])}$ is specified as follows:

- $D[a] \rightarrow \overline{B[b]}\ C[a]$ is in $R^{(A[a])}$ for each rule $D[a] \rightarrow B[b]\ C[a]$ in R;
- $D[a] \rightarrow B[a]\ \overline{C[c]}$ is in $R^{(A[a])}$ for each rule $D[a] \rightarrow B[a]\ C[c]$ in R;
- $D[a] \rightarrow B[a]$ is in $R^{(A[a])}$ for each rule $D[a] \rightarrow B[a]$ in R;
- $D[a] \rightarrow a$ is in $R^{(A[a])}$ for each rule $D[a] \rightarrow a$ in R.

Grammar $G^{(A[a])}$ might contain useless rules, that is, rules that never appear in a derivation of a string of the generated language, but this is irrelevant to the development of the results in this article.

We now introduce an equivalence relation on 2-LCFGs, which will be used later in the definition of splittability. As we will see, our equivalence relation is stronger than the usual weak equivalence between grammars, where the latter only requires that the languages generated by two grammars be the same. In addition to weak equivalence, we demand that two 2-LCFGs establish the same predicate–argument dependencies between lexical elements in the generated sentences, as will be explained here.

Definition 1
Two 2-LCFGs G_1 and G_2 are **d-equivalent** if the following conditions are all satisfied:

1. G_1 and G_2 have the same set of nonterminals that occur as maximal projections;
2. G_1 and G_2 have the same rules rewriting the start symbol, that is, the rules of the form $S \rightarrow A[a]$; and
3. for each nonterminal $A[a]$ that occurs as a maximal projection in G_1 and G_2, the grammars $G_1^{(A[a])}$ and $G_2^{(A[a])}$ generate the same languages.

Lemma 1
If two 2-LCFGs G_1 and G_2 are d-equivalent, then $L(G_1) = L(G_2)$.

1 One way to formalize this marking is to have a separate subset of delexicalized nonterminals that are part of each maximal projection occurrence and never a part of any head child occurrence.

Proof
We show the stronger statement that, for each maximal projection occurrence $A[a]$ from G_1 and G_2 (item 1 in Definition 1) and for each $w \in \Sigma^*$, we have $A[a] \Rightarrow^*_{G_1} w$ if and only if $A[a] \Rightarrow^*_{G_2} w$. The statement of the lemma easily follows from item 2 in Definition 1. We proceed by induction on $|w|$.

Assume $|w| = 1$, $w = a$. If $A[a] \Rightarrow^*_{G_1} a$, then $A[a] \Rightarrow^*_{G_1^{(A[a])}} a$. From item 3 in Definition 1 we also have $A[a] \Rightarrow^*_{G_2^{(A[a])}} a$ and hence $A[a] \Rightarrow^*_{G_2} a$. The converse statement is shown similarly.

Assume now $|w| > 1$. If $A[a] \Rightarrow^*_{G_1} w$, there must be a "head-driven" derivation rewriting maximal projection occurrence $A[a]$ into a through a chain of head child nonterminal occurrences. More precisely, we can write:

$$\begin{aligned} A[a] &\Rightarrow^*_{G_1} B_1[b_1] \cdots B_l[b_l] a C_1[c_1] \cdots C_r[c_r] \\ &\Rightarrow^*_{G_1} u a C_1[c_1] \cdots C_r[c_r] \\ &\Rightarrow^*_{G_1} uav = w \end{aligned} \tag{1}$$

with $l, r \geq 0$, $l + r > 0$ (because $|w| > 1$), and the indicated occurrence of a in uav is generated from $A[a]$ through a chain of rule applications always rewriting the head child of the previous rule and generating the maximal projection occurrences $B_i[b_i]$, $1 \leq i \leq l$, and $C_j[c_j]$, $1 \leq j \leq r$.

Due to Equation (1) we have $A[a] \Rightarrow^*_{G_1^{(A[a])}} \overline{B_1[b_1]} \cdots \overline{B_l[b_l]} a \overline{C_1[c_1]} \cdots \overline{C_r[c_r]}$ and from item 3 in Definition 1 we have $A[a] \Rightarrow^*_{G_2^{(A[a])}} \overline{B_1[b_1]} \cdots \overline{B_l[b_l]} a \overline{C_1[c_1]} \cdots \overline{C_r[c_r]}$. We thus conclude that $A[a] \Rightarrow^*_{G_2} B_1[b_1] \cdots B_l[b_l] a C_1[c_1] \cdots C_r[c_r]$.

From Equation (1) we can also see that there must be strings u_i, $1 \leq i \leq l$ and v_j, $1 \leq j \leq r$, such that $B_i[b_i] \Rightarrow^*_{G_1} u_i$ and $C_j[b_j] \Rightarrow^*_{G_1} v_j$, where $u_1 \cdots u_l = u$ and $v_1 \cdots v_r = v$. Because each $B_i[b_i]$ and each $C_j[c_j]$ is a maximal projection occurrence, we can apply the inductive hypothesis and conclude that $B_i[b_i] \Rightarrow^*_{G_2} u_i$ and $C_j[b_j] \Rightarrow^*_{G_2} v_j$, $1 \leq i \leq l$ and $1 \leq j \leq r$.

Putting together all of this, we have $A[a] \Rightarrow^*_{G_2} u_1 \cdots u_l a v_1 \cdots v_r = w$. With a similar argument we can show that $A[a] \Rightarrow^*_{G_2} w$ implies $A[a] \Rightarrow^*_{G_1} w$. ■

Besides enforcing the weak equivalence between two 2-LCFGs, the d-equivalence relation guarantees that the grammars establish the same dependencies between pairs of lexical elements in the generated sentences. To explain this, we borrow subsequently the notion of dependency structure from dependency grammars.

Let G be a 2-LCFG and let t be a parse tree assigned by G to a string $w \in \Sigma^*$. We can associate with t a **dependency structure**, by considering all rule occurrences in t that link two lexical elements from w. For each rule occurrence $A[a] \rightarrow B[b]\ C[a]$ in t, a link is constructed from the corresponding occurrence of b in w to the corresponding occurrence of a. Similarly, a rule occurrence $A[a] \rightarrow B[a]\ C[c]$ gives rise to a link from the corresponding occurrence of c to the corresponding occurrence of a. Notice that a dependency structure abstracts away from some topological aspects of the parse trees. For example, two rules $A[a] \rightarrow B[a]\ C[c]$ and $B[a] \rightarrow D[d]\ E[a]$ could give rise to the same dependency structures as the two rules $A[a] \rightarrow D[d]\ F[a]$ and $F[a] \rightarrow E[a]\ C[c]$.

It is not difficult to see that if two 2-LCFGs G_1 and G_2 are d-equivalent, then the sets of dependency structures they assign to strings in the common language (via the set of parse trees as sketched above) are identical.

Example 2
Consider the 2-LCFG G_1 defined by the rules:

$$\begin{array}{ll} S \rightarrow A[a], & A[a] \rightarrow a, \\ A[a] \rightarrow B[b]\ A[a], & B[b] \rightarrow b, \\ A[a] \rightarrow C[c]\ A[a], & C[c] \rightarrow c, \\ A[a] \rightarrow A[a]\ D[d], & D[d] \rightarrow d, \\ A[a] \rightarrow A[a]\ E[e], & E[e] \rightarrow e; \end{array}$$

and the 2-LCFG G_2 defined by the rules:

$$\begin{array}{ll} S \rightarrow A[a], & A'[a] \rightarrow a, \\ A[a] \rightarrow B[b]\ A[a], & B[b] \rightarrow b, \\ A[a] \rightarrow C[c]\ A[a], & C[c] \rightarrow c, \\ A[a] \rightarrow A'[a], & D[d] \rightarrow d, \\ A'[a] \rightarrow A'[a]\ D[d], & E[e] \rightarrow e, \\ A'[a] \rightarrow A'[a]\ E[e]. & \end{array}$$

G_1 and G_2 are d-equivalent, and thus generate the same language by Lemma 1. This language consists of all strings of the form uav, where $u \in \{b,c\}^*$ and $v \in \{d,e\}^*$. Furthermore, the set of dependency structures associated by G_1 and G_2 to any of the strings they can generate are the same. Figure 1 shows one instance of such a correspondence. Note that in addition to the tree in Figure 1(a), there are two more trees generated by grammar G_1 that correspond to the dependency structure in Figure 1(c).

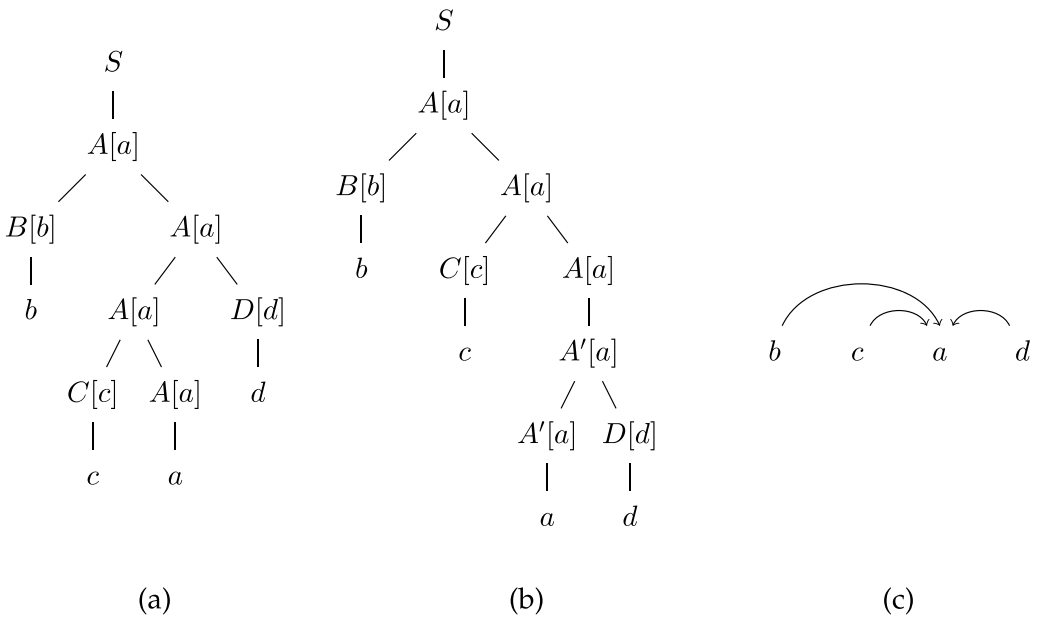

Figure 1
Two parse trees for the string *bcad* under grammars G_1 (a) and G_2 (b) from Example 2. Both trees correspond to the same associated dependency structure (c).

We conclude this section with the definition of splittability, which is central to this article. We first introduce a special form for 2-LCFGs, which we call split form. The intuition is that a 2-LCFG is in split form if, for each maximal projection nonterminal occurrence $A[a]$, all arguments on the left are generated first, followed by the generation of all arguments on the right.

Definition 2
A 2-LCFG $G = (\Sigma, N_D(\Sigma) \cup \{S\}, S, R)$ is in **split form** if, for each maximal projection occurrence of a nonterminal $A[a] \in N_D(\Sigma)$, the nonterminals in $G^{(A[a])}$ can be divided into two disjoint sets $N_l^{(A[a])}$ and $N_r^{(A[a])}$, such that $A[a] \in N_l^{(A[a])}$ and all of the following conditions are satisfied:

1. if $D[a] \to \overline{B[b]}\ C[a]$ in $R^{(A[a])}$, then $D[a], C[a] \in N_l^{(A[a])}$;
2. if $D[a] \to B[a]\ \overline{C[c]}$ in $R^{(A[a])}$, then $D[a], B[a] \in N_r^{(A[a])}$;
3. if $D[a] \to B[a]$ in $R^{(A[a])}$, then $D[a] \in N_l^{(A[a])}$ or $B[a] \in N_r^{(A[a])}$; and
4. if $D[a] \to a$ in $R^{(A[a])}$, then $D[a] \in N_r^{(A[a])}$.

Note that the left arguments of $A[a]$ are generated via the nonterminals in $N_l^{(A[a])}$, and the right arguments are generated via nonterminals in $N_r^{(A[a])}$. Furthermore, in the top–down generation of the arguments we can switch from nonterminals in $N_l^{(A[a])}$ to nonterminals in $N_r^{(A[a])}$ only once, through a rule of the form $D[a] \to B[a]$ such that $D[a] \in N_l^{(A[a])}$ and $B[a] \in N_r^{(A[a])}$, as stated by the third item in Definition 2. Grammar G_2 from Example 2 is in split form, with $N_l^{(A[a])} = \{A[a]\}$ and $N_r^{(A[a])} = \{A'[a]\}$.

The importance of 2-LCFGs in split form is that they allow parsing in time $\mathcal{O}(|w|^3)$, which is independent of the size of G assuming N_D is fixed. This contrasts with the time complexity $\mathcal{O}(|w|^4)$ for arbitrary 2-LCFGs.

We say a 2-LCFG G_1 is **splittable** if there is a 2-LCFG G_2 in split form such that G_1 and G_2 are d-equivalent. Grammar G_1 from Example 2 is splittable, because it is d-equivalent to grammar G_2 in the same example, and the latter is in split form.

Example 3
Consider the 2-LCFG defined by the following rules:

$$\begin{array}{lll} S & \to A[a], & A[a] \to a, \\ A[a] & \to B[b]\ A'[a], & B[b] \to b, \\ A'[a] & \to A[a]\ C[c], & C[c] \to c. \end{array}$$

Note that there is an equal number of arguments on either side of the head a. No 2-LCFG in split form can achieve this dependency, and therefore this grammar is not splittable.

We can now formally prove that if a 2-LCFG is splittable, then the left arguments of each head are independent of its right arguments, and vice versa, in the sense that only a finite amount of information can be passed between the two sides.

Theorem 1
Let G be a 2-LCFG. G is splittable if and only if, for each maximal projection occurrence $A[a]$ from G, the lin-CFG(a) $G^{(A[a])}$ generates a regular language.

Proof
If G is splittable, then there exists a split 2-LCFG G_s such that G and G_s are d-equivalent. Let $A[a]$ be a nonterminal of G_s that occurs as maximal projection, and consider the lin-CFG(a) $G_s^{(A[a])}$, whose nonterminals are partitioned into sets $N_l^{(A[a])}$ and $N_r^{(A[a])}$ as in Definition 2. It is not difficult to see that $L(G_s^{(A[a])})$ is a regular language. This regular language is the finite union of the languages:

$$L^{(B[a],C[a])} = \{uav \mid A[a] \Rightarrow^* uB[a] \Rightarrow uC[a] \Rightarrow^* uav\},$$

for all possible distinct choices of rules of the form $B[a] \to C[a]$ from $G_s^{(A[a])}$ with $B[a] \in N_l^{(A[a])}$ and $C[a] \in N_r^{(A[a])}$. For each such choice, the set of strings u is clearly regular, as it is generated by a right-linear subgrammar. Similarly, the set of strings v is regular. Because G and G_s are d-equivalent, it follows that each $G^{(A[a])}$ generates a regular language as well.

Conversely, assume that for each maximal projection occurrence $A[a]$ from G we have that $G^{(A[a])}$ generates a regular language. Then we can construct an alternative grammar in lin-CFG(a) generating the same language as $G^{(A[a])}$, but now in split form. To this end, assume a finite automaton M accepting $L(G^{(A[a])})$ with a finite set Q of states and a transition relation $\mapsto$. For each transition $q_1 \overset{a}{\mapsto} q_2$ of M, we can construct right-linear rules corresponding to transitions occurring on paths from the initial state of M to state q_1, and left-linear rules corresponding to transitions occurring on paths from state q_2 to any final state of M.

Concretely, for each transition $q \overset{\overline{B[b]}}{\mapsto} q'$ such that q_1 is reachable from q', we construct rule $q[a] \to \overline{B[b]}\, q'[a]$, and for each transition $q \overset{\overline{B[b]}}{\mapsto} q'$ such that q is reachable from q_2, we construct rule $q'[a] \to q[a]\, \overline{B[b]}$. We also construct rule $A[a] \to q_0[a]$, where q_0 is the initial state of M, and the rule $q_2[a] \to a$. For each final state q_f of M that is reachable from q_2 we furthermore construct the rule $q_1[a] \to q_f[a]$. The resulting grammar generates the strings that are accepted by M by computations that traverse the transition $q_1 \overset{a}{\mapsto} q_2$. Each transition $q_1 \overset{a}{\mapsto} q_2$ gives rise to one such grammar.

We can now take the disjoint union of such grammars for all transitions $q_1 \overset{a}{\mapsto} q_2$, taking different copies of automaton states q in nonterminals $q[A]$ for each combination of q_1 and q_2. The result is a grammar generating all strings accepted by M. Nonterminals $q[a]$ are defined to be in $N_l^{(A[a])}$ or in $N_r^{(A[a])}$ depending on whether a state q_1 in a transition $q_1 \overset{a}{\mapsto} q_2$ is reachable from q or whether q is reachable from a state q_2 in a transition $q_1 \overset{a}{\mapsto} q_2$. Naturally $A[a]$ is taken to be in $N_l^{(A[a])}$. The requirements on split form are now clearly satisfied. ■

4. Undecidability Results

This section presents the main results of this article. We start with some basic notions from computability theory and with the adopted notation for Turing machines.

We use single-tape, single-head, deterministic Turing machines. A **configuration** of a Turing machine $\mathcal{M}$ is a string over some alphabet Σ, encoding the current content of the tape along with the position of the head on the tape and the current state. In the initial configuration, the tape contains the input to the Turing machine, the head is placed at the start of the tape, and the machine is in its (unique) initial state. An accepting configuration is any configuration of $\mathcal{M}$ of which the current state belongs to a distinguished set of accepting states. An **accepting computation** of $\mathcal{M}$ is a sequence of configurations of $\mathcal{M}$ starting with an initial configuration and ending with a final

configuration, and such that consecutive configurations are related by a valid move of the machine. Without loss of generality, we assume that any accepting computation has an odd number of moves, and at least three.

For a given string w, we write $\widetilde{w}$ to denote the mirror image of w. Consider a Turing machine $\mathcal{M}$ whose configurations are defined over some alphabet Σ, and assume two fresh symbols \$ and # not in Σ. Following the notation of Bordihn, Holzer, and Kutrib (2005), we encode accepting computations of $\mathcal{M}$ as strings of the form:

$$w_0\$w_2\$\ldots\$w_{2k}\#\widetilde{w_{2k+1}}\$\ldots\$\widetilde{w_3}\$\widetilde{w_1} \tag{2}$$

Here, $w_0, \ldots, w_{2k+1}$ are configurations of $\mathcal{M}$, with w_0 an initial configuration and w_{2k+1} an accepting configuration. Furthermore, for each i with $0 \leq i \leq 2k$, configuration w_i is related to configuration w_{i+1} by a valid move of $\mathcal{M}$. We define VALC($\mathcal{M}$) to be the set of all valid computations of $\mathcal{M}$, represented as before. We say that a string u is **accepted** by $\mathcal{M}$ if u occurs in an initial configuration w_0 in some computation in VALC($\mathcal{M}$). The language $L(\mathcal{M})$ is defined to be the set of all strings accepted by $\mathcal{M}$.

We rely on the following result (Hopcroft and Ullman 1979, page 189):

Lemma 2
For a Turing machine $\mathcal{M}$, it is undecidable whether the language $L(\mathcal{M})$ is finite.

We also need the following result, which is essentially the same as a result reported by Hopcroft and Ullman (1979, Lemma 8.8, page 204). We provide a complete proof here because we need to adapt the result to our special encoding of valid computations, which is somewhat different from that used by Hopcroft and Ullman (1979), and because Hopcroft and Ullman only report a sketch of the proof.

Lemma 3
For a Turing machine $\mathcal{M}$, the language VALC($\mathcal{M}$) is context-free if and only if $L(\mathcal{M})$ is finite.

Proof
If $L(\mathcal{M})$ is finite, then VALC($\mathcal{M}$) is also finite (because $\mathcal{M}$ is deterministic). Hence VALC($\mathcal{M}$) is a context-free language.

If $L(\mathcal{M})$ is an infinite language, we need to show that VALC($\mathcal{M}$) cannot be a context-free language. To this end, we use the version of Ogden's lemma presented by Hopcroft and Ullman (1979, page 129). This lemma states that if L is a context-free language, there exists a constant n such that, if we arbitrarily mark n or more distinct positions in a string in L, then we can write that string as $uvwxy$ in such a way that v and x together include at least one marked position, vwx includes at most n marked positions, and uv^iwx^iy is in L for any $i \geq 0$.

Assume now that VALC($\mathcal{M}$) is a CFL. Because $L(\mathcal{M})$ is an infinite language, the length of its strings can be arbitrarily large. Therefore there must be a string in VALC($\mathcal{M}$) of the form:

$$\alpha = w_0\$w_2\$\ldots\$w_{2k}\#\widetilde{w_{2k+1}}\$\ldots\$\widetilde{w_3}\$\widetilde{w_1},$$

with w_2 having length greater than n. This is so because the initial configuration w_0, representing the input string, can be chosen with arbitrarily large length, and only two

moves of $\mathcal{M}$ have been applied to obtain w_2 from w_0. Let us mark all positions of α that fall within w_2.

Now let $\alpha = uvwxy$ be a factorization satisfying Ogden's lemma. If v or x contain one or more occurrences of the symbol \$, we can pump such a string in α and obtain a new string which is not a valid computation, which is a contradiction. More precisely, assume that v contains exactly one occurrence of \$, and write $v = v'\$v''$. If we choose $i = 3$ in Ogden's lemma, we obtain a new string with two instances of configuration $v''v'$ in it. This would imply a loop in the computation of a deterministic Turing machine, and therefore the resulting string cannot be an accepting computation. Using similar arguments, we can conclude that neither v nor x contains an occurrence of symbol \$.

Because v and x together must include at least one marked position, we can distinguish two (not necessarily distinct) cases. In the first case, v is entirely included in the string w_2. This means that we can pump w_2 to a new configuration of arbitrarily large length, without pumping w_0. But this is a contradiction, because w_0 and w_2 are only two moves apart. In the second case, x is entirely included in the string w_2. Similarly to the first case, we can pump w_2 to a new configuration of arbitrarily large length, without pumping w_1, but this is also a contradiction because the two configurations are related by a single move. Therefore VALC($\mathcal{M}$) cannot be a CFL. ■

We now define the set of invalid computations of $\mathcal{M}$, written INVALC($\mathcal{M}$), as:

$$\text{INVALC}(\mathcal{M}) = ((\Sigma \cup \{\$\})^* \cdot \{\#\} \cdot (\Sigma \cup \{\$\})^*) \setminus \text{VALC}(\mathcal{M}) \tag{3}$$

We deviate from Bordihn, Holzer, and Kutrib (2005) in that we only consider those strings in the complement of VALC($\mathcal{M}$) that contain exactly one occurrence of #. Because all the operations that need to be applied to obtain INVALC($\mathcal{M}$) from VALC($\mathcal{M}$), or vice versa, preserve regular languages, it follows that VALC($\mathcal{M}$) is a regular language if and only if INVALC($\mathcal{M}$) is a regular language.

The following result relates the language INVALC($\mathcal{M}$) to the class lin-CFG(#), that is, the linear context-free grammars with center marker #, introduced in Section 2. Our proof differs somewhat from proofs of similar statements in Hopcroft and Ullman (1979) for general linear CFGs, in that the center marker has a prominent role in the grammar, and the construction must ensure that # occurs exactly once.

Lemma 4
Given a Turing machine $\mathcal{M}$, one can effectively construct a grammar in lin-CFG(#) generating INVALC($\mathcal{M}$).

Proof
Intuitively, our grammar is the union of several subgrammars, which generate, respectively:

- a set of strings of the form (2) in which the w_i can be arbitrary strings over Σ, except that w_0 must *not* be an initial configuration,
- similarly, a set of strings of the form (2) in which w_{2k+1} must *not* be a final configuration,
- a set of strings of the form (2) in which for some i, w_{2i+1} must *not* be the successor of w_{2i},

- a set of strings of the form (2) in which for some i, w_{2i} must *not* be the successor of w_{2i-1},
- a set of strings of the form:

$$u_1\$ \ldots \$u_k\#v_1\$ \ldots \$v_m$$

where $u_i \in \Sigma^*$ $(1 \leq i \leq k)$, $v_j \in \Sigma^*$ $(1 \leq j \leq m)$ and $k < m$,

- a set of strings of the form:

$$u_1\$ \ldots \$u_k\#v_1\$ \ldots \$v_m$$

where $u_i \in \Sigma^*$ $(1 \leq i \leq k)$, $v_j \in \Sigma^*$ $(1 \leq j \leq m)$ and $m < k$.

It is straightforward to construct each of these subgrammars as linear grammars with center marker #, similarly to proofs from Hopcroft and Ullman (1979, pages 203 and 215). The statement of the theorem then follows from the fact that the class lin-CFG(#) is closed under (finite) union. ■

We now show that it is undecidable whether a linear context-free grammar with a center marker generates a regular language. This result is stronger than a similar result stating the undecidability of regularity for general linear context-free languages, originally obtained by Hunt III and Rosenkrantz (1974) and also reported by Hopcroft and Ullman (1979, Exercise 8.10a, page 214), in that our languages have an explicit center marker that unambiguously splits the string into two parts. Here we report a direct proof of the result for linear context-free grammars with a center marker, whereas the similar result for general linear context-free grammars is indirectly obtained by Hunt III and Rosenkrantz (1974, Corollary 2.7(5) and discussion at page 66) through sufficient conditions for the undecidability of a family of general predicates.

Theorem 2
It is undecidable whether a grammar in lin-CFG(#) generates a regular language.

Proof
Let $\mathcal{M}$ be a Turing machine. We start by claiming that $L(\mathcal{M})$ is finite if and only if INVALC($\mathcal{M}$) is regular. To show this, assume first that $L(\mathcal{M})$ is finite. Because our Turing machines are deterministic, VALC($\mathcal{M}$) is also finite, and therefore regular. From Equation (3), it follows that INVALC($\mathcal{M}$) is regular as well. Conversely, if INVALC($\mathcal{M}$) is regular, then so is VALC($\mathcal{M}$). This means that VALC($\mathcal{M}$) is context-free and, by Lemma 3, that $L(\mathcal{M})$ is finite.

Suppose now that it is decidable whether a grammar in lin-CFG(#) generates a regular language. Following Lemma 4, we can effectively construct a grammar in lin-CFG(#) generating INVALC($\mathcal{M}$). Under our assumption, we can now decide whether INVALC($\mathcal{M}$) is regular and hence, by our claim, whether $L(\mathcal{M})$ is finite. This contradicts Lemma 2. Hence, it must be undecidable whether a grammar in lin-CFG(#) generates a regular language. ■

Example 4
Consider the (unambiguous) grammar that is defined by the rules:

$$\begin{aligned} S &\rightarrow A \mid B \mid C \\ A &\rightarrow aAa \mid bAb \mid aAb \mid bAa \mid \# \\ B &\rightarrow aB \mid bB \mid aA \mid bA \\ C &\rightarrow Ca \mid Cb \mid Aa \mid Ab \end{aligned}$$

Despite rules such as $A \rightarrow aAa$ and $A \rightarrow bAb$, which by themselves may appear to make the language non-regular, the generated language is in fact $\{a, b\}^*\#\{a, b\}^*$.

We now return to bilexical context-free grammars. In Section 3 we have constructed grammars $G^{(A[a])}$ out of a 2-LCFG G, for each nonterminal $A[a]$ that occurs as maximal projection in rules from G. We then characterized splittability of G, in Theorem 1, in terms of the regularity of the languages generated by each $G^{(A[a])}$. Note that each $G^{(A[a])}$ is in lin-CFG(a). In order to obtain the main result of this article, we only need to look at a very simple type of 2-LCFG in which we can identify a given linear context-free grammar with a center marker.

Theorem 3
Splittability of bilexical CFGs is undecidable.

Proof
We show that the existence of an algorithm for deciding whether a 2-LCFG is splittable would imply an algorithm for deciding whether any grammar in lin-CFG(#) generates a regular language, contrary to Theorem 2.

Let G_1 be a grammar in lin-CFG(#). Without any loss of generality, we assume that:

- G_1 is in binary form;
- the start symbol of G_1 has the form $S[\#]$;
- each nonterminal symbol of G_1 has the form $A[\#]$, for some A; and
- each terminal symbol of G_1 is either # or a symbol of the form $\overline{B[b]}$, for some B and some b.

From G_1 we construct a 2-LCFG G_2 as follows. The start symbol of G_2 is $S^\dagger$, and G_2 has a single rule $S^\dagger \rightarrow S[\#]$ expanding the start symbol. Each rule from G_1 is copied to be a rule of G_2, after replacing each terminal symbol $\overline{B[b]}$ by a nonterminal $B[b]$. The remaining rules are of the form $B[b] \rightarrow b$ for each terminal symbol $\overline{B[b]}$ of G_1.

It is easy to see that $G_2^{(S[\#])} = G_1$. We also observe that, for each nonterminal $B[b]$ of G_2, grammar $G_2^{(B[b])}$ generates the regular language $\{b\}$. From Theorem 1 it then follows that G_2 is splittable if and only if G_1 is regular. ■

5. Discussion

Splittability of 2-LCFGs is of central importance to the processing efficiency of several models that are currently being used in statistical natural language parsing, as discussed in the Introduction. The notion of splittability has been originally introduced by Eisner (1997) and Eisner and Satta (1999) for the closely related formalism of head automaton grammars. In this article we gave an equivalent definition for 2-LCFGs, through the notion of d-equivalence, and we showed that splittability of 2-LCFGs is undecidable.

The result immediately carries over to head automaton grammars, through the standard mapping defined by Eisner and Satta (1999).

Our result is based on a characterization of the notion of splittability (theorem 1) in terms of a factorization of a bilexical CFG into linear context-free grammars with center markers, and on the regularity of these linear grammars. The same characterization has been claimed by Eisner and Satta (1999) relative to head automaton grammars, without a formal proof. Central to our result is a proof showing the undecidability of regularity for linear context-free grammars with center markers (Theorem 2). This strengthens an already known result stating the undecidability of regularity for general linear context-free languages.

The main implication of our results is that mechanical analysis of a 2-LCFG will not suffice to determine whether the grammar is splittable or not. There are trivial sufficient conditions for splittability, as for example requiring that the grammar is already in the split form of definition 2. Also, if $G^{(B[b])}$ has a single nonterminal, for each $B[b]$ that occurs as maximal projection in a 2-LCFG G, then G is trivially splittable. It is an open problem whether splittability is decidable in case the number of nonterminals in each $G^{(B[b])}$ is bounded by some constant larger than one. The proof techniques presented in this article do not seem to lend themselves to extension of our undecidability results in this direction. Other interesting sufficient conditions for splittability are not known.

To obtain a more general setting, one might depart from 2-LCFGs and consider Chomsky normal form context-free grammars where each binary rule classifies its two nonterminals into functor (or predicate projection) and argument. Also in this case, one can pose the question of whether the two processes of generating the left and the right arguments of each predicate are independent one of the other. The results reported in this article can be extended to show undecidability of this question as well.

Acknowledgments

We are indebted to the anonymous reviewers for many detailed and helpful comments. Among other things, the terminology has been improved relative to the first version of this article. This we owe to a valuable suggestion from one of the reviewers.

References

Alshawi, H. 1996. Head automata and bilingual tiling: Translation with minimal representations. In *34th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 167–176, Santa Cruz, CA.

Bordihn, H., M. Holzer, and M. Kutrib. 2005. Some non-semi-decidability problems for linear and deterministic context-free languages. In *Implementation and Application of Automata: 9th International Conference*, pages 68–79, Kingston.

Charniak, E. 2001. Immediate-head parsing for language models. In *39th Annual Meeting and 10th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 116–123, Toulouse.

Collins, M. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

Eisner, J. 1996. Three new probabilistic models for dependency parsing: An exploration. In *The 16th International Conference on Computational Linguistics*, volume 1, pages 340–345, Copenhagen.

Eisner, J. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *International Workshop on Parsing Technologies*, pages 54–65, Cambridge, MA.

Eisner, J. and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 457–464, College Park, MD.

Hopcroft, J. E. and J. D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Hunt III, H. B. and D. J. Rosenkrantz. 1974. Computational parallels between the regular and context-free languages. In *Conference Record of the Sixth Annual ACM Symposium on Theory of Computing*, pages 64–74, Seattle, WA.