A Tale of Conjunctive Grammars

Alexander Okhotin(✉)

St. Petersburg State University, 7/9 Universitetskaya nab., Saint Petersburg 199034, Russia
alexander.okhotin@spbu.ru

Abstract. Conjunctive grammars are an extension of ordinary ("context-free") grammars with a conjunction operator, which can be used in any rules to specify a substring that satisfies several syntactic conditions simultaneously. This family has been systematically studied since the turn of the century, and is a subject of current studies. This paper gives an overview of the current state of the art in the research on conjunctive grammars.

1 Introduction

Formal grammars are a logic for describing syntactic structures. This logic typically deals with propositions of the general form *"a string w has a property A"*, which associate syntactic categories to phrases forming a sentence. These are propositions such as *"a string* `x+x*x` *is an arithmetical expression"* and *"a string* `Every man is mortal` *is a sentence"*. In this logic, the structure of a string is defined recursively by combining smaller phrases into larger phrases. Phrases are typically substrings, although grammar models using discontinuous fragments also exist. Definitions use logical connectives, such as the choice between alternative syntactic conditions (the disjunction).

Various families of formal grammars differ by the allowed sets of logical connectives, as well as by any further restrictions imposed on the definitions. The most important and natural model is a formal grammar featuring unrestricted concatenation and disjunction, and no other operations. This model is widely known under the name *context-free grammar*, popularized in the famous early work of Chomsky [15], who regarded grammars as string-rewriting systems. However, the rewriting approach does not explain the new developments in the area of formal grammars, and thus the legacy term "context-free grammar" suggests a wrong outlook on this area. For that reason, this paper uses the name ***ordinary grammar***, which reflects the central place of this model in the theory, and is free of undesired connotations [59].

Conjunctive grammars are an extension of ordinary grammars with a conjunction operation. In total, there are three operations: concatenation of strings, disjunction of syntactic conditions, conjunction of syntactic conditions. Historically, the model was first mentioned in an unpublished Master's thesis by

Supported by the Russian Science Foundation, project 18-11-00100.

Szabari [66]. Later, conjunctive grammars were investigated by the author [41]; the same model was independently considered by Boullier [14] and by Lange [37]. As pioneers in the field, one can mention Heilbrunner and Schmitz [20] and Latta and Wall [38], who considered ordinary grammars with Boolean operations on top. An important inspiration for all this work was a 1988 paper by Rounds [65], who explained formal grammars as fragments of the FO(LFP) logic.

The early results on conjunctive grammars include parsing algorithms [41, 43,44,53,56] and an investigation of the expressive power in the unary case [23–29,62]. Recent work on conjunctive grammars includes, in particular, a parsing algorithm based on the generalization of pushdown automata by Aizikowitz and Kaminski [1,3]; algorithms for path queries using conjunctive grammars [21]; a connection with categorial grammars explored by Kuznetsov [34] and by Kuznetsov and Okhotin [35]; a learning algorithm by Clark et al. [16] and by Yoshinaka [72]; a stochastic generalization explored by Domaratzki and Zier-Vogel [74] and by Kanchan Devi and Arumugam [31]; an application to data analysis by Zhang and Su [73].

A few related models inspired by conjunctive grammars were introduced: these are *Boolean grammars* [45], which further include a negation operator, *grammars with context operators* [9,10], and *GF(2)-grammars* [6], which are ordinary grammars redefined in GF(2) logic instead of Boolean logic.

This paper gives an introduction to conjunctive grammars and briefly describes the main results on this model.

2 Definitions

Definition 1. *A* conjunctive grammar *is a quadruple $G = (\Sigma, N, R, S)$, with the following meaning of components.*

- *A finite set of symbols Σ is the alphabet.*
- *There is a finite set of syntactic categories N, called "nonterminal symbols".*
- *In a finite set of rules R, every rule defines a possible form of strings with the property A, as one or more concatenations $\alpha_1, \ldots, \alpha_m$, of zero or more symbols and nonterminal symbols each. These concatenations are called* conjuncts*, and are separated with a conjunction operator.*

$$A \to \alpha_1 \mathbin{\&} \ldots \mathbin{\&} \alpha_m \qquad (A \in N,\ \alpha_1, \ldots, \alpha_m \in (\Sigma \cup N)^*) \tag{1}$$

Each conjunct defines a certain structure for the string, and if a string satisfies all these conditions at the same time, then it is deemed to have the property A.
- *The initial symbol $S \in N$ stands for well-formed sentences of the language.*

If a grammar has a unique conjunct in every rule ($m = 1$), then it is an *ordinary grammar*, and all definitions of conjunctive grammars degenerate to the definitions of ordinary grammars.

Conjunctive grammars are normally constructed in the same way as ordinary grammars, but whenever a conjunction of multiple conditions is needed, it can always be expressed. The simplest use for the conjunction is to express an intersection of two languages, as in the following grammar, which shall be used as a running example for the basic definitions.

Example 1 (Szabari [66]*; see also Okhotin* [41]*).* The following conjunctive grammar describes the language $\{a^n b^n c^n \mid n \geqslant 0\}$.

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

The rules for the nonterminal symbols A, B, C and D do not use conjunction, and have the same meaning as in an ordinary grammar. The rule for S then defines the desired language as the following intersection.

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

As in the case of ordinary grammars, the informal understanding of the meaning of a grammar can be formalized in several equivalent ways.

2.1 Logical Inference

The most natural definition of ordinary grammars uses deduction of propositions of the form "a string w has the property A", denoted by $A(w)$. Each rule of a grammar is regarded as an *inference rule*; for instance, by a rule $S \to \mathrm{NP}\ \mathrm{VP}$, a proposition $S(\texttt{Every man is mortal})$ is inferred from $\mathrm{NP}(\texttt{Every man})$ and $\mathrm{VP}(\texttt{is mortal})$ as follows.

$$\frac{\mathrm{NP}(\texttt{Every man}) \quad \mathrm{VP}(\texttt{is mortal})}{S(\texttt{Every man is mortal})}$$

A *proof tree* in this deduction system is a standard *parse tree.* This is the correct understanding of grammars, which puts them in the right perspective. In particular, the definition by logical inference obviously extends to conjunctive grammars, with the derivation rules now involving larger sets of premises that represent multiple conjuncts in a rule.

Definition 1(I). *For a conjunctive grammar $G = (\Sigma, N, R, S)$, propositions are of the form "a string w has a property A", with $w \in \Sigma^*$ and $A \in N$, denoted by $A(w)$.*

Let $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m$ be any rule in R, and let each conjunct be denoted by $\alpha_i = u_{i,0}B_{i,1}u_{i,1}B_{i,2} \ldots u_{i,\ell_i-1}B_{i,\ell_i}u_{i,\ell_i}$, where $B_{i,1}, \ldots, B_{i,\ell_i} \in N$, with $\ell_i \geqslant 0$, *are all nonterminal symbols it refers to, while* $u_{i,0}, u_{i,1} \ldots, u_{i,\ell} \in \Sigma^*$ *are symbols between them. This rule is regarded as the following scheme for logical derivations, applicable as long as all strings* $w_i = u_{i,0}v_{i,1}u_{i,1}v_{i,2} \ldots u_{i,\ell_i-1}v_{i,\ell_i}u_{i,\ell_i}$*, with* $i \in \{1, \ldots, m\}$*, are actually the same string* $w = w_1 = \ldots = w_m$*.*

$$\frac{B_{1,1}(v_{1,1}) \quad \ldots \quad B_{1,\ell_1}(v_{1,\ell_1}) \quad \ldots \quad B_{m,1}(v_{m,1}) \quad \ldots \quad B_{m,\ell_m}(v_{m,\ell_m})}{A(w)}$$

Whenever a proposition $A(w)$ *can be deduced from the above axioms by the given deduction rules, this is denoted by* $\vdash A(w)$*. Define* $L_G(A) = \{w \mid \vdash A(w)\}$ *and* $L(G) = L_G(S) = \{w \mid \vdash S(w)\}$*.*

A derivation of a proposition $S(w)$ according to these inference rules forms a proof tree, which is a parse tree of w. Whenever a conjunction operator is used in a tree, the same symbol of w is used in multiple branches. Accordingly, these are *trees with shared leaves*, as illustrated in the following example.

Example 1**(I).** *Getting back to the grammar from Example 1, the following derivation establishes that the string abc is a well-formed sentence.*

$$\frac{\dfrac{A(\varepsilon)}{A(a)} \quad \dfrac{B(\varepsilon)}{B(bc)} \quad \dfrac{D(\varepsilon)}{D(ab)} \quad \dfrac{C(\varepsilon)}{C(c)}}{S(abc)}$$

The last step is by the rule $S \to AB \,\&\, DC$ *using two representations of* $w = abc$ *as* $a \cdot bc$ *and as* $ab \cdot c$*.*

As a parse tree, this derivation is presented in Fig. 1. For example, the first symbol a is referenced both in the inference of $A(a)$ *and in the inference of* $D(ab)$*, and thus the leaf a is shared between two inbound arcs.*

2.2 Term Rewriting

Chomsky's [15] string-rewriting approach to the definition of grammars is presented in most computer science textbooks. For example, using a rule $S \to$ NP VP, the initial symbol S is rewritten into NP VP, and then NP is rewritten with `Every man`, while VP is rewritten with `is mortal`, so that the rewriting ends with the whole sentence. Each intermediate object in the rewriting, such as "NP `is mortal`", is a scheme of a sentence as a concatenation of unspecified substrings with given properties (in this example, NP) and actual symbols occurring in the sentence (`is mortal`). Such a scheme is known as a *sentential form*.

Conjunctive grammars can also be defined by rewriting sentential forms. This time, sentential forms are *terms* using two operations: concatenation and conjunction.

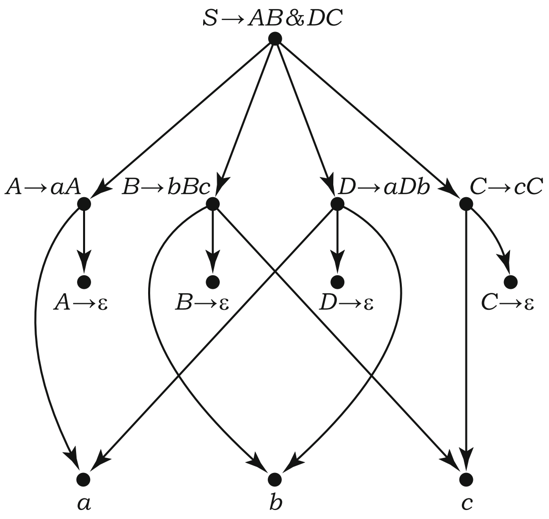

Fig. 1. Parse tree of the string *abc* according to the conjunctive grammar for the language $\{a^n b^n c^n \mid n \geqslant 0\}$ given in Example 1.

Definition 1(R) (Szabari [66], Okhotin [41])**.** *For a conjunctive grammar $G = (\Sigma, N, R, S)$, terms over concatenation and conjunction are defined as follows: (I) any symbol $a \in \Sigma$ is a term; (II) any nonterminal symbol $A \in N$ is a term; (III) a concatenation $t_1 \ldots t_\ell$ of finitely many terms $t_1, \ldots, t_\ell$, with $\ell \geqslant 0$, is a term; (IV) a conjunction $(t_1 \& \ldots \& t_m)$ of finitely many terms $t_1, \ldots, t_m$, with $m \geqslant 1$, is a term.*

Terms may be rewritten by two kinds of rewriting rules.

– *Any occurrence of a nonterminal symbol $A \in N$ in a term can be rewritten by the right-hand side of any rule $A \to \alpha_1 \& \ldots \& \alpha_m$ for A.*

$$\ldots A \ldots \Longrightarrow \ldots (\alpha_1 \& \ldots \& \alpha_m) \ldots$$

– *A conjunction of several identical strings $w \in \Sigma^*$ can be collapsed into one such string.*

$$\ldots (w \& \ldots \& w) \ldots \Longrightarrow \ldots w \ldots$$

If t can be rewritten to t' in zero or more steps, as defined above, this is denoted by $t \Longrightarrow^* t'$. The language of each term t is the set of all strings $w \in \Sigma^*$, into which t can be rewritten in zero or more steps.

$$L_G(t) = \{w \mid w \in \Sigma^*,\ t \Longrightarrow^* w\}$$

The language of the term S is the language described by the grammar: $L(G) = L_G(S)$.

For simplicity, when a single-conjunct rule $A \to \alpha$ is applied, parentheses can be omitted, so that A is rewritten with α, rather than with (α) enclosed in brackets.

Example 1(R). *For the grammar in Example 1, according to the definition by term rewriting, the string abc can be obtained by the following rewriting sequence.*

$S \Longrightarrow (AB \mathbin{\&} DC) \Longrightarrow (aAB \mathbin{\&} DC) \Longrightarrow (aB \mathbin{\&} DC) \Longrightarrow (abBc \mathbin{\&} DC) \Longrightarrow$
$(abc \mathbin{\&} DC) \Longrightarrow (abc \mathbin{\&} aDbC) \Longrightarrow (abc \mathbin{\&} abC) \Longrightarrow (abc \mathbin{\&} abcC) \Longrightarrow$
$(abc \mathbin{\&} abc) \Longrightarrow abc$

Unlike a sentential form in ordinary grammars, a conjunctive sentential form may define a contradictory set of conditions, so that no actual sentence satisfies them. For instance, in the above grammar, one can rewrite S to $(aaB \mathbin{\&} Dc)$, but the latter cannot be rewritten to any string.

2.3 Language Equations

The representation of ordinary grammars by equations with formal languages as unknowns was discovered by Ginsburg and Rice [17]; its modern treatment can be found in the handbook chapter by Autebert et al. [5]. Each nonterminal symbol $A \in N$ is treated as an unknown language, defined by an equation of the form $A = \varphi_A$, where the expression φ_A on the right-hand side may use any variables from N, any symbols from Σ, and the operations of union and concatenation.

This setting extends to conjunctive grammars by implementing each conjunction in the rules as an intersection operation in the equations. The resulting system is bound to have a least solution by the same basic lattice-theoretic argument as in the ordinary case with disjunction only, since it only relies on mononicity and continuity of operations.

Definition 1(E) (Okhotin [42]). *For every conjunctive grammar $G = (\Sigma, N, R, S)$, the associated system of language equations is a system of equations in variables N, with each variable representing an unknown language over Σ, which contains the following equation for every variable A.*

$$A = \bigcup_{A \to \alpha_1 \,\&\ldots\&\, \alpha_m \in R} \ \bigcap_{i=1}^{m} \alpha_i \qquad (\textit{for all } A \in N)$$

Each α_i in the equation is a concatenation of variables and constant languages $\{a\}$ representing symbols of the alphabet, or constant $\{\varepsilon\}$ if α_i is the empty string. Let $A = L_A$ for all $A \in N$ be the least solution of this system. Then $L_G(A)$ is defined as L_A for each $A \in N$, and $L(G) = L_S$.

This least solution can be obtained as a limit of a sequence of vectors of languages that is ascending under componentwise inclusion. The first element is $\bot = (\varnothing, \ldots, \varnothing)$. Every next element is obtained by applying the right-hand sides of the system as a vector function $\varphi\colon \big(2^{\Sigma^*}\big)^{|N|} \to \big(2^{\Sigma^*}\big)^{|N|}$ to the previous element. The resulting sequence $\{\varphi^k(\bot)\}_{k \to \infty}$ is ascending, and its limit (componentwise infinite union) $\bigsqcup_{k \geqslant 0} \varphi^k(\bot)$ is the least solution.

Example 1(E). *According to the definition by language equations, the grammar in Example 1 is represented by the following system.*

$$\begin{cases} S = (A \cdot B) \cap (D \cdot C) \\ A = \big(\{a\} \cdot A\big) \cup \{\varepsilon\} \\ B = \big(\{b\} \cdot B \cdot \{c\}\big) \cup \{\varepsilon\} \\ C = \big(\{c\} \cdot C\big) \cup \{\varepsilon\} \\ D = \big(\{a\} \cdot D \cdot \{b\}\big) \cup \{\varepsilon\}, \end{cases}$$

This system has a unique solution, with $S = \{a^n b^n c^n \mid n \geqslant 0\}$, $A = a^*$, $B = \{b^m c^m \mid m \geqslant 0\}$, $C = c^*$ and $D = \{a^m b^m \mid m \geqslant 0\}$.

These three definitions of conjunctive grammars are equivalent as follows.

Theorem 1. *Let $G = (\Sigma, N, R, S)$ be a conjunctive grammar, as in Definition 1. For every $A \in N$ and $w \in \Sigma^*$, the following three statements are equivalent:* **(I)** *$\vdash A(w)$;* **(R)** *$A \Longrightarrow^* w$;* **(E)** *w is in the A-component of $\bigsqcup_{k \geqslant 0} \varphi^k(\bot)$.*

3 Methods for Grammar Construction

3.1 Iterating the Conjunction

Quite a few languages can be defined by using the conjunction operator to intersect separately defined languages, as in Example 1. For instance, one can easily construct conjunctive grammars for such languages as $\{a^m b^n c^m d^n \mid m, n \geqslant 0\}$ and $\{w \mid w \in \{a, b, c\}^*,\ |w|_a = |w|_b = |w|_c\}$.

However, the expressive power of conjunctive grammars is not limited by this unsophisticated technique. Descriptions of a different kind can be obtained by *iterating the conjunction*, in the sense that a rule for some nonterminal A refers to A in one of its conjuncts. This way, one can express that an unbounded number of overlapping substrings are described by A. The simplest illustration of this method is given in the following grammar.

Example 2. The following conjunctive grammar describes the language $L = \{b\,ab\,a^2b\,a^3b \ldots a^n b \mid n \geqslant 0\}$.

$$\begin{aligned} S &\to ECb \,\&\, SAb \mid b \\ A &\to aA \mid \varepsilon \\ E &\to AbE \mid \varepsilon \\ C &\to aCa \mid ba \end{aligned}$$

The grammar defines this language by ensuring, for some string, that each prefix ending with b and containing at least two symbols b ends with $ba^{n-1}ba^n b$. This condition on the form of the suffix is defined by ECb. The second conjunct in the rule $S \to ECb \,\&\, SAb$ ensures that the same string without its last block must also belong to the language L: this is where the conjunction is iterated.

Of course, the language in Example 2 can be defined by intersecting two languages defined by ordinary grammars. The goal of the above example is to illustrate a principle of grammar construction, which shall later be used for more sophisticated languages.

3.2 Comparing Identifiers

The language $\{wcw \mid w \in \{a,b\}^*\}$ is among the most common examples of languages not described by any ordinary grammar. It represents identifier checking in programming languages. As proved by Wotschke [71], this language is not expressible as an intersection of finitely many languages defined by ordinary grammars. A conjunctive grammar for this language iterates the conjunction in the same way as in Example 2.

Example 3 (Okhotin [41]*).* The following conjunctive grammar describes the language $\{wcw \mid w \in \{a,b\}^*\}$.

$$\begin{aligned}
S &\to C \,\&\, D \\
C &\to XCX \mid c \\
D &\to aA \,\&\, aD \mid bB \,\&\, bD \mid cE \\
A &\to XAX \mid cEa \\
B &\to XBX \mid cEb \\
E &\to XE \mid \varepsilon \\
X &\to a \mid b
\end{aligned}$$

The nonterminal symbol C specifies the general form of the string as xcy, with $x, y \in \{a,b\}^*$ and $|x| = |y|$. Thus, the conjunction with C in the rule for S ensures that the string consists of two parts of equal length separated by a center marker. The other conjunct D checks that the symbols in corresponding positions are the same. The actual language defined by D is $L(D) = \{uczu \mid u, z \in \{a,b\}^*\}$, and it is defined inductively as follows: a string is in $L(D)$ if and only if either it is in $c\{a,b\}^*$ (the base case: no symbols to compare, the rule $D \to cE$), or its first symbol is the same as the corresponding symbol on the opposite side, *and* the string without its first symbol is in $L(D)$. The latter condition ensures that the rest of the symbols are compared in the same way. If the first symbol is a, this is handled by the rule $D \to aA \,\&\, aD$, where A is a nonterminal symbol that checks that the corresponding symbol on the right-nand side is a as well; if the first symbol is b, the rule $D \to bB \,\&\, bD$ similarly locates the matching symbol and verifies that it is b.

The above grammar essentially uses the center marker, and therefore this method cannot be applied to constructing a conjunctive grammar for the language $\{ww \mid w \in \{a,b\}^*\}$. The question of whether $\{ww \mid w \in \{a,b\}^*\}$ can be described by any conjunctive grammar remains an open problem.

Exercise 1. Construct conjunctive grammars for the following languages: (a) $\{(wc)^{|w|} \mid w \in \{a,b\}^*\}$; (b) $\{wwc^{|w|} \mid w \in \{a,b\}^*\}$; (c) $\{wcww \mid w \in \{a,b\}^*\}$.

3.3 Declaration Before Use

The grammar in the next example defines the requirement of *declaration before use*.

Example 4. Strings of the following form encode sequences of "declarations" da^i and "calls" ca^i, with the condition that every call ca^i requires an earlier declaration da^i.

$$\{s_1a^{i_1}\dots s_na^{i_n} \mid n, i_1, \dots, i_n \geqslant 0;\ \forall j,\ \text{if } s_j = c,\ \text{then } \exists k < j : i_k = i_j\}$$

This language is described by the following grammar.

$$\begin{aligned}
S &\to SdA \mid ScA \,\&\, EdB \mid \varepsilon \\
A &\to aA \mid \varepsilon \\
B &\to aBa \mid Ec \\
E &\to cAE \mid dAE \mid \varepsilon
\end{aligned}$$

The grammar applies generally the same technique of inductive definitions as in Example 3. The rule $S \to \varepsilon$ asserts that an empty sequence of declarations and calls has the required property. The rule $S \to SdA$ appends a new declaration (dA) to a well-formed string with all references preceded by declarations (S). The other rule $S \to ScA \,\&\, EdB$ similarly appends a call (cA), and at the same time ensures that this call has a preceding declaration (EdB). Here E defines an arbitrary sequence of declarations and calls, and the concatenation EdB defines a suitable partition of the string, where the symbol d begins the appropriate declaration, and B ensures that the number of symbols a is the same in the declaration and in the call.

Exercise 2. Construct a conjunctive grammar for the following language, which adopts the encoding from Example 4 and represents the condition of having no duplicate declarations.

$$\{da^{i_1}\dots da^{i_n} \mid n, i_1, \dots, i_n \geqslant 0,\ \text{and } i_1, \dots, i_n \text{ are pairwise distinct}\}$$

The languages of identifier comparison (Example 3), declaration before use (Example 4) and duplicate declarations (Exercise 2) represent several syntactic constructs common for artificial languages. The conjunctive grammars for these abstract languages can be combined into a single grammar for a simple programming language [49].

3.4 One-Symbol Alphabet

The question of whether conjunctive grammars over a one-symbol alphabet can define any non-regular languages used to be an open problem, until Jeż [23] found the following example.

Example 5 (Jeż [23]). The following conjunctive grammar with the initial symbol A_1 describes the language $\{a^{4^n} \mid n \geqslant 0\}$.

$$\begin{aligned}
A_1 &\to A_1A_3 \,\&\, A_2A_2 \mid a \\
A_2 &\to A_1A_1 \,\&\, A_2A_6 \mid aa \\
A_3 &\to A_1A_2 \,\&\, A_6A_6 \mid aaa \\
A_6 &\to A_1A_2 \,\&\, A_3A_3
\end{aligned}$$

In particular, each nonterminal symbol A_i defines the language $\{a^{i\cdot 4^n} \mid n \geqslant 0\}$.

This grammar is best explained in terms of base-4 representations of string length. Each symbol A_i, with $i \in \{1, 2, 3\}$, describes base-4 numbers of the form $i\mathtt{0}\ldots\mathtt{0}$, whereas the numbers described by A_6 are of the form $\mathtt{120}\ldots\mathtt{0}$. The form of base-4 representations is given by regular expressions $i\mathtt{0}^*$ and $\mathtt{120}^*$, and the corresponding unary languages shall be denoted by $a^{(i\mathtt{0}^*)_4}$ and $a^{(\mathtt{120}^*)_4}$.

Concatenation *adds* these numbers. In the rule for A_1, the first concatenation A_1A_3 produces all numbers with the base-4 representations $\mathtt{10}^*\mathtt{30}^*$, $\mathtt{30}^*\mathtt{10}^*$ and $\mathtt{10}^+$, of which the latter is the intended set, while the rest are regarded as garbage.

$$A_1A_3 = a^{(\mathtt{10}^*)_4}a^{(\mathtt{30}^*)_4} = a^{(\mathtt{10}^+)_4} \cup a^{(\mathtt{10}^*\mathtt{30}^*)_4} \cup a^{(\mathtt{30}^*\mathtt{10}^*)_4}$$

The second concatenation A_2A_2 yields $\mathtt{20}^*\mathtt{20}^*$ and $\mathtt{10}^+$.

$$A_2A_2 = a^{(\mathtt{20}^*)_4}a^{(\mathtt{20}^*)_4} = a^{(\mathtt{10}^+)_4} \cup a^{(\mathtt{20}^*\mathtt{20}^*)_4}$$

Although both concatenations contain some garbage, it turns out that *the garbage in the two concatenations is disjoint*, and is accordingly filtered out by the conjunction, which produces exactly the numbers with the base-4 representation $\mathtt{10}^+$, that is, the language $\{a^{4^n} \mid n \geqslant 1\}$. For $n = 0$, the string is given by the rule $A_1 \to a$. The construction for the rest of nonterminals works similarly. This way, the grammar inductively defines longer strings by concatenating shorter ones.

4 Equivalent Models

4.1 Normal Forms

Conjunctive grammars can be transformed to several normal forms. The first normal form generalizes the Chomsky normal form for ordinary grammars: a conjunctive grammar $G = (\Sigma, N, R, S)$ is said to be in the *binary normal form*, if all rules are of the following form.

$$\begin{aligned} &A \to B_1C_1 \mathbin{\&} \ldots \mathbin{\&} B_mC_m && (m \geqslant 1,\ B_i, C_i \in N) \\ &A \to a && (a \in \Sigma) \end{aligned}$$

A rule $S \to \varepsilon$ is also allowed, as long as S is never referenced in any rules. Every conjunctive grammar can be transformed to this normal form, even though the size of the resulting grammar may be exponential [41]. No lower bound on the complexity of the transformation is known.

The Greibach normal form is naturally extended to conjunctive grammars.

$$A \to a\alpha_1 \mathbin{\&} \ldots \mathbin{\&} a\alpha_m \qquad (a \in \Sigma,\ m \geqslant 1,\ \alpha_1, \ldots, \alpha_m \in (\Sigma \cup N)^*)$$

However, it is not known whether every conjunctive grammar has a grammar in this normal form that describes the same language [55]. Transformation is possible for a unary alphabet [60].

There exists another normal form, inspired by the operator normal form for ordinary grammars. A conjunctive grammar is in the *odd normal form*, if all rules are of the following form.

$$\begin{array}{ll} A \to B_1 a_1 C_1 \,\&\, \dots \,\&\, B_m a_m C_m & (m \geqslant 1,\ B_i, C_i \in N,\ a_i \in \Sigma) \\ A \to a & (a \in \Sigma) \end{array}$$

If S is never used in the right-hand sides of any rules, then rules of the form $S \to aA$, with $a \in \Sigma$ and $A \in N$, and $S \to \varepsilon$ are also allowed.

There is also a normal form of another kind: a conjunctive grammar is said to be *with restricted disjunction*, if, for each nonterminal symbol, there is at most one rule referring to any other nonterminals, whereas the rest of the rules define finitely many fixed strings.

$$A \to \alpha_1 \,\&\, \dots \,\&\, \alpha_m \mid w_1 \mid \dots \mid w_k \qquad (k \geqslant 0,\ w_1, \dots, w_k \in \Sigma^*)$$

Every conjunctive grammar can be transformed to a grammar with restricted disjunction [60]. On the other hand, the corresponding subfamily of ordinary grammars, the *single tree grammars*, is known to be weaker in power than ordinary grammars of the general form [19].

4.2 Conjunctive Categorial Grammars

A *categorial grammar* [4] uses a set of *primitive categories* N, which can be combined by using quotient operators, as follows. Let a sentence (S) and a noun ($\boldsymbol{n}$) be primitive categories. Then, a category $\boldsymbol{n}\backslash S$, obtained by left-quotient, means "any string that, after concatenating a noun ($\boldsymbol{n}$) on the left, becomes a sentence (S)". A categorial grammar consists of rules assigning such a category to every symbol of the alphabet. For instance, `Sun` is a noun, which is given by a rule $\boldsymbol{n} \to \texttt{Sun}$. A verb `shines` is defined by a rule $\boldsymbol{n}\backslash S \to \texttt{shines}$. Then, a sentence formed of these two words is derived as follows.

$$\frac{\boldsymbol{n}(\texttt{Sun}) \quad \boldsymbol{n}\backslash S(\texttt{shines})}{S(\texttt{Sun shines})}$$

Bar-Hillel et al. [7] proved these grammars' equivalence to ordinary grammars.

An extension of categorial grammars with conjunction was first considered by Kanazawa [30], and Kuznetsov [34] was the first to investigate their relation to conjunctive grammars. Recently, Kuznetsov and Okhotin [35] developed a simpler definition of *conjunctive categorial grammars*. In these grammars, a conjunction of primitive categories is a category, and the corresponding inference rules are added. These grammars are equivalent to conjunctive grammars, which is proved similarly to Bar-Hillel et al. [7], with the help of the odd normal form.

4.3 Contextual Binary Feature Grammars

Contextual binary feature grammars were introduced by Clark et al. [16] as a formalism for learning algorithms.

A contextual binary feature grammar is a triple $G = (\Sigma, F, R)$, where $F \subset \Sigma^* \times \Sigma^*$ is a finite set of pairs of strings called *contexts*, with $(\varepsilon, \varepsilon) \in F$. Subsets of F are regarded as categories, and the rules in R are of the following two forms.

$$\begin{aligned} &A \to a && (A \subseteq F,\ a \in \Sigma) \\ &A \to BC && (A, B, C \subseteq F) \end{aligned}$$

The grammar defines the set of contexts for every string as follows: $f(\varepsilon) = \varnothing$, $f(a) = \bigcup\{A \mid A \to a \in R\}$ and $f(w) = \bigcup\{A \mid A \to BC \in R,\ w = uv,\ B \subseteq f(u),\ C \subseteq f(v)\}$. The language defined by the grammar is $L(G) = \{w \mid (\varepsilon, \varepsilon) \in f(w)\}$.

Yoshinaka [72] proved these grammars equivalent to conjunctive grammars.

4.4 Pushdown Automata with Tree Stack

The representation of ordinary grammars by nondeterministic pushdown automata (NPDA) was among the earliest results in formal language theory. It is important for its deterministic case: the idea of a deterministic machine with a stack gave rise to LR(k) grammars [32], which were then proved equivalent to DPDA.

A generalization of NPDA equivalent to conjunctive grammars was defined by Aizikowitz and Kaminski [1]. Their stack is a *tree:* it has a single root and may split into branches, so that it has multiple top nodes. Each top node has its own state. A transition is made at one top node at a time, so that it changes its state and pushes a *subtree*, rather than a string of stack symbols. If all branches coming out of a single node are exhausted, the automaton synchronizes by checking that all branches have computed the same state. An input symbol is read if all top nodes are ready to read it. The model also has a deterministic case [3].

5 Parsing Algorithms

Quite a few efficient parsing algorithms for ordinary grammars are known. There are algorithms applicable to an arbitrary grammar, as well as more efficient algorithms for certain subclasses. It turns out that many of these algorithms extend to conjunctive grammars with almost no changes, and others have generalizations.

5.1 Tabular Algorithms

Tabular algorithms, also known as *chart parsers*, use dynamic programming to determine all properties of all substrings of the input string, inductively from shorter to longer substrings. The simplest of these algorithms is the Cocke–Kasami–Younger algorithm: given a string $w = a_1 \ldots a_n$, it constructs, for each substring $a_{i+1} \ldots a_j$, the set $T_{i,j} = \{A \mid a_{i+1} \ldots a_j \in L_G(A)\}$. In the end, it is sufficient to test whether S is in $T_{0,n}$. Its running time is $\Theta(n^3)$ and it uses $\Theta(n^2)$ space.

The Cocke–Kasami–Younger algorithm applies to conjunctive grammars in the binary normal form without any changes, providing cubic-time parsing.

There is an asymptotically faster algorithm for ordinary grammars, namely, *Valiant's algorithm* [70], which computes the same table $T_{i,j}$, but organizes the same Boolean operations to be computed in a different order, so that they are grouped into *matrix products* for pairs of Boolean matrices of size from 1×1 to $2^k \times 2^k$, with $2^k \approx \frac{n}{4}$. Then, using matrix multiplication, the parsing algorithm can run in time $O(n^\omega)$, where $\omega < 3$ is a constant determined by the matrix multiplication algorithm; asymptotically best algorithms provide $\omega \approx 2.4$. After some simplifications were made to Valiant's algorithm, it also turned out that it applies to conjunctive grammars without any changes [56], retaining the same running time $O(n^\omega)$.

Finally, the square-time Kasami–Torii parsing algorithm for unambiguous grammars extends to unambiguous conjunctive grammars, also preserving the time complexity $O(n^2)$ [53]. Overall, the ideas of chart parsers are unaffected by adding the conjunction.

5.2 LL Parsing

The LL(k) parsing method applies to a subclass of ordinary grammars and runs in linear time for every applicable grammar. These grammars are called *LL(k)*, where k is the number of lookahead symbols. An LL(k) parser reads the string from left to right, maintaining its expectations on the form of the remaining input in a stack containing symbols and nonterminal symbols. When the parser has read a prefix u of a string uv and has $\alpha \in (\Sigma \cup N)^*$ in the stack, this means that it expects to parse v as α. At each moment of its computation, the parser pops the top stack symbol. If it is a symbol $a \in \Sigma$, then the parser reads the next input symbol, making sure that this is also a. If a nonterminal symbol $A \in N$ is popped, then the parser identifies a rule $A \to \alpha$ on the basis of the next k input symbols, and pushes α onto the stack. Initially, the parser has S in the stack, representing its intention to parse the entire input as S. In order to accept, the parser has to empty its stack as it finishes reading the string.

LL(k) parsing has a generalization for conjunctive grammars, in which *the stack is tree-structured*, with a single bottom node and with potentially multiple leaves as top stack symbols. Parallel branches converging in a single node define multiple representations for a single substring [43]. Whenever a nonterminal symbol $A \in N$ in one of the leaves is processed, a rule $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m$ is determined, and the leaf is replaced with m parallel branches labelled with α_1, $\ldots$, α_m. In order to read an input symbol, the same symbol must be popped from each branch of the stack.

The representation of an ordinary LL(k) parser by recursive descent also has an extension to conjunctive grammars. In a conjunctive recursive descent parser, the conjunction is implemented by scanning a single substring multiple times. The algorithm may spend exponential time on some grammars and inputs, but with the help of the memoization technique, its running time can be kept linear in the length of the input [51].

LL(k) conjunctive grammars are known to be less powerful that conjunctive grammars of the general form: in particular, these grammars cannot define any non-regular unary languages [54]. LL(k) linear conjunctive grammars are even less powerful [54]. However, these results were obtained by *ad hoc* proofs, and no general method for proving languages not to be LL(k) conjunctive is known. It would be interesting to adapt some of the ideas in the classical paper on LL parsing by Rosenkrantz and Stearns [64]. In particular, is there a Greibach normal form for LL conjunctive grammars? If this is resolved in the affirmative, some interesting results shall follow.

5.3 LR Parsing

The LR(k) parsing algorithm, introduced by Knuth [32], applies to a strictly larger subclass of ordinary grammars than the LL(k) method. It processes the string from left to right, storing partial parses of the prefix it has read in a stack containing symbols and nonterminal symbols. When the parser has read a prefix u and has $\alpha \in (\Sigma \cup N)^*$ in the stack, this means that u has been parsed as α. At each moment in its computation, the parser either reads the next symbol and "shifts" it onto the stack, or identifies the right-hand side of some rule $A \to \eta$ at the top of the stack, and "reduces" η to A. LR(k) grammars are those for which these decisions can be made deterministically, while looking at the next k input symbols.

There is an extension of LR parsing, introduced by Lang [36] and by Tomita [69] and known as the *Generalized LR (GLR)*. This algorithm is applicable to every ordinary grammar; and if the grammar is not LR(k), nondeterminism in the decisions is handled by storing all possible contents of an LR parser's stack as a graph. Under a careful implementation, the algorithm works in time $O(n^3)$, but if the grammar is LR(k), its graph-structured stack becomes a path, and the algorithm works in linear time.

The GLR is extended to conjunctive grammars as follows [44,50]. The processing of multiple conjuncts in a single rule $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m$ is initiated as if these were multiple rules $A \to \alpha_1, \ldots, A \to \alpha_m$, so that the parser tries to parse a substring as each of them. If all parses are successful, the parser eventually constructs m paths emerging from a single node and labelled with $\alpha_1, \ldots, \alpha_m$. These paths are detected and "reduced" to A. The algorithm can be implemented to run in worst-case cubic time, but on easier grammars it can be much faster.

An extension of deterministic LR parsing to conjunctive grammars was proposed by Aizikowitz and Kaminski [3]: their *LR(0) conjunctive grammars* are based on the deterministic case of their pushdown automata with a tree stack [1].

6 Theoretical Results

6.1 Linear Conjunctive Grammars and Cellular Automata

A conjunctive grammar $G = (\Sigma, N, R, S)$ is said to be *linear*, if every conjunct of every rule refers to at most one nonterminal symbol. In other words, all rules are of the following form.

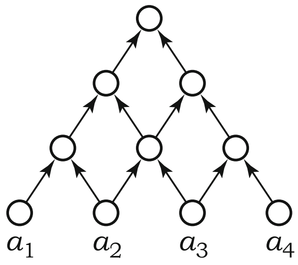

Fig. 2. A computation carried out by a trellis automaton.

$$\begin{aligned} &A \to u_1 B_1 v_1 \,\&\, \dots \,\&\, u_m B_m v_m && (m \geqslant 1,\ u_i, v_i \in \Sigma^*,\ B_i \in N) \\ &A \to w && (w \in \Sigma^*) \end{aligned}$$

Linear conjunctive grammars are notable for their representation by *one-way real-time cellular automata*, also known as *trellis automata*.

Definition 2. *A trellis automaton is a quintuple $(\Sigma, Q, I, \delta, F)$, where Q is the set of states, $I \colon \Sigma \to Q$ sets the initial value of each cell, $\delta \colon Q \times Q \to Q$ determines the next value of each cell as a function of its state and the state of its right neighbour, and $F \subseteq Q$ is the set of accepting states.*

An input string is mapped to the first row of states, $I(a_1 \dots a_n) = I(a_1) \dots I(a_n)$, and then, at each step, a row of states $q_1 \dots q_k$ is replaced with $\Delta(q_1 \dots q_k) = \delta(q_1, q_2)\delta(q_2, q_3) \dots \delta(q_{k-1}, q_k)$. On an input string $w \in \Sigma^+$, after $|w| - 1$ steps, the automaton computes a single state, and w is accepted if this state is in F.

Each state computed by a trellis automaton in its computation on some string corresponds to one of the string's substrings. The dependence of states on each other within a single computation is illustrated in Fig. 2.

Theorem 2 (Okhotin [46,47]). *A language $L \subseteq \Sigma^+$ is described by a linear conjunctive grammar if and only if it is recognized by a trellis automaton.*

Accordingly, many interesting theoretical results on trellis automata apply to linear conjunctive grammars. In particular, numerous results on this model were obtained by Ibarra and Kim [22]. Some limitations of trellis automata were determined by Terrier [67,68], who, in particular, established a general lemma for proving non-representability of languages by trellis automata, and used it to prove the non-closure of this family under concatenation [67]. This is all that is known about proving non-representability of languages by conjunctive grammars, and, hopefully, some of these ideas could eventually be extended to their general, non-linear case.

Another equivalent representation of linear conjunctive grammars was studied by Aizikowitz and Kaminski [2]: this is an extension of one-turn NPDA for the case of a tree stack.

6.2 Unary Languages

Since the first example of a non-regular unary language defined by a conjunctive grammar (Example 5), quite a few general results on representing unary languages have been obtained. The first such result is that if there is a finite automaton over an alphabet of k-ary digits that recognizes some set of numbers given to it in base-k notation, then there exists a conjunctive grammar that defines the same set of numbers, but this time in unary notation.

Theorem 3 (Jeż [23]). *Let $\Sigma_k = \{0, 1, \ldots, k-1\}$, with $k \geqslant 2$, be an alphabet of k-ary digits, and let $L \subseteq \Sigma_k^* \setminus 0\Sigma_k^*$ be a regular language. Then, the language $\{a^n$ the k-ary representation of n is in $L\}$ is described by a conjunctive grammar.*

Theorem 3 actually holds true for L defined by any *linear conjunctive grammar* [24]. Under some techical restrictions of L, such as on the allowed digits in base-k representations, the theorem asserts the existence of an *unambiguous conjunctive grammar* [29].

This theorem led to several undecidability results for unary conjunctive grammars [24]. It was also crucial for establishing the computational completeness of language equations over a unary alphabet [28,40].

Further results on this class include the following. There exists an EXPTIME-complete set that, *in unary notation*, can be defined by a conjunctive grammar [25]. Next, any unary conjunctive grammar can be encoded in a grammar with one nonterminal symbol that defines a closely related language [26,62]. For unambiguous conjunctive grammars, there is a similar result encoding any grammar in a two-nonterminal grammar [27].

There is also the following complexity upper bound for unary conjunctive grammars. The membership of a string a^n in a language can be tested in time $O(n^2)$ by adapting the basic cubic-time algorithm in Sect. 5.1. As shown by Okhotin and Reitwießner [61], this algorithm can be accelerated by using *Boolean convolution*, in the same way as Valiant's algorithm uses Boolean matrix multiplication. The resulting algorithm works in time $n \log^{O(1)} n$ [61].

6.3 Hardest Languages

There is a famous theorem by Greibach [18], which states that all languages defined by ordinary grammars are representable as inverse homomorphic images of a single *hardest language* L_0, which is also defined by some ordinary grammar. There is a similar result for conjunctive grammars.

Theorem 4 (Okhotin [58]). *There exists a fixed language $L_0 \subseteq \Sigma_0^+$ described by a conjunctive grammar, so that every language $L \subseteq \Sigma^+$ described by some conjunctive grammar is representable as $h^{-1}(L(G_0))$.*

In the absence of a Greibach normal form for conjunctive grammars, the proof of this result had to rely on the odd normal form, which made the constructed fairly elaborate. It is also worth mentioning that the language family defined by conjunctive grammars is closed under inverse homomorphisms [39], and therefore the class of homomorphic images of L_0 is precisely the family defined by conjunctive grammars.

It is an interesting open question whether a similar result exists for linear conjunctive grammars.

7 Further Grammar Families

The results on conjunctive grammars demonstrate that many basic properties of ordinary grammars are not broken by allowing the conjunction operator alongside the disjunction, and many recurring ideas of formal grammars still apply in this case. The question is, how far can the formalism be extended to preserve those properties?

7.1 Context Operators

In actual languages, grammatical constructs may depend on the context in which they occur. The idea of having grammar rules with context restrictions has a long history of research attempts. In the early days of formal language theory, Chomsky [15] aimed to model this by rewriting a substring $\eta A \theta$ with $\eta \alpha \theta$, where the strings η and θ are the "contexts" *within the sentential form*. However, it was quickly found that this model is equivalent to NSPACE(n); these are nondeterministic Turing machines, and the "contexts" they define refer to nothing but neighbouring bits in the memory. This makes the model quite irrelevant to language description.

Grammars with context operators were introduced by Barash and Okhotin [9, 10] as another attempt to model this syntactic phenomenon. This is an extension of conjunctive grammars featuring special operators for referring to the contexts of a substring *within the string*. Using such an operator, the applicability of a rule may be restricted to some particular contexts.

For every partition $w = xyz$, the prefix x is the *left context* of y, whereas the concatenation xy is the *extended left context* of y. The rules in a grammar with contexts are of the following form, where each β_i describes the structure of x, whereas each γ_i describes the structure of xy.

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \trianglelefteq\gamma_1 \,\&\, \ldots \,\&\, \trianglelefteq\gamma_n,$$

Informally, such a rule asserts that every substring representable as each concatenation α_i, written in a left context representable as each β_i and in an extended left context representable as γ_i, therefore has the property A. This is formalized by logical inference using propositions of the form "a substring $v \in \Sigma^*$ in the left context $u \in \Sigma^*$ has the property $X \in \Sigma \cup N$", denoted by $X\big(u\langle v\rangle\big)$. These propositions are derived as follows.

Example 6. The following grammar with left contexts $G = (\Sigma, N, R, S)$ defines a single string ab.

$$\begin{aligned} S &\to AB \\ A &\to a \\ B &\to b \,\&\, \lhd A \end{aligned} \qquad \dfrac{A\big(\varepsilon\langle a\rangle\big) \quad \dfrac{A\big(\varepsilon\langle a\rangle\big)}{B\big(a\langle b\rangle\big)}}{S\big(\varepsilon\langle ab\rangle\big)}$$

In the derivation of $S\big(\varepsilon\langle ab\rangle\big)$, deriving the proposition $B\big(a\langle b\rangle\big)$ requires a left context of the form A. The concatenation of $A\big(\varepsilon\langle a\rangle\big)$ and $B\big(a\langle b\rangle\big)$ needed to infer S respects contexts.

Example 7. The following grammar with one-sided context operators describes the language $\{ww \mid w \in \{a,b\}^*\}$.

$$\begin{aligned} S &\to XT \\ T &\to XXT \,\&\, A \;\;\&\;\; \lhd A \mid XXT \,\&\, B \;\;\&\;\; \lhd B \mid a \,\&\, \lhd A \mid b \,\&\, \lhd B \\ A &\to XAX \mid a \\ B &\to XBX \mid b \\ X &\to a \mid b \end{aligned}$$

Assuming that conjunctive grammars cannot describe this language, grammars with one-sided contexts should be strictly more powerful. A more convincing demonstration of the expressive power of these grammars was given by Barash [8], who constructed a grammar for the syntax of a simple programming language, including not only declaration before use, but also type checking.

A few results on grammars with one-sided contexts are known, which further elaborate the ideas extended from ordinary grammars to conjunctive grammars. There is a normal form with rules of the form $A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m$ and $A \to a \,\&\, \lhd D$ [57]. There is a straightforward cubic-time parsing algorithm [9], which can be accelerated to $O(\frac{n^3}{\log n})$ [57]. GLR parsing is also possible [12], and parsing in DSPACE(n) is inherited as well [13]. A subclass of *linear grammars with one-sided contexts* is represented by a generalization of trellis automata [11].

Grammars with two-sided context operators have been introduced as well [10]. Little is known about them, besides polynomial-time parsing: a direct extension of Cocke–Kasami–Younger works in time $O(n^4)$ [10], whereas a more sophisticated algorithm by Rabkin [63] uses time $O(n^3)$.

7.2 Boolean Operations

Boolean grammars [45] are an extension of conjunctive grammars that further allows negation in the rules. The rules in Boolean grammars are of the following general form, with $m + n \geqslant 1$ and $\alpha_i, \beta_j \in (\Sigma \cup N)^*$.

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

Such a rule asserts that if a string can be represented according to each positive conjunct α_i, and cannot be represented according to any negative conjunct β_j, then this string has the property A. This intuitively clear definition is not so easy to define formally, because a contradiction can be expressed: what is the grammar $S \to \neg S$ supposed to mean?

All known definitions are based on language equations, with negation represented by complementation. The first, naïve definition, given by the author [45], required that the system corresponding to the grammar has a unique solution in a certain strong sense. Although mathematically correct, that definition behaved counter-intuitively on some extreme examples. An improved definition by Kountouriotis et al. [33], based on three-valued logic, follows the intuition.

There is a variant of Boolean grammars that does not use negation at all. This is an extension of conjunctive grammars with *the logical dual of concatenation* [48,52], defined as follows.

$$K \mathbin{\bar{\cdot}} L = \{w \mid \text{for each partition } w = uv, \text{ it holds that } u \in K \text{ or } v \in L\} = \overline{\overline{K} \cdot \overline{L}}$$

This grammar family is defined using inference rules or using language equations, and it is known to be equivalent to Boolean grammars in power [48].

Many results for conjunctive grammars discussed above also apply to Boolean grammars: the binary normal form [45]; all tabular parsing algorithms [45,53,56,61]; LL and GLR parsing [50,51]; limitations of LL parsing [54]; representation of linear Boolean grammars by trellis automata [46]; the hardest language theorem [58].

7.3 Further Models

A recently introduced grammar model are the *GF(2)-grammars* [6]. This is a variant of ordinary grammars, in which the disjunction operation is replaced with the exclusive OR, whereas the following variant of the concatenation based on GF(2) logic is used instead of the classical concatenation.

$$K \odot L = \{w \mid \text{the number of partitions } w = uv, \text{ with } u \in K \text{ and } v \in L, \text{ is odd}\}$$

These grammars can be defined by language equations, but there is an easier equivalent definition: a GF(2) grammar defines the set of all strings with an odd number of parse trees.

This model is notable for preserving all tabular parsing algorithms, as well as parsing in NC^2 [6]. Nothing else is yet known. By their expressive power, these grammars are most likely incomparable with conjunctive grammars.

8 Conclusion

What has been achieved over almost two decades of research on conjunctive grammars? First, some basic properties of conjunctive grammars, inherited from ordinary grammars, were determined, such as normal forms, parsing algorithms, and complexity upper bounds. These were simple results indeed. However, they were crucial, first of all, for justifying the value of the new grammar family, confirming that it is not a nonsense model, after all. These results also imply the potential usefulness of conjunctive grammars in applications; the very fact that there are so few new ideas in the basic properties of conjunctive grammars is an argument in favour of their usefulness!

The results on conjunctive grammars inherited from ordinary grammars also improve the general understanding of the recurring ideas common to various grammar families. This further justifies the outlook on formal grammars as fragments of a logic for inductive definitions, and suggests a systematic approach to grammars based on the logical connectives allowed in these fragments. Perhaps it is time to retire the Chomskian outlook on grammars as "context-free" or "context-sensitive", and accordingly adopt the proposed term **ordinary grammar**, which reflects today's state of the art much better [59].

However, as far as theoretical research on conjunctive grammars is concerned, still very little is known about their basic properties. We still have no methods for proving that any particular language is not described by any conjunctive grammar. In fact, it is not even known whether the family defined by conjunctive grammars is different from DSPACE(n). Knowing the limitations of conjunctive grammars is essential for any further theoretical study of this model, and investigating the methods for proving these limitations is proposed as a challenging research subject.

References

1. Aizikowitz, T., Kaminski, M.: Conjunctive grammars and alternating pushdown automata. Acta Informatica **50**(3), 175–197 (2013). https://doi.org/10.1007/978-3-540-69937-8_6
2. Aizikowitz, T., Kaminski, M.: Linear conjunctive grammars and one-turn synchronized alternating pushdown automata. Int. J. Found. Comput. Sci. **25**(6), 781–802 (2014). https://doi.org/10.1142/S0129054114500336
3. Aizikowitz, T., Kaminski, M.: LR(0) conjunctive grammars and deterministic synchronized alternating pushdown automata. J. Comput. Syst. Sci. **82**(8), 1329–1359 (2016). https://doi.org/10.1016/j.jcss.2016.05.008
4. Ajdukiewicz, K.: Die syntaktische Konnexität. In: Ajdukiewicz, K., Ingarden, R., Twardowski, K. (eds.) Studia Philosophica, vol. 1, pp. 1–27 (1935)
5. Autebert, J., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 111–174. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_3
6. Bakinova, E., Basharin, A., Batmanov, I., Lyubort, K., Okhotin, A., Sazhneva, E.: Formal languages over GF(2). In: Klein, S.T., Martín-Vide, C., Shapira, D. (eds.) LATA 2018. LNCS, vol. 10792, pp. 68–79. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77313-1_5
7. Bar-Hillel, Y., Gaifman, H., Shamir, E.: On categorial and phrase structure grammars. Bull. Res. Counc. Isr. **9F**, 155–166 (1960)
8. Barash, M.: Programming language specification by a grammar with contexts. In: NCMA (2013)
9. Barash, M., Okhotin, A.: An extension of context-free grammars with one-sided context specifications. Inf. Comput. **237**, 268–293 (2014). https://doi.org/10.1016/j.ic.2014.03.003

10. Barash, M., Okhotin, A.: Two-sided context specifications in formal grammars. Theor. Comput. Sci. **591**, 134–153 (2015). https://doi.org/10.1016/j.tcs.2015.05.004
11. Barash, M., Okhotin, A.: Linear grammars with one-sided contexts and their automaton representation. RAIRO Informatique Théorique et Applications **49**(2), 153–178 (2015). http://dx.doi.org/10.1051/ita/2015004
12. Barash, M., Okhotin, A.: Generalized LR parsing algorithm for grammars with one-sided contexts. Theory Comput. Syst. **61**(2), 581–605 (2017). https://doi.org/10.1007/s00224-016-9683-3
13. Barash, M., Okhotin, A.: Linear-space recognition for grammars with contexts. Theor. Comput. Sci. **719**, 73–85 (2018). https://doi.org/10.1016/j.tcs.2017.11.006
14. Boullier, P.: A cubic time extension of context-free grammars. Grammars **3**(2–3), 111–131 (2000). https://doi.org/10.1023/A:1009907814595
15. Chomsky, N.: Three models for the description of language. IRE Trans. Inf. Theory **2**(3), 113–124 (1956). https://doi.org/10.1109/TIT.1956.1056813
16. Clark, A., Eyraud, R., Habrard, A.: Using contextual representations to efficiently learn context-free languages. J. Mach. Learn. Res. **11**, 2707–2744 (2010). http://www.jmlr.org/papers/v11/clark10a.html
17. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. J. ACM **9**, 350–371 (1962). https://doi.org/10.1145/321127.321132
18. Greibach, S.A.: A new normal-form theorem for context-free phrase structure grammars. J. ACM **12**, 42–52 (1965). https://doi.org/10.1145/321250.321254
19. Greibach, S.A., Shi, W., Simonson, S.: Single tree grammars. In: Ullman, J.D. (ed.) Theoretical Studies in Computer Science, pp. 73–99. Academic Press, Cambridge (1992)
20. Heilbrunner, S., Schmitz, L.: An efficient recognizer for the Boolean closure of context-free languages. Theor. Comput. Sci. **80**, 53–75 (1991). https://doi.org/10.1016/0304-3975(91)90205-G
21. Hellings, J.: Conjunctive context-free path queries. In: 17th International Conference on Database Theory, ICDT 2014, Athens, Greece, 24–28 March 2014, pp. 119–130 (2014)
22. Ibarra, O.H., Kim, S.M.: Characterizations and computational complexity of systolic trellis automata. Theor. Comput. Sci. **29**, 123–153 (1984). https://doi.org/10.1016/0304-3975(84)90015-X
23. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. Int. J. Found. Comput. Sci. **19**(3), 597–615 (2008). https://doi.org/10.1142/S012905410800584X
24. Jeż, A., Okhotin, A.: Conjunctive grammars over a unary alphabet: undecidability and unbounded growth. Theory Comput. Syst. **46**(1), 27–58 (2010). https://doi.org/10.1007/s00224-008-9139-5
25. Jeż, A., Okhotin, A.: Complexity of equations over sets of natural numbers. Theory Comput. Syst. **48**(2), 319–342 (2011). https://doi.org/10.1007/s00224-009-9246-y
26. Jeż, A., Okhotin, A.: One-nonterminal conjunctive grammars over a unary alphabet. Theory Comput. Syst. **49**(2), 319–342 (2011). https://doi.org/10.1007/s00224-011-9319-6
27. Jeż, A., Okhotin, A.: On the number of nonterminal symbols in unambiguous conjunctive grammars. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 183–195. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31623-4_14
28. Jeż, A., Okhotin, A.: Computational completeness of equations over sets of natural numbers. Inf. Comput. **237**, 56–94 (2014). https://doi.org/10.1016/j.ic.2014.05.001

29. Jeż, A., Okhotin, A.: Unambiguous conjunctive grammars over a one-symbol alphabet. Theor. Comput. Sci. **665**, 13–39 (2017). https://doi.org/10.1016/j.tcs.2016.12.009
30. Kanazawa, M.: The Lambek calculus enriched with additional connectives. J. Log. Lang. Inf. **1**, 141–171 (1992). https://doi.org/10.1007/BF00171695
31. Kanchan Devi, K., Arumugam, S.: Probabilistic conjunctive grammar. In: Arumugam, S., Bagga, J., Beineke, L.W., Panda, B.S. (eds.) ICTCSDM 2016. LNCS, vol. 10398, pp. 119–127. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64419-6_16
32. Knuth, D.E.: On the translation of languages from left to right. Inf. Control **8**(6), 607–639 (1965). https://doi.org/10.1016/S0019-9958(65)90426-2
33. Kountouriotis, V., Nomikos, C., Rondogiannis, P.: Well-founded semantics for Boolean grammars. Inf. Comput. **207**(9), 945–967 (2009). https://doi.org/10.1016/j.ic.2009.05.002
34. Kuznetsov, S.: Conjunctive grammars in greibach normal form and the lambek calculus with additive connectives. In: Morrill, G., Nederhof, M.-J. (eds.) FG 2012-2013. LNCS, vol. 8036, pp. 242–249. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39998-5_15
35. Kuznetsov, S., Okhotin, A.: Conjunctive categorial grammars. In: Proceedings of the 15th Meeting on the Mathematics of Language, MOL 2017, London, UK, 13–14 July 2017, pp. 141–151. ACL (2017)
36. Lang, B.: Deterministic techniques for efficient non-deterministic parsers. In: Loeckx, J. (ed.) ICALP 1974. LNCS, vol. 14, pp. 255–269. Springer, Heidelberg (1974). https://doi.org/10.1007/3-540-06841-4_65
37. Lange, M.: Alternating context-free languages and linear time μ-calculus with sequential composition. Electron. Notes Theor. Comput. Sci. **68**(2), 70–86 (2002). https://doi.org/10.1016/S1571-0661(05)80365-2
38. Latta, M., Wall, R.: Intersective context-free languages. In: Martin-Vide, C. (ed.) 9th Congress on Natural and Formal Languages, Reus, Spain, 20–22 December 1993, pp. 15–43 (1993)
39. Lehtinen, T., Okhotin, A.: Boolean grammars and GSM mappings. Int. J. Found. Comput. Sci. **21**(5), 799–815 (2010). https://doi.org/10.1142/S0129054110007568
40. Lehtinen, T., Okhotin, A.: On language equations $XXK = XXL$ and $XM = N$ over a unary alphabet. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 291–302. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14455-4_27
41. Okhotin, A.: Conjunctive grammars. J. Autom. Lang. Comb. **6**(4), 519–535 (2001)
42. Okhotin, A.: Conjunctive grammars and systems of language equations. Program. Comput. Soft. **28**(5), 243–249 (2002). https://doi.org/10.1023/A:1020213411126
43. Okhotin, A.: Top-down parsing of conjunctive languages. Grammars **5**(1), 21–40 (2002). https://doi.org/10.1023/A:1014219530875
44. Okhotin, A.: LR parsing for conjunctive grammars. Grammars **5**(2), 81–124 (2002). https://doi.org/10.1023/A:1016329527130
45. Okhotin, A.: Boolean grammars. Inf. Comput. **194**(1), 19–48 (2004). https://doi.org/10.1016/j.ic.2004.03.006
46. Okhotin, A.: On the equivalence of linear conjunctive grammars to trellis automata. Informatique Théorique et Applications **38**(1), 69–88 (2004). https://doi.org/10.1051/ita:2004004
47. Okhotin, A.: On the number of nonterminals in linear conjunctive grammars. Theor. Comput. Sci. **320**(2–3), 419–448 (2004). https://doi.org/10.1016/j.tcs.2004.03.002

48. Okhotin, A.: The dual of concatenation. Theor. Comput. Sci. **345**(2–3), 425–447 (2005). https://doi.org/10.1016/j.tcs.2005.07.019
49. Okhotin, A.: On the existence of a Boolean grammar for a simple programming language. In: Automata and Formal Languages, Proceedings of AFL 2005, Dobogókő, Hungary, 17–20 May 2005
50. Okhotin, A.: Generalized LR parsing algorithm for Boolean grammars. Int. J. Found. Comput. Sci. **17**(3), 629–664 (2006). https://doi.org/10.1142/S0129054106004029
51. Okhotin, A.: Recursive descent parsing for Boolean grammars. Acta Informatica **44**(3–4), 167–189 (2007). https://doi.org/10.1007/s00236-007-0045-0
52. Okhotin, A.: Notes on dual concatenation. Int. J. Found. Comput. Sci. **18**(6), 1361–1370 (2007). https://doi.org/10.1142/S0129054107005406
53. Okhotin, A.: Unambiguous Boolean grammars. Inf. Comput. **206**, 1234–1247 (2008). https://doi.org/10.1016/j.ic.2008.03.023
54. Okhotin, A.: Expressive power of LL(k) Boolean grammars. Theor. Comput. Sci. **412**(39), 5132–5155 (2011). https://doi.org/10.1016/j.tcs.2011.05.013
55. Okhotin, A.: Conjunctive and Boolean grammars: the true general case of the context-free grammars. Comput. Sci. Rev. **9**, 27–59 (2013). https://doi.org/10.1016/j.cosrev.2013.06.001
56. Okhotin, A.: Parsing by matrix multiplication generalized to Boolean grammars. Theor. Comput. Sci. **516**, 101–120 (2014). https://doi.org/10.1016/j.tcs.2013.09.011
57. Okhotin, A.: Improved normal form for grammars with one-sided contexts. Theor. Comput. Sci. **588**, 52–72 (2015). https://doi.org/10.1016/j.tcs.2015.03.041
58. Okhotin, A.: The hardest language for conjunctive grammars. In: Kulikov, A.S., Woeginger, G.J. (eds.) CSR 2016. LNCS, vol. 9691, pp. 340–351. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34171-2_24
59. Okhotin, A.: Underlying principles and recurring ideas of formal grammars. In: Klein, S.T., Martín-Vide, C., Shapira, D. (eds.) LATA 2018. LNCS, vol. 10792, pp. 36–59. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77313-1_3
60. Okhotin, A., Reitwießner, C.: Conjunctive grammars with restricted disjunction. Theor. Comput. Sci. **411**(26–28), 2559–2571 (2010). https://doi.org/10.1016/j.tcs.2010.03.015
61. Okhotin, A., Reitwießner, C.: Parsing Boolean grammars over a one-letter alphabet using online convolution. Theor. Comput. Sci. **457**, 149–157 (2012). https://doi.org/10.1016/j.tcs.2012.06.032
62. Okhotin, A., Rondogiannis, P.: On the expressive power of univariate equations over sets of natural numbers. Inf. Comput. **212**, 1–14 (2012). https://doi.org/10.1016/j.ic.2012.01.004
63. Rabkin, M.: Recognizing two-sided contexts in cubic time. In: Hirsch, E.A., Kuznetsov, S.O., Pin, J.É., Vereshchagin, N.K. (eds.) CSR 2014. LNCS, vol. 8476, pp. 314–324. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06686-8_24
64. Rosenkrantz, D.J., Stearns, R.E.: Properties of deterministic top-down grammars. Inf. Control **17**, 226–256 (1970). https://doi.org/10.1016/S0019-9958(70)90446-8
65. Rounds, W.C.: LFP: a logic for linguistic descriptions and an analysis of its complexity. Comput. Linguist. **14**(4), 1–9 (1988)
66. Szabari, A.: Alternujúce Zásobníkové Automaty (Alternating Pushdown Automata), in Slovak, diploma work (M.Sc. thesis), University of Košice (Czechoslovakia), 45 pp. (1991)

67. Terrier, V.: On real-time one-way cellular array. Theor. Comput. Sci. **141**(1–2), 331–335 (1995). https://doi.org/10.1016/0304-3975(94)00212-2
68. Terrier, V.: Some computational limits of trellis automata. In: Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E. (eds.) AUTOMATA 2017. LNCS, vol. 10248, pp. 176–186. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58631-1_14
69. Tomita, M.: An efficient augmented context-free parsing algorithm. Comput. Linguist. **13**(1), 31–46 (1987)
70. Valiant, L.G.: General context-free recognition in less than cubic time. J. Comput. Syst. Sci. **10**(2), 308–314 (1975). https://doi.org/10.1016/S0022-0000(75)80046-8
71. Wotschke, D.: The Boolean closures of the deterministic and nondeterministic context-free languages. In: Brauer, W. (ed.) GI Gesellschaft für Informatik e. V. LNCS, vol. 1, pp. 113–121. Springer, Heidelberg (1973). https://doi.org/10.1007/978-3-662-41148-3_11
72. Yoshinaka, R.: Learning conjunctive grammars and contextual binary feature grammars. In: Dediu, A.-H., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) LATA 2015. LNCS, vol. 8977, pp. 623–635. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15579-1_49
73. Zhang, Q., Su, Z.: Context-sensitive data-dependence analysis via linear conjunctive language reachability. In: Principles of Programming Languages (POPL 2017), pp. 344–358 (2017). http://dx.doi.org/10.1145/3009837.3009848
74. Zier-Vogel, R., Domaratzki, M.: RNA pseudoknot prediction through stochastic conjunctive grammars. In: The Nature of Computation: Logic, Algorithms, Applications (CiE 2013, Milan, Italy, 1–5 July 2013), Informal Proceedings, pp. 80–89 (2013)