Alexander Okhotin

Unambiguous Boolean grammars

Turku Centre *for* Computer Science

TUCS Technical Report
No 802, January 2007

Unambiguous Boolean grammars

Alexander Okhotin

Academy of Finland, *and*
Department of Mathematics, University of Turku, Turku, Finland, *and*
Turku Centre for Computer Science
alexander.okhotin@utu.fi

TUCS Technical Report
No 802, January 2007

Abstract

Boolean grammars are an extension of context-free grammars, in which all propositional connectives can be explicitly used in the rules. In this paper, the notion of ambiguity in Boolean grammars is defined. It is shown that the known transformation of a Boolean grammar to the binary normal form preserves unambiguity, and that every unambiguous Boolean language can be parsed in time $O(n^2)$. Linear conjunctive languages are shown to be unambiguous, while the existence of languages inherently ambiguous with respect to Boolean grammars is left open.

Keywords: ambiguity, parsing, Boolean grammars, conjunctive grammars, language equations

TUCS Laboratory
Discrete Mathematics for Information Technology

1 Introduction

Unambiguous context-free grammars are those that define a unique parse tree for every string they generate, that is, a syntactic structure is unambiguously assigned to every grammatical sentence.

A theoretical study of this class of grammars was carried out already in the first years of formal language theory. The undecidability of the problem whether a given context-free grammar is ambiguous was first proved by Floyd [4], while Greibach [7] extended this result to one-nonterminal linear context-free grammars. Some properties of unambiguous languages were determined by Ginsburg and Ullian [6]. In the later years a sophisticated theory was developed around the notion of ambiguity, leading, in particular, to deep results on the degree of ambiguity recently obtained by Wich [23].

As compared to context-free grammars of the general form, unambiguous context-free grammars are notable for their lower parsing complexity. A logarithmic-time parallel algorithm was proposed by Rytter [21], and later improved by Rossmanith and Rytter [20]. An adaptation of the well-known Cocke–Kasami–Younger algorithm for unambiguous grammars developed by Kasami and Torii [10] works in square time, while Earley's [3] algorithm achieves square-time performance on unambiguous grammars without any special modifications. The subclasses of even lower parsing complexity, the LR(k) and LL(k) context-free grammars, are notable for being the most practically used families of formal grammars.

This paper is the first to consider the notion of ambiguity in *Boolean grammars* [16], which are an extension of context-free grammars with explicit propositional connectives. Besides giving a greater freedom of constructing grammars, Boolean grammars are capable of specifying many non-context-free languages [16], as well as a simple model programming language [17]. On the other hand, the extended expressive power of Boolean grammars does not increase the complexity of parsing, which can still be done in time $O(n^3)$ using variants of Cocke–Kasami–Younger and Generalized LR [16, 19].

These results give a hope that Boolean grammars could be useful in practice, and it is worthwhile to investigate the unambiguous subclass of these grammars. Following an overview of the family of Boolean grammars given in Section 2, a definition of ambiguity for this family is given in Section 3. *Ambiguity in the choice of a rule* means that two distinct rules for some nonterminal can produce the same string; it is shown that this kind of ambiguity can be effectively eliminated in a given grammar. *Ambiguity of concatenation* means multiple factorizations of some string according to the body of some rule. A grammar is unambiguous if neither type of ambiguity occurs.

Given this definition, Section 4 reconsiders the known transformation of a Boolean grammar to a normal form, and it is shown that if the given grammar is unambiguous, then the resulting grammar in the normal form will be unambiguous as well. This result is useful for establishing an upper

bound on parsing complexity for unambiguous Boolean grammars, which is done in Section 5, where a new parsing algorithm for Boolean grammars in the normal form is obtained. The algorithm can be regarded as another variant of the Cocke–Kasami–Younger algorithm, obtained by refactoring both its data structures and its loops. Finally, the family of languages generated by unambiguous Boolean grammars is considered in Section 6 and compared to the related families. Some candidates for being inherently ambiguous languages are presented.

2 Boolean grammars

Definition 1 ([16]). *A Boolean grammar is a quadruple $G = (\Sigma, N, P, S)$, where Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols respectively; P is a finite set of rules of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \tag{1}$$

where $m+n \geqslant 1$, $\alpha_i, \beta_i \in (\Sigma \cup N)^*$; $S \in N$ is the start symbol of the grammar.

For each rule (1), the objects $A \to \alpha_i$ and $A \to \neg\beta_j$ (for all i, j) are called *conjuncts*, *positive* and *negative* respectively; the set of all conjuncts is denoted $conjuncts(P)$. Conjuncts of unknown sign will be referred to as *unsigned conjuncts*, and denoted $A \to \pm\alpha_i$ and $A \to \pm\beta_j$, Let $uconjuncts(P)$ be the set of all unsigned conjuncts.

A Boolean grammar is called a *conjunctive grammar* [13], if negation is never used, that is, $n = 0$ for every rule (1). It is a *context-free grammar* if neither negation nor conjunction are allowed, that is, $m = 1$ and $n = 0$ for each rule. Another important particular case of Boolean grammars is formed by *linear conjunctive grammars*, in which every conjunct is of the form $A \to uBv$ or $A \to w$, where $u, v, w \in \Sigma^*$, $w \in N$. (1). Linear conjunctive grammars are equal in power to *linear Boolean grammars* with conjuncts $A \to \pm uBv$ or $A \to w$, as well as to trellis automata, also known as one-way real-time cellular automata [2, 14]. The relation between these families and other common families of formal languages is shown in Figure 1, where arrows indicate inclusions, proper unless labelled by a question mark.

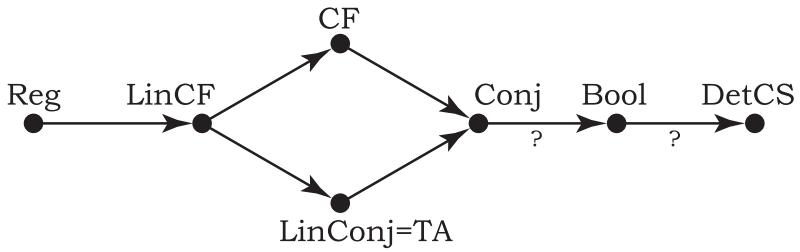

Figure 1: The hierarchy of language families.

Intuitively, a rule (1) of a Boolean grammar can be read as follows: every string w over Σ that satisfies each of the syntactical conditions represented by $\alpha_1, \ldots, \alpha_m$ and none of the syntactical conditions represented by $\beta_1, \ldots, \beta_m$ therefore satisfies the condition defined by A. Though this is not yet a formal definition, this understanding is sufficient to construct grammars.

Example 1. *The following grammar generates the language* $\{a^n b^n c^n | n \geqslant 0\}$*:*

$$\begin{aligned} S &\to AB\&DC \\ A &\to aA \mid \varepsilon \\ B &\to bBc \mid \varepsilon \\ C &\to cC \mid \varepsilon \\ D &\to aDb \mid \varepsilon \end{aligned}$$

This grammar, which is actually conjunctive, represents this language as an intersection of two context-free languages:

$$\underbrace{\{a^n b^n c^n \mid n \geqslant 0\}}_{L(S)} = \underbrace{\{a^i b^j c^k \mid j = k\}}_{L(AB)} \cap \underbrace{\{a^i b^j c^k \mid i = j\}}_{L(DC)}$$

A related non-context-free language can be specified by inverting the sign of one of the conjuncts in this grammar.

Example 2. *The following Boolean grammar generates the language* $\{a^m b^n c^n \mid m, n \geqslant 0, m \neq n\}$*:*

$$\begin{aligned} S &\to AB\&\neg DC \\ A &\to aA \mid \varepsilon \\ B &\to bBc \mid \varepsilon \\ C &\to cC \mid \varepsilon \\ D &\to aDb \mid \varepsilon \end{aligned}$$

This grammar is based upon the following representation.

$$\underbrace{\{a^n b^m c^m \mid m, n \geqslant 0, m \neq n\}}_{L(S)} = \{a^i b^j c^k \mid j = k \text{ and } i \neq j\} = L(AB) \cap \overline{L(DC)}$$

Let us now consider formal definitions of the language generated by a Boolean grammar, which can be given in several different ways [11, 16]. Each definition departs, explicitly or implicitly, from the interpretation of a grammar as a system of equations with formal languages as unknowns:

Definition 2. *Let* $G = (\Sigma, N, P, S)$ *be a Boolean grammar. The system of language equations associated with* G *is a resolved system of language*

equations over Σ in variables N, in which the equation for each variable $A \in N$ is

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \cap \bigcap_{j=1}^{n} \overline{\beta_j} \right] \tag{2}$$

Each instance of a symbol $a \in \Sigma$ in such a system defines a constant language $\{a\}$, while each empty string denotes a constant language $\{\varepsilon\}$. A solution of such a system is a vector of languages $(\ldots, L_C, \ldots)_{C \in N}$, such that the substitution of L_C for C, for all $C \in N$, turns each equation (2) into an equality.

The semantics of the special case of conjunctive and context-free grammars is easy to define [13], because in the absence of negation a system (2) always has a least solution with respect to componentwise inclusion.

This is no longer the case for Boolean grammars of the general form, for which this system may have no solutions or multiple pairwise incomparable solutions. If there are no solutions, the grammar is always considered invalid, and if there are solutions, there is a room for interpretation: one of these solutions can be deemed "the right one" and used to define the semantics, or the grammar could also be considered invalid. Different ways of settling this question are known as semantics for Boolean grammars [11, 16].

Definition 3. *A semantics of Boolean grammars is a certain law, according to which some quadruples $G = (\Sigma, N, P, S)$ as in Definition 1 are considered well-formed grammars, and for every such quadruple, a certain solution $(\ldots, L_C, \ldots)$ of the associated system of language equations is specified. Then, for every $A \in N$, the language $L_G(A)$ is defined as L_A (with respect to this semantics). This notation is extended to any expressions formed of terminal and nonterminal symbols, concatenation and Boolean operations as follows: $L_G(\varepsilon) = \{\varepsilon\}$, $L_G(a) = \underline{\{a\}}$, $L_G(\psi \mid \xi) = L_G(\psi) \cup L_G(\xi)$, $L_G(\psi \& \xi) = L_G(\psi) \cap L_G(\xi)$, $L_G(\neg\psi) = \overline{L_G(\psi)}$, $L_G(\psi \cdot \xi) = L_G(\psi) \cdot L_G(\xi)$. The language generated by the grammar is $L(G) = L_G(S)$.*

A requirement that naturally comes to mind is the uniqueness of a solution. However, the resulting class of languages is too broad to be useful: it is known that every recursive language can be represented as a unique solution of such a system [16], and the way it is represented contradicts the intuitive semantics of Boolean grammars given above. In order to give a satisfactory formal definition of Boolean grammars, some stronger condition must be assumed [16].

The most straightforward semantics is defined by the following condition of *strong uniqueness* of a solution. Other semantics have also been considered [11, 16], but these details of formal definition are beyond the scope of this paper.

Definition 4. *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar, let (2) be the associated system of language equations. Suppose that for every finite language $M \subset \Sigma^*$ (such that for every $w \in M$ all substrings of w are also in M) there exists a unique vector of languages $(\ldots, L_C, \ldots)_{C \in N}$ ($L_C \subseteq M$), such that a substitution of L_C for C, for each $C \in N$, turns every equation (2) into an equality modulo intersection with M.*

Let us now define *parse trees* for Boolean grammars [16]. These are, strictly speaking, finite acyclic graphs rather than trees. A parse tree of a string $w = a_1 \ldots a_{|w|}$ from a nonterminal A contains a leaf labelled a_i for every i-th position in the string; the rest of the vertices are labelled with rules from P. The subtree accessible from any given vertex of the tree contains leaves in the range between $i+1$ and j, and thus corresponds to a substring $a_{i+1} \ldots a_j$. In particular, each leaf a_i corresponds to itself.

For each vertex labelled with a rule

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n$$

and associated to a substring $a_{i+1} \ldots a_j$, the following conditions hold:

1. It has exactly $|\alpha_1 \ldots \alpha_m|$ direct descendants corresponding to the symbols in positive conjuncts. For each nonterminal in $\alpha_1 \ldots \alpha_m$, the corresponding descendant is labelled with some rule for that nonterminal, and for each terminal $a \in \Sigma$, the descendant is a leaf labelled with a.

2. For each t-th positive conjunct of this rule, let $\alpha_t = s_1 \ldots s_\ell$. There exist numbers $i_1, \ldots, i_{\ell-1}$, where $i = i_0 \leqslant i_1 \leqslant \ldots \leqslant i_{\ell-1} \leqslant i_\ell = j$, such that each descendant corresponding to s_r encompasses the substring $a_{i_{r-1}+1} \ldots a_{i_r}$.

3. For each t-th negative conjunct of this rule, $a_{i+1} \ldots a_j \notin L_G(\beta_t)$.

The root is the unique node with no incoming arcs; it is labelled with any rule for the nonterminal A, and all leaves are reachable from it. To consider the uniqueness of a parse tree for different strings, it is useful to assume that only terminal leaves can have multiple incoming arcs.

The condition 3 ensures that the requirements imposed by negative conjuncts are satisfied. However, nothing related to these negative conjuncts is reflected in the actual trees. For some grammars, this effectively means that the tree does not convey any information. Consider the following example.

Example 3 (cf. [16, Example 4])**.** *The language $\{a^{2^n} \mid n \geqslant 0\}$ is generated*

by the following Boolean grammar:

$$
\begin{aligned}
S &\to A\&\neg aA \mid aB\&\neg B \mid aC\&\neg C\\
A &\to aBB\\
B &\to E\&\neg CC\\
C &\to E\&\neg DD\\
D &\to E\&\neg A\\
E &\to aE \mid \varepsilon
\end{aligned}
$$

Looking at the positive conjuncts in this grammar, it can be seen that the parse tree of any a^{2^n} from S reflects only the decomposition of the string as aE or aEE, which has no meaning. However, this is quite an artificial grammar, while in more common cases, such as in Example 2, the parse tree contains useful partial information.

3 Defining ambiguity

Unambiguous context-free grammars can be defined in two ways:

1. for every string generated by the grammar there is a unique parse tree (in other words, a unique leftmost derivation);

2. for every nonterminal A and for every string $w \in L(A)$ there exists a unique rule $A \to s_1 \dots s_\ell$, such that $w \in L(s_1 \dots s_\ell)$, and a unique factorization $w = u_1 \dots u_\ell$, such that $u_i \in L(s_i)$.

Assuming that $L(A) \neq \varnothing$ for every nonterminal A, these definitions are equivalent.

The first definition becomes useless in the case of Boolean grammars, because the parse tree does not reflect the full information about the membership of the string in the language (negative conjuncts are not accounted for). The requirement of parse tree uniqueness can be trivially satisfied as follows. Given any grammar G over an alphabet $\Sigma = \{a_1, \dots, a_m\}$ and with a start symbol S, one can define a new start symbol S' and additional symbols $\widehat{S}$ and A, with the following rules:

$$
\begin{aligned}
S' &\to A\&\neg\widehat{S}\\
\widehat{S} &\to A\&\neg S\\
A &\to a_1A \mid \ \dots\ \mid a_mA \mid \varepsilon
\end{aligned}
$$

This grammar generates the same language, and every string in $L(G)$ has a unique parse tree, which reflects only the nonterminal A and hence bears no essential information.

Let us generalize the second approach to defining ambiguity.

Definition 5. *A Boolean grammar $G = (\Sigma, N, P, S)$ is said to be unambiguous if*

I. Different rules for every single nonterminal A generate disjoint languages, that is, for every string w there exists at most one rule

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n,$$

such that $w \in L_G(\alpha_1) \cap \ldots \cap L_G(\alpha_m) \cap \overline{L_G(\beta_1)} \cap \ldots \cap \overline{L_G(\beta_n)}$.

II. All concatenations are unambiguous, that is, for every conjunct $A \to \pm s_1 \ldots s_\ell$ and for every string w there exists at most one factorization $w = u_1 \ldots u_\ell$, such that $u_i \in L_G(s_i)$ for all i.

Note that this definition is applicable to any semantics of Boolean grammars, as long as $L_G(\gamma)$ is defined according to Definition 3.

For instance, both grammars in Examples 1 and 2 are unambiguous. To see that condition II is satisfied with respect to the conjunct $S \to AB$, consider that a factorization $w = uv$, where $u \in L(A)$ and $v \in L(B)$, implies that $u = a^*$ and $v \in b^*c^*$, so the boundary between u and v cannot be moved. The same argument applies to the conjuncts $S \to DC$ and $S \to \neg DC$. Different rules for each of A, B, C, D clearly generate disjoint languages.

On the other hand, the grammar in Example 3 is ambiguous. To see this, consider the string $w = aa$ and the conjunct $A \to aBB$: there exist factorizations $w = a \cdot \varepsilon \cdot a$ and $w = a \cdot a \cdot \varepsilon$, where $\varepsilon, a \in L(B)$, so the concatenation $\{a\} \cdot L(B) \cdot L(B)$ is ambiguous.

Though, as mentioned above, the uniqueness of a parse tree does not guarantee that the grammar is unambiguous, the converse holds:

Proposition 1. *For any unambiguous Boolean grammar, for any nonterminal $A \in N$ and for any string $w \in L_G(A)$, there exists a unique parse tree of w from A (assuming that only terminal nodes may have multiple incoming arcs).*

Another thing to note is that the first condition in the definition of unambiguity can be met for every grammar using simple transformations. Consider any nonterminal A and assume each of its rules consists of a single positive conjunct, that is, its rules are

$$A \to \alpha_1 \mid \ldots \mid \alpha_n \quad (\text{where } \alpha_i \in (\Sigma \cup N)^*) \tag{3}$$

There is no loss of generality in this assumption, because any rule for A can be replaced with a rule of the form $A \to A'$, where A' is a new nonterminal with a single rule replicating the original rule for A. Then the rules (3) can be

replaced with the following n rules, which clearly generate disjoint languages:

$$\begin{array}{lcl} A & \to & \alpha_1 \\ A & \to & \alpha_2 \& \neg \alpha_1 \\ A & \to & \alpha_3 \& \neg \alpha_1 \& \neg \alpha_2 \\ & \vdots & \\ A & \to & \alpha_n \& \neg \alpha_1 \& \neg \alpha_2 \& \ldots \& \neg \alpha_{n-1} \end{array} \tag{3'}$$

If this transformation is applied to every nonterminal, the resulting grammar will satisfy condition I. Additionally, condition II, if it holds, will be preserved by the transformation.

Proposition 2. *For every Boolean grammar there exists a Boolean grammar generating the same language, for which the condition I is satisfied. If the original grammar satisfies the condition II, then so will the constructed grammar.*

This property does not hold for context-free grammars. Consider the standard example of an inherently ambiguous context-free language:

$$\{a^i b^j c^k \mid i, j, k \geqslant 0,\ i = j \text{ or } j = k\}.$$

Following is the most obvious ambiguous context-free grammar generating this language:

$$\begin{array}{lcl} S & \to & AB \mid DC \\ A & \to & aA \mid \varepsilon \\ B & \to & bBc \mid \varepsilon \\ C & \to & cC \mid \varepsilon \\ D & \to & aDb \mid \varepsilon \end{array}$$

The condition II is satisfied for the same reasons as in Examples 1 and 2. On the other hand, the condition I is failed for the nonterminal S and for strings of the form $a^n b^n c^n$, which can be obtained using each of the two rules, and this is what makes this grammar ambiguous.

If the above context-free grammar is regarded as a Boolean grammar (ambiguous as well), then the given transformation disambiguates it in the most natural way by replacing the rules for the start symbol with the following rules:

$$S \to AB \mid DC \& \neg AB\,.$$

We have thus seen that ambiguity in the choice of a rule represented by condition I can be fully controlled in a Boolean grammar, which is a practically very useful property not found in the context-free grammars. On the other hand, ambiguity of concatenations formalized in condition II seems to be, in general, beyond such control.

4 Normal forms

Definition 6. *A Boolean grammar is said to be in the binary normal form [16], if all of its rules are of the form*

$$A \to B_1C_1\& \ldots \&B_mC_m\&\neg D_1E_1\& \ldots \&\neg D_nE_n\&\neg\varepsilon \quad (m \geqslant 1, n \geqslant 0)$$
$$A \to a$$
$$S \to \varepsilon \quad \text{(only if } S \text{ does not appear in right-hand sides of rules)}$$

It is known that every Boolean grammar can be transformed to an equivalent grammar in the binary normal form [16]. Let us refine this result by showing that this known transformation converts an unambiguous Boolean grammar to an unambiguous grammar in the normal form.

The transformation of a Boolean grammar $G = (\Sigma, N, P, S)$ to the binary normal form proceeds as follows. For every $s_1 \ldots s_\ell \in (\Sigma \cup N)^*$, denote

$$\rho(s_1 \ldots s_\ell) = \{s_{i_1} \ldots s_{i_k} \mid k \geqslant 1,\ 1 \leqslant i_1 < \ldots < i_k \leqslant \ell,\ \text{and}\\ j \notin \{i_1, \ldots, i_k\} \text{ implies } \varepsilon \in L_G(s_j)\}$$

At the first step, a new grammar $G_1 = (\Sigma, N, P_1, S)$ is constructed, where, for every rule

$$A \to \alpha_1\& \ldots \&\alpha_m\&\neg\beta_1\& \ldots \&\neg\beta_n,$$

from P, in which $\rho(\alpha_i) = \{\mu_{i1}, \ldots, \mu_{ik_i}\}$ and $\rho(\beta_j) = \{\nu_{j1}, \ldots, \nu_{j\ell_j}\}$, the set P_1 contains a rule

$$A \to \mu_{1t_1}\& \ldots \&\mu_{mt_m}\&\neg\nu_{11}\& \ldots \&\neg\nu_{1\ell_1}\& \ldots \&\neg\nu_{n1}\& \ldots \&\neg\nu_{1\ell_n}\&\neg\varepsilon,$$

for every vector of numbers $(t_1, \ldots, t_m)$ ($1 \leqslant t_i \leqslant k_i$ for all i). It is known that, for every $A \in N$, $L_{G_1}(A) = L_G(A) \setminus \{\varepsilon\}$ [16].

At the second step, another Boolean grammar $G_2 = (\Sigma, N, P_2, S)$ is constructed on the basis of G_1. This grammar is free of *unit conjuncts* of the form $A \to \pm B$. The most important property of the constructed grammar is as follows. Let $R = \{\gamma \mid A \to \pm\gamma \in uconjuncts(P_1), \gamma \notin N\} = \{\eta_1, \ldots, \eta_\ell\}$. Then every rule in P_2 is of the general form

$$A \to \alpha_1\& \ldots \&\alpha_m\&\neg\beta_1\& \ldots \&\neg\beta_n, \quad \text{where } \{\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_n\} = R \tag{4}$$

In other words, the body of every conjunct in P_1 appears either positively or negatively in every rule in P_2.

The rest of the transformation is obvious. At the third step, every "long" conjunct of the form $A \to \pm s\alpha$, where $s \in \Sigma \cup N$ and $|\alpha| \geqslant 2$, is shortened by adding a new nonterminal A' with a rule $A' \to \alpha$ and by replacing the body of the original conjunct with sA'. This is done until every conjunct is either $A \to \pm\alpha$ with $|\alpha| = 1, 2$ (where $|\alpha| = 1$ implies $\alpha = a \in \Sigma$) or $A \to \neg\varepsilon$.

Let $G_3 = (\Sigma, N \cup N', P_3, S)$ be the resulting grammar; obviously, $L(G_3) = L(G_2)$. At the final fourth step every conjunct $A \to \pm as$ or $A \to \pm sa$, where $a \in \Sigma$ and $s \in \Sigma \cup N$, has its body replaced with $X_a s$ or $s X_a$, respectively, where X_a is a new nonterminal with a rule $X_a \to a$. The resulting grammar $G_4 = (\Sigma, N \cup N' \cup N'', P_4, S)$ generates the same language $L(G_4) = L(G_3)$.

The following property of this transformation is important for this paper.

Lemma 1. *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar compliant to the semantics of strongly unique solution, assume $L_G(A) \neq \varnothing$ for any $A \in N$. Let G_1, G_2, G_3 and G_4 be obtained from G by the above construction. Then*

- *The grammar G_2, as well as the subsequent grammars obtained, satisfies condition I from the definition of an unambiguous grammar.*
- *If G satisfies condition II, then each grammar obtained satisfies condition II.*

Proof. Let us first prove that the grammar G_2 satisfies condition I. Assume the contrary, then for some $A \in N$ there exist two distinct rules of the form (4), such that some string $w \in \Sigma^*$ can be obtained from either rule. Both rules are formed from the same set of unsigned conjuncts, but some of them may have different signs in different rules. Since the rules are distinct, at least one pair of conjuncts with different signs should exist; let one rule contain a conjunct $A \to \gamma$ and let the other contain $A \to \neg\gamma$. Each rule generates w by assumption, and hence $w \in L(\gamma)$ and $w \notin L(\gamma)$, which forms a contradiction.

It is easy to see that condition I is preserved in the transformation of G_2 to G_3 and G_4. For each nonterminal $A \in N$, there is a one-to-one correspondence between rules for A in P_2 and in P_3 (or in P_4), such that the corresponding rules generate the same languages, and thus these languages remain disjoint. Each of the new nonterminals in N' and N'' has a unique rule, so condition I is again met.

Now assume G satisfies condition II, and let us prove that each step of the transformation preserves this property. Consider the first step. For every $A \to \pm s_1 \dots s_\ell \in \mathit{uconjuncts}(P)$, $\mathit{uconjuncts}(P_1)$ contains every $A \to \pm s_{i_1} \dots s_{i_k}$, such that $k \geqslant 1$, $1 \leqslant i_1 < \ldots < i_k \leqslant \ell$ and for every j in $\{1, \dots, \ell\} \setminus \{i_1, \dots, i_k\}$, $\varepsilon \in L_G(s_j)$. Every conjunct in G_1 is formed in this way, with the exception of $A \to \neg\varepsilon$. Consider any two representations of any string as $L_{G_1}(s_{i_1}) \cdot \ldots \cdot L_{G_1}(s_{i_k})$:

$$w = u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k} \quad (\text{where } u_{i_t}, v_{i_t} \in L_{G_1}(s_{i_t}) \text{ for all } t) \tag{5}$$

Consider that $L_{G_1}(s_{i_j}) \subseteq L_G(s_{i_j})$, and define $u_j = v_j = \varepsilon \in L_G(s_j)$ for all $j \in \{1, \dots, \ell\} \setminus \{i_1, \dots, i_k\}$. Then $u_1 \dots u_\ell = v_1 \dots v_\ell = w$, and since G satisfies condition II, $u_j = v_j$ for all $j \in \{1, \dots, \ell\}$. Hence, the factorizations (5) are actually the same, and, since the choice of the conjunct and the string was arbitrary, G_1 satisfies condition II.

In the next phase, when G_1 is converted to G_2, no new conjunct bodies are created, and thus condition II is trivially preserved. The conversion of G_2 to G_3 is a series of elementary steps, and it is sufficient to prove the correctness of one such step. Let $\widehat{G}$ be a grammar with a conjunct $A \to \pm s_1 s_2 \dots s_\ell$, and let $\widetilde{G}$ be constructed by replacing this conjunct with $A \to \pm s_1 A'$, where A' is a new nonterminal with the rule $A' \to s_2 \dots s_\ell$.

Suppose some string $w \in \Sigma^*$ can be represented as $w = u_2 \dots u_\ell = v_2 \dots v_\ell$, where $u_j, v_j \in L_{\widetilde{G}}(s_j) = L_{\widehat{G}}(s_j)$ for $j = 2, \dots, \ell$. Since it is assumed that $L_{\widehat{G}(s_1)} \neq \varnothing$, there exists a string $x \in L_{\widehat{G}(s_1)}$. Let $u_1 = v_1 = x$, then the string xw can be represented as $xw = u_1 u_2 \dots u_\ell = v_1 v_2 \dots v_\ell$, where $u_j, v_j \in L_{\widehat{G}}(s_j)$ for $j = 1, \dots, \ell$. By assumption, $\widehat{G}$ satisfies condition II, hence $u_j = v_j$ for all j, and hence the given factorization of w as $L_{\widetilde{G}}(s_2) \cdot \ldots \cdot L_{\widetilde{G}}(s_\ell)$ is unambiguous.

Consider the other case of a conjunct $A \to \pm s_1 A'$. Suppose some string $w \in \Sigma^*$ can be represented as $w = u_1 u' = v_1 v'$, where $u_1, v_1 \in L_{\widetilde{G}}(s_1)$ and $u', v' \in L_{\widetilde{G}}(A')$ Since $L_{\widetilde{G}}(A') = L_{\widehat{G}}(s_2 \dots s_\ell)$, there exist factorizations $u' = u_2 \dots u_\ell$ and $v' = v_2 \dots v_\ell$, where $u_j, v_j \in L_{\widehat{G}}(s_j)$ for $j = 2, \dots, \ell$ Thus we obtain two factorizations of w as $w = u_1 u_2 \dots u_\ell = v_1 v_2 \dots v_\ell$, such that $u_j, v_j \in L_{\widehat{G}}(s_j)$ for $j = 1, \dots, \ell$. Since $\widehat{G}$ satisfies condition II, this implies $u_1 = v_1$ and hence $u' = v'$. This completes the proof that G_2 satisfies condition II.

In the final step, a conjunct $A \to \pm as$ is replaced with $A \to \pm X_a s$, where X_a generates $\{a\}$, and $A \to \pm sa$ is treated similarly. The factorizations as and $X_a s$ are the same with respect to ambiguity. □

Theorem 1. *For every Boolean grammar there exists and can be effectively constructed a Boolean grammar in the binary normal form (as in Definition 6) generating the same language. If the original Boolean grammar is unambiguous, then so will be the constructed Boolean grammar.*

5 Parsing unambiguous languages

One of the important properties of unambiguous context-free grammars is efficient parsing. Some cubic-time algorithms for general context-free grammars work in square time in the unambiguous case [3, 10]. Similarly, log-square-time parallel algorithms can be sped up to logarithmic time [21]. As we shall see later on, Rytter's [21] parallel algorithm is very unlikely to have an analogue for unambiguous Boolean grammars.

On the other hand, the idea behind Earley's [3] argument that his algorithm works in square time for unambiguous context-free grammars is generally applicable to parsing algorithms for Boolean grammars, provided that the data flow inside the algorithm arranges for different conjuncts of the same rule to be considered together. If the grammar contains a rule $A \to \alpha \& \beta$, and it holds that $u \in L_G(\alpha)$ and $u \in L_G(\beta)$ for some substring u of the input

string, then the algorithm should determine the membership of u in $L_G(\alpha)$ and in $L_G(\beta)$ at the same time, so that $u \in L_G(A)$ could be deduced. Since no such property is required for context-free grammars, Earley's algorithm can consider $u \in L_G(\alpha)$ and $u \in L_G(\beta)$ (for a grammar containing the rules $A \to \alpha$ and $A \to \beta$) at different times. Therefore, a parsing algorithm for unambiguous Boolean grammars has to be constructed from scratch.

The proposed algorithm is applicable to all Boolean grammars in the binary normal form, as in Definition 6. In view of Theorem 1, there is no loss of generality in this assumption. The algorithm uses dynamic programming to construct a two-dimensional table E indexed by positions in the input and nonterminals. Each entry of this table assumes the value of a set of positions in the input string, which are stored as a list in an ascending order. The element corresponding to a position k ($1 \leqslant k \leqslant n$) and a nonterminal $A \in N$ is denoted $E_k[A]$. By definition, i should be in $E_k[A]$ if and only if $0 \leqslant i < k$ and $a_{i+1} \dots a_k \in L_G(A)$. In the end of the computation, each list $E_k[A]$ will contain exactly these numbers. Then, accordingly, the entire string $a_1 \dots a_n$ is in $L(G)$ if and only if the position 0 is in $E_n[S]$.

Algorithm 1. *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar in binary normal form. For every $X \subseteq conjuncts(G)$, define*

$$f(X) = \{A \mid \exists A \to B_1C_1\& \dots \&B_\ell C_\ell \& \neg D_1E_1 \& \dots \& \neg D_mE_m \& \neg\varepsilon \in P, \text{ such that } A \to B_1C_1, \dots, A \to B_\ell C_\ell \in X \text{ and } A \to \neg D_1E_1, \dots, A \to \neg D_mE_m \notin X\}$$

Let $w = a_1 \dots a_n$, where $n \geqslant 1$ and $a_i \in \Sigma$, be an input string. For all $j \in \{1, \dots, n\}$, let $E_j[A]$ be a variable ranging over subsets of $\{0, \dots, j-1\}$; for all $k \in \{0, \dots, n-1\}$, $T[k]$ ranges over subsets of $uconjuncts(P)$.

```
1: let E_j[A] = ∅ for all j = 1, ..., n and A ∈ N
2: for j = 1 to n do
3:     for all all A ∈ N do
4:         if A → a_j ∈ P then
5:             E_j[A] = {j − 1}
6:         else
7:             E_j[A] = ∅
8:     let T[k] = ∅ for all k (0 ⩽ k < j − 1)
9:     for k = j − 1 to 1 do
10:        for all A → ±BC ∈ uconjuncts(P) do
11:            if k ∈ E_j[C] then
12:                for all i ∈ E_k[B] do
13:                    T[i] = T[i] ∪ {A → ±BC}
14:        for all A ∈ f(T[k − 1]) do
15:            E_j[A] = E_j[A] ∪ {k − 1}
16: accept iff 0 ∈ E_n[S]
```

Each $E_j[A]$ is stored as a list, with elements sorted in an ascending order. The operations on this data structure are implemented as follows:

Lines 1, 5 and 7: *A one-element list or an empty list is created.*

Line 11: *The first element in the list is checked. If it is not k, it is assumed that k is not in the list.*

Line 12: *The list is traversed.*

Line 15: *The new element is inserted in the beginning of the list.*

Line 16: *As in line 11, only the first element is checked.*

The goal of each j-th iteration of the outer loop (line 2) is to determine, for all $A \in N$, the membership in $L_G(A)$ of substrings of the input string ending at its j-th position. This information is stored in $E_j[A]$. The first nested loop in lines 3–7 handles substrings of length 1, that is, it records in $E_j[A]$ whether a_j is in $L_G(A)$. Substrings of greater length ending at the j-th position are processed in the second nested loop by k (line 9).

This loop constructs an auxiliary data structure T: for each $i \in \{0, \ldots, j-2\}$, $T[i]$ is meant to contain all conjuncts $A \to \pm BC$, such that the substring starting from the position $i+1$ and ending at the position j is in $L_G(BC)$. Every k-th iteration of this loop, denoted (j, k), considers substrings of various length starting at any position $i+1 \in \{1, 2, \ldots, k\}$ and ending at the position j. The goal is to determine all such substrings, which belong to $L_G(BC)$ for some unsigned conjunct $A \to \pm BC$, and in which the middle point in their factorization into $u \in L_G(B)$ and $v \in L_G(C)$ is exactly $k+1$, that is, the first part u ends at the position k and the second part v starts at the position $k+1$. These substrings uv are identified by first considering the appropriate unsigned conjunct, then checking the membership of the second substring in $L_G(C)$ (line 11), and finally by enumerating all appropriate first parts using the data in $E_k[B]$.

This is used to fill the elements of T, namely $T[k-1], T[k-2], \ldots, T[0]$, with appropriate conjuncts. An element $T[k-1]$ gets completely filled in course of iteration (j, k), and at this point the set of nonterminals generating the substring starting from the position k and ending at the position j can be obtained as $f(T[k-1])$, which is done in lines 14–15.

To verify the algorithm's correctness, there are three properties to establish: first, that the given implementation of $E_j[A]$ by lists faithfully represents the high-level set operations. Second, it has to be shown that the algorithm is a correct recognizer, that is, it accepts w if and only if $w \in L(G)$. Third, it remains to demonstrate that the algorithm works in time $O(n^2)$ on every unambiguous grammar.

Let us see that, indeed, the lists $E_j[A]$ stay sorted in course of the computation, and the tests in lines 11, 16 and the insertion in line 15 can be implemented as described.

Lemma 2. *Each list $E_j[A]$ always remains sorted. Each time the algorithm checks the condition in line 11, every set $E_j[A]$ does not contain elements less than k. Each time the algorithm is about to execute line 15, the set $E_j[A]$ does not contain elements less than k.*

Proof. An element $k-1$ ($1 \leqslant k < j$) can be added to $E_j[A]$ only at the iteration (j,k). Hence, in the beginning of each iteration (j,k) the current value of $E_j[A]$ is a subset of $\{k, k+1, \ldots, j-1\}$. As a result, if $E_j[A]$ is sorted before the assignment in line 15, it remains sorted after the assignment. All three claims follow. □

Let us continue with the correctness statement of the algorithm, which claims what values should the variables have at certain points of the computation.

To unify the notation, let us refer to the point before the iteration $j = 1$, that is, to the very beginning of the execution, as "after the iteration 0". Similarly, the point before the iteration $(j, k = j-1)$, that is, inside iteration j right before the loop by k is entered, will be referred to as "after the iteration (j,j)". Then the statement of correctness can be succinctly formulated as follows:

Lemma 3 (Correctness of Algorithm 1)**.** *For every Boolean grammar in the binary normal form, in the computation of the above algorithm on a string $w \in \Sigma^+$,*

i. after iteration j, for each $A \in N$ and for each $t \in \{1, \ldots, j\}$, the set $E_t[A]$ equals

$$\{i \mid 0 \leqslant i < t \text{ and } a_{i+1} \ldots a_t \in L_G(A)\}; \tag{6}$$

ii. after iteration (j,k), every $E_j[A]$ ($A \in N$) equals

$$\{i \mid k-1 \leqslant i < j \text{ and } a_{i+1} \ldots a_j \in L_G(A)\}; \tag{7}$$

iii. after iteration (j,k), every $T[i]$ ($0 \leqslant i < j$) equals

$$\{A \to \pm BC \mid \exists \ell (k \leqslant \ell < j) : a_{i+1} \ldots a_\ell \in L(B) \text{ and } a_{\ell+1} \ldots a_j \in L(C)\}. \tag{8}$$

Proof. The proof is by a nested induction corresponding to the structure of the loops. The outer claim (i) is proved by induction on j.

Basis: the beginning of the execution is the point "after iteration $j = 0$", when each $E_j[A]$ equals $\varnothing$. Here claim (i) trivially holds, because there are no applicable ts.

Induction step: It has to be proved that every j-th iteration of the outer loop effectively assigns

$$E_j[A] = \{i \mid 0 \leqslant i < j \text{ and } a_{i+1} \ldots a_j \in L_G(A)\} \quad \text{(for every } A \in N\text{)}.$$

To prove this, an inner induction is used to establish claims (ii–iii).

Basis, $k = j$**:** The point "after iteration (j, j)" is reached when lines 3–8 have been executed, and the nested loop by k is about to be entered. Let us substitute $k = j$ into claim (ii):

$$\{i | \underbrace{j-1 \leqslant i < j}_{i=j-1} \text{ and } \underbrace{a_{i+1} \dots a_j}_{a_j} \in L_G(A)\} = \begin{cases} \varnothing, & \text{if } a_j \notin L_G(A) \\ \{j-1\}, & \text{if } a_j \in L_G(A) \end{cases}$$

Since the grammar is in the normal form, $a_i \in L_G(A)$ if and only if $A \to a_i \in P$, and hence the lines 3–7 assign the appropriate values. A similar substitution of $k = j$ into claim (iii) results in $\{A \to \pm BC \mid \exists \ell\ (j \leqslant \ell < j) : \langle \dots \rangle\} = \varnothing$, which is consistent with line 8.

Induction step $k + 1 \to k$ $(j > k \geqslant 1)$**:** Assume iterations $(j, j-1), (j, j-2), \dots, (j, k+2), (j, k+1)$, have already been executed, and the iteration (j, k) has just started, in which line 10 is about to be executed. By the (inner) induction hypothesis, at this point, for each $A \in N$,

$$E_j[A] = \{i \mid k \leqslant i < j \text{ and } a_{i+1} \dots a_j \in L_G(A)\}, \tag{9}$$

while for each i,

$$T[i] = \{A \to \pm BC \mid \text{there exists } \ell\ (k+1 \leqslant \ell < j), \text{ such that } a_{i+1} \dots a_\ell \in L(B) \text{ and } a_{\ell+1} \dots a_j \in L(C)\}.$$

Let us first show that the execution of lines 10–13 sets every $T[i]$ to (8). It has to be proved that an unsigned conjunct $A \to \pm BC$ is added to $T[i]$ if and only if $a_{i+1} \dots a_k \in L(B)$ and $a_{k+1} \dots a_j \in L(C)$.

Suppose these statements hold. Then, according to the outer induction hypothesis, $i \in E_k[B]$ (since $k < j$), and by (9), $k \in E_j[C]$. Therefore, once the conjunct $A \to \pm BC$ is considered in line 10, the condition in line 11 will be true, then the loop in line 12 will be executed and will eventually find i in the list, and $A \to \pm BC$ will be added to $T[i]$ in line 13. Conversely, if $A \to \pm BC$ is added to $T[i]$, then $i \in E_k[B]$ and $k \in E_j[C]$, which implies $a_{i+1} \dots a_k \in L(B)$ and $a_{k+1} \dots a_j \in L(C)$, respectively.

We have thus proved that when the iteration (j, k) proceeds with the second inner loop starting in line 9, each $T[i]$ is already of the form (8). This, in particular, implies

$$T[k-1] = \{A \to \pm BC \mid a_k \dots a_j \in L(BC)\}, \tag{10}$$

because the middle point in the factorization of $a_k \dots a_j$ as $L(B) \cdot L(C)$ is always in $\{k, \dots, j-1\}$. Each $E_j[A]$ remains as in (9) at this point, and the claim is that the lines 14–15 set $E_j[A]$ to (ii), for each $A \in N$.

It suffices to prove that $k-1$ is added to $E_j[A]$ if and only if $a_k \dots a_j \in L_G(A)$.

Note that $|a_k \dots a_j| \geqslant 2$, since $k < j$. Then $a_k \dots a_j \in L_G(A)$ if and only if there exists a rule

$$A \to B_1C_1\& \dots \& B_\ell C_\ell \& \neg D_1E_1\& \dots \& \neg D_mE_m \& \neg\varepsilon,$$

such that $a_k \dots a_j \in L_G(B_iC_i)$ and $a_k \dots a_j \notin L_G(D_tE_t)$ for all appropriate i and t. By (10), this is equivalent to $A \to \pm B_iC_i \in T[k-1]$ and $A \to \pm D_tE_t \notin T[k-1]$ for all i and t, which in turn holds if and only if $A \in f(T[k-1])$. This completes the proof of the inner induction step, the outer induction step and the entire lemma. □

Lemma 4 (Algorithm 1 on unambiguous grammars)**.** *Assume G satisfies condition II in the definition of an unambiguous grammar, let w be an n-symbol input string. Then the assignment statement $T[i] = T[i] \cup \{A \to \pm BC\}$ in the inner loop is executed at most $|uconjuncts(G)| \cdot n^2$ times.*

Proof. Let us prove that for every j, for every conjunct $A \to \pm BC$ and for every i there exists at most one number k, such that iteration $(j, k, A \to \pm BC, i)$ of four nested loops is executed.

Suppose there exist two such numbers, k and k'. For the inner loop in lines 12–13 to be executed, both k and k' have to be in $E_j[C]$. Then, by Lemma 3(ii),

$$a_{k+1} \dots a_j \in L(C) \quad \text{and} \tag{11a}$$
$$a_{k'+1} \dots a_j \in L(C). \tag{11b}$$

Furthermore, for the corresponding iterations of the inner loop to be executed, i must be both in $E_k[B]$ and in $E_{k'}[B]$. By Lemma 3(i), this means the following:

$$a_{i+1} \dots a_k \in L(B), \tag{12a}$$
$$a_{i+1} \dots a_{k'} \in L(B). \tag{12b}$$

Combining (12a) with (11a) and (12b) with (11b), one obtains two factorizations of $a_{i+1} \dots a_j$ as $u \cdot v$, where $u \in L(B)$ and $v \in L(C)$. By the condition II from the definition of an unambiguous grammar, which holds by assumption, there is at most one such factorization. Therefore, the constructed factorizations are the same, that is, $k = k'$. □

Theorem 2. *For every Boolean grammar $G = (\Sigma, N, P, S)$ in binary normal form and for every input string $w \in \Sigma^*$, Algorithm 1 accepts if and only if $w \in L(G)$. Implemented on a random access machine, it terminates after $O(n^3)$ elementary steps, where $n = |w|$, or after $O(n^2)$ elementary steps, if the grammar is unambiguous.*

Proof. The correctness of the algorithm is given by Lemma 3(i): for $j = n$ and $A = S$, the final value of $E_j[A]$ is

$$E_n[S] = \{i \mid 0 \leqslant i < n \text{ and } a_{i+1} \dots a_n \in L(G)\},$$

and therefore $0 \in E_n[S]$ if and only if $a_1 \dots a_n \in L(G)$.

Next, let us note that each statement of the algorithm is executed in a constant number of elementary steps. Indeed, the only data of non-constant size are the lists $E_j[A]$, and the implementation notes in the end of Algorithm 1 cover each reference to these variables in the algorithm. Then the cubic time upper bound for the execution time is evident.

Note that these are lines 14–15 that are responsible for cubic time, and each of the rest of the statements is visited $O(n^2)$ times in any computation. Since, by Lemma 4, on any unambiguous grammar lines 14–15 are visited $O(n^2)$ times as well, this implies the algorithm's square-time performance on any unambiguous grammar. □

Corollary 1. *For every unambiguous Boolean grammar G there exists and can be effectively constructed an algorithm to test the membership of given strings in $L(G)$ in time $O(n^2)$.*

6 The family of unambiguous languages

Let us consider the language family generated by unambiguous Boolean grammars, which will be denoted $\mathsf{UnambBool}$. The family generated by unambiguous conjunctive grammars will be similarly denoted $\mathsf{UnambConj}$. As in the theory of context-free languages, let us say that a language is *inherently ambiguous* with respect to conjunctive (Boolean) grammars, if it is generated by some conjunctive (Boolean) grammar, but all conjunctive (Boolean) grammars generating it are ambiguous. The sets of inherently ambiguous languages are $\mathsf{UnambConj} \setminus \mathsf{Conj}$ and $\mathsf{UnambBool} \setminus \mathsf{Bool}$, respectively.

Concerning linear conjunctive languages, we shall now see each of them is unambiguous, so there is no third family to consider.

Theorem 3. *For every linear conjunctive grammar there exists and can be effectively constructed an equivalent unambiguous linear conjunctive grammar.*

Note that every linear conjunctive grammar satisfies condition II on uniqueness of factorization, because there is at most one nonterminal in every conjunct. So it remains to reconstruct the grammar so that different rules for any nonterminal generate disjoint languages.

The construction is based upon the representation of linear conjunctive languages by trellis automata [14]. *Trellis automata* [2, 9], also known as one-way real-time cellular automata, are defined as quadruples $(\Sigma, Q, I, \delta, F)$,

where Σ is an input alphabet, Q is a finite nonempty set of states, $I : \Sigma \to Q$ is the *initial function*, $\delta : Q \times Q \to Q$ is the *transition function*, and F is the set of *accepting states*. The computation on a string $a_1 \dots a_n$, where $n \geqslant 1$ and $a_i \in \Sigma$, is arranged as a triangle of states $\langle q_{ij} \rangle_{1 \leqslant i \leqslant j \leqslant n}$, where the bottom row is obtained from the symbols of the input string as $q_{ii} = I(a_i)$, while each of the rest of the states is computed from two of its predecessors as $q_{ij} = \delta(q_{i,j-1}, q_{i+1,j})$. The string is accepted if $q_{1n} \in F$.

Proof of Theorem 3. Construct a trellis automaton $M = (\Sigma, Q, I, \delta, F)$ generating $L \setminus \{\varepsilon\}$, where L is the language generated by the original grammar.

Let us use a known transformation of a trellis automaton to a linear conjunctive grammar [14]. A grammar $G = (\Sigma, \{A_q \mid q \in Q\} \cup \{S\}, P, S)$ is constructed, where P consists of the following rules:

$$S \to A_q \quad (\text{for all } q \in F) \tag{13a}$$
$$S \to \varepsilon \quad (\text{if } \varepsilon \in L) \tag{13b}$$
$$A_{I(a)} \to a \quad (\text{for all } a \in \Sigma) \tag{13c}$$
$$A_{\delta(q_1,q_2)} \to A_{q_1} c \& b A_{q_2} \quad (\text{for all } q_1, q_2 \in Q \text{ and } b, c \in \Sigma) \tag{13d}$$

For this grammar it is known [14, Lemma 2] that $L_G(A_q) = \{w | \Delta(I(w)) = q\}$, $L(G) \cap \Sigma^+ = L(M)$ and $L(G) = L$. It will now demonstrated that this grammar is unambiguous.

The condition II satisfied. Suppose condition I is not met for some string $w \in \Sigma^*$ and for some nonterminal A_q, that is, there exist two distinct rules,

$$A_q \to A_{q_1} c \& b A_{q_2} \quad \text{and} \tag{14a}$$
$$A_q \to A_{q_3} c' \& b' A_{q_4}, \tag{14b}$$

such that w belongs to each of the four languages $L_G(A_{q_1}c)$, $L_G(bA_{q_2})$, $L_G(A_{q_3}c')$, $L_G(b'A_{q_4})$. Note that this implies $b = b'$, $c = c'$ and $w = buc$, where $bu \in L_G(A_{q_1})$, $bu \in L_G(A_{q_3})$, $uc \in L_G(A_{q_2})$ and $uc \in L_G(A_{q_4})$. The latter, according to the correctness statement of the construction [14, Lemma 2], in turn implies $\Delta(I(bu)) = q_1$, $\Delta(I(bu)) = q_3$, $\Delta(I(uc)) = q_2$ and $\Delta(I(uc)) = q_4$. Therefore, $q_1 = q_3$ and $q_2 = q_4$, and the rules (14) coincide, which contradicts the assumption.

It remains to show that Condition I holds for the start symbol S. This is so, because any languages $L_G(A_q) = \{w \mid \Delta(I(w)) = q\}$ and $L_G(A_{q'}) = \{w \mid \Delta(I(w)) = q'\}$, where $q \neq q'$, are disjoint, and each of them is disjoint with $\{\varepsilon\}$. □

This result immediately provides us with many interesting examples of unambiguous languages.

Proposition 3. *The language $\{wcw \mid w \in \{a, b\}^*\}$ is a linear conjunctive language [13] and hence it is unambiguous.*

Proposition 4. *Let M be a Turing machine over an alphabet Σ, let Γ be an alphabet, let $C_M(w)$, where $w \in L(M)$, be an appropriate encoding of its accepting computation on w [18], let $\natural \notin \Sigma \cup \Gamma$. Then the language of computations of M,*

$$\mathrm{VALC}(M) = \{w \natural C_M(w) \mid w \in L(M)\},$$

is an intersection of two LL(1) linear context-free languages [18], hence it is linear conjunctive, and therefore unambiguous.

Recalling some known examples of P-complete linear conjunctive languages [9, 15], one can conclude that there exist unambiguous grammars for these languages.

Proposition 5. *There exists a P-complete unambiguous linear conjunctive language.*

Since the language $\mathrm{VALC}(M)$ is unambiguous, this implies some basic undecidability results for unambiguous linear conjunctive grammars, which carry on to unambiguous Boolean grammars.

Proposition 6. *The following problems are undecidable for unambiguous linear conjunctive grammars, unambiguous conjunctive grammars and unambiguous Boolean grammars: emptiness, universality, finiteness, regularity, equality, inclusion, etc.*

The existence of a P-complete language claimed in Proposition 5 indicates that efficient parallel parsing for unambiguous Boolean grammars is quite unlikely, because the non-existence of efficient parallel algorithms for P-complete languages is one of the common current assumptions of the complexity theory ($\mathsf{P} \neq \mathsf{NC}$).

Proposition 7. *Unless $\mathsf{P} = \mathsf{NC}$, there can be no polylogarithmic-time parallel parsing algorithm for unambiguous conjunctive (Boolean) grammars.*

Let us now find a place for the two new families of languages in the hierarchy of language families shown in the earlier Figure 1. The updated hierarchy is presented in Figure 2. The family *UnambConj* could be placed between *LinConj* and *Conj* by Theorem 3. The inclusion *UnambCF* $\subset$ *UnambConj* is proper, because there exist non-context-free linear conjunctive languages, such as $\{a^n b^n c^n \mid n \geqslant 0\}$ [13] or the language in Proposition 3. None of the inclusions *LinConj* $\subseteq$ *UnambConj* and *UnambConj* $\subseteq$ *Conj* is known to be proper, which is indicated in the figure by question marks upon the arrows; however, at least one of them must be proper, because *LinConj* $\subset$ *Conj* [14].

The four families *UnambConj*, *UnambBool*, *Conj* and *Bool* naturally form four inclusions among themselves, none of which is known to be proper. Of these inclusions, *Conj* $\subseteq$ *Bool* holds because conjunctive grammars are a

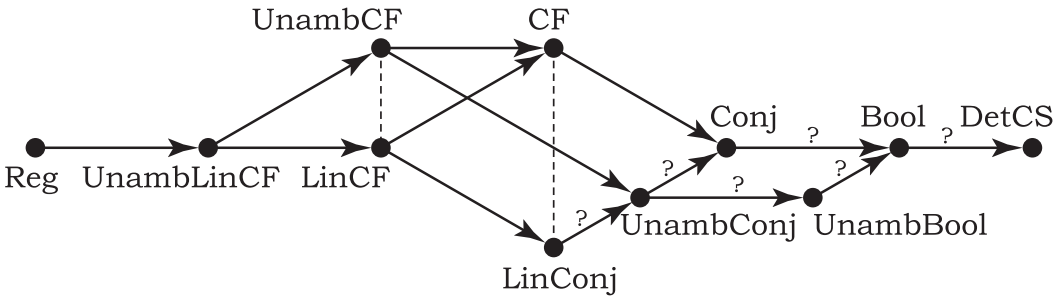

Figure 2: Unambiguous language families in the overall hierarchy.

particular case of Boolean grammars, but it remains unknown whether their expressive power is different [16]. For the inclusion in the case of unambiguous grammars, *UnambConj* $\subseteq$ *UnambBool*, is also unknown whether it is strict. The inclusions *UnambConj* $\subseteq$ *Conj* and *UnambBool* $\subseteq$ *Bool* are obvious, and the question of whether they are proper is exactly the problem of the existence of inherently ambiguous conjunctive (Boolean) languages. Even the inclusion *UnambConj* $\subseteq$ *Bool* is not known to be proper, so the possibility of all four families collapsing into one cannot be ruled out.

Though no proofs of inherent ambiguity of any languages have been found so far, there is a certain evidence that both inherently ambiguous conjunctive languages and inherently ambiguous Boolean languages do exist. Consider the opposite; then, by Corollary 1, the language generated by any conjunctive grammar could be parsed in time $O(n^2)$, which would be faster than the asymptotically best known general context-free parsing algorithms.

Consider some candidate languages for being inherently ambiguous. One of them is the language $\{a^{2^n} \mid n \geqslant 0\}$ given in Example 3. It was shown in Section 3 that the known Boolean grammar for this language is ambiguous. In fact, the grammar in Example 3 employs ambiguous concatenations as a principal device of expressing the language $L(A)$ through itself. It is conjectured that this kind of ambiguity cannot be avoided when expressing this language by a Boolean grammar.

Another candidate is the language $\{ww \mid w \in \{a,b\}^*\}$. The only known way of representing this language by a Boolean grammar essentially uses a context-free grammar for its complement and a negation on top of it. However, since $\overline{\{ww \mid w \in \{a,b\}^*\}}$ is most likely inherently ambiguous as a context-free language, any Boolean grammar constructed in this way also has to be ambiguous.

One more question made apparent by Figure 2 is whether the families *UnambCF* and *LinConj* are incomparable. It is known from Terrier [22] that context-free grammars and trellis automata have incomparable expressive power; hence, *CF* and *LinConj* are incomparable. However, the only known example of a context-free language not representable by trellis automata, due to Terrier [22], is inherently ambiguous.

Proposition 8. *The language L_T^2, where*

$$L_T = \{a^m b^m \mid m \geqslant 0\} \cup \{a^n bxab^n \mid n \geqslant 0,\ x \in \{a,b\}^*\},$$

which is a context-free language that is not linear conjunctive [22], is an inherently ambiguous context-free language.

Sketch of a proof. Consider the intersection

$$L_T^2 \cap a^+b^+a^+b^+a^+b^+ = \{a^i b^j a^k b^\ell a^m b^n | (i = j \text{ and } k = n) \text{ or } (i = \ell \text{ and } m = n)\}$$

The inherent ambiguity of this language follows by a straightforward modification of the well-known proof based upon Ogden's lemma [12] that the language $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$ is inherently ambiguous, see Harrison [8, Th. 7.2.2].

Suppose that this language is generated by an unambiguous context-free grammar. Let p_0 be the constant given by Ogden's lemma, let $p = p_0!$ and consider the string

$$a^{3p}b^{4p}a^{4p}b^{3p}a^{4p}b^{4p} = a^{3p}b^{4p}a^{4p}b^p\underbrace{b^p}b^p a^{4p}b^{4p} = xuyvz$$

with the specified distinguished positions. Standard case analysis shows that the only factorization satisfying the conditions of Ogden's lemma is of the form

$$\begin{aligned} x &= a^s, \\ u &= a^k, \\ y &= a^{3p-k-s}b^{4p}a^{4p}b^{p+t}, \\ v &= b^k, \\ z &= b^{2p-k-t}a^{4p}b^{4p}, \end{aligned}$$

for some $k \leqslant p_0$ and $s, t \geqslant 0$. The string is then pumped to xu^iyv^iz for any $i \geqslant 0$. Taking $i = p/k$ (which divides evenly), the string $a^{4p}b^{4p}a^{4p}b^{4p}a^{4p}b^{4p}$ is obtained. By a symmetric argument, $a^{4p}b^{4p}a^{3p}b^{4p}a^{4p}b^{3p}$ can be pumped to $a^{4p}b^{4p}a^{4p}b^{4p}a^{4p}b^{4p}$. Since the regions of pumping overlap, two different parse trees of $a^{4p}b^{4p}a^{4p}b^{4p}a^{4p}b^{4p}$ are constructed.

It follows that L_T^2 is inherently ambiguous, because unambiguous context-free languages are known to be closed under intersection with regular languages. □

Proposition 8 points out a certain gap in our knowledge on linear conjunctive grammars and trellis automata:

Remark 1. *It is not known whether there exists any unambiguous context-free language, which is not linear conjunctive (equivalently, "which cannot be recognized by a trellis automaton").*

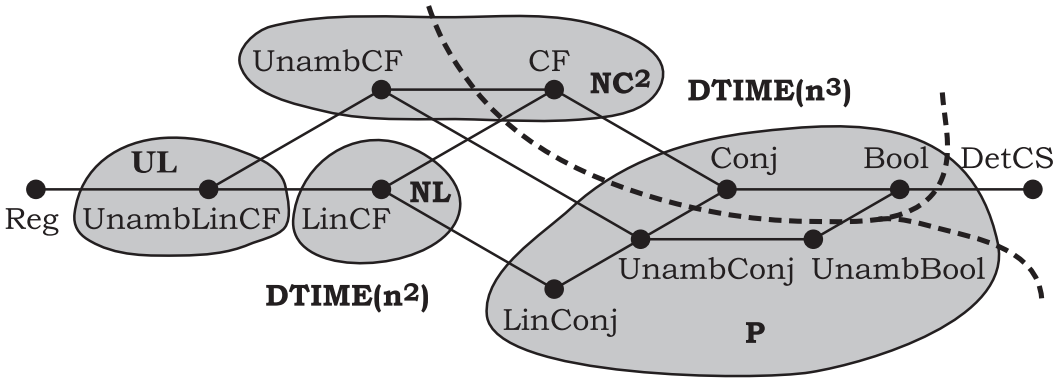

Figure 3: Complexity of languages.

Compare the complexity of the families of languages considered in this paper. According to the time complexity of recognition, all unambiguous classes of grammars are contained in deterministic square time, and no better bound is known even for unambiguous linear context-free grammars. Context-free grammars can be parsed as fast as matrices can be multiplied, which is $\mathit{DTIME}(n^{2.376})$, while practical general algorithms work in worst-case cubic time. Cubic time remains the best known theoretical upper bound for conjunctive and Boolean grammars. This partition is shown in Figure 3 by dotted lines.

In relation to the complexity-theoretic hierarchy, the families are separated into four classes, UL, NL, NC^2 and P. It is known that linear context-free grammars can be parsed in nondeterministic logarithmic space (NL), while their unambiguous subfamily can obviously be parsed in unambiguous logarithmic space (UL, see Álvarez and Jenner [1]). Parallel parsing algorithms for context-free grammars can be formalized by circuits of height $\log^2 n$, that is, context-free languages belong to NC^2. Finally, the languages generated by Boolean grammars are contained in P, and already linear conjunctive grammars can specify P-complete languages [9, 15].

Finally, consider the closure properties of the unambiguous classes. Some straightforward positive results can be given:

Proposition 9. *The family* UnambConj *is closed under intersection. The family* UnambBool *is closed under all Boolean operations.*

Other properties, such as whether unambiguous conjunctive languages are closed under union, complementation, concatenation and star, and whether the last two operations preserve unambiguous Boolean languages, are left as research problems.

7 Conclusion

The notion of ambiguity has been extended to Boolean grammars. The main practical properties of the class generalized well from the context-free case. On the other hand, the inherent ambiguity proofs did not generalize, and, accordingly, no closure properties besides the obvious could be established.

This picture is familiar, since everything related to Boolean grammars has been like that so far: attractive practical properties are found, while nontrivial theoretical problems have to be left open. Let us hope that yet another result on efficient parsing for Boolean grammars will attract further attention to this theoretical model, and the language-theoretic open questions shown in Figure 2 will be answered.

References

[1] C. Álvarez, B. Jenner, "A very hard log-space counting class", *Theoretical Computer Science*, 107:1 (1993), 3–30.

[2] K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata", I and II, *International Journal of Computer Mathematics*, 15 (1984), 195–212, and 16 (1984), 3–22.

[3] J. Earley, "An efficient context-free parsing algorithm", *Communications of the ACM*, 13:2 (1970), 94–102.

[4] R. W. Floyd, "On ambiguity in phrase structure languages", *Communications of the ACM*, 5:10 (1962), pp. 526, 534.

[5] S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.

[6] S. Ginsburg, J. S. Ullian, "Ambiguity in context free languages", *Journal of the ACM*, 13:1 (1966), 62–89.

[7] S. A. Greibach, "The undecidability of the ambiguity problem for minimal linear grammars", *Information and Control*, 6:2 (1963), 119–125.

[8] M. A. Harrison, *Introduction to formal language theory*, Addison-Wesley, 1978.

[9] O. H. Ibarra, S. M. Kim, "Characterizations and computational complexity of systolic trellis automata", *Theoretical Computer Science*, 29 (1984), 123–153.

[10] T. Kasami, K. Torii, "A syntax-analysis procedure for unambiguous context-free grammars", *Journal of the ACM*, 16:3 (1969), 423–431.

[11] V. Kountouriotis, Ch. Nomikos, P. Rondogiannis, "Well-founded semantics for Boolean grammars", *Developments in Language Theory* (DLT 2006, Santa Barbara, USA, June 26–29, 2006), LNCS 4036, 203–214.

[12] W. F. Ogden, "A helpful result for proving inherent ambiguity", *Mathematical Systems Theory*, 2:3 (1968), 191–194.

[13] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

[14] A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *RAIRO Informatique Théorique et Applications*, 38:1 (2004), 69–88.

[15] A. Okhotin, "The hardest linear conjunctive language", *Information Processing Letters*, 86:5 (2003), 247–253.

[16] A. Okhotin, "Boolean grammars", *Information and Computation*, 194:1 (2004), 19–48.

[17] A. Okhotin, "On the existence of a Boolean grammar for a simple programming language", *Proceedings of AFL 2005* (May 17–20, 2005, Dobogókő, Hungary).

[18] A. Okhotin, "Language equations with symmetric difference", *Computer Science in Russia* (CSR 2006, St. Petersburg, Russia, June 8–12, 2006), LNCS 3967, 292–303.

[19] A. Okhotin, "Generalized LR parsing algorithm for Boolean grammars", *International Journal of Foundations of Computer Science*, 17:3 (2006), 629–664.

[20] P. Rossmanith, W. Rytter, "Observation on $\log(n)$ time parallel recognition of unambiguous cfl's", *Information Processing Letters*, 44:5 (1992), 267–272.

[21] W. Rytter, "Parallel time $O(\log n)$ recognition of unambiguous context-free languages", *Information and Computation*, 73:1 (1987), 75–86.

[22] V. Terrier, "On real-time one-way cellular array", *Theoretical Computer Science*, 141 (1995), 331–335.

[23] K. Wich, "Sublogarithmic ambiguity", *Theoretical Computer Science*, 345:2–3 (2005), 473–504.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978-952-12-1846-0
ISSN 1239-1891