Unambiguous Boolean grammars

Alexander Okhotin*

[1] Academy of Finland
[2] Department of Mathematics, University of Turku, Turku FIN–20014, Finland
alexander.okhotin@utu.fi

Abstract. Boolean grammars are an extension of context-free grammars, in which all propositional connectives are allowed. 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, 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 grammar is ambiguous was first proved by Floyd [3], while Greibach [6] extended this result to one-nonterminal linear context-free grammars. Some properties of unambiguous languages were determined by Ginsburg and Ullian [5]. In the later years a sophisticated theory was developed around the notion of ambiguity, and recent results of Wich [18] on the degree of ambiguity are worth particular attention.

As compared to context-free grammars of the general form, unambiguous context-free grammars are notable for lower parsing complexity. A logarithmic-time parallel algorithm was proposed by Rytter [16]. An adaptation of the well-known Cocke–Kasami–Younger algorithm for unambiguous grammars developed by Kasami and Torii [8] works in square time. 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* [13], 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 [13], as well as a simple model programming language [14]. On the other hand,

* Supported by the Academy of Finland under grant 118540.

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 [13,15].

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.

2 Boolean grammars

Definition 1 ([13]). *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.

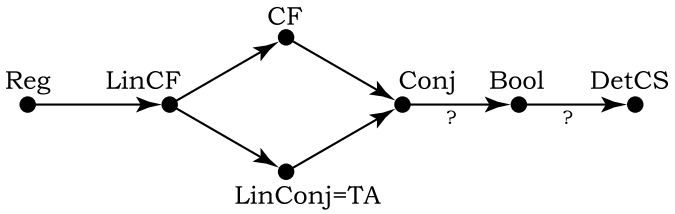

Fig. 1. The hierarchy of language families.

A Boolean grammar is called a *conjunctive grammar* [10], 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 one-way real-time cellular automata [12]. These three language families are denoted *Bool*, *Conj* and *LinConj* in the hierarchy in Figure 1 [13], where the rest of the classes are regular (*Reg*), linear context-free (*LinCF*), context-free (*CF*) and deterministic context-sentitive 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 grammar generates the language $\{a^n b^n c^n \mid n \geqslant 0\}$:

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

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 $\{a^m b^n c^n \mid m, n \geqslant 0, m \neq n\}$:

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

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 give a formal definition of the language generated by a Boolean grammar. Actually, this definition can be given in several different ways [9,13], which ultimately yield the same class of languages. For simplicity, we shall use the most straightforward of these definitions, but it can be mentioned that the results of this paper are applicable to other semantics as well. This simplest definition departs 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.

Definition 3. *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.*

Then, for every $A \in N$, the language $L_G(A)$ is defined as L_A, while the language generated by the grammar is $L(G) = L_G(S)$.

A useful property of Boolean grammars is that they define *parse trees* of the strings they generate [13], which represent parses of a string according to positive conjuncts in the rules.

See Appendix A.

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

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 $\{a_1, \ldots, a_m\}$ 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:

$$\begin{aligned} S' &\to A \& \neg \widehat{S} \\ \widehat{S} &\to A \& \neg S \\ A &\to a_1 A \mid \ \ldots \ \mid a_m A \mid \varepsilon \end{aligned}$$

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 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

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

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

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 conjuncts $S \to DC$ and $S \to \neg DC$ are treated similarly. Different rules for each of A, B, C, D clearly generate disjoint languages.

On the other hand, here is an example of an ambiguous grammar, and it is not known whether there exists an unambiguous grammar for the same language:

Example 3 (cf. [13, Example 4]). The language $\{a^{2^n} \mid n \geqslant 0\}$ is generated by the following ambiguous Boolean grammar:

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

To see that the grammar is ambiguous, 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 $\varepsilon, a \in L(B)$, and thus condition II is violated.

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

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

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:

$$\begin{array}{l} A \to \alpha_1 \\ A \to \alpha_2\&\neg\alpha_1 \\ A \to \alpha_3\&\neg\alpha_1\&\neg\alpha_2 \\ \quad \vdots \\ A \to \alpha_n\&\neg\alpha_1\&\neg\alpha_2\&\dots\&\neg\alpha_{n-1} \end{array} \tag{3′}$$

If this transformation is applied to every nonterminal, the resulting grammar will satisfy condition I. Additionaly, 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:

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

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.

4 Normal forms

It is known that every Boolean grammar can be transformed to an equivalent grammar in the *binary normal form* [13], in which all rules are of the form

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

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

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

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 $A \in N$, $L_{G_1}(A) = L_G(A) \setminus \{\varepsilon\}$ [13].

At the second step, another 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

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

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 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 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 $A \in N$. Then, in course of the transformation of G to the binary normal form according to the above description,*

- *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.*

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 [2,8]. Similarly, log-square-time parallel algorithms can be speeded up to logarithmic time [16]. While logarithmic-time parallel parsing seems to have no analogues for Boolean grammars, the square-time Kasami–Torii algorithm [8], can be extended to unambiguous Boolean grammars, once its data structures and loops are refactored. Such an algorithm will be constructed 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 binary normal form. For every $X \subseteq conjuncts(G)$, define*

$$f(X) = \{A \mid \exists A \to B_1C_1 \& \dots \& B_\ell C_\ell \& \neg D_1E_1 \& \dots \& \neg D_mE_m \& \neg\varepsilon \in P,$$
$$s.t.\ A \to B_1C_1, \dots, A \to B_\ell C_\ell \in X, A \to \neg D_1E_1, \dots, A \to \neg D_mE_m \notin X\}$$

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

Now 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.*

See Appendix C.

Let us continue with the correctness statement of the algorithm, which claims what values should the variables have at certain points of the computation. It will refer to iterations of the outer loop by j (line 2) and of the nested loop by k (line 9), the latter will be denoted by pairs (j, k). 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 $t = 1, \ldots, j$, the set $E_t[A]$ equals

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

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

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

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

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

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.*

Theorem 1. *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.*

The algorithm relies upon the normal form, but Lemma 1 shows that there is no loss of generality in this assumption.

Theorem 2. *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 class 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 $\mathsf{UnambConj}$. The hypothetical corresponding sets of inherently ambiguous languages are $\mathsf{UnambBool} \setminus \mathsf{Bool}$ and $\mathsf{UnambConj} \setminus \mathsf{Conj}$. As for linear conjunctive languages, it is not hard to prove that each of them is unambiguous.

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 proposed construction is based upon the representation of linear conjunctive languages by trellis automata [12].

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 [10] 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,*

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

is a linear conjunctive language, and hence it is unambiguous.

The last example implies some standard undecidability results for unambiguous linear conjunctive grammars (such as the undecidability of emptiness problem, etc.) which carry on to unambiguous Boolean grammars.

Recalling some known examples of P-complete linear conjunctive languages [7,11], one can construct unambiguous grammars for these languages.

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

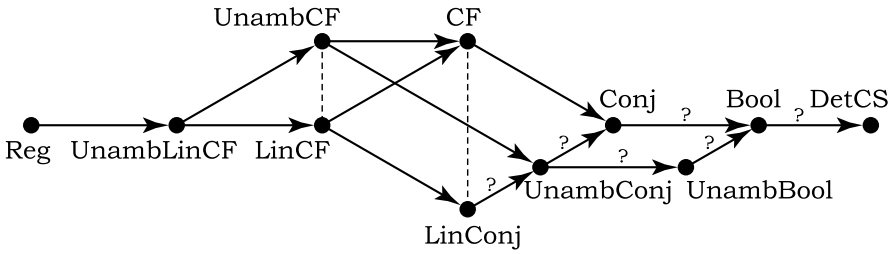

Fig. 2. Unambiguous language families in the overall hierarchy.

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 hypothetical sets of inherently ambiguous languages are $\mathsf{UnambBool} \setminus \mathsf{Bool}$ and $\mathsf{UnambConj} \setminus \mathsf{Conj}$, but it is interesting indeed whether these sets are nonempty.

Let us consider some candidate languages for being inherently ambiguous. One of them, given in Example 3, is the language $\{a^{2^n} \mid n \geqslant 0\}$. Another pair of candidates are $L_{ww} = \{ww \mid w \in \{a,b\}^*\}$ and its complement, $\overline{L_{ww}}$. The former is non-context-free, and the only known way of representing it by a Boolean grammar essentially uses a context-free grammar for $\overline{L_{ww}}$ and a negation on top of it. However, $\overline{L_{ww}}$ is an inherently ambiguous context-free language, so any Boolean grammar constructed in this way is also ambiguous. Unfortunately, no proofs of inherent ambiguity have so far been obtained; determining whether $\mathsf{UnambBool} = \mathsf{Bool}$ and whether $\mathsf{UnambConj} = \mathsf{Conj}$ are the main open problems on unambiguous variants of conjunctive and Boolean grammars.

Having introduced two new families of languages, $\mathsf{UnambConj}$ and $\mathsf{UnambBool}$, let us place them in the hierarchy of language families. The updated hierarchy is shown in Figure 2. The natural expectation is that all inclusions are proper, and every two families not connected by a directed path are incomparable. However, at present as many as six inclusions are not known to be proper. Also, for a few pairs of classes it is not known whether they are incomparable or not; these are: $\mathsf{UnambCF}$ and $\mathsf{LinConj}$, CF and $\mathsf{UnambConj}$, CF and $\mathsf{UnambBool}$, Conj and $\mathsf{UnambBool}$. These challenging open problems deserve investigation.

Let us compare the parsing complexity of these families of languages. Another partition is given by the degree of the polynomial of the time complexity of recognition algorithms: some classes of languages, The previously known families $\mathsf{UnambLinCF}$, LinCF, $\mathsf{UnambCF}$ and $\mathsf{LinConj}$ can be parsed in time $O(n^2)$, and both families introduced in this paper, $\mathsf{UnambConj}$ and $\mathsf{UnambBool}$, join them in the same class. For conjunctive and Boolean grammars of the general form, no better bound than $O(n^3)$ is known, while context-free grammars can be parsed at least as quckly as matrix multiplication, which, according to the current knowledge, can be done in time $O(n^{2.376})$.

To conclude, a useful subclass of Boolean grammars with an improved worst-case parsing time has been introduced, which becomes one more step towards practical applicability of these theoretically attractive grammars. This justifies continued attention to these grammars, and will hopefully lead to a conjectured family of *deterministic Boolean grammars*.

References

1. 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.
2. J. Earley, "An efficient context-free parsing algorithm", *Communications of the ACM*, 13:2 (1970), 94–102.
3. R. W. Floyd, "On ambiguity in phrase structure languages", *Communications of the ACM*, 5:10 (1962), pp. 526, 534.
4. S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.
5. S. Ginsburg, J. S. Ullian, "Ambiguity in context free languages", *Journal of the ACM*, 13:1 (1966), 62–89.
6. S. A. Greibach, "The undecidability of the ambiguity problem for minimal linear grammars", *Information and Control*, 6:2 (1963), 119–125.
7. O. H. Ibarra, S. M. Kim, "Characterizations and computational complexity of systolic trellis automata", *Theoretical Computer Science*, 29 (1984), 123–153.
8. T. Kasami, K. Torii, "A syntax-analysis procedure for unambiguous context-free grammars", *Journal of the ACM*, 16:3 (1969), 423–431.
9. 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.
10. A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.
11. A. Okhotin, "The hardest linear conjunctive language", *Information Processing Letters*, 86:5 (2003), 247–253.
12. A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *RAIRO Informatique Théorique et Applications*, 38:1 (2004), 69–88.
13. A. Okhotin, "Boolean grammars", *Information and Computation*, 194:1 (2004), 19–48.
14. A. Okhotin, "On the existence of a Boolean grammar for a simple programming language", *Automata and Formal Languages* (Proceedings of AFL 2005, May 17–20, 2005, Dobogókő, Hungary).
15. A. Okhotin, "Generalized LR parsing algorithm for Boolean grammars", *International Journal of Foundations of Computer Science*, 17:3 (2006), 629–664.
16. W. Rytter, "Parallel time $O(\log n)$ recognition of unambiguous context-free languages", *Information and Computation*, 73:1 (1987), 75–86.
17. V. Terrier, "On real-time one-way cellular array", *Theoretical Computer Science*, 141 (1995), 331–335.
18. K. Wich, "Sublogarithmic ambiguity", *Theoretical Computer Science*, 345:2–3 (2005), 473–504.

A Parse trees for Boolean grammars

Parse trees for Boolean grammars [13] are, strictly speaking, finite acyclic graphs rather than trees. A parse tree of a string $w = a_1 \dots a_{|w|}$ 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 $a_{i+1} \dots a_j$. In particular, each leaf a_i corresponds to itself.

For each vertex labelled with a rule

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

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

1. It has exactly $|\alpha_1 \dots \alpha_m|$ direct descendants corresponding to the symbols in positive conjuncts. For each nonterminal in $\alpha_1 \dots \alpha_m$, 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 $\alpha_t = s_1 \dots s_\ell$. There exist numbers $i_1, \dots, i_{\ell-1}$, where $i = i_0 \leqslant i_1 \leqslant \dots \leqslant i_{\ell-1} \leqslant i_\ell = j$, such that each descendant corresponding to s_r encompasses the substring $a_{i_{r-1}+1} \dots a_{i_r}$.
3. For each t-th negative conjunct of this rule, $a_{i+1} \dots a_j \notin L_G(\beta_t)$.

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.

B Normal form for unambiguous grammars

Proof (Lemma 1 on page 7). 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

$$A \to \alpha_1 \& \dots \& \alpha_m \& \neg\beta_1 \& \dots \& \neg\beta_n, \quad \text{where } \{\alpha_1, \dots, \alpha_m, \beta_1, \dots, \beta_n\} = P$$

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 rule for A in P_2 and in P_3 (or in P_4), such that the correponding 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 $1 \leqslant i_1 < \ldots < i_k \leqslant \ell$ 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, unless it is $A \to \neg\varepsilon$. Consider any two representations of any string as $L_{G_1}(s_{i_1}) \cdot \ldots \cdot L_{G_1}(s_{i_k})$:

$$w = u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k} \quad (\text{where } u_{i_t}, v_{i_t} \in L_{G_1}(s_{i_t}) \text{ for all } t) \tag{4}$$

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 (4) are actually the same, and, since the choice of the conjunct and the word 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. The case of the conjunct $A \to \pm s_1 A'$ has an even simpler proof, which is too elementary to be included.

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

C Correctness of the parsing algorithm

Proof (Lemma 2 on 9). An element $k-1$ can be added to $E_j[A]$ only at the iteration (j, k) of loops in lines 2 and 9. Hence, in the beginning of each iteration (j, k) the following condition holds: $E_j[A] \subseteq \{k, k+1, \dots, j-2\}$. Hence, if $E_j[A]$ is sorted before the assignment in line 15, it remains sorted after the assignment. All three claims follow. □

Proof (Lemma 3 on page 9). 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

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

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):

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

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 $A \in N$,

$$E_j[A] = \{i \mid k \leqslant i < j \text{ and } a_{i+1} \dots a_j \in L_G(A)\}, \tag{5}$$

while for each i,

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

Let us first show that the execution of lines 10–13 sets every $T[i]$ to

$$\{A \to \pm BC \mid \exists \ell\, (k \leqslant \ell < j) : a_{i+1} \dots a_\ell \in L(B) \text{ and } a_{\ell+1} \dots a_j \in L(C)\}. \tag{6}$$

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 (5), $k \in E_j[C]$. 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 3, each $T[i]$ is already of the form (6). This, in particular, implies

$$T[k-1] = \{A \to \pm BC \mid a_k \dots a_j \in L(BC)\}, \tag{7}$$

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 (5) 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

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

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, t. By (7), 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, 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. □

Proof (Lemma 4 on page 9). 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),

$$a_{k+1} \dots a_j \in L(C) \quad \text{and} \tag{8a}$$
$$a_{k'+1} \dots a_j \in L(C). \tag{8b}$$

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:

$$a_{i+1} \dots a_k \in L(B), \tag{9a}$$
$$a_{i+1} \dots a_{k'} \in L(B). \tag{9b}$$

Combining (9a) with (8a) and (9b) with (8b), 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, $k = k'$. □

Proof (Theorem 1 on page 9). 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. □

D Making linear conjunctive grammars unambiguous

Trellis automata [1,7], 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 (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 [12]. A grammar $G = (\Sigma, \{A_q \mid q \in Q\} \cup \{S\}, P, S)$ is constructed, where P consists of the following rules:

$$S \to A_q \quad \text{(for all } q \in F\text{)} \tag{10a}$$

$$A_{I(a)} \to a \quad \text{(for all } a \in \Sigma\text{)} \tag{10b}$$

$$A_{\delta(q_1,q_2)} \to A_{q_1} c \& b A_{q_2} \quad \text{(for all } q_1, q_2 \in Q \text{ and } b, c \in \Sigma\text{)} \tag{10c}$$

For this grammar it is known [12, Lemma 2] that $L_G(A_q) = \{w \mid \Delta(I(w)) = q\}$ and $L(G) = L(M)$. 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,

$$A_q \to A_{q_1} c \& b A_{q_2} \quad \text{and} \tag{11a}$$

$$A_q \to A_{q_3} c' \& b' A_{q_4}, \tag{11b}$$

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 [12, 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 (11) 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. □