Information and Computation 206 (2008)1234–1247

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Unambiguous Boolean grammars☆

Alexander Okhotin

Academy of Finland
Department of Mathematics, University of Turku, Turku FIN-20014, Finland

ARTICLE INFO

Article history:
Received 2 July 2007
Revised 19 February 2008
Available online 8 June 2008

Keywords:
Boolean grammars
Conjunctive grammars
Ambiguity
Parsing

ABSTRACT

Boolean grammars are an extension of context-free grammars, in which conjunction and negation may 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.

1. Introduction

Unambiguous context-free grammars are those that define a unique parse tree for every string they generate. In other words, 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. In particular, 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 [26].

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 [24], and later improved by Rossmanith and Rytter [23]. An adaptation of the well-known Cocke–Kasami–Younger algorithm for unambiguous grammars developed by Kasami and Torii [12] 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* [17], which are an extension of context-free grammars with a complete basis of propositional connectives. Besides giving a greater freedom of grammar construction, Boolean grammars are capable of specifying many key examples of non-context-free languages [17], as well as a simple model programming language [18]. 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 [17,19]. Recursive descent parsing is known to have a generalization for Boolean grammars [20,21].

An important subclass of Boolean grammars are *conjunctive grammars* [14], which, in addition to the disjunction implicitly present in the context-free grammars, may also contain an explicit conjunction operation, but not the negation found in

☆ Supported by Academy of Finland under Grant 118540.
E-mail address: alexander.okhotin@utu.fi

0890-5401/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2008.03.023

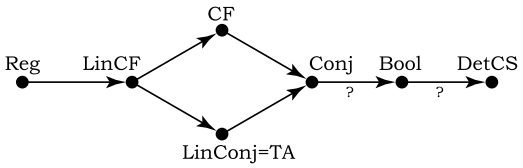

Fig. 1. The hierarchy of language families.

Boolean grammars. For parsing algorithms, conjunctive grammars are a simple case of Boolean grammars; however, their expressive power is still substantial [14], and no proofs of separation of conjunctive grammars from Boolean grammars are known. An important contribution to the study of conjunctive grammars has recently been made by Jeż [10], who constructed a conjunctive grammar over a unary alphabet generating the nonregular language $\{a^{4^n} \mid n \geqslant 0\}$, thus contradicting the earlier intuition that such grammars should generate only regular languages.

The useful practical properties of conjunctive and Boolean grammars give a reasonable hope that these formal models can be useful for applications, and it is worthwhile to investigate their unambiguous subclasses. This paper starts with a definition of ambiguity in Boolean grammars and establishes their basic theory. 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 proved that if the given grammar is unambiguous, then the resulting grammar in the normal form will be unambiguous as well. The resulting normal form theorem for unambiguous Boolean grammars is then used to establish 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: it is applicable to any Boolean grammar, works in time $O(n^3)$ in the general case, and in time $O(n^2)$ on any unambiguous grammar.

Finally, the family of languages generated by unambiguous Boolean grammars is considered in Section 6 and compared to the related language families. Some candidates for being inherently ambiguous languages are presented.

2. Boolean grammars

Definition 1 ([17]). 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 $A \in N$, $m + n \geqslant 1$ and $\alpha_i, \beta_i \in (\Sigma \cup N)^*$; $S \in N$ 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 an 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* [14], 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 $A \to w$, with $u,v,w \in \Sigma^*$ and $B \in N$. 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 [2,16].

The relation between the families of languages generated by Boolean grammars, conjunctive grammars and linear conjunctive grammars, as well as other common families of formal languages, is shown in Fig. 1 [17], in which the three new families are denoted *Bool*, *Conj* and *LinConj*, respectively. The rest of the classes in the figure are regular (*Reg*), linear context-free (*LinCF*), context-free (*CF*) and deterministic context-sensitive languages (*DetCS*).

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 Boolean grammars generate the languages $\{a^n b^n c^n \mid n \geqslant 0\}$ and $\{a^m b^n c^n \mid m,n \geqslant 0, m \neq n\}$, respectively:

$S \to AB\&DC$ | $S \to AB\&\neg DC$
$A \to aA \mid \varepsilon$ | $A \to aA \mid \varepsilon$
$B \to bBc \mid \varepsilon$ | $B \to bBc \mid \varepsilon$
$C \to cC \mid \varepsilon$ | $C \to cC \mid \varepsilon$
$D \to aDb \mid \varepsilon$ | $D \to aDb \mid \varepsilon$

The first grammar, which is actually conjunctive, represents its language as an intersection of two context-free languages:

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

The second grammar is obtained by inverting the sign of one of the conjuncts in the rule for S. It generates the following language:

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

Example 2. The following Boolean grammar generates the language $\{ww \mid w \in \{a,b\}^*\}$:

$S \to \neg AB\&\neg BA\&C$
$A \to XAX \mid a$
$B \to XBX \mid b$
$C \to XXC \mid \varepsilon$
$X \to a \mid b$

According to the intuitive semantics of Boolean grammars described above, the nonterminals A, B and C generate context-free languages

$$L(A) = \{uav \mid u,v \in \{a,b\}^*, |u| = |v|\},$$
$$L(B) = \{ubv \mid u,v \in \{a,b\}^*, |u| = |v|\},$$
$$L(C) = \{aa,ab,ba,bb\}^*.$$

Then

$$L(AB) = \{uavxby \mid u,v,x,y \in \{a,b\}^*, |u| = |x|, |v| = |y|\},$$

in other words, $L(AB)$ is the set of all strings of even length with a mismatch a on the left and b on the right (in any position). Similarly,

$$L(BA) = \{ubvxay \mid u,v,x,y \in \{a,b\}^*, |u| = |x|, |v| = |y|\}$$

specifies the mismatch formed by b on the left and a on the right. Then the rule for S specifies the set of strings of even length without such mismatches:

$$L(S) = \overline{L(AB)} \cap \overline{L(BA)} \cap \{aa,ab,ba,bb\}^* = \{ww \mid w \in \{a,b\}^*\}.$$

Let us now give a formal definition of the language generated by a Boolean grammar. Actually, this definition can be given in several different ways [13,17], which ultimately yield the same class of languages; these details of formal definition are beyond the scope of this paper. This paper uses the most straightforward of these definitions, though it is worth mention that the results are applicable to other semantics as well. This simplest definition begins with 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 idea is to use a solution of this system to define the languages generated by nonterminals of a grammar. However, a system of equations of this form may have no solutions or multiple solutions. Even in the case when it has a unique solution,

it is known that its components may be arbitrary recursive sets [17]. Such results are far from the intuitive meaning of a Boolean grammar. The simplest formal definition corresponding to this intuitive meaning is by restricting these systems as follows:

Definition 3. Let $G = (\Sigma,N,P,S)$ be a Boolean grammar, let (2) be the associated system of language equations. Assume that for every finite language $M \subset \Sigma^*$ (in which 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 $(\ldots,L_C,\ldots)_{C\in N}$ be a unique solution of this system. Then, for every $A \in N$, the language $L_G(A)$ is defined as L_A, while the language generated by the grammar is $L(G) = L_G(S) = L_S$.

If this condition is failed, a grammar is considered ill-formed. In practice this happens only for artificially constructed grammars, such as $S \to S$ or $S \to \neg S$.

The full strength of this definition is used in the proof of the normal form theorem for Boolean grammars [17]. In most other cases, such as in this paper, the conditions on the system are assumed to hold, and then the equation (2) may be used as an explicit expression for $L_G(A)$.

A useful property of Boolean grammars is that they define *parse trees* of the strings they generate [17], which represent parses of a string according to positive conjuncts in the rules. 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 ith 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 $|\alpha_1| + \cdots + |\alpha_m|$ direct descendants corresponding to the symbols in positive conjuncts. For each nonterminal in each α_k, 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 kth positive conjunct of this rule, let $\alpha_k = s_1 \ldots s_\ell$. There exist numbers $i_1,\ldots,i_{\ell-1}$, with $i = i_0 \leqslant i_1 \leqslant \ldots \leqslant i_{\ell-1} \leqslant i_\ell = j$, such that each descendant corresponding to each s_t encompasses the substring $a_{i_{t-1}+1} \ldots a_{i_t}$.
(3) For each kth negative conjunct of this rule, $a_{i+1} \ldots a_j \notin L_G(\beta_k)$.

The root is the unique vertex 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.

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 instance, parse trees of the second grammar from Example 1 reflect only the conjunct $S \to AB$, and thus are plain context-free trees. On the other hand, parse trees corresponding to any conjunctive grammar, such as the first grammar in Example 1, reflect full information about the membership of a string in the language.

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 $A \to s_1 \ldots s_\ell$ with $w \in L(s_1 \ldots s_\ell)$, and a unique factorization $w = u_1 \ldots u_\ell$ with $u_i \in L(s_i)$.

Assuming that $L(A) \neq \varnothing$ for every nonterminal A, these definitions are equivalent. In the case of Boolean grammars, the first definition becomes useless, because negative conjuncts are not accounted for in a parse tree. The requirement of parse tree uniqueness can be trivially satisfied as follows. Given any grammar G over an alphabet $\Sigma = \{a_1,\ldots,a_m\}$ 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:

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

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.

Trying to generalize the second approach for Boolean grammars in the least restrictive way, one may produce the following definition:

- for every nonterminal A and for every string $w \in L(A)$ there exists a unique rule

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \tag{3}$$

with $w \in L_G(\alpha_t)$ and $w \notin L_G(\beta_t)$ for all t, such that for every positive conjunct $\alpha_t = s_1 \ldots s_\ell$ there exists a unique factorization $w = u_1 \ldots u_\ell$ with $u_i \in L(s_i)$.

However, this definition can be trivialized similarly to the previous case. Given a Boolean grammar G, replace every rule (3) with

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

where every new nonterminal C_α has a unique rule $C_\alpha \to \neg\alpha$. The resulting grammar generates the same language and contains only negative conjuncts, and so the condition on factorizations in positive conjuncts is trivially satisfied (while the choice of a rule can be made unique as well using some additional transformations).

Therefore, a proper definition of ambiguity for Boolean grammars must take into account factorizations of strings according to negative conjuncts. The following definition is obtained.

Definition 4. A Boolean grammar $G = (\Sigma, N, P, S)$ is 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

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

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

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 $w = u_1 \ldots u_\ell$ with $u_i \in L_G(s_i)$ for all i.

Note that Condition II applies to positive and negative conjuncts alike. In the case of a positive conjunct belonging to some rule, this means that a string that is potentially generated by this rule must be uniquely factorized according to this conjunct. For a negative conjunct $A \to \neg DE$, Condition II requests that a factorization of $w \in L_G(DE)$ into $L_G(D) \cdot L_G(E)$ is unique even though w is *not generated* by any rule involving this conjunct. As argued above, this condition cannot be relaxed.

Consider some examples. Both grammars in Example 1 are unambiguous. To see that Condition II is satisfied with respect to the conjunct $S \to AB$, consider that a factorization $w = uv$, with $u \in L(A)$ 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 2 is ambiguous because Condition II does not hold. Consider the string $w = aabb$ and the conjunct $S \to \neg AB$. This string has two factorizations $w = a \cdot abb = aab \cdot b$, with $a \in L(A)$, $abb \in L(B)$, $aab \in L(A)$ and $b \in L(B)$. This, by definition, means that the grammar is ambiguous. It is not known whether there exists an unambiguous Boolean grammar generating the same language.

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 vertices 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. Assume that every nonterminal A has either a unique rule (1) of an arbitrary form, or multiple rules each containing a single positive conjunct:

$$A \to \alpha_1 \mid \ldots \mid \alpha_n \quad (\text{where } \alpha_i \in (\Sigma \cup N)^*) \tag{4}$$

There is no loss of generality in this assumption, because any multiple-conjunct rule for A can be replaced with a rule of the form $A \to A'$, where A' is a new nonterminal with a unique rule replicating the original rule for A. Then, for every nonterminal with multiple rules of the form (4), these rules can be replaced with the following n rules, which clearly generate disjoint languages:

$$\begin{aligned} &A \to \alpha_1 \\ &A \to \alpha_2 \& \neg\alpha_1 \\ &A \to \alpha_3 \& \neg\alpha_1 \& \neg\alpha_2 \\ &\quad\vdots \\ &A \to \alpha_n \& \neg\alpha_1 \& \neg\alpha_2 \& \ldots \& \neg\alpha_{n-1} \end{aligned} \tag{4'}$$

The grammar obtained by this transformation 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 Condition I is satisfied. If the original grammar satisfies 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:

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

Condition II is satisfied for the same reasons as in Example 1. On the other hand, 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:

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

So it has been demonstrated 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

Consider the following normal form for Boolean grammars that generalizes Chomsky normal form for context-free grammars:

Definition 5 (*[17]*)**.** A Boolean grammar is said to be in the binary normal form [17], if all of its rules are of the form

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

In particular, every grammar in this normal form satisfies Definition 3. It is known that every Boolean grammar can be effectively transformed to an equivalent grammar in the binary normal form. 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 by

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

the set of all strings obtained from $s_1\ldots s_\ell$ by removing some of the symbols generating ε. 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, for 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

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

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 $A\in N$, $L_{G_1}(A)=L_G(A)\setminus\{\varepsilon\}$ [17].

At the second step, another Boolean grammar $G_2=(\Sigma,N,P_2,S)$ with $L(G_2)=L(G_1)$ 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 $R=\{\gamma \mid A\to\pm\gamma\in \mathit{uconjuncts}(P_1),\ \gamma\notin N\}=\{\eta_1,\ldots,\eta_\ell\}$. Then every rule in P_2 is of the general form

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

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 $A \to \pm s\alpha$, with $s \in \Sigma \cup N$ 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 sX_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 proof of the correctness of this transformation can be found in the cited paper [17]. The new contribution of this paper is that it preserves unambiguity.

Lemma 1. *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar compliant to Definition 3. Assume $L_G(A) \neq \varnothing$ for every $A \in N$. 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 (regardless of whether G satisfies this condition).*
- *If G satisfies Condition II, then each grammar obtained satisfies Condition II.*

In particular, if G satisfies Condition II, then the normal form grammar G_4 is unambiguous.

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

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

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 $A \to \pm s_1 \ldots s_\ell \in \mathit{uconjuncts}(P)$, the set $\mathit{uconjuncts}(P_1)$ contains every $A \to \pm s_{i_1} \ldots s_{i_k}$, with $k \geqslant 1$, $1 \leqslant i_1 < \ldots < i_k \leqslant \ell$ and $\varepsilon \in L_G(s_j)$ for every j in $\{1, \ldots, \ell\} \setminus \{i_1, \ldots, i_k\}$. 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 $L_{G_1}(s_{i_1}) \cdot \ldots \cdot L_{G_1}(s_{i_k})$:

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

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 $j \in \{1, \ldots, \ell\} \setminus \{i_1, \ldots, i_k\}$. Then $u_1 \ldots u_\ell = v_1 \ldots v_\ell = w$, and since G satisfies Condition II, $u_j = v_j$ for all $j \in \{1, \ldots, \ell\}$. 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 $A \to \pm s_1 s_2 \ldots s_\ell$, 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 $A' \to s_2 \ldots s_\ell$.

Suppose some string $w \in \Sigma^*$ can be represented as $w = u_2 \ldots u_\ell = v_2 \ldots v_\ell$, where $u_j, v_j \in L_{\widetilde{G}}(s_j) = L_{\widehat{G}}(s_j)$ for $j = 2, \ldots, \ell$. 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 $xw = u_1 u_2 \ldots u_\ell = v_1 v_2 \ldots v_\ell$, where $u_j, v_j \in L_{\widehat{G}}(s_j)$ for $j = 1, \ldots, \ell$. By assumption, $\widehat{G}$ satisfies Condition II, hence $u_j = v_j$ for all j, and thus 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 $L_{\widetilde{G}}(A') = L_{\widehat{G}}(s_2 \ldots s_\ell)$, there exist factorizations $u' = u_2 \ldots u_\ell$ and $v' = v_2 \ldots v_\ell$, where $u_j, v_j \in L_{\widehat{G}}(s_j)$ for $j = 2, \ldots, \ell$. Thus two factorizations of w are obtained: $w = u_1 u_2 \ldots u_\ell = v_1 v_2 \ldots v_\ell$, with $u_j, v_j \in L_{\widehat{G}}(s_j)$ for $j = 1, \ldots, \ell$. 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. □

Together with the correctness statement for the given transformation [17, Th. 8], the above Lemma 1 implies the following result:

Theorem 1. *For every unambiguous Boolean grammar there exists an unambiguous Boolean grammar in the binary normal form generating the same language.*

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 [3,12]. Similarly, log-square-time parallel algorithms can be sped up to logarithmic time [24].

While logarithmic-time parallel parsing seems to have no analogues for Boolean grammars, the idea behind the square-time algorithms of Earley [3] and Kasami and Torii [12] 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. Meeting this requirement requires refactoring data structures and loops of existing algorithms [12,17]. A new algorithm based upon these ideas will be constructed from scratch in this section.

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 the binary normal form. For every $X \subseteq \mathit{uconjuncts}(G)$, define

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

Let $w = a_1 \dots a_n$, where $n \geqslant 1$ and $a_i \in \Sigma$, be an input string. For each $j \in \{1, \dots, n\}$, 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 $\mathit{uconjuncts}(P)$.

```
1:  let E_j[A] = ∅ for all j = 1,...,n and A ∈ N
2:  for j = 1 to n do
3:    for all 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 if and only if 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 purpose of each jth 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 jth 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 jth position are processed in the second nested loop by k (line 9).

This loop constructs an auxiliary data structure T: for each $i \in \{0, \dots, j-2\}$, $T[i]$ is meant to contain all conjuncts $A \to \pm BC$, for which the substring starting from the position $i+1$ and ending at the position j is in $L_G(BC)$. Every kth iteration of this loop, denoted (j,k), considers substrings of various length starting at any position $i+1 \in \{1,2, \dots, k\}$ 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 $T[k-1]$,$T[k-2]$,...,$T[0]$, 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 be established: 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 $E_j[A]$ with $A \in N$ 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 $T[i]$ with $0 \leqslant i < j$ 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 values of t.

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

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

under the assumption that (i) holds for $t \in \{1,\ldots,j-1\}$. First, 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):

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

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 $\{A \to \pm BC \mid \exists \ell\ (j \leqslant \ell < j) : \langle\ldots\rangle\} = \varnothing$, which is consistent with line 8.

Induction step $k+1 \to k$ ($j > k \geqslant 1$): Assume iterations $(j,j-1)$,$(j,j-2)$,...,$(j,k+2)$,$(j,k+1)$, 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,

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

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 $a_{k+1} \dots a_j \in L(C)$. It has thus been proved that when the iteration (j,k) proceeds with the second inner loop starting in line 14, 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 and 15 set $E_j[A]$ to (7), 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

$$A \to B_1 C_1 \& \dots \& B_\ell C_\ell \& \neg D_1 E_1 \& \dots \& \neg D_m E_m \& \neg \varepsilon,$$

such that $a_k \dots a_j \in L_G(B_i C_i)$ and $a_k \dots a_j \notin L_G(D_t E_t)$ for all appropriate i and t. By (10), this is equivalent to $A \to \pm B_i C_i \in T[k-1]$ and $A \to \pm D_t E_t \notin T[k-1]$ 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.

Getting back to the outer induction, claim (ii) for $k = 1$ asserts that after all iterations of the loop in lines 9–15 are complete, $E_j[A]$ is exactly as in (6), which proves 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 and 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 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 machine instructions. 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 and 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 and 15 are visited $O(n^2)$ times as well, this implies the algorithm's square-time performance on any unambiguous grammar. □

The algorithm relies upon the normal form, but Theorem 1 shows that there is no loss of generality in this assumption. Hence the following result has been established:

Theorem 3. *For every unambiguous Boolean grammar G there exists 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 *UnambBool*. The family generated by unambiguous conjunctive grammars will be similarly denoted *UnambConj*. These languages will be called *unambiguous* with respect to conjunctive (Boolean) grammars. As in the theory of the 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.

Concerning linear conjunctive languages, it is not hard to prove that each of them is unambiguous.

Theorem 4. *For every linear conjunctive grammar there exists and can be effectively constructed an equivalent unambiguous linear conjunctive grammar.*

To establish this theorem, it is convenient to use an automaton model equivalent to linear conjunctive grammars [16]. These are *trellis automata* [2,9], also known as one-way real-time cellular automata, defined as quadruples (Σ,Q,I,δ,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 states with $i = j$ are 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 result of the computation is the state q_{1n}, denoted by $\Delta(I(a_1 \dots a_n))$. The input string is accepted if $q_{1n} \in F$.

It is known that a language $L \subseteq \Sigma^+$ is generated by a linear conjunctive grammar if and only if it is recognized by a trellis automaton. This gives the following easy proof of the theorem:

Proof (*Proof of Theorem 4*). 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 [16]. A grammar $G = (\Sigma, \{A_q \mid q \in Q\} \cup \{S\},P,S)$ is constructed, where P consists of the following rules:

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

For this grammar it is known [16, Lemma 2] that $L_G(A_q) = \{w \mid \Delta(I(w)) = q\}$, $L(G) \cap \Sigma^+ = L(M)$ and $L(G) = L$. Note that the nonterminals $\{A_q\}_{q\in Q}$ generate pairwise disjoint languages, because every string w belongs only to $L_G(A_q)$ with $q = \Delta(I(w))$.

It will now be demonstrated that this grammar is unambiguous. Condition II is satisfied because concatenation is linear. 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,

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

such that w belongs to each of the four languages $L_G(A_{q_1}c), L_G(bA_{q_2}), L_G(A_{q_3}c'), L_G(b'A_{q_4})$. Then, clearly, $b = b'$, $c = c'$ 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 $uc \in L_G(A_{q_4})$. Since different nonterminals generate disjoint languages, it follows that $A_{q_1} = A_{q_3}$ and $A_{q_2} = A_{q_4}$, and the rules (14) coincide, which contradicts the assumption.

Condition I holds for the start symbol S because the languages $L_G(A_q)$ and $L_G(A_{q'})$ with $q \neq q'$ are disjoint, and each of them is disjoint with $\{\varepsilon\}$. □

This result immediately leads to many interesting examples of unambiguous languages.

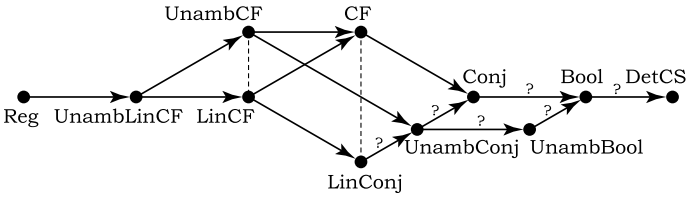

Fig. 2. Unambiguous language families in the overall hierarchy.

Proposition 3. *The language* $\{wcw \mid w \in \{a,b\}^*\}$ *is a linear conjunctive language [14] 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, 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 a linear conjunctive language, and hence it is unambiguous.

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

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

Since the language 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.*

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 ($P \neq NC$).

Proposition 7. *Unless* $P = NC$, *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 Fig. 1. The updated hierarchy is presented in Fig. 2. The family *UnambConj* could be placed between *LinConj* and *Conj* by Theorem 4. The inclusion *UnambCF* $\subset$ *UnambConj* is proper, because there exist non-context-free linear conjunctive languages, such as $\{a^n b^n c^n \mid n \geqslant 0\}$ [14] 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* [16].

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 particular case of Boolean grammars, but it remains unknown whether their expressive power is different [17]. 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 Theorem 3, any language generated by a conjunctive grammar could be parsed in time $O(n^2)$, which would be faster than the asymptotically best known general context-free parsing algorithms.

A candidate language for being inherently ambiguous is the language $\{ww \mid w \in \{a,b\}^*\}$, for which an ambiguous grammar has been given in Example 2. This language is known to be non-context-free, but its complement is context-free, and the only known way of representing this language by a Boolean grammar essentially uses a context-free grammar for $\overline{L_{ww}}$ and a negation on top of it. However, since $\overline{L_{ww}}$ is most likely inherently ambiguous as a context-free language, any Boolean grammar constructed in this way must also be ambiguous. Unfortunately, no proofs of inherent ambiguity have so far been obtained; determining whether *UnambBool* = *Bool* and whether *UnambConj* = *Conj* are the main open problems on unambiguous variants of conjunctive and Boolean grammars.

One could conjecture that unambiguous Boolean grammars over a unary alphabet generate only regular languages. Until recently, all known Boolean grammars for unary nonregular languages essentially used both negation and ambiguity. However, the recent results by Jeż [10] completely disprove this intuition by showing that conjunctive grammars not only can generate nonregular unary languages, but can do so without using ambiguity [11].

One more question made apparent by Fig. 2 is whether the families *UnambCF* and *LinConj* are incomparable. It is known from Terrier [25] that context-free grammars and trellis automata have incomparable expressive power; that is, *CF* and *LinConj* are incomparable. However, the only known example of a context-free language not representable by trellis automata, due to Terrier [25], 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 [25], is an inherently ambiguous context-free language.

Proof (*Sketch of a proof*)**.** Consider the intersection

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

The inherent ambiguity of this language follows by a straightforward modification of the well-known proof based upon Ogden's lemma [22] 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

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

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 $i = \frac{p}{k} + 1$ (where p divides by k 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 that is not linear conjunctive (equivalently, "that cannot be recognized by a trellis automaton").

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 $DTIME(n^{2.376})$, 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 Fig. 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 languages 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].

At last, 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.*

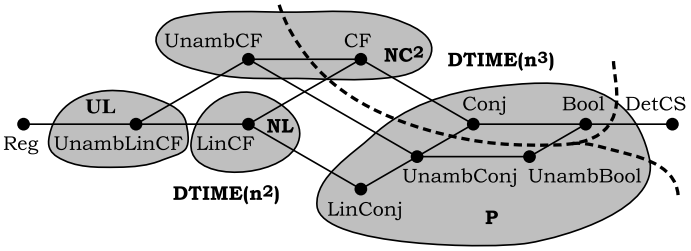

Fig. 3. Complexity of languages.

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 open. Unfortunately, in the absence of proof techniques for inherent ambiguity, no closure properties besides the obvious could be established.

7. Conclusion

A useful subfamily of Boolean grammars with an improved worst-case parsing time of $O(n^2)$ has been introduced, which becomes one more step towards practical applicability of these theoretically attractive grammars. The next step could be to find a natural subfamily of unambiguous Boolean grammars that could be parsed in time $o(n^2)$, hopefully $O(n)$. Such a subfamily, if it exists, would deserve a name of *deterministic Boolean grammars*, and would bring Boolean grammars one step closer to use in applications.

Meanwhile, many theoretical questions on unambiguous Boolean grammars remain open and their study could yield important results. This paper raises a new question of the existence of inherently ambiguous languages with respect to Boolean grammars. Inventing new proof techniques for inherent ambiguity seems to be crucial for the further study of this class, and is proposed as a question for research.

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–II, International Journal of Computer Mathematics, 15 (1984), 195–212; 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) 526.
[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] A. Jeż, Conjunctive grammars can generate non-regular unary languages, International Journal of Foundations of Computer Science 19 (3) (2008) 597–615.
[11] A. Jeż, Personal communication, April 2007.
[12] T. Kasami, K. Torii, A syntax-analysis procedure for unambiguous context-free grammars, Journal of the ACM 16 (3) (1969) 423–431.
[13] 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.
[14] A. Okhotin, Conjunctive grammars, Journal of Automata, Languages and Combinatorics 6 (4) (2001) 519–535.
[15] A. Okhotin, The hardest linear conjunctive language, Information Processing Letters 86 (5) (2003) 247–253.
[16] A. Okhotin, On the equivalence of linear conjunctive grammars to trellis automata, RAIRO Informatique Théorique et Applications 38 (1) (2004) 69–88.
[17] A. Okhotin, Boolean grammars, Information and Computation 194 (1) (2004) 19–48.
[18] A. Okhotin, On the existence of a Boolean grammar for a simple programming language, in: Proceedings of AFL 2005, May 17–20, 2005, Dobogókő, Hungary.
[19] A. Okhotin, Generalized LR parsing algorithm for Boolean grammars, International Journal of Foundations of Computer Science 17 (3) (2006) 629–664.
[20] A. Okhotin, Recursive descent parsing for Boolean grammars, Acta Informatica 44 (3-4) (2007) 167–189.
[21] A. Okhotin, Expressive power of LL(k) Boolean grammars, Fundamentals of Computation Theory (FCT 2007, Budapest, Hungary, August 27–30, 2007), LNCS 4639, 446–457.
[22] W.F. Ogden, A helpful result for proving inherent ambiguity, Mathematical Systems Theory 2 (3) (1968) 191–194.
[23] P. Rossmanith, W. Rytter, Observation on log(n) time parallel recognition of unambiguous cfl's, Information Processing Letters 44 (5) (1992) 267–272.
[24] W. Rytter, Parallel time $O(\log n)$ recognition of unambiguous context-free languages, Information and Computation 73 (1) (1987) 75–86.
[25] V. Terrier, On real-time one-way cellular array, Theoretical Computer Science 141 (1995) 331–335.
[26] K. Wich, Sublogarithmic ambiguity, Theoretical Computer Science 345 (2–3) (2005) 473–504.