Boolean grammars [1]

Alexander Okhotin

School of Computing, Queen's University, Kingston, Ontario, Canada K7L3N6.

Abstract

A new generalization of context-free grammars is introduced: *Boolean grammars* allow the use of all set-theoretic operations as an integral part of the formalism of rules. Rigorous semantics for these grammars is defined by language equations in a way that allows to generalize some techniques from the theory of context-free grammars, including Chomsky normal form, Cocke–Kasami–Younger cubic-time recognition algorithm and some limited extension of the notion of a parse tree, which together allow to conjecture practical applicability of the new concept.

Key words: Context-free grammar, intersection, complement, language equation, parsing, conjunctive grammar, trellis automaton, cellular automaton

1 Introduction

Context-free grammars are the most intuitively obvious syntactical formalism, and their formal properties, such as the decidability and complexity, are close to perfection. However, their generative power often turns out to be insufficient for denoting languages that arise in practice. Already Chomsky considered a more potent class of transformational grammars, *context-sensitive grammars*, but they proved to be much too powerful – equivalent to **NSPACE**(n), – which, besides grave implications on the complexity of decision problems, makes them an intricate programming language rather than a tool for describing syntax. The position of a language specification formalism adequate to the intuitive notion of syntax thus remained vacant.

Email address: `okhotin@cs.queensu.ca` (Alexander Okhotin).

[1] An earlier version of this paper was presented at the DLT 2003 conference held in Szeged, Hungary, July 7–11, 2003, and its extended abstract appeared in: Z. Ésik, Z. Fülöp (Eds.), *Developments in Language Theory*, LNCS 2710, 398–410.

Consequently, the search for formalisms with good properties has been a subject of efforts of formal language theorists for many years. Most of the attempts started from context-free grammars and proceeded with extending them with extra constructs. Among the formalisms thus obtained, let us mention *indexed grammars* [1], in which a stack of special symbols is attached to any nonterminal, and context-free derivation is modified to manipulate with these stacks; numerous types of *grammars with controlled derivation* [11], where some sequences of applications of context-free rules are disallowed by the means of a control language; *linear indexed grammars* [13], a computationally feasible subclass of indexed grammars; *tree-adjoining grammars* [17], which define transformations of context-free parse trees and contribute a new operation of inserting a subtree in the middle of an existing tree; *head grammars*, which transform pairs of words and contain a wrapping operation. The last three formalisms mentioned were eventually proved equivalent [18,35], and an extensive parsing theory for them was developed.

One more recently introduced extension of context-free grammars, *conjunctive grammars* [21] feature an explicit intersection operation. While context-free rules are of the form $A \to \alpha$, the rules in conjunctive grammars allow the use of conjunction: $A \to \alpha_1 \& \ldots \& \alpha_n$ ($n \geqslant 1$). The semantics of these grammars can be defined either by derivation [21], or using language equations with union, intersection and concatenation [22]. Several parsing algorithms for conjunctive grammars with worst-case cubic time performance, including extensions of LL(k) and generalized LR, were developed and implemented in a parser generator [23].

Conjunction is an intuitively obvious operation, as it denotes a set of strings that satisfy several conditions simultaneously. Another related operation is negation, which expresses that a string should *not* have some property, and including it in the formalism of rules is no less justified than including conjunction. The goal might appear clear – to introduce a class of grammars with rules of the form

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \quad (m + n \geqslant 1) \tag{1}$$

but meeting this goal presents certain difficulties. For instance, what to do with clearly contradictory rules like $S \to \neg S$? Should such grammars be considered ill-formed, what grammars are well-formed then, and how to define their semantics?

These difficulties associated with negation have already been encountered in the literature, and no direct way to solve them has been found. In the formal first-order theory over strings developed by Rounds [27], every variable is syntactically required to be within the scope of an even number of negations; this solves all the problems at once, but at the expense of generality. And the mere fact that one is forced to "exclude such cases for reasons of smoothness" [27] even in a *logic system* clearly points at the semantical nontriviality of negation. In the related subsequent work on negative range concatenation grammars, equipped with explicit Boolean operations and reduplication, Boullier [4] simply dismisses these definition problems with a brief remark that "some grammars are inconsistent".

This paper takes the challenge of adding syntactically unrestricted negation to the context-free grammars, and doing it with the appropriate rigorousness. The language generated by a grammar is defined using language equations with all Boolean operations and concatenation [25]. Two semantics for language equations are proposed in Section 2; one of them has a solid justification in terms of solutions of equations, while the other relies upon an adaptation of the *partial fixed point* method [16,34] to generalize the notion of derivability from context-free and conjunctive grammars [21]. The next Section 3 introduces Boolean grammars, using language equations as a formal semantics, and defines parse trees for them. In Section 4, a normal form for Boolean grammars that naturally extends the binary normal form for conjunctive [21] and Chomsky normal form for context-free grammars is proposed, and it is shown that every Boolean grammar can be effectively transformed to this normal form. Two recognition and parsing algorithms for Boolean grammars are developed in Section 5: one of them, operating in time $O(n^3)$ and space $O(n^2)$, naturally generalizes the similar algorithm for conjunctive grammars [21], which is in turn an extension of the Cocke–Kasami–Younger algorithm for context-free grammars [36]; the other uses space $O(n)$ (at the expense of exponential time), thus proving that any language generated by a Boolean grammar is deterministic context-sensitive. Section 7 summarizes the theoretical properties of the language family generated by Boolean grammars, and compares it to other families of languages.

2 Semantics for language equations

2.1 Language equations

Definition 1 (Language formula) *Let Σ be a finite nonempty alphabet and let $X = (X_1, \ldots X_n)$ ($n \geqslant 1$) be a vector of language variables. Language formulae over the alphabet Σ in variables X are defined inductively as follows:*

- *the empty string ε is a formula;*
- *any symbol from Σ is a formula;*
- *any variable from X is a formula;*
- *if φ and ψ are formulae, then $(\varphi \cdot \psi)$, $(\varphi \& \psi)$, $(\varphi \vee \psi)$ and $(\neg\varphi)$ are formulae.*

As in logic formulae, the parentheses will be omitted whenever possible, and the following default precedence of operators will be used: the concatenation has the highest precendence and is followed by the logical connectives arranged in their usual order: $\neg$, $\&$ and $\vee$. If needed, this default precedence will be overridden with parentheses; the dot for concatenation will be most of the time omitted. For instance, $XY \vee \neg aX \& aY$ means the same as $(X \cdot Y) \vee ((\neg(a \cdot X)) \& (a \cdot Y))$. Note that all the mentioned binary logical operations, as well as concatenation, are associative, and therefore there is no need to disambiguate formulae like XYZ or $X \vee Y \vee Z$ with extra parentheses.

The syntax of formulae has been defined; let us now define their semantics by interpreting the connectives with operations on languages, thus associating a language function with every formula:

Definition 2 (Value of a formula) *Let φ be a formula over an alphabet Σ in variables $X = (X_1, \ldots, X_n)$. Let $L = (L_1, \ldots, L_n)$ be a vector of languages over Σ. The value of the formula φ on the vector of languages L, denoted as $\varphi(L)$, is defined inductively on the structure of φ: $\varepsilon(L) = \{\varepsilon\}$, $a(L) = \{a\}$ for every $a \in \Sigma$, $X_i(L) = L_i$ for every i ($1 \leqslant i \leqslant n$), $\psi\xi(L) = \psi(L) \cdot \xi(L)$, $(\psi \vee \xi)(L) = \psi(L) \cup \xi(L)$, $(\psi \& \xi)(L) = \psi(L) \cap \xi(L)$ and $(\neg\psi)(L) = \Sigma^* \setminus \psi(L)$.*

The value of a vector of formulae $\varphi = (\varphi_1, \ldots, \varphi_\ell)$ on a vector of languages $L = (L_1, \ldots, L_n)$ is the vector of languages $\varphi(L) = (\varphi_1(L), \ldots, \varphi_\ell(L))$.

Definition 3 (System of equations) *Let Σ be an alphabet. Let $n \geqslant 1$. Let $X = (X_1, \ldots, X_n)$ be a set of language variables. Let $\varphi = (\varphi_1, \ldots, \varphi_n)$ be a vector of formulae in variables X over the alphabet Σ. Then*

$$\begin{cases} X_1 = \varphi_1(X_1, \ldots, X_n) \\ \quad\vdots \\ X_n = \varphi_n(X_1, \ldots, X_n) \end{cases} \tag{2}$$

is called a resolved system of equations over Σ in variables X. (2) can also be denoted in vector form as $X = \varphi(X)$.

A vector of languages $L = (L_1, \ldots, L_n)$ is said to be a solution of the system (2), if for every i ($1 \leqslant i \leqslant n$) it holds that $L_i = \varphi_i(L_1, \ldots, L_n)$. In the vector form, this is denoted $L = \varphi(L)$.

Let us introduce some simple language-theoretic terminology that will be used in the following. For a pair of languages $L_1, L_2 \subseteq \Sigma^*$ and another language $M \subseteq \Sigma^*$, we say that L_1 and L_2 are equal modulo M if $L_1 \cap M = L_2 \cap M$; this is denoted $L_1 = L_2 \pmod{M}$. The relation of equality modulo M is easily extended to vectors of languages. A vector of languages L is a solution of a system $X = \varphi(X)$ modulo M, if the vectors L and $\varphi(L)$ are equal modulo M.

For every string $w \in \Sigma^*$, $y \in \Sigma^*$ is a *substring* of w if $w = xyz$ for some $x, z \in \Sigma^*$; y is a *proper substring* of w if additionally $|y| < |w|$. A language L is *closed under substring*, if all substrings of every $w \in L$ are in L.

Vectors of languages are partially ordered with respect to componentwise inclusion as follows: $(L'_1, \ldots, L'_n) \preccurlyeq (L''_1, \ldots, L''_n)$ if and only if $L'_i \subseteq L''_i$ for all i ($1 \leqslant i \leqslant$

n).

Example 1 *The following system of equations over the alphabet* $\Sigma = \{a, b\}$

$$X_1 = \neg X_2 X_3 \& \neg X_3 X_2 \& X_4 \qquad X_3 = (a \vee b) X_3 (a \vee b) \vee b$$
$$X_2 = (a \vee b) X_2 (a \vee b) \vee a \qquad X_4 = (aa \vee ab \vee ba \vee bb) X_4 \vee \varepsilon$$

has the unique solution $(\{ww \mid w \in \{a,b\}^*\},\ \{xay \mid x, y \in \{a,b\}^*, |x| = |y|\},\ \{xby \mid x, y \in \{a,b\}^*, |x| = |y|\},\ \{u \mid u \in \{a,b\}^{2n},\ n \geqslant 0\})$.

If the first variable is interpreted as a "start symbol", then the system of language equations given in Example 1 can be said to denote the language $\{ww \mid w \in \{a,b\}^*\}$. This abstract language is often given as an example that captures the notion of "reduplication", which is viewed as an essential property of natural languages. The language $\{ww \mid w \in \{a,b\}^*\}$ is co-context-free, but not context-free; moreover, it is known not to be representable as a finite intersection of context-free languages. Although conjunctive grammars can denote a very similar language $\{wcw \mid w \in \{a,b\}^*\}$ [21], it is not known whether $\{ww \mid w \in \{a,b\}^*\}$ is a conjunctive language, or, equivalently, whether it can be denoted using a system of language equations containing concatenation, union and intersection, but not negation.

For language equations with negation *only* it was shown by Leiss [20] that there exists a single language equation that has unique solution $L = \{a^n \mid \exists k \geqslant 0, \text{ such that } 2^{3k} \leqslant n < 2^{3k+2}\}$, which is a nonregular unary language. Using symmetric difference, one can denote the language $L \,\triangle\, aL = \{a^n \mid \exists k \geqslant 0, \text{ such that } n = 2^{3k} \text{ or } n = 2^{3k+2}\}$ [12], which is "almost" $\{a^{2^n} \mid n \geqslant 0\}$, one more standard example of a non-context-free language. Let us define exactly the latter language:

Example 2 *Let* φ^2 *abbreviate* $\varphi \cdot \varphi$*. The system*

$$S = (X \& \neg aX) \vee (\neg X \& aX) \vee (Z \& \neg aZ) \vee (\neg Z \& aZ) \tag{3a}$$
$$X = a(\neg(\neg(\neg X)^2)^2)^2 \tag{3b}$$
$$Y = aa(\neg(\neg(\neg Y \& T)^2 \& T)^2 \& T)^2 \tag{3c}$$
$$Z = Y \vee aY \tag{3d}$$
$$T = aaT \vee \varepsilon \tag{3e}$$

over the alphabet $\{a\}$ *in variables* $\{S, X, Y, Z, T\}$ *has the unique solution* $(\{a^{2^n} \mid n \geqslant 0\}, \{a^n \mid \exists k \geqslant 0 :\ 2^{3k} \leqslant n \leqslant 2^{3k+2} - 1\}, \{a^n \mid \exists k \geqslant 0 :\ 2^{3k+1} \leqslant n \leqslant 2^{3k+3} - 2,\ \textit{and } n \textit{ is even}\}, \{a^n \mid \exists k \geqslant 0 :\ 2^{3k+1} \leqslant n \leqslant 2^{3k+3} - 1\}, (aa)^*)$.

The equation (3b) is from Leiss [20]; the equation (3c) uses the same technique to construct the language of strings twice as long. The next equation (3d) adds a string of odd length to each string in (3c), thus filling the gaps between the strings of even length. Now S is the union of two symmetic differences, $X \bigtriangleup aX = \{a^n \mid \exists k \geqslant 0 : n = 2^{3k} \text{ or } n = 2^{3k+2}\}$ and $Z \bigtriangleup aZ = \{a^n \mid \exists k \geqslant 0 : \ n = 2^{3k+1} \text{ or } n = 2^{3k+3}\}$. This union equals precisely $\{a^{2^n} \mid n \geqslant 0\}$.

2.2 Semantics of unique solution in the strong sense

In Examples 1 and 2, languages are defined as first components of unique solutions of systems; it would be natural to use this as a semantics for language equations. However, it has recently been proved [25] that the set of systems that have exactly one solution is in the second level of the arithmetical hierarchy, and even worse, the class of languages defined in this way is exactly the class of recursive sets. This is definitely too much; in order to use language equations as a basis for a practical language specification formalism, the source of this enormous expressive power has to be located, and a natural way to limit it has to be invented.

To begin with, consider the following first-order characterization of systems with a unique solution:

Theorem 1 (Criterion of solution uniqueness [25]) *A system of language equations has a unique solution if and only if for every finite language M closed under substring there exists a finite language $M' \supseteq M$ closed under substring, such that there exists at least one solution of the system modulo M', and all the solutions modulo M' are equal modulo M.*

So, if a system has a unique solution and one wants to check the membership of a string w in the components of this solution, then one can set M to be the set of substrings of w and look for a finite modulus $M' \supseteq M$ that satisfies the condition formulated in Theorem 1. The existence of such M' is guaranteed by the theorem, the domain of search is countable and effectively enumerable, and hence M' will eventually be found. Once it is found, the solutions modulo M' coincide modulo M, which gives (modulo M) the unique solution of the system.

The only problem is that there is no *a priori* lower bound on the cardinality of M' and hence on the time of search. The characterization of recursive sets by unique solutions of language equations [25] is based upon a peculiar way to extract the language recognized by a Turing machine out of the language of its computations; so, M' can well contain computations of some Turing machine on all strings from M. It is known that $|M'|$ is a recursive function of $|M|$ [25], but that can be an *arbitrary* recursive function!

Taking note of the origin of unbounded complexity, let us impose an additional restriction upon the systems of equations with a unique solution. We require that a system has a unique solution modulo every finite M closed under substring. M' in Theorem 1 is thus forced to be always equal to M, making the search for it immediate.

Definition 4 *A system of language equations is said to be compliant to the semantics of the unique solution in the strong sense if for every finite M closed under substring the system has a unique solution modulo M.*

As mentioned above, Definition 4 implies the condition of Theorem 1, and thus a system compliant to this semantics indeed has a unique solution. This solution can be computed modulo every finite M by simply finding the unique solution modulo M, which can be done by an exhaustive search.

This search is best conducted by increasing M string by string. If the unique solution modulo M is known, then the unique solution modulo $M \cup \{u\}$ can be determined by trying 2^n possible candidates:

Proposition 1 *Let a system of equations $X = \varphi(X)$ have a unique solution modulo every language closed under substring. Let M be a finite language closed under substring, let $u \in \Sigma^*$ be a string not in M, such that all of its proper substrings are in M. If $L = (L_1, \ldots, L_n)$ ($L_i \subseteq M$) is the unique solution modulo M, then the unique solution modulo $M \cup \{u\}$ is of the form $(L_1 \cup L'_1, \ldots, L_n \cup L'_n)$ for some $L'_1, \ldots, L'_n \subseteq \{u\}$.*

It should be noted that one cannot effectively decide whether a system complies to this semantics.

Theorem 2 *The set of systems compliant to the semantics of unique solution in the strong sense is co-**RE**-complete.*

The membership in co-**RE** is witnessed by a nondeterministic Turing machine that recognizes the complement of the problem by guessing a finite modulus M and then accepting if and only if the given system has none or multiple solutions modulo M. Co-**RE**-hardness is proved by a standard reduction from the complement of the Post Correspondence Problem, very similar to the proof of co-**RE**-hardness of the solution existence problem [25, Theorem 2].

2.3 Semantics of naturally reachable solution

The previous section gives a well-formed semantics for language equations, which associates a language with every compliant system. What it lacks, is an assignment of a syntactical structure to strings, and this deficiency appears to be inherent. Consider the system

$$\begin{aligned} X &= X \\ Y &= \neg Y \& \neg X \end{aligned} \tag{4}$$

It is easy to see that it has unique solution $(\Sigma^*, \varnothing)$, its solution modulo every language is unique, and thus the system denotes the language Σ^* by the first component. However, if one considers *why* some particular string w is in this language, the only answer will be that the second equation would form a contradiction otherwise, which explanation can hardly count as a syntactical parse.

In derivation-based formal grammars, syntactical structure is given by *derivation trees*, and thus the process of proving a string to be in the language is linked to producing its parse. In terms of language equations, this is the semantics of the least fixed point of a system of equations with union [7,14] or with union and intersection [21,22]. This essentially relies on the monotonicity of the operations involved.

Can one extend derivability for the nonmonotone negation? A similar problem has been encountered in restricted applied logics, and a solution was proposed by Vardi [34, Section 4]. In loose terms, his approach can be described as trying to compute the solution iteratively, using a certain method that always converges to the least solution if negation is not used. In the presence of negation, the process is not guaranteed to terminate; if it does terminate, it converges to a solution, but not necessarily to the least one. This method became known in the literature under the name of a *partial fixed point* [16].

Applying a variation of this method to the language equations, the inductive approach of Proposition 1 is slightly modified to define the parse-oriented *semantics of the naturally reachable solution*:

Definition 5 (Naturally reachable solution) *Let $X = \varphi(X)$ be a system of equations. A vector $L = (L_1, \ldots, L_n)$ is called a naturally reachable solution of the system if for every finite modulus M closed under substring and for every string $u \notin M$ (such that all proper substrings of u are in M) every sequence of vectors of the form*

$$L^{(0)}, L^{(1)}, \ldots, L^{(i)}, \ldots \tag{5}$$

(where $L^{(0)} = (L_1 \cap M, \ldots, L_n \cap M)$ and every next vector $L^{(i+1)} \neq L^{(i)}$ in the sequence is obtained from the previous vector $L^{(i)}$ by substituting some j-th component with $\varphi_j(L^{(i)}) \cap (M \cup \{u\})$) converges to

$$(L_1 \cap (M \cup \{u\}), \ldots, L_n \cap (M \cup \{u\})) \tag{6}$$

in finitely many steps regardless of the choice of components at each step.

Note that such a sequence can only converge to a solution modulo $M \cup \{u\}$ – otherwise further transformations would be applicable to (6), and hence (5) would never actually converge to it.

Lemma 1 (Consistency of Definition 5) *A naturally reachable solution is a solution. A system cannot have more than one naturally reachable solution.*

PROOF. Since a naturally reachable solution is a solution modulo every finite M closed under substring, it is known to be a solution [25, Lemma 2]. If L', L'' both satisfy Definition 5, then they can be proved to coincide modulo every finite M closed under substring, inductively on the cardinality of M: indeed, if $L' = L'' \pmod{M}$, then the sequence (5) should converge to a single vector modulo $M \cup \{u\}$, and therefore $L' = L'' \pmod{M \cup \{u\}}$. Hence [25, Proposition 1], $L' = L''$. $\square$

Also note that all vectors forming the sequence (5) are equal modulo M (because the initial term is a solution modulo M), and therefore the derivation is confined to transforming Boolean vectors of the membership of u in the components, similarly to Proposition 1. It follows that a sequence (5) cannot be longer than 2^n, because otherwise it will go into an infinite loop and consequently violate the definition. Thus the naturally reachable solution modulo every finite language can be computed by following Definition 5 and carrying out the derivation (5), repeating this inductively on the cardinality of a modulus.

According to this semantics, some systems with multiple solutions become well-formed. Consider

$$\begin{aligned} X &= \neg Y \\ Y &= Y \end{aligned} \tag{7}$$

Although $(L, \overline{L})$ $(L \subseteq \Sigma^*)$ are all solutions of (7), one of them is the naturally reachable solution – $(\Sigma^*, \varnothing)$ – and this distinction is quite deserved: indeed, from the intuitive point of view, Y can "derive" nothing, and hence X denotes Σ^*.

While the semantics of the naturally reachable solution does not have the same clear theoretical justification as the semantics of the unique solution in the strong sense has, the new semantics has a pleasant property of being "backward compatible" with conjunctive and context-free grammars.

Theorem 3 *If a system $X = \varphi(X)$ contains no negation, then its least solution is naturally reachable.*

PROOF. It is known that such a system has least solution [22]; the proof is a straightforward adaptation of the fixed point techniques used for the context-free grammars [2], showing that the sequence $\{\varphi^k(\varnothing, \ldots, \varnothing)\}_{k=0}^{\infty}$ monotonely increases and converges to the least solution of the system. Let L be the limit of this sequence, and let us prove that for every M and u it satisfies Definition 5. Fix a sequence (5).

The first thing to prove is that the sequence (5) is increasing, i.e., $L^{(i)} \preccurlyeq L^{(i+1)}$ for all $i \geqslant 0$. This is proved by induction on i, and it suffices to show that if

$u \in L_j^{(i)}$, then $u \in \varphi_j(L^{(i)})$. Indeed, if $u \in L_j^{(i)}$, then u was added at some step i_0 ($0 < i_0 \leqslant i$), which means that $u \in \varphi_j(L^{(i_0-1)})$. Since $i_0 - 1 < i < i+1$, $L^{(i_0-1)} \preccurlyeq L^{(i)}$ holds by the induction hypothesis, and hence $\varphi_j(L^{(i_0-1)}) \preccurlyeq \varphi_j(L^{(i)})$ by the monotonicity of φ_j. Therefore, $u \in \varphi_j(L^{(i)})$.

Thus the sequence (5) converges to some language L', which must be a solution modulo $M \cup \{u\}$. Let us take this L' and iteratively apply φ to it, obtaining a sequence $\{\varphi^k(L')\}_{k=0}^{\infty}$, such that $\varphi^k(L') = L' \pmod{M \cup \{u\}}$ for all $k \geqslant 0$ (established by an induction on k). This sequence is monotone, i.e.,

$$L' \preccurlyeq \varphi(L') \preccurlyeq \varphi^2(L') \preccurlyeq \ldots \preccurlyeq \varphi^k(L') \preccurlyeq \ldots \tag{8}$$

and hence converges to some language $\sup_k \varphi^k(L') = L''$, which equals L' modulo $M \cup \{u\}$. L'' can be proved to be a solution of the system using the standard method based upon applying φ to each term of (8), showing that the resulting sequence converges to $\varphi(L'')$ (the lattice-theoretic continuity of φ is essentially used here), and then observing that this sequence is the same as (8), and thus their respective limits, L'' and $\varphi(L'')$, have to coincide. Now, since L is the least solution, $L \preccurlyeq L''$, and therefore

$$L \preccurlyeq L'' = L' \pmod{M \cup \{u\}} \tag{9}$$

Because the sequence $\{\varphi^k(\varnothing, \ldots, \varnothing)\}$ converges to L, there exists a number $k_0 \geqslant 0$, such that $\varphi^{k_0+k}(\varnothing, \ldots, \varnothing) = L \pmod{M}$ for all $k \geqslant 0$. Let us prove that

$$\varphi^{k_0+k}(\varnothing, \ldots, \varnothing) \succcurlyeq L^{(k)} \quad \text{(for all } k \geqslant 0) \tag{10}$$

Induction on k. The basis case, $k = 0$, holds because $\varphi^{k_0}(\varnothing, \ldots, \varnothing) = L \pmod{M}$, $L^{(0)} = L \pmod{M}$ and $L^{(0)} = \varnothing \pmod{\Sigma^* \setminus M}$. For the induction step, let i be the component modified in the $(k+1)$-th element of (5), and denote $\tilde{L} = \varphi^{k_0+k}(\varnothing, \ldots, \varnothing)$; then:

$$\begin{aligned}\varphi^{k_0+k+1}(\varnothing, \ldots, \varnothing) =& (\varphi_1(\tilde{L}), \ldots, \varphi_{i-1}(\tilde{L}), \varphi_i(\tilde{L}), \varphi_{i+1}(\tilde{L}), \ldots, \varphi_n(\tilde{L})) \succcurlyeq \\ & (\tilde{L}_1, \ldots, \tilde{L}_{i-1}, \varphi_i(\tilde{L}), \tilde{L}_{i+1}, \ldots, \tilde{L}_n) \succcurlyeq \\ & (L_1^{(k)}, \ldots, L_{i-1}^{(k)}, \varphi_i(L^{(k)}), L_{i+1}^{(k)}, \ldots, L_n^{(k)}) = L^{(k+1)},\end{aligned}$$

where the first relation is based upon the monotonicity of $\{\varphi^k(\varnothing, \ldots, \varnothing)\}$, while the second relation employs the induction hypothesis. The claim (10) is proved.

By (10), $L \succcurlyeq \varphi^{k_0+k'}(\varnothing, \ldots, \varnothing) \succcurlyeq L^{(k')} = L' \pmod{M \cup \{u\}}$, where k' is the number of the last element of (5). This, together with (9), proves that $L' = L \pmod{M \cup \{u\}}$ $\square$

The semantics of the naturally reachable solution suffers from the same unfortunate undecidability as the semantics of the unique solution in the strong sense. The following result can be proved exactly as Theorem 2.

Theorem 4 *The set of systems compliant to the semantics of naturally reachable solution is co-RE-complete.*

It should be noted that if a system complies to both semantics, then it defines the same vector under both semantics, because the solution is unique by the first semantics, and therefore the second semantics cannot define any other solution. Thus they do not contradict each other, but the classes of systems compliant to the two semantics are easily seen to be incomparable: consider (4) and (7). Later on it will be proved that these two semantics nevertheless define a single class of languages.

3 Definition of Boolean grammars

3.1 Grammars

The class of Boolean grammars can now be defined, using systems of language equations as formal semantics.

Definition 6 *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, each of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \quad (m + n \geqslant 1,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*), \quad (11)$$

where objects of the form $A \to \alpha_i$ and $A \to \neg\beta_j$ are called conjuncts, positive and negative respectively; $S \in N$ is the start symbol of the grammar.

The right hand side of every rule is a formula, and a grammar is interpreted as a system of equations over Σ in variables N of the form

$$A = \bigvee_{A \to \varphi \in P} \varphi \quad (\textit{for all } A \in N) \quad (12)$$

The vector of languages L generated by a grammar is then defined using either the semantics of unique solution in the strong sense or the semantics of naturally reachable solution (see Sections 2.2 and 2.3 respectively).

For every formula φ, denote the language of the formula $L_G(\varphi) = \varphi(L)$. Denote the language generated by the grammar as $L(G) = L_G(S)$.

Every conjunctive grammar is a Boolean grammar, in which every rule (11) contains only positive conjuncts, i.e., $m \geqslant 1$ and $n = 0$; it is compliant to the semantics of the naturally reachable solution by Theorem 3. In particular, every context-free grammar is a Boolean grammar, in which every rule (11) contains a single positive conjunct ($m = 1$, $n = 0$).

Consider the system of equations from Example 1, and let us use its general idea to produce the following Boolean grammar:

Example 3 *Let $G = (\{a, b\}, \{S, A, B, C, X\}, P, S)$ be a Boolean grammar, where P consists of the following rules:*

$$\begin{array}{lllll} S \to \neg AB \& \neg BA \& C & A \to XAX & B \to XBX & C \to XXC & X \to a \\ & A \to a & B \to b & C \to \varepsilon & X \to b \end{array}$$

Then $L(G) = \{ww \mid w \in \{a, b\}^*\}$ with respect to either of the two semantics.

The system from Example 2 can be written as a grammar in a similar way:

Example 4 *Let $G = (\{a\}, \{S, X, X', X'', X''', Y, Y', Y'', Y''', Z, T\}, P, S)$ be a Boolean grammar, where P contains the rules*

$$\begin{array}{llll} S \to X \& \neg aX & X \to aX'X' & Y \to aaY'Y' & Z \to Y \\ S \to \neg X \& aX & X' \to \neg X''X'' & Y' \to \neg Y''Y'' \& T & Z \to aY \\ S \to Z \& \neg aZ & X'' \to \neg X'''X''' & Y'' \to \neg Y'''Y''' \& T & T \to aaT \\ S \to \neg Z \& aZ & X''' \to \neg X & Y''' \to \neg Y \& T & T \to \varepsilon \end{array}$$

Then $L(G) = \{a^{2^n} \mid n \geqslant 0\}$ under both semantics.

Although Boolean grammars, unlike context-free and conjunctive grammars, are defined using language equations, and the arrow in their rules has lost its original interpretation of string rewriting, they nevertheless have a lot in common with these classes of transformational grammars. In fact, much of the theory of context-free grammars can be equally developed using language equations over the algebraic operations of union and concatenation [19], and now somewhat similar methods shall be developed for a formalism that has neither string rewriting nor semiring theory behind. The first thing to be generalized is the notion of a parse tree.

3.2 Parse trees

The reason for introducing the semantics of the naturally reachable solution was to associate a syntactical structure with sentences. Using Boolean grammars, this association will now be defined.

Let $G = (\Sigma, N, P, S)$ be a Boolean grammar compliant to this semantics, and suppose without loss of generality that every rule in P contains at least one positive conjunct (every grammar can be converted to this form by adding a new nonterminal that generates Σ^*, and by referring to it in every rule). Let $r = |N|$. A parse tree of a string $w \in L_G(A)$ ($A \in N$) from A is an acyclic directed graph with shared leaves that has a terminal leaf for every symbol in w. Define it inductively on the length of w.

Let $L^{(0)}, \ldots, L^{(z)}$ be a sequence of vectors satisfying Definition 5 for a string w and a modulus M. For all p ($0 \leqslant p \leqslant z$), let $L^{(p)} = (L_1^{(p)}, \ldots, L_r^{(p)})$. Let us construct the sequence $\{(t_1^{(p)}, \ldots, t_r^{(p)})\}_{p=0}^{z}$ of vectors of trees corresponding to the sequence of vectors of languages. For the initial term of the sequence, define all $t_i^{(0)}$ to be empty. For every next p-th term, if the string u is added to some k-th component, then there should exist a rule

$$A_k \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \tag{13}$$

such that $w \in \alpha_i(L^{(p-1)})$ for all i and $w \notin \beta_j(L^{(p-1)})$ for all j. In the construction, the negative conjuncts are completely ignored and the positive ones are used. The goal is to construct a tree with a root labeled with (13), which will have $|\alpha_1| + \ldots + |\alpha_m|$ descendants corresponding to all symbols from these positive conjuncts.

For each $\alpha_i = s_1 \ldots s_\ell$ there are ℓ descendants to add. There exists a factorization $u = v_1 \ldots v_\ell$, such that $v_j \in s_j(L^{(p-1)})$ for all j. For $s_j \in \Sigma$, a leaf labeled s_j is simply added. For $s_j \in N$, if the corresponding v_j is shorter than u, then, by the induction hypothesis, a parse tree of v_j from s_j is known, and hence can be used as a subtree. If v_j is of the same length as u, then $v_j = u$, and therefore $u \in L_j^{(p-1)}$. Then this subtree is already stored in $t_j^{(p-1)}$ and can now also be connected to the new root.

The subtrees collected for all positive conjuncts have the same set of terminal leaves – those corresponding to the symbols from u. So the corresponding leaves in these subtrees are identified (i.e., glued together), connected to a single root and the newly constructed tree is placed in $t_k^{(p)}$. In the end, $t_A^{(z)}$ contains the required tree.

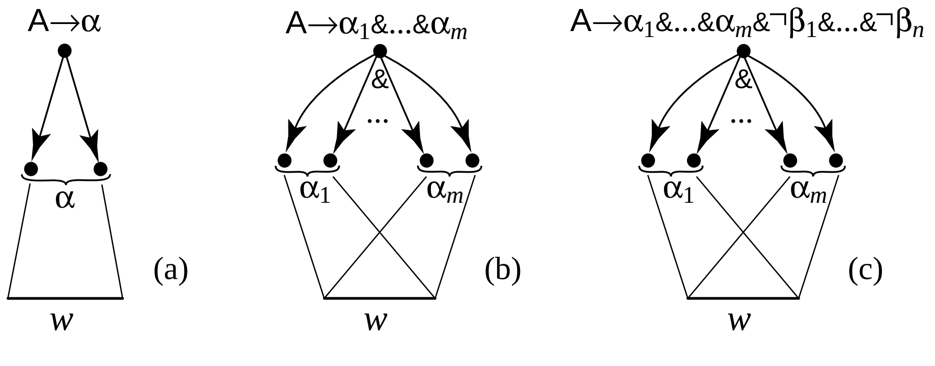

Fig. 1. Parse trees in (a) context-free (b) conjunctive (c) Boolean grammars.

Note that, in view of Theorem 3, this technique is a generalization of the tree construction method for conjunctive grammars, which in turn generalizes the context-free case (see Figure 1). For grammars that use negation as the principal logical connective, such as those from Examples 3 and 4, this method does not have much sense, because the resulting tree contains no meaningful information. But if negation is used sparingly, such trees can contain enough “positive” information on the syntactic structure of the string according to the grammar.

4 Normal form

One of the most important context-free techniques to generalize is the Chomsky normal form and an effective algorithm for transforming a grammar to this normal form. All of this has been generalized for the case of conjunctive grammars [21] without any major difficulties, simply by extending the proof techniques due to Bar-Hillel, Perles and Shamir [3], based on removing ε rules and then unit rules; a further generalization of the method is presented in this section.

In the case of Boolean grammars, the proof methods substantially deviate from the prototype and are technically more difficult, but the main schedule of removing epsilon rules first and unit rules next is preserved.

4.1 Epsilon conjuncts

Given a Boolean grammar that generates a vector of languages $L = (L_1, \ldots, L_n)$ under one of the mentioned semantics, the goal is to construct a Boolean grammar that generates $L' = (L_1 \setminus \{\varepsilon\}, \ldots, L_n \setminus \{\varepsilon\})$ under the same semantics. As in the cases of context-free and conjunctive grammars, this is being done by removing the so-called *positive epsilon conjuncts* of the form $A \to \varepsilon$. The case of Boolean grammars is more complicated, because positive epsilon conjuncts are not the only way of generating the empty string: in fact, putting negation over any formula that preserves ε-freeness immediately creates it. In order to ban ε entirely, the formal constuction includes a *negative epsilon conjunct* of the form $A \to \neg\varepsilon$ in every rule.

Let $G = (\Sigma, N, P, S)$ by a Boolean grammar, and let $L^\varepsilon = \langle L^\varepsilon_A \rangle_{A \in N}$ be a solution modulo $\{\varepsilon\}$ of the corresponding system. With respect to G and L^ε, define $\rho(\alpha)$ (for each $\alpha \in (\Sigma \cup N)^*$) to be the set of all nonempty strings $\alpha' = s_1 \ldots s_k$ ($k \geqslant 1$, $s_i \in \Sigma \cup N$), such that $\alpha = \nu_0 s_1 \nu_1 s_2 \ldots \nu_{k-1} s_k \nu_k$ for some $\nu_0, \ldots, \nu_k \in Nullable^*$, where $Nullable \subseteq N$ denotes $\{A \mid \varepsilon \in L^\varepsilon_A\}$.

Lemma 2 *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar, let L^ε be a vector of languages, let ρ be defined with respect to these. Let L, L' be vectors of languages, such that $L = L' \pmod{\Sigma^+}$, $L = L^\varepsilon \pmod{\{\varepsilon\}}$ and $L' = \varnothing \pmod{\{\varepsilon\}}$. Then, for every $w \in \Sigma^+$ and $\alpha \in (\Sigma \cup N)^*$, $w \in \alpha(L)$ holds if and only if $w \in \alpha'(L')$ for some $\alpha' \in \rho(\alpha)$.*

PROOF. If $\alpha = \varepsilon$, then $w \notin \alpha(L)$ and $\rho(\alpha) = \varnothing$; the statement trivially holds.

Let $\alpha = s_1 \ldots s_\ell$, where $s_i \in \Sigma \cup N$ and $\ell \geqslant 1$.

$\ominus\!\!\!\!\Rightarrow$ If $w \in \alpha(L)$, then w can be factorized as $w = u_1 \ldots u_\ell$, where $u_i \in s_i(L)$ for all i ($1 \leqslant i \leqslant \ell$). Let $1 \leqslant i_1 < \ldots < i_k \leqslant \ell$ be the numbers of all nonempty factors ($u_{i_j} \neq \varepsilon$); $k \geqslant 1$, because $w \neq \varepsilon$. For every empty factor $u_t = \varepsilon$ ($t \notin \{i_j\}$) we know that $\varepsilon \in s_t(L)$ and therefore $s_t \in \{A \mid \varepsilon \in L_A\}$ and $s_t \in \{A \mid \varepsilon \in L^\varepsilon_A\}$. So, $\alpha = s_1 \ldots s_{i_1-1} s_{i_1} s_{i_1+1} \ldots s_{i_k-1} s_{i_k} s_{i_k+1} \ldots s_\ell$, where $s_1 \ldots s_{i_1-1}, s_{i_1+1} \ldots s_{i_2-1}, \ldots, s_{i_k+1} \ldots s_\ell \in Nullable^*$. and thus $\alpha' = s_{i_1} \ldots s_{i_k}$ is in $\rho(\alpha)$ by the definition of ρ. On the other hand, since every u_{i_j} ($1 \leqslant j \leqslant k$) is nonempty, $u_{i_j} \in s_{i_j}(L)$ implies $u_{i_j} \in s_{i_j}(L')$. Therefore, $w = u_{i_1} \ldots u_{i_k} \in s_{i_1}(L') \ldots s_{i_k}(L') = \alpha'(L')$.

$\ominus\!\!\!\!\Leftarrow$ Let $w \in \alpha'(L')$, where $\alpha' \in \rho(\alpha)$. Let $\alpha' = s_1 \ldots s_k$. Then there exists a factorization $w = u_1 \ldots u_k$, where $u_i \in s_i(L')$. Since L' is ε-free, all the strings u_i are nonempty, and therefore $u_i \in s_i(L)$. By the definition of $\rho(\alpha)$, $\alpha = \nu_0 s_1 \nu_1 s_2 \ldots \nu_{k-1} s_k \nu_k$, where $\nu_0, \ldots, \nu_k \in Nullable^*$, i.e., $\varepsilon \in \nu_i(L^\varepsilon)$ and hence $\varepsilon \in \nu_i(L)$. Consequently, $w = \varepsilon \cdot u_1 \cdot \varepsilon \cdot \ldots \cdot \varepsilon \cdot u_k \cdot \varepsilon \in \nu_0(L) s_1(L) \nu_1(L) \ldots \nu_{k-1}(L) s_k(L) \nu_k(L) = \alpha(L)$. $\square$

Construct a Boolean grammar $G' = (\Sigma, N, P', S)$, such that for every rule

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \tag{14}$$

from P, where $\rho(\alpha_i) = \{\mu_{i1}, \ldots, \mu_{ik_i}\}$ ($k_i \geqslant 0$; for all i) and $\rho(\beta_j) = \{\nu_{j1}, \ldots, \nu_{jl_j}\}$ ($l_i \geqslant 0$; for all i), the set P' contains the rule

$$A \to \mu_{1t_1} \& \ldots \& \mu_{mt_m} \& \neg\nu_{11} \& \ldots \& \neg\nu_{1l_1} \& \ldots \& \neg\nu_{n1} \& \ldots \& \neg\nu_{1l_n} \& \neg\varepsilon \tag{15}$$

for every vector of numbers $(t_1, \ldots, t_m)$ ($1 \leqslant t_i \leqslant k_i$ for all i).

Lemma 3 *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar. Let L^ε be a solution modulo $\{\varepsilon\}$ of the system corresponding to G. Let the Boolean grammar G' be constructed out of G and L^ε by the method above. Let $X = \varphi(X)$ and $X = \varphi'(X)$ be systems of language equations corresponding to G and G' respectively.*

Let $L = (L_1, \ldots, L_n)$ (where $L_i \subseteq \Sigma^*$ and $n = |N|$) be a vector of languages that equals L^ε modulo $\{\varepsilon\}$. Let $L' = (L_1 \setminus \{\varepsilon\}, \ldots, L_n \setminus \{\varepsilon\})$. Then $\varphi(L) = \varphi'(L') \pmod{\Sigma^+}$.

PROOF. It has to be proved that for every nonempty string w and for every t-th component, $w \in \varphi_t(L)$ if and only if $w \in \varphi'_t(L')$.

$w \in \varphi_t(L)$ if and only if there exists a rule of the form (14) for the t-th nonterminal in the original grammar, such that $w \in \alpha_i(L)$ for all i and $w \notin \beta_j(L)$ for all j. By Lemma 2, this holds if and only if there exists a rule of the form (14) in the original grammar, such that for every i-th positive conjunct there exists $\mu_{ip} \in \rho(\alpha_i)$, such that $w \in \mu_{ip}(L')$, and for every j-th negative conjunct and for every $\nu_{jq} \in \rho(\beta_j)$ it holds that $w \notin \nu_{jq}(L')$. By the construction of the new grammar, this is equivalent to the existence of a rule of the form (15) in the new grammar, such that all of its positive conjuncts and none of its negative conjuncts produce w when evaluated on L'. The latter statement is true if and only if $w \in \varphi'_t(L')$. $\square$

Lemma 4 *Under the conditions of Lemma 3, for every M closed under substring, L is a solution of the first system if and only if L' is a solution of the second system.*

PROOF. $\textcircled{\Rightarrow}$ It is known that $L' = L \pmod{\Sigma^+}$. $L = \varphi(L) \pmod{M}$, because L is a solution of the first system. $\varphi(L) = \varphi'(L') \pmod{\Sigma^+}$ by Lemma 3. All these equalities hold modulo $\Sigma^+ \cap M$ as follows:

$$L' = L = \varphi(L) = \varphi'(L') \pmod{M \setminus \{\varepsilon\}}, \tag{16}$$

which implies that $L' = \varphi'(L') \pmod{M \setminus \{\varepsilon\}}$. Since L' is an ε-free vector by definition, while $\varphi'(L')$ is ε-free because every φ'_i is a disjunction of expressions (15) each containing a negative epsilon conjunct, in follows that $L' = \varphi'(L') \pmod{\{\varepsilon\}}$ as well, and one can conclude that L' is a solution of $X = \varphi'(X)$ modulo M.

$\textcircled{\Leftarrow}$ Again, $L = L' \pmod{\Sigma^+}$. Since L' is a solution of the second system modulo M, $L' = \varphi'(L') \pmod{M}$. By Lemma 3, $\varphi(L) = \varphi'(L') \pmod{\Sigma^+}$. This implies

$$L = L' = \varphi'(L') = \varphi(L) \pmod{M \setminus \{\varepsilon\}} \tag{17}$$

On the other hand, $L = L^\varepsilon \pmod{\{\varepsilon\}}$, and therefore L satisfies the first system modulo $\{\varepsilon\}$. This proves that L is a solution of the first system modulo M. $\square$

Theorem 5 *For every Boolean grammar $G = (\Sigma, N, P, S)$ compliant to the semantics of the unique solution in the strong sense there exists and can be effectively constructed a Boolean grammar G' compliant to the semantics of the unique solution in the strong sense, such that $L(G') = L(G) \setminus \{\varepsilon\}$.*

PROOF. Let L^ε be the unique solution of the system corresponding to G modulo $\{\varepsilon\}$, and construct the grammar G' out of G and L^ε as specified above.

Existence of solution. Let $L = (L_1, \ldots, L_n)$ be the unique solution of the system corresponding to G. Then $L = L^\varepsilon \pmod{\{\varepsilon\}}$, $L' = (L_1 \setminus \{\varepsilon\}, \ldots, L_n \setminus \{\varepsilon\})$ is a solution of the second system by Lemma 4, and its first component is $L(G) \setminus \{\varepsilon\}$.

Uniqueness of solution modulo every language. If the second system had two distinct solutions modulo some M, they would differ modulo $M \setminus \{\varepsilon\}$, and by Lemma 4 that would imply that the first system also has two distinct solutions modulo M, which is untrue. $\square$

A similar statement can be proved for the semantics of the naturally reachable solution. First, an auxiliary result.

Lemma 5 *Let the grammar G comply to the semantics of the naturally reachable solution, and let L^ε be this solution modulo $\{\varepsilon\}$. Let G' be constructed out of G and L^ε as specified above. Let M be a finite language closed under substring. Let all proper substrings of $w \notin M$ be in M. Let $L = (L_1, \ldots, L_n)$ ($L_i \subseteq M \cup \{w\}$) be a vector that equals L^ε modulo $\{\varepsilon\}$.*

Then L can be derived (in the sense of Definition 5) from $(L_1 \cap M, \ldots, L_n \cap M)$ with respect to G in i ($i \geqslant 0$) steps if and only if $(L_1 \setminus \{\varepsilon\}, \ldots, L_n \setminus \{\varepsilon\})$ can be derived from $((L_1 \cap M) \setminus \{\varepsilon\}, \ldots, (L_n \cap M) \setminus \{\varepsilon\})$ with respect to G' in i steps.

PROOF. Induction on i. The basis, 0-step derivation, is trivial. Let $X = \varphi(X)$ and $X = \varphi'(X)$ be the systems corresponding to the grammars G and G'. If a vector is derivable with respect to G in $i+1$ steps, it is of the form

$$(L_1, \ldots, \varphi_j(L), \ldots, L_n), \tag{18}$$

where $L = (L_1, \ldots, L_j, \ldots, L_n)$ is derivable with respect to G in i steps. By the induction hypothesis, this implies $L' = (L_1 \setminus \{\varepsilon\}, \ldots, L_j \setminus \{\varepsilon\}, \ldots, L_n \setminus \{\varepsilon\})$ being derivable with respect to G' in i steps. Then

$$(L_1 \setminus \{\varepsilon\}, \ldots, \varphi'_j(L'), \ldots, L_n \setminus \{\varepsilon\}) \tag{19}$$

is derivable with respect to G' in $i+1$ steps. Similarly, the derivability of (19) for G' implies the derivability of (18) for G. $\varepsilon \notin \varphi'_j(L')$ is evident from the construction of φ'. It remains to prove that these two vectors are equal modulo Σ^+: for the components other than j this is given by the induction hypothesis, while $\varphi_j(L) = \varphi'_j(L') \pmod{\Sigma^+}$ by Lemma 3. $\square$

Theorem 6 *For every Boolean grammar $G = (\Sigma, N, P, S)$ compliant to the semantics of the naturally reachable solution there exists and can be effectively constructed a Boolean grammar G' compliant to the semantics of the naturally reachable solution, such that $L(G') = L(G) \setminus \{\varepsilon\}$.*

PROOF. Let $L = (L_1, \ldots, L_n)$ be the naturally reachable solution of G, let L^ε be L taken modulo $\{\varepsilon\}$. Compute L^ε and construct G' with respect to it. It has to be proved that the vector $L' = (L_1 \setminus \{\varepsilon\}, \ldots, L_n \setminus \{\varepsilon\})$ is the naturally reachable solution of G'. The proof is an induction on the cardinality of a finite modulus, as in Definition 5. For $\varnothing$, it clearly holds. Consider an arbitrary finite language M closed under substring, and a string $w \notin M$, such that all proper substrings of w are in M.

$L \pmod{M}$ derives $L \pmod{M \cup \{w\}}$ with respect to G by the assumption. Hence, by Lemma 5, $L' \pmod{M}$ derives $L' \pmod{M \cup \{w\}}$ with respect to G'.

It remains to show that nothing else can be derived with respect to G'. Suppose, $L' \pmod{M}$ derives some $L'' = (L''_1, \ldots, L''_n)$ ($L''_i \subseteq M \cup \{w\}$), such that $L'' \neq L' \pmod{M \cup \{w\}}$. Then, as L'' is clearly ε-free and equal to L' modulo M, $L'' \neq L' \pmod{\{w\}}$. Hence, by Lemma 5, it is possible to derive, with respect to G, some vector not equal to $L \pmod{M \cup \{w\}}$, which contradicts the compliance of G to the semantics of the naturally reachable solution. $\square$

4.2 Unit conjuncts

Conjuncts of the form $A \to B$ and $A \to \neg B$ are called *positive* and *negative unit conjuncts* respectively. They will be collectively referred to as *unit conjuncts*, and our next challenge is to devise an algorithm to get rid of them.

Let $G = (\Sigma, N, P, S)$ be a Boolean grammar compliant to one of the two semantics, such that $\varepsilon \notin L_G(A)$ for every $A \in N$. Let $M \subset \Sigma^*$ be a finite language closed under substring and let $w \notin M$ be a string, such that all of its proper substrings are in M. Let $L = (L_1, \ldots, L_n)$ be a solution of the system modulo M and let $L' = (L_1 \cup L'_1, \ldots, L_n \cup L'_n)$ be a vector, such that $L'_1, \ldots, L'_n \subseteq \{w\}$. Note that the membership of w in $\alpha(L')$ depends on L alone if $\alpha \notin N$ and on $\{L'_i\}$ alone if $\alpha = A \in N$.

Let $R \subseteq (\Sigma \cup N)^* \setminus N$ be a finite set of strings that contains a string $\gamma \notin N$ if and only if there is a conjunct $A \to \gamma$ or $A \to \neg\gamma$ in the grammar. A fixed solution L modulo M defines a certain *assignment to conjuncts*: a mapping $f_{M,w,L} : R \to \{0, 1\}$, such that $f_{M,w,L}(\alpha) = 1$ if and only if $w \in \alpha(L)$. Once the non-unit conjuncts are assigned values dependent on L, what remains is a system of Boolean equations, which contains all the information necessary to determine the membership of w in the solution (unique or naturally reachable) modulo $M \cup \{w\}$ (as can be inferred from Proposition 1 or Definition 5). Let us put this formally.

Definition 7 *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar and let $\varepsilon \notin L_G(A)$ for all $A \in N$. Let M be a finite modulus closed under substring and let all proper substrings of $w \notin M$ be in M. Let L be a solution modulo M of the system corresponding to G.*

Define R (with respect to G) and $f : R \to \{0, 1\}$ (with respect to M, L and

w) as above. Take the set of Boolean variables $x = (x_1, \ldots, x_k)$ ($k = |N|$), and define $f' : R \cup N \to \{0, 1, x_1, \ldots, x_k\}$ as follows: $f'(\alpha) = f(\alpha)$ for $\alpha \in R$ and $f'(X_i) = x_i$.

Now define a system $x_i = \varphi_i^f(x)$ ($1 \leqslant i \leqslant k$) of k Boolean equations: for the right hand side $\varphi_i = \bigvee(\alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n)$ of every equation in the system of language equations corresponding to G, let φ_i^f in the new system of Boolean equations be $\bigvee(f'(\alpha_1) \& \ldots \& f'(\alpha_m) \& \neg f'(\beta_1) \& \ldots \& \neg f'(\beta_n))$.

Proving the following two statements is just the matter of reformulating the notation.

Lemma 6 *Let G, M, w, L, f and $x = \varphi^f(x)$ be as in Definition 7, let $b = (b_1, \ldots, b_k)$ be a Boolean vector. Then $L_b = (L_1 \cup \{w \mid \text{if } b_1 = 1\}, \ldots, L_k \cup \{w | \text{if } b_k = 1\})$ is a solution of $X = \varphi(X)$ if and only if b is a solution of $x = \varphi^f(x)$.*

Lemma 7 *Let G, M, w, L, f and $x = \varphi^f(x)$ be as in Definition 7, let $b = (b_1, \ldots, b_k)$ be a Boolean vector. Then $L_b = (L_1 \cup \{w \mid \text{if } b_1 = 1\}, \ldots, L_k \cup \{w | \text{if } b_k = 1\})$ can be derived from L using the method of Definition 5 if and only if b can be derived from $(0, \ldots, 0)$ using the following rule: $x = (x_1, \ldots, x_i, \ldots, x_k)$ can be followed by $x' = (x_1, \ldots, \varphi_i^f(x), \ldots, x_k)$, provided that $\varphi_i^f(x) = \neg x_i$.*

Now, given G, let us construct an equivalent grammar free of unit conjuncts. The idea of the construction is to precompute processing of unit conjuncts for every possible assignment to non-unit conjuncts. For every assignment to conjuncts $f : R \to \{0, 1\}$ (there are $2^{|R|}$ such assignments), determine the unique Boolean vector using the method of Lemma 6 or Lemma 7. If the method fails – i.e., none or multiple solutions are found, – then, taking into account that the grammar is assumed to be compliant to the chosen semantics, this means that this situation is artificial and could never happen on a real modulus M and string w. If the method succeeds and produces a set $N' \subseteq N$ of nonterminals that evaluate to true, then for every nonterminal $A \in N'$ construct a rule

$$A \to \mu_1 \& \ldots \& \mu_k \& \neg\nu_1 \& \ldots \& \neg\nu_l, \tag{20}$$

which lists *all* strings from R, where $f(\mu_1) = \ldots = f(\mu_k) = 1$ and $f(\nu_1) = \ldots = f(\nu_l) = 0$. Note that the set R is common to both grammars.

Lemma 8 *Let G be a Boolean grammar compliant to the semantics of the unique solution in the strong sense (naturally reachable solution, resp.). Let M, w and f be as in Definition 7. Let G' be as constructed above. Let $X = \varphi(X)$ and $X = \psi(X)$ be the systems corresponding to G and G' respectively. Let L be the unique (naturally reachable, resp.) solution modulo M of $X = \varphi(X)$.*

Then the system $x = \psi^f(x)$ is of the form $x_i = b_i$ ($1 \leqslant i \leqslant k$, $b_i \in \{0, 1\}$), and the Boolean vector $b = (b_1, \ldots, b_k)$ is the solution of $x = \varphi^f(x)$ (unique or reachable *in the sense of Lemma 7, resp.).*

PROOF. The system $x = \varphi^f(x)$ has unique solution $(b_1, \dots b_k)$ by Lemma 6 (reachable solution by Lemma 7, resp.). One has to prove that $\psi_i^f \equiv b_i$ for all i.

Let $b_i = 1$ for the i-th nonterminal A. Then, by the construction of G', there is a rule (20), such that $f(\mu_1) = \dots = f(\mu_k) = 1$ and $f(\nu_1) = \dots = f(\nu_l) = 0$. Hence, $\psi_A^f = (1\& \dots \& 1 \& \neg 0 \& \dots \& \neg 0) \vee \dots = 1$.

If $b_i = 0$, then each rule (20) for A is constructed with respect to some $g \not\equiv f$. Since $\{\mu_i, \nu_j\}$ is the exhaustive list of the common domain of f and g, $f \not\equiv g$ implies $f(\mu_i)$=0 or $f(\nu_j) = 1$ for some i, j and thus the disjunct corresponding to (20) is 0. The overall formula is $\psi_A^f = (0\& \dots) \vee \dots \vee (0 \& \dots) = 0$. $\square$

Theorem 7 *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar that generates an ε-free vector of languages under the semantics of unique solution in the strong sense* ***or*** *under the semantics of naturally reachable solution. Then there exists and can be effectively constructed a Boolean grammar $G' = (\Sigma, N, P', S)$ without unit conjuncts, such that G' generates L under* ***both*** *semantics.*

PROOF. Construct the grammar as above. Let $X = \varphi(X)$ and $X = \psi(X)$ be the systems corresponding to G and G' respectively. It is claimed that the solution (unique in the strong sense or naturally reachable) of $X = \varphi(X)$ is both the unique solution in the strong sense and the naturally reachable solution of $X = \psi(X)$. The proof is an induction on the cardinality of a modulus, trivially true for $\varnothing$.

Consider the modulus $M \cup \{w\}$, where all proper substrings of w are in M. Let $L = (L_1, \dots, L_k)$ be the unique (naturally reachable, resp.) solution of $X = \varphi(X)$ modulo M, and define $f : R \to \{0, 1\}$ with respect to M, w and L. By Lemma 6 (Lemma 7, resp.), the unique (naturally reachable, resp.) solution of $X = \varphi(X)$ modulo $M \cup \{w\}$ is of the form $L_b = (L_1 \cup \{w | \text{if } b_1 = 1\}, \dots, L_k \cup \{w | \text{if } b_k = 1\})$, where the Boolean vector $b = (b_1, \dots, b_k)$ is the unique (reachable in the sense of Lemma 7, resp.) solution of $x = \varphi^f(x)$.

According to Lemma 8, the system $x = \psi^f(x)$ is of the form $x_i = b_i$ ($1 \leqslant i \leqslant k$), and thus b is its unique solution and is easily seen to be naturally reachable. Therefore, by Lemma 6, L_b is the unique solution of $X = \psi(X)$ modulo $M \cup \{w\}$; similarly, by Lemma 7, the definition of naturally reachable solution is satisfied for $X = \psi(X)$, M, w and the naturally reachable solution of $X = \varphi(X)$. $\square$

Corollary 1 *The classes of languages defined by Boolean grammars under the semantics of the unique solution in the strong sense and under the semantics of the naturally reachable solution coincide.*

4.3 *The binary normal form*

Finally we come to a generalization of the context-free Chomsky normal form.

Definition 8 *A Boolean grammar $G = (\Sigma, N, P, S)$ is said to be in the binary normal form if every rule in P is of the form*

$$A \to B_1C_1\& \ldots \&B_mC_m\&\neg D_1E_1\& \ldots \&\neg D_nE_n\&\neg\varepsilon \quad (m \geqslant 1, n \geqslant 0) \tag{21a}$$

$$A \to a \tag{21b}$$

$$S \to \varepsilon \quad \text{(only if } S \text{ does not appear in right hand sides of rules)} \tag{21c}$$

As in the context-free and conjunctive case [21], the transformation of a grammar to the binary normal form can be carried out by first removing the epsilon conjuncts, then eliminating the unit conjuncts, cutting the bodies of the "long" conjuncts by adding extra nonterminals, and, if the original grammar generated the empty string, by adding a new start symbol with the rule (21c).

Theorem 8 *For every Boolean grammar $G = (\Sigma, N, P, S)$ that generates some language L under some of the two given semantics, there exists and can be effectively constructed a Boolean grammar $G' = (\Sigma, N', P', S')$ in the binary normal form that generates this language L under both semantics.*

Note that the effectiveness of the transformation does not extend to checking whether the original grammar actually complies to the semantics. The latter property, as we already know, is undecidable.

5 Recognition and parsing

5.1 *A cubic-time algorithm*

Binary normal form allows to devise a cubic-time recognition and parsing algorithm for every language generated by a Boolean grammar. This algorithm is an extension of a similar algorithm for conjunctive grammars [21], which in turn generalizes the well-known Cocke–Kasami–Younger algorithm [36] for context-free grammars in Chomsky normal form. The idea is to compute the sets of nonterminals deriving all substrings of the input string, starting from the shorter substrings, continuing with the longer ones, and ending with the whole string.

Let $G = (\Sigma, N, P, S)$ be a Boolean grammar in the binary normal form. Let $w =$

$a_1 \dots a_n \in \Sigma^+$ ($n \geqslant 1$) be the input string. For all $0 \leqslant i < j \leqslant n$, define:

$$T_{i,j} = \{A \mid A \in N,\ a_{i+1} \dots a_j \in L_G(A)\} \tag{22}$$

The actual task is to determine whether $S \in T_{0,n}$. For this purpose, the algorithm computes all $T_{i,j}$ starting from $T_{0,1}, \dots, T_{n-1,n}$ and ending with $T_{0,n}$, where every $T_{i,j}$ is obtained out of $T_{i,k}, T_{k,j}$ ($i < k < j$).

The following result is instrumental in reducing problems to subproblems:

Lemma 9 *For all i, j ($j - i \geqslant 2$), $a_{i+1} \dots a_j \in L(B)L(C)$ if and only if*

$$(B, C) \in \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j} \tag{23}$$

PROOF. $a_{i+1} \dots a_j \in L(B)L(C)$ if and only if there exists a number k ($i \leqslant k \leqslant j$), such that $a_{i+1} \dots a_k \in L(B)$ and $a_{k+1} \dots a_j \in L(C)$; by (22), this is equivalent to $B \in T_{i,k}$ and $C \in T_{k,j}$, or $(B, C) \in T_{i,k} \times T_{k,j}$. Since $\varepsilon \notin L(B)$ and $\varepsilon \notin L(C)$, the statement holds if and only if there is k ($i < k < j$), such that $(B, C) \in T_{i,k} \times T_{k,j}$, which is equivalent to (23). $\square$

Algorithm 1 *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar in the binary normal form. For each $R \subseteq N \times N$ denote*

$$\begin{aligned} f(R) = \{A \mid A \in N,\ &\text{there exists a “long” rule (21a), such that} \\ &(B_s, C_s) \in R \text{ and } (D_t, E_t) \notin R \text{ for all } s, t\ (1 \leqslant s \leqslant m,\ 1 \leqslant t \leqslant n)\} \end{aligned} \tag{24}$$

Let $w = a_1 \dots a_n \in \Sigma^*$ ($n \geqslant 1$) be the input string. For all i, j, such that $0 \leqslant i < j \leqslant n$, compute $T_{i,j}$.

```
for $i = 1$ to $n$
    $T_{i-1,i} = \{A \mid A \to a_i \in P\}$
for $d = 1$ to $n$
    for $i = 0$ to $n - d$
    {
        let $j = i + d$
        let $R = \varnothing$ ( $R \subseteq N \times N$ )
        for $k = i + 1$ to $j - 1$
            $R = R \cup T_{i,k} \times T_{k,j}$
        $T_{i,j} = f(R)$
    }
```

To compute each $T_{i,j}$, the algorithm does *conjunct gathering*, i.e., collects the data for checking (23); this is done in the inner loop, and the set of pairs is accumulated in R. Then (24) is computed in constant time. Since there are $O(n^2)$ entries to compute, the time complexity is thus $O(n^3)$.

Theorem 9 *Algorithm 1 is correct, i.e., every assignment to a variable $T_{i,j}$ assigns the value (22).*

PROOF. $a \in L(A)$ if and only if there is the rule $A \to a$; none of the long rules (21a) can produce a, because $\varepsilon \notin L(B_1)$ and $\varepsilon \notin L(C_1)$, and hence there are no strings of length 1 in $L(B_1C_1)$.

$a_{i+1} \ldots a_j \in L(A)$ if and only if there exists a rule (21a), such that $a_{i+1} \ldots a_j \in L(B_s)L(C_s)$ for all $1 \leqslant s \leqslant m$ and $a_{i+1} \ldots a_j \notin L(D_t)L(E_t)$ for all $1 \leqslant t \leqslant n$. By Lemma 9, this is equivalent to $(B_s, C_s) \in \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$ and $(D_t, E_t) \notin \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$ respectively.

For all i, j, the algorithm computes $R = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$. In this notation, the statement can be equivalently rewritten as follows: there exists a rule (21a), such that $(B_s, C_s) \in R$ for all s ($1 \leqslant s \leqslant m$) and $(D_t, E_t) \notin R$ for all t ($1 \leqslant t \leqslant n$). Using the notation (24), this holds if and only if $A \in f(R)$. Thus the only assignment to $T_{i,j}$ sets it precisely to the set of nonterminals A, such that $a_{i+1} \ldots a_j \in L(A)$. $\square$

Once this algorithm determines that a string is in the language, the table $T_{i,j}$ can be used to construct its parse tree (as defined in Section 3.2). This is done in the same way as for the Cocke–Kasami–Younger algorithm: a recursive procedure $parse(\text{int } i, \text{int } j, A \in N)$ is defined, which, assuming that $A \in T_{i,j}$, constructs the parse tree of $a_{i+1} \ldots a_j \in L_G(A)$ and returns a pointer to the root. Then a call to $parse(0, n, S)$ gives a parse tree of the whole input string.

5.2 Recognition in linear space

Algorithm 1 uses space $O(n^2)$, as does its context-free prototype, and in both cases this is the best known upper bound for practical universal algorithms. However, in the context-free case it is possible to trade time for space and use as little as $O(\log^2 n)$ memory. How little space would be enough to recognize the languages generated by Boolean grammars?

Constructing a recognizer that would use $C \cdot n \log n$ space is straightforward: Algorithm 1 can be modified to use a recursive procedure $T(i, j, A)$ instead of the dynamic programming table $T_{i,j}$; in this case the depth of recursion is at most n, while each instance of the procedure requires $O(\log n)$ bits to store its variables that range over positions in the input. In this section, using a much more involved technique based upon a formal term-rewriting system, an $O(n)$ upper bound for space is established; this also demonstrates inclusion in the deterministic context-sensitive languages.

Let $G = (\Sigma, N, P, S)$ be a Boolean grammar in the binary normal form. Assume that P is linearly ordered. Let P' be the set of "long" rules of G with marked conjuncts, i.e., $P' = \{p_k \mid p \text{ is of the form (21a)}, 1 \leqslant k \leqslant m + n\}$. Let $N^+ = \{A^+ \mid A \in N\}$ and $N^- = \{A^- \mid A \in N\}$. For every $p = A \to \varphi \in P$, denote $L(p) = L_G(\varphi)$; if the k-th conjunct of p is $A \to \psi$ (where $\psi = BC$ or $\psi = \neg BC$), denote $L(p_k) = L_G(\psi)$.

Terms are defined over the alphabet $\Sigma \cup N \cup N^+ \cup N^- \cup P' \cup \{\text{"("}, \text{")"}\}$ as follows:

- For every $u \in \Sigma^+$ and $A \in N$, the following are terms (for A): $A(u)$, $A^+(u)$, $A^-(u)$.
- If p_k is a conjunct ($A \to BC$ or $A \to \neg BC$) and t_1, t_2 are terms (for B and C respectively), then $p_k(t_1 t_2)$ is a term (for A).

Define *the string value* $\sigma(t)$ of a term t as

- $\sigma(A(u)) = \sigma(A^+(u)) = \sigma(A^-(u)) = u$ for all $u \in \Sigma^+$.
- $\sigma(p_k(t_1 t_2)) = \sigma(t_1) \cdot \sigma(t_2)$.

Define the notion of a *true term*:

- $A(u)$ is always true. $A^+(u)$ is true if and only if $u \in L(A)$. $A^-(u)$ is true if and only if $u \notin L(A)$.
- $t = p_k(t_1 t_2)$ is true if and only if all of the following conditions hold:
  - (I) both subterms t_1 and t_2 are true;
  - (II) for every "long" rule r for A (where A is the nonterminal on the left hand side of r) that precedes p it holds that $\sigma(t) \notin L(r)$;
  - (III) for every conjunct p_i of the rule p that precedes p_k (i.e., $i < k$) it holds that $\sigma(t) \in L(p_k)$;
  - (IV) for every factorization $\sigma(t_1)\sigma(t_2) = uv$, such that $0 < |u| < |\sigma(t_1)|$ it holds that $u \notin L(B)$ or $v \notin L(C)$, where p_k is $A \to BC$ or $A \to \neg BC$.

Define the following set of *rewriting rules*:

(1) A term $A(a)$, where $a \in \Sigma$, is rewritten with $A^+(a)$ if $A \to a \in P$, with $A^-(a)$ otherwise.

(2) A term $A(u)$, where $u = a_1 \dots a_m$ and $m \geqslant 2$, is rewritten with $p_1(B(a_1)C(a_2 \dots a_m))$, where p is the first rule for A, and its first conjunct is $A \to BC$ or $A \to \neg BC$.

(3) A term $p_k(B^+(u)C^+(v))$ is rewritten as follows:
- If the k-th conjunct of p_k is positive ($A \to BC$), then: (*)
  - · If the k-th conjunct is the last, rewrite with $A^+(uv)$;
  - · If there are more conjuncts, rewrite with $p_{k+1}(D(a_1)E(a_2 \dots a_m))$, where $uv = a_1 a_2 \dots a_m$ and the $(k+1)$-th conjunct is $A \to DE$ or $A \to \neg DE$.
- Else, if the k-th conjunct of p_k is negative ($A \to \neg BC$), then: (**)

· If p is the last rule for A, rewrite with $A^-(uv)$;
· If r is the next rule for A after p, rewrite with $r_1(D(a_1)E(a_2 \ldots a_m))$, where $uv = a_1 a_2 \ldots a_m$ and the first conjunct of r is $A \to DE$ or $A \to \neg DE$.

(4) Any of the terms $p_k(B^+(u)C^-(v))$, $p_k(B^-(u)C^+(v))$ or $p_k(B^-(u)C^-(v))$ is rewritten as follows:

- If $|v| > 1$, let $v = ax$ ($a \in \Sigma$, $x \in \Sigma^+$) and rewrite with $p_k(B(ua)C(x))$.
- If $|v| = 1$ and the k-th conjunct is positive, then do as in (**) above.
- If $|v| = 1$ and the k-th conjunct is negative, do as in (*) above.

Claim 1 *The rewriting preserves truth, i.e., a true term is rewritten with a true term.*

In order to prove Claim 1, each case of rewriting has to be examined. Let us give a complete treatment of one of these numerous cases. Consider the rewriting of

$$p_k(B^-(a_1 \ldots a_{m-1})C^+(a_m)) \tag{25}$$

with

$$r_1(D(a_1)E(a_2 \ldots a_m)), \tag{26}$$

where the conjunct p_k is positive ($A \to BC$) and r is the next rule for A. Since (25) is true, the condition (IV) implies that $y \notin L(B)$ or $z \notin L(C)$ for all factorizations $a_1 \ldots a_m = yz$ ($y, z \in \Sigma^+$), such that $|y| < m - 1$. The condition (I) gives that $a_1 \ldots a_{m-1} \notin L(B)$. Putting these together, $y \notin L(B)$ or $z \notin L(C)$ for *every* factorization $a_1 \ldots a_m = yz$ ($y, z \in \Sigma^+$). Therefore, $a_1 \ldots a_m \notin L(BC)$.

The conjunct p_k is positive, and so $a_1 \ldots a_m \notin L(p_k)$. Therefore, $a_1 \ldots a_m \notin L(p)$. By the condition (II) for the true term (25), $a_1 \ldots a_m \notin L(q)$ was the case for all rules q that precede p. Combining this with the now known failure of p, $a_1 \ldots a_m \notin L(q)$ for every rule q for A that precedes r, the immediate successor of p.

Hence, the condition (II) for the correctness of (26) is fulfilled. The conditions (III) and (IV) are fulfilled simply because r_1 is the first conjunct of r and $a_1 \cdot (a_2 \ldots a_m)$ is the first factorization of the string. The subterms of (26) are true by definition, which completes the proof of this case. The rest of the cases are proved – indeed – similarly.

Claim 2 *The rewriting preserves string value of terms.*

Claim 3 *Every term with string value w contains at most $7|w| - 3$ symbols.*

These two properties are easily verifiable: Claim 2 can be directly observed from the rewriting rules above, while Claim 3 is checked by a simple induction on the structure of a term.

Claims 1–3 together imply that a rewriting that starts from a true term goes over true terms of linearly bounded size. In order to show that the rewriting eventually terminates and the final (true) term confers all relevant information, define a linear order on the set of terms. If $\sigma(t_1)$ is lexicographically less than (greater than) $\sigma(t_2)$, then $t_1 < t_2$ ($t_1 > t_2$, resp.). For terms with the same string value w, the general order is $A_1(w) < \ldots < A_m(w) < p_k(t_1t_2) < A_1^+(w) < \ldots < A_m^+(w) < A_1^-(w) < \ldots < A_m^-(w)$. It is left to define the relation between $p_k(t_1t_2)$ and $r_l(t_3t_4)$: if $p \neq r$, then the terms compare by the order on P; if $p = r$ and $k < l$ ($k > l$), then $p_k(t_1t_2) < p_l(t_3t_4)$ ("$>$", resp.); finally, the terms $p_k(t_1t_2), p_k(t_3t_4)$ compare lexicographically as (t_1, t_2) and (t_3, t_4).

The following statement, together with its corollaries, motivates this order:

Claim 4 *The rewriting strictly increases the terms with respect to the given order.*

Claim 5 *Every term is eventually converted to a term of the form $A^+(w)$ or $A^-(w)$.*

Claim 6 *A term $A(w)$ is eventually converted either to $A^+(w)$, or to $A^-(w)$.*

Claims 1 and 6 show that this term rewriting system effectively works as a recognizer for the source grammar. Direct algorithmic simulation of this rewriting yields the following result:

Theorem 10 *Every language generated by a Boolean grammar is deterministic context-sensitive, i.e., is in **DSPACE**(n).*

PROOF. Consider an arbitrary grammar $G = (\Sigma, N, P, S)$ and assume without loss of generality that it is in the binary normal form. Construct the following Turing machine: given a string w,

(1) If $w = \varepsilon$, accept it if $\varepsilon \in L(G)$, reject otherwise.
(2) Write the term $S(w)$ to the tape.
(3) While possible, transform subterms as specified above.
(4) If the resulting term is $S^+(w)$, accept; if it is $S^-(w)$, reject.

The assumption that the resulting term will be either $S^+(w)$ or $S^-(w)$ is valid by Claim 6. Since the initial term $S(w)$ is true, and, according to Claim 1, truth is being preserved in course of the rewriting, the resulting term $S^+(w)$ or $S^-(w)$ must also be true, which allows to make a justified conclusion on whether $w \in L_G(S) = L(G)$ or $w \notin L(G)$.

Observing that all terms have string value w, each of them fits into $7|w| - 3$ symbols by Claim 3. Hence the constructed TM can be converted to a deterministic LBA by compressing seven symbols into one, which means that the language is deterministic context-sensitive. $\square$

6 Linear Boolean grammars

6.1 Definition and a normal form

Linear context-free grammars are an important and a much-studied subclass of context-free grammars. The concatenation in their rules is restricted to linear, i.e., all rules are of the form $A \to uBv$ or $A \to w$. For conjunctive grammars, the subclass of linear conjunctive grammars [21,24] is defined by similarly restricting the rules to be $A \to u_1B_1v_1\& \ldots \& u_mB_mv_m$ or $A \to w$; this class is worth particular interest due to its recently discovered equivalence [26] to trellis automata [9,10,15].

It makes sense to try to restrict Boolean grammars in a similar way and to see what comes out of it.

Definition 9 *A Boolean grammar $G = (\Sigma, N, P, S)$ is said to be linear if every rule in P is of the form $A \to u_1B_1v_1\& \ldots \& u_mB_mv_m \& \neg x_1C_1y_1\& \ldots \& \neg x_nC_ny_n$, where $B_i, C_j \in N$, $u_i, v_i, x_jy_j \in \Sigma^*$ and $m + n \geqslant 1$, or $A \to w$, where $w \in \Sigma^*$.*

The following normal form is a natural generalization of linear normal forms for linear context-free and linear conjunctive grammars.

Definition 10 *A linear Boolean grammar $G = (\Sigma, N, P, S)$ is said to be in the linear normal form if every rule in P is of the form*

$$\begin{aligned} A \to bB_1\& \ldots bB_m \& C_1c\& \ldots \& C_nc \& \neg bD_1 \& \ldots \& \neg bD_k \& \\ \& \neg E_1c \& \ldots \& \neg E_lc \ (m, n, k, l \geqslant 0;\ m + n \geqslant 1), \end{aligned} \quad (27a)$$

$$A \to a \quad (27b)$$

$$S \to \varepsilon \quad \text{(only if } S \text{ does not appear in right hand sides of rules)} \quad (27c)$$

The result on effective transformation to linear normal form can be proved very much like Theorem 8, if one notes that the transformations given in Sections 4.1 and 4.2 preserve the linearity of a grammar.

Theorem 11 *For every linear Boolean grammar G that generates some language L under some of the two given semantics, there exists and can be effectively constructed a linear Boolean grammar G' in the linear normal form that generates the same language L under both semantics.*

Using linear normal form, linear Boolean grammars can be completely characterized: it turns out that, by an effective transformation almost identical to the one applicable to linear conjunctive grammars [26], every given Boolean grammar in the linear normal form can be converted to an equivalent trellis automaton [9]. Let us first give a short introduction to trellis automata.

6.2 Trellis automata

Trellis automata [9] were introduced by Culik II, Gruska and A. Salomaa as a model of a massively parallel computer with simple identical processors connected in a uniform pattern. In the most common type of trellis automata, called *homogeneous*, *triangular* or *real-time*, the connections between nodes form a figure of triangular shape, such as the one shown in Figure 2.

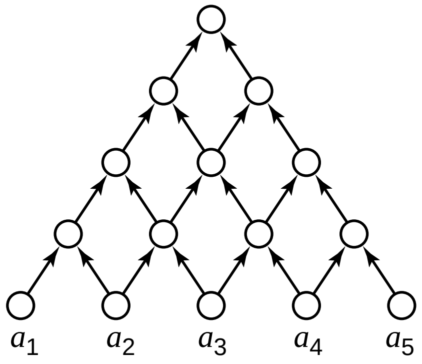

Fig. 2. Computation of a trellis automaton.

Trellis automata are used as acceptors of strings loaded from the bottom, and the acceptance is determined by the topmost element. In their original definition they cannot handle the empty string, because a triangular trellis, as in Figure 2, cannot be of size zero. However, as other models equivalent to trellis automata [15,26] do not have problems of this kind, it becomes natural to augment trellis automata with an unsophisticated means to accept or reject the empty string:

Definition 11 ([26]) *A trellis automaton (TA) is a sextuple $M = (\Sigma, Q, I, \delta, F, e)$, where Σ is the input alphabet, Q is a finite nonempty set of states, $I : \Sigma \to Q$ is a function that sets the initial states, $\delta : Q \times Q \to Q$ is the transition function, $F \subseteq Q$ is the set of final states, and the bit $e \in \{0, 1\}$ determines whether ε is accepted or rejected.*

Given a nonempty string $a_1 \ldots a_n$ ($a_i \in \Sigma$, $n \geqslant 1$), every node of a trellis corresponds to a certain substring $a_i \ldots a_j$ ($1 \leqslant i \leqslant j \leqslant n$) of symbols on which its value depends. The value of a bottom node corresponding to one symbol of the input is $I(a_i)$; the value of a successor of two nodes is δ of the values of these ancestors. Denote the value of a node corresponding to $a_i \ldots a_j$ as $\Delta(I(a_i \ldots a_j)) \in Q$: here $I(a_i \ldots a_j)$ is a string of states (the bottom row of the trellis), while Δ denotes the result (a single state) of a computation starting from a row of states. By definition, $\Delta(I(a_i)) = I(a_i)$ and $\Delta(I(a_i \ldots a_j)) =$

$\delta(\Delta(I(a_i \dots a_{j-1})), \Delta(I(a_{i+1} \dots a_j)))$. Define the language recognized by M as

$$L(M) = \{w \in \Sigma^+ \mid \Delta(I(w)) \in F\} \cup \{\varepsilon \mid \text{if } e = 1\} \tag{28}$$

Once introduced [9], trellis automata were quickly noted to be isomorphic to one-way real-time cellular automata [6], and a characterization in terms of sequential machines was found [15]. Two decades later linear conjunctive grammars were proved to generate the same family of languages [26], and now one more language specification formalism joins this company.

6.3 The equivalence result

Let $G = (\Sigma, N, P, S)$ be a linear Boolean grammar in the linear normal form, and let us show how an equivalent TA can be constructed. Using subset construction similar to the one used for linear conjunctive grammars [26], construct the TA $M = (\Sigma, Q, I, \delta, F, e)$, where $Q = \Sigma \times 2^N \times \Sigma$ and

$$I(a) = (a, \{A \mid A \to a \in P\}, a), \tag{29a}$$

$$\delta((b, Q, b'), (c', R, c)) = (b, \{A \mid \text{there is a rule (27a), such that } B_i \in R, C_j \in Q, D_s \notin R, E_t \notin Q \text{ for all } i, j, s, t\}, c), \tag{29b}$$

$$F = \{(a, R, b) \mid S \in R\}, \tag{29c}$$

while e equals 1 if $\varepsilon \in L(G)$, $e = 0$ otherwise. The correctness of this construction is stated in the following lemma:

Lemma 10 *Let $w \in \Sigma^+$ and let $\Delta(I(w)) = (b, R, c)$. Then the first symbol of w is b, the last symbol of w is c, and for each nonterminal $A \in Q$, $w \in L_G(A)$ if and only if $A \in R$.*

Lemma 10 can be proved similarly to the corresponding statement for linear conjunctive grammars [26, Lemma 1]. Then, using (29c), it is easy to see that, for every $w \in \Sigma^+$, $\Delta(I(w)) \in F$ if and only if $w \in L_G(S) = L(G)$. The case of the empty string follows from the construction of e, showing that the automaton is completely equivalent to the grammar.

Theorem 12 *For every linear Boolean grammar $G = (\Sigma, N, P, S)$ compliant to any of the two semantics, there exists and can be effectively constructed a trellis automaton $M = (\Sigma, Q, I, \delta, F, e)$, such that $L(M) = L(G)$.*

So, linear Boolean grammars can be simulated by trellis automata, while trellis automata can be in turn simulated by linear conjunctive grammars [26], which form a subclass of linear Boolean grammars. This brings us to the following conclusion:

Theorem 13 *The family of languages generated by linear Boolean grammars coincides with the family generated by linear conjunctive grammars and the family* *recognized by trellis automata.*

Although this equivalence result means that linear Boolean grammars give nothing new in terms of generative power in comparison with linear conjunctive grammars, the additional operation they offer makes them more convenient as a practical tool for specifying formal languages. It has been proposed that the relationship between trellis automata and linear conjunctive grammars resembles that between finite automata and regular expressions [26]. Perhaps linear Boolean grammars can be compared to extended regular expressions [37] then.

7 General properties

Let us summarize the basic properties of Boolean grammars and compare them with some other language specification formalisms – namely, with finite automata (Reg), linear context-free grammars (LinCF), context-free grammars (CF), linear conjunctive grammars (LinConj), conjunctive grammars (Conj) and context-sensitive grammars (CS). The results are put together in Table 1.

It can be proved by a straightforward construction that the language family denoted by Boolean grammars is closed under all set-theoretic operations, concatenation, reversal and star. It is not closed under homomorphism, because homomorphic images of linear conjunctive languages already constitute all recursively enumerable sets [10,24]. The closure under inverse homomorphism is left as an open problem; a positive answer can be conjectured.

Turning to the decidability results, the membership problem for Boolean grammars can be solved according to Theorem 8 and Algorithm 1, provided that the given grammar complies to one of the two semantics (which property, however, cannot be effectively decided). Formally, the membership problem for *verified representations*, i.e., for Boolean grammars with attached axiomatic proofs of compliance to one of the semantics, can be called decidable in the most rigorous sense. On the other hand, most other decision problems, such as emptiness or equivalence, are undecidable already for linear conjunctive grammars [24].

Every language generated by a Boolean grammar can be recognized in time $O(n^3)$ using Algorithm 1; this is the same as in the best known algorithms for conjunctive grammars [21]. On the other hand, for context-free grammars, using Valiant's algorithm [33] together with the matrix multiplication method of Coppersmith and Winograd [8] allows to obtain an asymptotically better result: $O(n^{2.376})$. It seems that this method cannot be generalized for conjunctive and Boolean grammars, since Algorithm 1, despite all its similarity to Cocke–Kasami–Younger, can no longer be reduced to computing the transitive closure of a matrix.

	Reg	LinCF	CF	LinConj	Conj	Bool	CS
Closure properties							
$\cup$	$+$	$+$	$+$	$+$	$+$	$+$	$+$
$\cap$	$+$	$-$	$-$	$+$	$+$	$+$	$+$
$\sim$	$+$	$-$	$-$	$+$	$?$	$+$	$+$
$\cdot$	$+$	$-$	$+$	$-$	$+$	$+$	$+$
$*$	$+$	$-$	$+$	$-$	$+$	$+$	$+$
h	$+$	$+$	$+$	$-$	$-$	$-$	$-$
h^{-1}	$+$	$+$	$+$	$+$	$+$	$?$	$+$
Decision problems							
Membership	$+$	$+$	$+$	$+$	$+$	$+$	$+$
Emptiness	$+$	$+$	$+$	$-$	$-$	$-$	$-$
Equivalence	$+$	$-$	$-$	$-$	$-$	$-$	$-$
Complexity of recognition							
Lower bound		**NL**[31]	**NL**[31]	**P**[15]	**P**	**P**	**PSPACE**
Upper bound		**NL**	**NC**^{2}[29,5,30]	**P**[9,21]	**P**[21]	**P**	**PSPACE**
Known algorithm	n	n^2[36]	$n^{2.376}$[33,8]	n^2[21]	n^3[21]	n^3	C^n

Table 1
Basic properties of Boolean grammars, compared to other classes.

Every language generated by a Boolean grammar is in **P**. Since already linear conjunctive grammars can denote **P**-complete languages [15], this **P** upper bound for complexity is tight. These complexity results put Boolean grammars, together with conjunctive and linear conjunctive grammars, into the single class of formalisms of "medium complexity". Context-free languages are much easier, as they are all in **NC**^{2} (demonstrated by the Brent–Goldschlager–Rytter parsing algorithm [5,30] that works in parallel time $O(\log^2 n)$ on polynomially many processors), while the hardest known language is in **NL** [31]. On the other hand, context-sensitive languages are considerably harder, as there are **PSPACE**-complete languages among them. In Figure 3 these classes are grouped according to their complexity.

Let us discuss the containment of these families of languages in each other, shown by arrows in Figure 3. It is known that context-free and linear conjunctive languages are incomparable subsets of conjunctive languages [32,26], which is denoted in the figure by a dotted line. Every conjunctive grammar is a Boolean grammar by Theorem 3, whence an inclusion, not known to be proper. Every language generated by a Boolean grammar is deterministic context-sensitive by Theorem 10. Hence,

$$\mathcal{L}(Conjunctive) \subseteq \mathcal{L}(Boolean) \subseteq \mathcal{L}(DetCS) \subseteq \mathcal{L}(CS) \qquad (30)$$

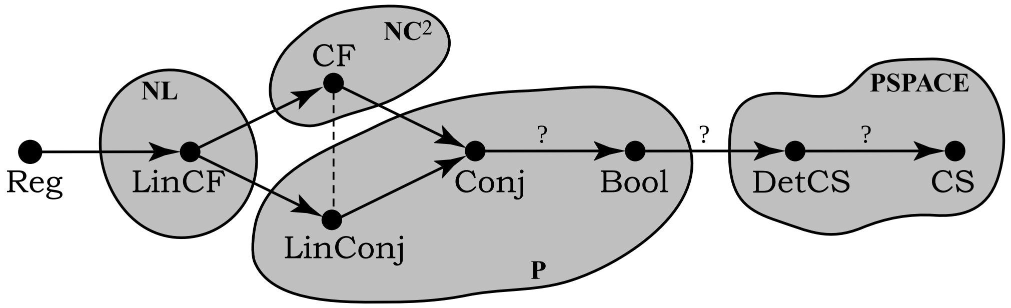

Fig. 3. Hierarchy of language families and its relation to complexity classes.

and whether any of these three inclusions is strict is not known. It is conjectured that $\{ww \mid w \in \{a,b\}^*\}$ could witness the strictness of the first inclusion, while any **PSPACE**-complete language is expected to separate the middle two classes (otherwise **P** would equal **PSPACE**).

It was conjectured before that every unary conjunctive language is regular [21], which would imply proper inclusion in deterministic context-sensitive languages. If this conjecture were proved, it would mean that the first inclusion in (30) is proper, because Boolean grammars can generate some nonregular unary languages (which can be inferred from Example 2). However, no such results for unary conjunctive languages have been obtained so far.

8 Conclusion

Yet another family of formal grammars has been introduced. It offers an extended but still intuitively clear formalism of rules, allows to denote many non-context-free constructs, and at the same inherits a lot of appealing properties of context-free grammars. Giving a logically consistent definition without unjustified simplifications was the most complicated part of the study, especially in light of the previous attempts at incorporating negation in formal grammars, but in the end a well-defined mathematical concept was successfully produced.

To understand Boolean grammars and related formalisms (such as linear conjunctive and conjunctive grammars) better, it would be important to compare them to the most well-known derivation-based formalisms, including indexed, linear indexed and tree-adjoining grammars. In order to do that, and generally for any further study of Boolean grammars, a method for proving particular languages not to belong to this family would be very instrumental. As usual in the field, finding one promises to be a hard problem. For instance, no such method is known even for conjunctive grammars; only for linear conjunctive grammars one can use quite elaborate counting arguments for the trellis automaton representation [32]. Coming up with a method for showing languages to be not generable by Boolean grammars and proving the strictness of the first two inclusions in (30) could be proposed as a worthy theoretical problem.

Another issue is the practical applicability of Boolean grammars, and here one can be reasonably optimistic: the formalism is easy to understand, the class of languages is sufficiently large, the complexity of parsing is low. In light of Algorithm 1, it looks that generalizing some of the parsing algorithms for conjunctive grammars [23] for the case of Boolean grammars is quite feasible. Once done, this will confirm the practical value of the newly introduced concept.

References

[1] A. V. Aho, "Indexed grammars", *Journal of the ACM*, 15:4 (1968), 647–671.

[2] J. Autebert, J. Berstel and L. Boasson, "Context-Free Languages and Pushdown Automata", in [28], 111–174.

[3] Y. Bar-Hillel, M. Perles, E. Shamir, "On formal properties of simple phrase-structure grammars", *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14 (1961), 143–177.

[4] P. Boullier, "A cubic time extension of context-free grammars", *Grammars*, 3 (2000), 111–131.

[5] R. P. Brent, L. M. Goldschlager, "A parallel algorithm for context-free parsing", *Australian Computer Science Communications*, 6:7 (1984) 7.1–7.10.

[6] C. Choffrut, K. Culik II, "On real-time cellular automata and trellis automata", *Acta Informatica*, 21 (1984), 393–407.

[7] N. Chomsky, M. P. Schützenberger, "The algebraic theory of context-free languages", in: Braffort, Hirschberg (Eds.), *Computer Programming and Formal Systems*, North-Holland Publishing Company, Amsterdam, 1963, 118–161.

[8] D. Coppersmith, S. Winograd, "Matrix multiplication via arithmetic progressions", *Journal of Symbolic Computation*, 9:3 (1990), 251–280.

[9] 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.

[10] K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata: stability, decidability and complexity", *Information and Control*, 71 (1986), 218–230.

[11] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[12] M. Domaratzki, personal communication, February 2003.

[13] G. Gazdar, "Applicability of indexed grammars to natural languages", Reyle, Rohrer (Eds.), *Natural Language Parsing and Linguistic Theories*, D. Reidel Publishing Company, 1988, 69–94.

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

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

[16] N. Immerman, *Descriptive complexity*, Springer-Verlag, New York, 1998.

[17] A. K. Joshi, L. S. Levy, M. Takahashi, “Tree adjunct grammars”, *Journal of Computer and System Sciences*, 10:1 (1975), 136–163.

[18] A. Joshi, K. Vijay-Shanker, D. Weir, “The convergence of mildly context-sensitive grammatical formalisms, *Foundational Issues in Natural Language Processing*, MIT Press, 1991.

[19] W. Kuich, “Semirings and Formal Power Series: Their Relevance to Formal Language and Automata”, in [28], 609–677.

[20] E. L. Leiss, “Unrestricted complementation in language equations over a one-letter alphabet”, *Theoretical Computer Science*, 132 (1994), 71–93.

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

[22] A. Okhotin, “Conjunctive grammars and systems of language equations”, *Programming and Computer Software*, 28 (2002), 243–249.

[23] A. Okhotin, “Whale Calf, a parser generator for conjunctive grammars”, *Implementation and Application of Automata* (Proceedings of CIAA 2002, Tours, France, July 3–5, 2002), LNCS 2608, 213–220.

[24] A. Okhotin, “On the closure properties of linear conjunctive languages”, *Theoretical Computer Science*, 299 (2003), 663–685.

[25] A. Okhotin, “Decision problems for language equations with Boolean operations”, *Automata, Languages and Programming* (Proceedings of ICALP 2003, Eindhoven, The Netherlands, June 30–July 4, 2003), LNCS 2719, 239–251.

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

[27] W. C. Rounds, “LFP: a logic for linguistic descriptions and an analysis of its complexity”, *Computational Linguistics*, 14:4 (1988), 1–9.

[28] G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages, Vol. I*, Springer-Verlag, 1997.

[29] W. L. Ruzzo, “On uniform circuit complexity”, *Journal of Computer and System Sciences*, 22:3 (1981), 365–383.

[30] W. Rytter, “On the recognition of context-free languages”, *5th Symposium on Fundamentals of Computation Theory* (1985), LNCS 208, 315–322.

[31] I. H. Sudborough, “A note on tape-bounded complexity classes and linear context-free languages”, *Journal of the ACM*, 22:4 (1975), 499–500.

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

[33] L. G. Valiant, “General context-free recognition in less than cubic time”, *Journal of Computer and System Sciences*, 10 (1975), 308–315.

[34] M. Vardi, “Complexity of relational query languages”, *14th Symposium on Theory of Computation* (1982), 137–146.

[35] K. Vijay-Shanker, D. J. Weir, “The equivalence of four extensions of context-free grammars”, *Mathematical Systems Theory*, 27:6 (1994), 511–546.

[36] D. H. Younger, “Recognition and parsing of context-free languages in time n^3”, *Information and Control*, 10 (1967), 189–208.

[37] S. Yu, “Regular Languages”, in [28], 41–110.