Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 302 (2003) 365–399

Theoretical
Computer Science

www.elsevier.com/locate/tcs

A recognition and parsing algorithm for arbitrary conjunctive grammars

Alexander Okhotin[1]

Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L3N6

Received 1 November 2001; received in revised form 30 September 2002; accepted 1 November 2002
Communicated by M.A. Harrison

Abstract

Conjunctive grammars are basically context-free grammars with an explicit set intersection operation added to the formalism of rules. This paper presents a cubic-time recognition and parsing algorithm for this family of grammars, which is applicable to an arbitrary conjunctive grammar without any initial transformations.

The algorithm is in fact an extension of the context-free recognition and parsing algorithm due to Graham, Harrison and Ruzzo, and it retains the cubic time complexity of its prototype. It is shown that for the case of linear conjunctive grammars this algorithm can be modified to work in quadratic time and use linear space.

The given algorithm is then applied to solve the membership problem for conjunctive grammars in polynomial time, and subsequently to prove the problem's **P**-completeness, as well as **P**-completeness of the membership problem for linear conjunctive grammars.

Keywords: Conjunctive grammars; Recognition; Parsing; Complexity

1. Introduction

Conjunctive grammars were introduced in [7] with an intention to obtain a generative device with greater generative power than that of the context-free grammars, while retaining most of the attractive properties of the latter, such as derivation trees and polynomial-time parsing.

E-mail address: okhotin@cs.queensu.ca (A. Okhotin).

[1] This work was initiated during the author's studies at the Faculty of Computational Mathematics and Cybernetics of Moscow State University.

0304-3975/03/$ - see front matter
PII: S0304-3975(02)00853-8

A simple $O(n^3)$ parsing algorithm for conjunctive grammars, akin to the classical Cocke–Kasami–Younger (CKY) context-free parsing algorithm, was presented in [7]. It is applicable to conjunctive grammars in the binary normal form (which is a generalization of the context-free Chomsky normal form for the case of conjunctive grammars), and it has been proved that every conjunctive grammar can be effectively transformed to this normal form. However, since there is no known polynomial upper bound for the complexity of this transformation, the time complexity of the existing version of the algorithm in the worst case depends exponentially on the size of the grammar. In addition, the normal form requirement is especially inconvenient for practical applications, where the derivation tree corresponding to the original grammar is being sought.

A top-down LL(k)-style parsing algorithm for conjunctive grammars that does not require the grammar to be in the normal form was developed in [9]. Like its context-free prototype, this algorithm attempts to construct the leftmost derivation of a given string; the main innovation is the *tree-structured pushdown*, which is used to organize derivations from several conjuncts at the same time and to ensure that all of them derive the same string, thus simulating intersection of languages. This top-down algorithm can also be implemented using the recursive descent technique. While the complexity of the algorithm can be as high as exponential for some grammars, it is typically linear, and it is known to be linear for the intersection closure of the context-free LL(k) languages. However, deterministic parsing tables for the algorithm given in [9] can be constructed only for a proper subset of all conjunctive grammars; moreover, this subset is not even recursively enumerable, and therefore some approximation techniques must be used to compute the parsing table, which potentially lead to extra entries in the table and thus to nondeterminism.

The task of constructing a universal recognition and parsing algorithm for conjunctive grammars that would be applicable to any grammar and have polynomial time complexity has so far remained unsolved.

Turning back to the context-free grammars (which can naturally be viewed as a special case of conjunctive grammars, where the use of intersection is prohibited), the first parsing algorithm for general context-free languages was independently discovered by Cocke, Kasami and Younger (see bibliography in [4]); it requires the grammar to be in Chomsky normal form, which means quadratic blowup of the size of the grammar. An efficient parsing algorithm for arbitrary context-free grammars was developed by Earley [1]; its data structures are quite different from those of CKY. Finally, the ideas of CKY and Earley were put together in a universal parsing algorithm due to Graham, Harrison and Ruzzo [3–5], which uses simple data structures similar to CKY, performs computations similar to Earley and is able to compute null derivations and chain derivations in one step. Fast performance even on large grammars and relatively easy implementation makes the algorithm very suitable for nontrivial practical applications, e.g., natural language processing.

This paper develops a new cubic-time parsing algorithm for general conjunctive grammars of the general form, which is in fact an extension of Graham–Harrison–Ruzzo algorithm. This new algorithm has been implemented in the parser generator [12].

In Section 2 the definition of a conjunctive grammar is given, and several important results are quoted. The new parsing algorithm is presented and explained in Section 3; a proof of its correctness is provided. Section 4 describes two different methods of constructing derivation trees of recognized strings, and gives a way to represent these trees in memory using no more than quadratic space. The case of linear conjunctive grammars is investigated in Section 5, where it is shown that the given algorithm can be modified to work in $O(n^2)$ time and use $O(n)$ space (where n is the length of the string being recognized) on this subclass of conjunctive grammars. Theoretical implications of the algorithm are discussed in Section 6, where the membership problem for conjunctive grammars and the membership problem for linear conjunctive grammars are shown to be **P**-complete.

2. Preliminaries

2.1. Basic properties

A conjunctive grammar [7] is defined in the same way as a context-free grammar, but its rules may consist of multiple *conjuncts*.

Definition 1. A conjunctive 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 grammar rules, each of the form

$$A \to \alpha_1 \& \ldots \& \alpha_n \quad (A \in N,\ n \geqslant 1,\ \alpha_i \in (\Sigma \cup N)^* \text{ for all } i), \tag{1}$$

where the strings α_i are distinct, and their order is considered insignificant; $S \in N$ is a nonterminal designated as the start symbol.

For each rule of the form (1) and for each i $(1 \leqslant i \leqslant n)$, $A \to \alpha_i$ is called a conjunct. Let conjuncts (G) denote the sets of all conjuncts.

Let V be the union of Σ and N. We shall use three special symbols: '**(**', '**&**' and '**)**'; it is assumed that none of them is in V. Define $\bar{V} = \Sigma \cup N \cup \{$'**(**', '**&**', '**)**'$\}$.

A conjunctive grammar generates strings by deriving them from the start symbol, generally in the same way as the context-free grammars do. Intermediate objects used in course of a derivation are in this case formulae under the basis of concatenation and conjunction, represented as strings over $\bar{V}$:

Definition 2. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Conjunctive formulae are defined inductively on their structure:

- Every terminal symbol, every nonterminal symbol, and the empty string ε are formulae.
- If $\mathscr{A}$ and $\mathscr{B}$ $(\mathscr{A}, \mathscr{B} \neq \varepsilon)$ are formulae, then $\mathscr{A}\mathscr{B}$ is a formula.
- If $\mathscr{A}_1, \ldots, \mathscr{A}_n$ $(n \geqslant 1)$ are formulae, then $(\mathscr{A}_1 \& \ldots \& \mathscr{A}_n)$ is a formula.

Denote the set of conjunctive formulae as $\mathscr{F}$.

Definition 3. Let $G=(\Sigma,N,P,S)$ be a conjunctive grammar. Define $\overset{G}{\Longrightarrow}$, the relation of derivability in one step on the set of conjunctive formulae.

(i) For any $s',s''\in\tilde{V}^*$ and $A\in N$, such that $s'As''\in\mathscr{F}$, and for all $A\to\alpha_1\&\ldots\&\alpha_n\in P$,

$$s'As'' \overset{G}{\Longrightarrow} s'(\alpha_1\&\ldots\&\alpha_n)s'' \tag{2}$$

(ii) (the gluing rule) For any $s',s''\in\tilde{V}^*$, $n\geqslant 1$ and $w\in\Sigma^*$, such that $s'(\underbrace{w\&\ldots\&w}_{n})s''\in\mathcal{F}$,

$$s'(\underbrace{w\&\ldots\&w}_{n})s'' \overset{G}{\Longrightarrow} s'ws'' \tag{3}$$

Let $\overset{G}{\Longrightarrow}{}^*$ denote the reflexive and transitive closure of $\overset{G}{\Longrightarrow}$, and denote the transitive closure of $\overset{G}{\Longrightarrow}$ as $\overset{G}{\Longrightarrow}{}^+$.

Definition 4. Let $G=(\Sigma,N,P,S)$ be a conjunctive grammar. The language of a formula is the set of all terminal strings derivable from the formula: $L_G(\mathscr{A})=\{w\in\Sigma^*\mid\mathscr{A}\overset{G}{\Longrightarrow}{}^* w\}$. The language generated by the grammar is the language generated by its start symbol: $L(G)=L_G(S)$.

Since each application of a rule from P adds a pair of parentheses to the formula and each application of the gluing rule deletes a pair of parentheses, it is obvious that any successful derivation of a terminal string from a nonterminal has even length. Accordingly, some inductive proofs on the length of a derivation will have basis 0 or 2 and induction step $n\to n+2$.

The following theorem quoted from [7] shows that the language of a formula inductively depends on its structure, and the operations with formulae correspond to operations with languages.

Theorem 1. *Let $G=(\Sigma,N,P,S)$ be a conjunctive grammar. Let $\mathscr{A}_1,\ldots,\mathscr{A}_n,\mathscr{B}$ be formulae, let $A\in N$, let $a\in\Sigma$. Then,*

$$L_G(\varepsilon)=\{\varepsilon\}, \tag{4a}$$

$$L_G(a)=\{a\}, \tag{4b}$$

$$L_G(A)=\bigcup_{A\to\alpha_1\&\ldots\&\alpha_m\in P} L_G((\alpha_1\&\ldots\&\alpha_m)), \tag{4c}$$

$$L_G(\mathscr{A}\mathscr{B})=L_G(\mathscr{A})\cdot L_G(\mathscr{B}), \tag{4d}$$

$$L_G((\mathscr{A}_1\&\ldots\&\mathscr{A}_n))=\bigcap_{i=1}^{n} L_G(\mathscr{A}_i). \tag{4e}$$

The total number of symbols in the string representation of a grammar will be used as the measure of its descriptional complexity:

Definition 5. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Define the size of the grammar,

$$|G| = \sum_{A \to \alpha_1 \& \ldots \& \alpha_n \in P} (n + 1 + \sum_{i=1}^{n} |\alpha_i|). \tag{5}$$

We shall also compute and use the set of *nullable* nonterminals of a given grammar — i.e., those nonterminals that derive the empty string.

Definition 6. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Define

$$\text{NULLABLE}(G) = \{A \mid A \in N,\ A \Longrightarrow^* \varepsilon\}. \tag{6}$$

The set $\text{NULLABLE}(G)$ can be computed by the same nested-set technique as in the case of context-free grammars [7]; let us also observe that the time complexity of that algorithm on a RAM is $O(|G|^2)$.

2.2. Derivation trees

Each conjunctive derivation of the form $A \Longrightarrow \cdots \Longrightarrow \mathscr{A}$ $(A \in N)$ can be represented as a tree with shared leaves. The leaves of the tree are labeled with symbols from $\Sigma \cup N \cup \{\varepsilon\}$. Nonepsilon leaves correspond to terminal and nonterminal symbols from $\mathscr{A}$, and a leaf can have in-degree of more than one only if it is labeled with a terminal symbol. Internal vertices of the tree are labeled with the rules used in the derivation. For each vertex, all outgoing arcs are considered ordered.

A derivation tree is constructed inductively on the length of derivation:

- The tree corresponding to a 0-step derivation $A \Longrightarrow^* A$ is a single vertex labeled with A.
- The tree corresponding to an $(n+1)$-step derivation of the form

$$A \Longrightarrow \ldots \Longrightarrow s_1 B s_2 \Longrightarrow s_1(\beta_1 \& \ldots \& \beta_m) s_2 \tag{7}$$

is made from the tree corresponding to the first n steps of the derivation (7) in the following way:

(1) The leaf corresponding to B (currently labeled with the nonterminal B) is relabeled with the rule $B \to \beta_1 \& \ldots \& \beta_m$.
(2) For each string $\beta_i = s_{i1} \ldots s_{il_i}$ $(s_{ij} \in \Sigma \cup N,\ l_i \geqslant 0)$, if $|\beta_i| > 0$, then $|\beta_i|$ new leaves labeled with the symbols $s_{i1}, \ldots, s_{il_i}$, are created; if $|\beta_i| = 0$, then a single new leaf labeled with ε is created.
(3) The vertex recently relabeled with $B \to \beta_1 \& \ldots \& \beta_m$ is connected to all newly created leaves; the arcs are ordered in accordance with the order of the symbols in the rule.

This case is shown in Fig. 1(a); the signs "&" between the arcs are there entirely for illustrative purposes; in fact, the position of these signs can be inferred from the rule $B \to \beta_1 \& \ldots \& \beta_m$.

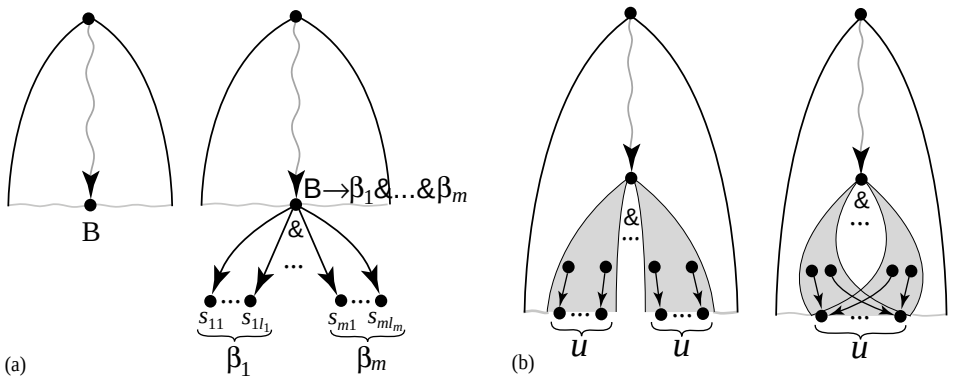

Fig. 1. Construction of derivation trees.

- The tree corresponding to an $(n+1)$-step derivation

$$A \Longrightarrow \ldots \Longrightarrow s_1(u\&\ldots\&u)s_2 \Longrightarrow s_1us_2 \tag{8}$$

is made from the tree corresponding to the first n steps of (8) by identifying for every fixed i $(1 \leqslant i \leqslant |u|)$ the leaves corresponding to ith characters of all instances of u being glued. This case is illustrated in Figure 1(b).

For each internal vertex of the tree labeled with some rule $A \to \alpha_1\&\ldots\&\alpha_m$ and for each i $(1 \leqslant i \leqslant m)$, the immediate descendants of the vertex corresponding to the symbols from α_i, together with the subtrees they start, will be collectively referred to as *a group of descendants corresponding to the conjunct* $A \to \alpha_i$.

It is essential to label internal vertices with rules rather than with nonterminals, because otherwise it would not be possible to determine the boundary between groups of descendants — e.g., given a derivation tree and a vertex A with descendants b, B and C, there would be no immediate way to distinguish between the rules $A \to bB\&C$, $A \to b\&BC$, $A \to bBC$ and $A \to b\&B\&C$.

Every internal vertex of a tree and every leaf labeled with a nonterminal or with the empty string is connected to the root vertex by exactly one path (this easily follows from the construction of the tree). On the other hand, a terminal leaf of a tree can be connected to the root vertex by multiple paths, which makes the derivation trees for conjunctive grammars somewhat harder to deal with than the context-free derivation trees.

Let a vertex z be called a *common ancestor* of vertices x and y if there exist directed paths from z to x and from z to y. A vertex z is a *least common ancestor* of x and y if z is a common ancestor of x and y and there are no other common ancestors of x and y among the descendants of z. In a tree in the strict mathematical sense, every two vertices have a uniquely determined least common ancestor. This is not the case in respect to the derivation trees with shared leaves we have just defined, due to the potential multiplicity of paths from the root vertex to a leaf: while every pair of vertices is still guaranteed to have at least one least common ancestor, two

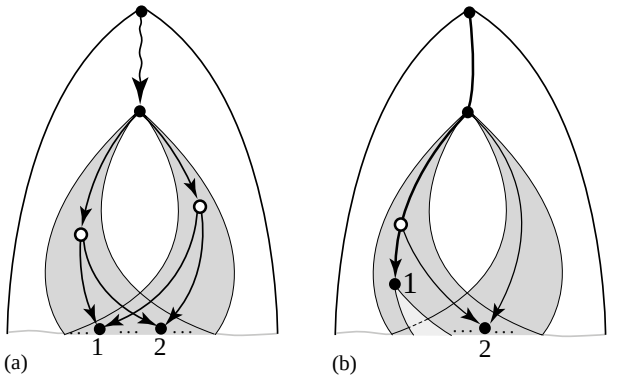

Fig. 2. (a) Multiple least common ancestors of two terminal leaves, 1 and 2; (b) The unique least common ancestor of an internal vertex 1 and a terminal leaf 2.

terminal leaves can possibly have multiple least common ancestors, as shown in the example in Fig. 2(a), in which arrows denote paths. However, since every internal vertex is connected to the root by a single path, an internal vertex and an arbitrary vertex always have unique least common ancestor (see Fig. 2(b)), because all of their common ancestors are located on this single path, and thus one and only one of them is the least.

Let us summarize the mentioned properties of the conjunctive derivation trees:

Proposition 1. *Every derivation* $A \Longrightarrow \cdots \Longrightarrow \mathscr{A}$ $(A \in N)$ *can be represented as a tree with shared leaves, in which the leaves are labeled with elements of* $\Sigma \cup N \cup \{\varepsilon\}$ *corresponding to the symbols forming* $\mathscr{A}$, *and the internal vertices are labeled with the rules used in the derivation.*

Every internal vertex or a leaf labeled with $N \cup \{\varepsilon\}$ *has in-degree of one and is connected to the root by a single path*; *terminal leaves may have multiple ancestors and thus can be connected to the root by multiple paths.*

Every group of two or more vertices of the tree has one or more least common ancestors; unless all of these vertices are terminal leaves, their least common ancestor is certain to be unique.

3. Construction and explanation of the algorithm

In this section we introduce, explain and prove correct the new parsing algorithm for conjunctive grammars which generalizes the Graham–Harrison–Ruzzo (GHR) context-free algorithm. While it is not required for the reader to be familiar with the original algorithm, this knowledge will definitely help to understand how the new algorithm works, as most of the underlying concepts are in this or that form derived from the similar concepts used in GHR algorithm. In the following, the original GHR parsing algorithm shall be referred to as *the context-free case*. For every conjunctive grammar

having only one-conjunct rules, the algorithm we shall now define behaves exactly like its prototype.

Let us start with introducing some additional terminology. Some of it — namely, prefix reachability and the mappings *finished* and $\overline{finished}$ — is new (although similar notions exist in the context-free case, they are too trivial to have proper names); the rest of terms are inherited from the context-free case and generalized for conjunctive grammars. Our terminology is most close to the version of GHR algorithm described in [5].

3.1. Prefix reachability

Every parsing algorithm in one or another form considers several derivations of the input string, until it finds one that succeeds or proves that there are none. The amount of work to be done can be substantially reduced if we rule out obviously impossible derivations by checking some easily computable necessary condition of nonterminal's "relevancy" at some particular position in the input string.

In the context-free case, for a prefix u of the input string uv and for a nonterminal $B \in N$, it is being checked whether $S \Longrightarrow^* uB\delta$ for some $\delta \in V^*$ (using the terminology of [4], B *follows* u) or, more generally, whether for given nonterminals A and B and for a string $u \in \Sigma^*$ there is $\delta \in V^*$, such that $A \Longrightarrow^* uB\delta$. If this does not hold for some S, B and u, then, clearly, there is no point in considering derivations from B at this point of computation.

We shall now extend this relation for the case of conjunctive grammars:

Definition 7 (Prefix reachability). Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. A nonterminal B is said to be *reachable* from nonterminal A *by prefix string* $u \in \Sigma^*$ (denoted as $A \overset{u}{\rightsquigarrow} B$), if there exists a number $p \geqslant 0$, a partition

$$u = u_1 \dots u_p \quad (u_i \in \Sigma^*) \tag{9}$$

and a sequence of nonterminals

$$C_0, \dots, C_p \quad (C_0 = A, C_p = B), \tag{10}$$

such that for any r $(1 \leqslant r \leqslant p)$ there is a rule

$$C_{r-1} \to \eta_1 \& \dots \& \eta_m, \tag{11}$$

and a representation of one of the conjuncts of that rule as

$$\eta_l = \gamma_r C_r \delta_r \quad (\gamma_r, \delta_r \in (\Sigma \cup N)^*), \tag{12}$$

where

$$\gamma_r \Longrightarrow^* u_r \tag{13}$$

The definition of prefix reachability is illustrated in Fig. 3, where every vertex C_{r-1} labeled with rule (11) is assumed to have additional groups of descendants corresponding to the conjuncts of (11) other than the selected conjunct (12); nothing is supposed

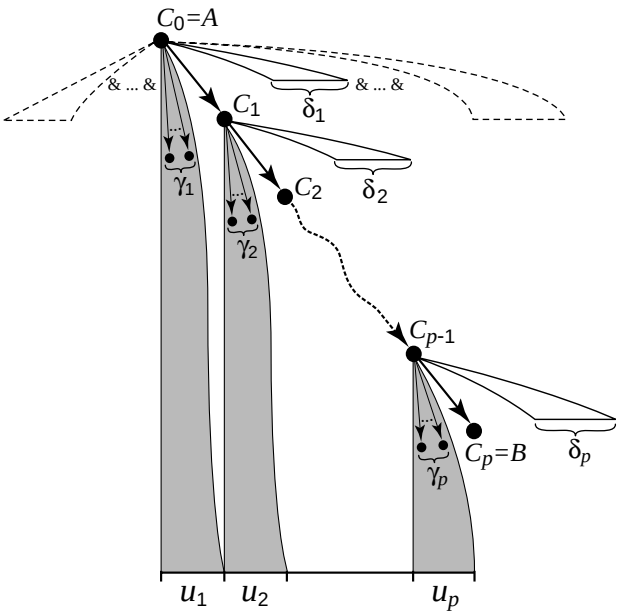

Fig. 3. Prefix reachability from A to B by the string $u_1 \dots u_p$.

to be derived from these additional conjuncts, and in the figure they are actually shown (in dotted lines) only for the root vertex of the tree. The labels (11) of the vertices are omitted for the reasons of clarity and the names of the corresponding nonterminals $C_0, \dots, C_p$ are written instead. The subtrees shown in grey are complete derivations trees of the strings u_r (for all $1 \leqslant r \leqslant p$) from the strings γ_r.

Prefix reachability can be called "reflexive" in the sense that each nonterminal is reachable from itself by the prefix ε. The following property of prefix reachability might be regarded as its "transitivity":

Lemma 1. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar, let $A, B, C \in N$, $u, v \in \Sigma^*$. If $A \overset{u}{\rightsquigarrow} B$ and $B \overset{v}{\rightsquigarrow} C$, then $A \overset{uv}{\rightsquigarrow} C$.*

The proof of Lemma 1 directly follows from Definition 7: the necessary factorization of the concatenation $u \cdot v$ is obtained by concatenating the factorizations for u and v.

It follows that the binary relation $\overset{\varepsilon}{\rightsquigarrow} \subseteq N \times N$ is reflexive and transitive. In fact, $\overset{\varepsilon}{\rightsquigarrow}$ equals the reflexive and transitive closure of the relation $\{(A, B) \mid$ there exists $A \to \alpha B \beta \in \text{conjuncts}(G)$, such that $\alpha \Longrightarrow^* \varepsilon\}$.

In this algorithm, prefix reachability of a nonterminal A from S by substring u of input string uv, as we shall shortly see, is made into a prerequisite for considering any rules for A in position $|u|$. This condition is being "enforced" by the so-called Predictor described later in Section 3.4.

3.2. Dotted conjuncts

Dotted conjuncts are objects of exactly the same form as the dotted rules used in the context-free case.

Definition 8. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. $A \to \alpha \cdot \beta$ is called a dotted conjunct, if $A \to \alpha\beta$ is a conjunct, i.e. if there is a rule $A \to \eta_1 \& \ldots \& \eta_n \in P$, such that $\alpha\beta = \eta_i$ for some i.

The set of all dotted conjuncts will be denoted as $dottedconjuncts(G)$; for every G it is a finite set.

The following definition of an (i,j)-*consistent dotted conjunct* generalizes the notion of (i,j)-consistent dotted rule used in the context-free case:

Definition 9 ((i,j)-consistent dotted conjunct). Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $w = a_1 \ldots a_n \in \Sigma^*$ $(n \geqslant 0)$ be a string. Let $0 \leqslant i \leqslant j \leqslant n$. A dotted conjunct $A \to \alpha \cdot \beta$ is said to be (i,j)-consistent, if

$$S \overset{a_1 \ldots a_i}{\rightsquigarrow} A, \tag{14a}$$

$$\alpha \Longrightarrow^* a_{i+1} \ldots a_j. \tag{14b}$$

Definition 10 ((i,j)-consistent and (i,j)-complete sets). Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $w = a_1 \ldots a_n \in \Sigma^*$ $(n \geqslant 0)$ be a string. A set of dotted conjuncts is called (i,j)-consistent if all of its members are (i,j)-consistent. A set of dotted conjuncts is called (i,j)-complete if it contains all (i,j)-consistent dotted conjuncts.

It must be noted that (i,j)-consistent conjuncts and (i,j)-consistent/complete sets are defined with respect to a fixed input string, and the numbers i and j refer to positions in this string.

Given an input string $w = a_1 \ldots a_n \in \Sigma^*$, the algorithm will construct an upper-triangular matrix $\{t_{ij}\}$ $(0 \leqslant i \leqslant j \leqslant n)$ of sets of dotted conjuncts, such that every entry t_{ij} will be (i,j)-consistent and complete. Afterwards, the $(0,n)$-consistent and complete set t_{0n} will allow to determine whether w is in $L(G)$, and, in case of a positive answer, it will be possible to use the whole matrix to construct the derivation tree of w.

3.3. From sets of dotted conjuncts to sets of nonterminals

Consider the following problem: given a set of dotted conjuncts R, determine the set of nonterminals, such that for every string $w = a_1 \ldots a_n$ and for every $1 \leqslant i \leqslant j \leqslant n$ the (i,j)-consistency of R would imply that each of these nonterminals derives the string $a_{i+1} \ldots a_j$.

An obvious answer is the set of all nonterminals A, such that some rule for A consists entirely of the conjuncts that appear in R with the dot at the end: for example,

if $A \to \alpha \cdot \in R$, $A \to \beta \cdot \in R$ and there is a rule $A \to \alpha\&\beta$, then A is included in the resulting set.

A more thorough approach is to consider *indirect* implications of the dotted conjuncts in R. For instance, if $A \to \alpha \cdot \in R$, $B \to \beta \cdot \in R$, $C \to \gamma \cdot \in R$, there are rules $A \to \alpha$, $B \to \beta$ and $C \to \gamma\&\mu\&\nu$, where μ can derive the formula **((***B***))** and ν can derive **(((***A***)&(***B***))**, then it is possible first to conclude that A and B must be in the resulting set, and consequently find out that C should also be in this set.

In the context-free case this approach means essentially the precomputation of chain derivations — those of the form $A \Longrightarrow \cdots \Longrightarrow B$ $(A, B \in N)$. In the case of conjunctive grammars we have to precompute more complex relations between entities, which cannot be represented in terms of derivations alone; an algorithmic nested-set definition shall be used.

Now define the mappings $\mathit{finished}, \overline{\mathit{finished}} : 2^{\mathit{dottedconjuncts}(G)} \to 2^N$ corresponding to the first and the second approaches respectively:

Definition 11. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let R be a set of dotted conjuncts, let ρ_R be a monotone operator on the set of subsets of N, defined as

$$\rho_R(Q) = Q \cup \{A \in N \mid \text{ there is a rule } A \to \alpha_1 \& \ldots \& \alpha_m : \text{ for all } i \\ (1 \leqslant i \leqslant m)\ A \to \alpha_i \in R \textbf{ or } \alpha_i \in \textsc{Nullable}^* \cdot Q \cdot \textsc{Nullable}^*\}. \tag{15}$$

The sets $\mathit{finished}(R), \overline{\mathit{finished}}(R)$ are defined as

$$\mathit{finished}(R) = \rho_R(\emptyset), \tag{16a}$$

$$\overline{\mathit{finished}}(R) = \mu Q \cdot \rho_R(Q), \tag{16b}$$

where $\mu Q \cdot \rho_R(Q)$ is the least fixed point of the operator ρ_R.

Intuitively, ρ_R^k "sees" all chain derivations up to depth k, while its least fixed point fathoms chain derivations of unbounded depth. Due to the finiteness of the set N, this least fixed point can be represented as a finite superposition $\rho_R^{n_0}(\emptyset)$ for some n_0. Obviously, $\mathit{finished}(R) \subseteq \overline{\mathit{finished}}(R)$ for any R.

The following three technical lemmas state the main properties of these mappings:

Lemma 2. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $w = a_1 \ldots a_n$ be a string. Let R be an (i,j)-consistent set of dotted conjuncts, let $A \in \overline{\mathit{finished}}(R)$. Then $A \Longrightarrow^* a_{i+1} \ldots a_j$.*

Proof. Let $k = k(A)$ be the minimal number such that $A \in \rho_R^k(\emptyset)$. The proof is an induction on k.

Let $k \geqslant 1$ and let $A \in \rho_R^k(\emptyset) \setminus \rho_R^{k-1}(\emptyset)$. Then there is a rule $A \to \alpha_1 \& \ldots \& \alpha_m$, such that for all l $(1 \leqslant l \leqslant m)$ $A \to \alpha_l \cdot \in R$ or $\alpha_l \Longrightarrow^* B$ for some $B \in \rho_R^{k-1}(\emptyset)$. In the former case, $\alpha_l \Longrightarrow^* a_{i+1} \ldots a_j$ due to (i,j)-consistency of $A \to \alpha_l \cdot$. In the latter case (which is possible only if $k \geqslant 2$ – i.e., in the proof of the induction step), $B \Longrightarrow^* a_{i+1} \ldots a_j$

by the induction hypothesis, and hence we can construct a derivation

$$\alpha_l \Longrightarrow B \Rightarrow a_{i+1} \dots a_j \tag{17}$$

The derivation of the string $a_{i+1} \dots a_j$ from A is easily constructed out of the derivations from the conjuncts of the rule $A \to \alpha_1 \& \dots \& \alpha_m$. □

Lemma 2 provides a necessary condition of the membership of nonterminals in $\overline{finished}(R)$; let us now devise some sufficient conditions.

A simple sufficient condition is provided in the following lemma:

Lemma 3. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $w = a_1 \dots a_n$ be a string. Let R be an (i,j)-complete set of dotted conjuncts. Let A be a nonterminal, such that $S \overset{a_1 \dots a_j}{\rightsquigarrow} A$ and $A \Longrightarrow^* a_{i+1} \dots a_j$. Then $A \in finished(R)$.*

Proof. Since $A \Longrightarrow^* a_{i+1} \dots a_j$, there exists a rule $A \to \alpha_1 \& \dots \& \alpha_m$, such that every α_k $(1 \leqslant k \leqslant m)$ derives $a_{i+1} \dots a_j$.

Together with $S \overset{a_1 \dots a_j}{\rightsquigarrow} A$, this implies that every dotted conjunct $A \to \alpha_k \cdot$ is (i,j)-consistent and therefore must be in the (i,j)-complete set R. Consequently, $A \in finished(R)$. □

The condition in Lemma 3 relies upon (i,j)-completeness of the table and thus is somewhat hard to meet before the construction of the table is completed. The following lemma gives a sufficient condition of the membership of nonterminals in $\overline{finished}(R)$ under weaker assumptions about the set R:

Lemma 4. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $A \in N$. Let*

$$A \Longrightarrow \dots \Longrightarrow w \tag{18}$$

be a derivation of some string $w \in \Sigma^*$ from the nonterminal A. Let $\mathcal{N}$ be the set of all those internal vertices in the derivation tree of (18) that are common ancestors of all terminal leaves at once (by Proposition 1, $|\mathcal{N}| \geqslant 1$). Let $\mathcal{N}' \subseteq \mathcal{N}$ be those among these vertices that have at least one group of descentants containing no vertices from $\mathcal{N}$.

Let R be a set of dotted conjuncts, such that for every vertex from $\mathcal{N}'$ (labeled with some rule $B \to \beta_1 \& \dots \& \beta_n$) and for each i-th group of its descentants containing no vertices from $\mathcal{N}$, the dotted conjunct $B \to \beta_i \cdot$ is in R.

Then the nonterminal A is in $\overline{finished}(R)$.

Before proceeding to the proof, let us explain the construction of the sets $\mathcal{N}$ and $\mathcal{N}'$ on a simple example. Consider the schematic derivation tree in Fig. 4, in which all the vertices shown are assumed to be labeled with some rules and have descendants corresponding to these rules. Here the set $\mathcal{N}$ consists of all the vertices above the grey dotted line, i.e., the root vertex, the vertices x_1, x_2 and x_5 from the light grey part and the vertices y_1 and y_2 from the dark grey part. The vertices x_3 and x_4 are not in $\mathcal{N}$, because each of them is an ancestor only to a proper substring of the whole string uv.

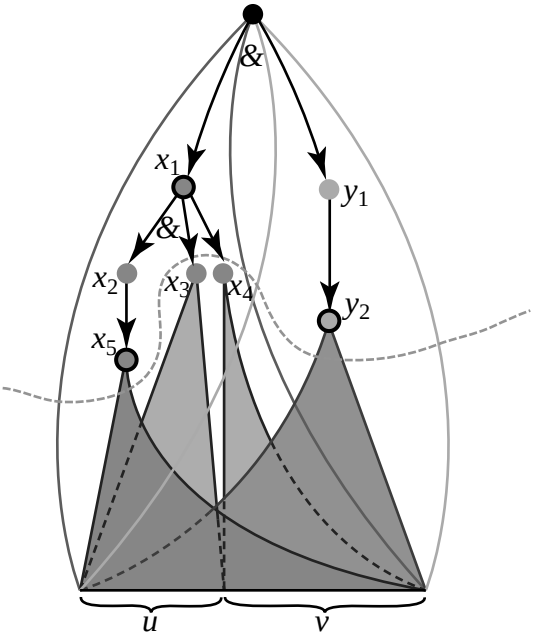

Fig. 4. An example referring to Lemma 4.

The immediate descendants of x_5 and y_2 are also assumed to be such that none of them is an ancestor to all terminal leaves of the tree.

Let us see which vertices from $\mathscr{N}$ are in the set $\mathscr{N}'$. The root vertex has two groups of descentants (light grey and dark grey), and each of these groups contains vertices from $\mathscr{N}$; therefore, the root vertex is not in $\mathscr{N}'$. The vertex x_1 has two groups of descendants; one of them contains vertices from $\mathscr{N}$ (x_2 and x_5), while the other does not; due to the latter, $x_1 \in \mathscr{N}'$. The vertices x_2 and y_1 have one group of descendants each, and for both of them this only group contains vertices from $\mathscr{N}$ — therefore, neither x_2 nor y_1 is in $\mathscr{N}'$. Finally, x_5 and y_2 are both in $\mathscr{N}'$, because none of their descentants are in $\mathscr{N}$. In Fig. 4, the vertices from $\mathcal{N}'$ are marked with black rims.

Lemma 4 actually states that if for some derivation tree the set R contains finished dotted conjuncts for any group of descendants (of any vertex from $\mathscr{N}'$) that contains no vertices from $\mathscr{N}$ (e.g., the descendants of x_5 and y_2, and the second group of descendants of x_1 in Fig. 4), then the mapping *finished* will be able to "reconstruct" the whole upper part of the derivation tree and make a conclusion that the nonterminal associated with the root vertex derives the string under consideration.

Proof of Lemma 4. We prove that for every nonterminal A satisfying the conditions of the lemma there exists a number $k > 0$, such that $A \in \rho_R^k(\emptyset)$. The argument is an induction on the length of derivation (18), which essentially means induction on the structure of the corresponding derivation tree.

Let v_A denote the root vertex of the derivation tree of (18), labeled with some rule

$$A \to \alpha_1 \& \ldots \& \alpha_m \tag{19}$$

For each conjunct $A \to \alpha_i$ from this rule, we consider two possible cases; only the first of them is possible in the basis of induction, when the derivation (18) has length two, and both are possible in the induction step:

- The group of descendants corresponding to this conjunct contains no vertices from $\mathcal{N}$. Then, by the condition of the lemma, $A \to \alpha_i \cdot \in R$.
- The corresponding group of descendants of v_A contains one or more vertices from $\mathcal{N}$. Let $\alpha_i = s_1 \ldots s_l$ ($l > 0$), where $s_j \in \Sigma \cup N$ (the case $\alpha_i = \varepsilon$ is clearly impossible). For each j ($1 \leqslant j \leqslant l$), let u_j be the substring of w formed by the terminal descendants of s_j; obviously, $w = u_1 \ldots u_l$.
  It is easy to prove that there exists a number t ($1 \leqslant t \leqslant l$), such that $s_t \in N$ and $u_t = w$. Indeed, if that did not hold, then either (i) none of the strings $u_1, \ldots, u_l$ would coincide with the whole of w, or (ii) there would exist t, such that $u_t = w$ and $s_t \in \Sigma$ – this would imply that $w = u_t = s_t$. In both cases none of the l subtrees starting from the descendants of v_A corresponding to $s_1, \ldots, s_l$ would contain vertices that are common ancestors of all terminal leaves, which would contradict our assumption that the i-th group of descendants of v_A contains vertices from $\mathcal{N}$.
  So let t be a number, such that $s_t \in N$ and $u_t = w$. Then $u_1 = \cdots = u_{t-1} = u_{t+1} = \ldots = u_l = \varepsilon$ and consequently $s_1, \ldots, s_{t-1}, s_{t+1}, \ldots, s_l \in \text{NULLABLE}$. Denote $B = s_t$ and denote the corresponding vertex of the derivation tree as v_B. Clearly, v_B is in $\mathcal{N}$; applying the induction hypothesis to the subtree with v_B as a root, we obtain that there is a number $k_i \geqslant 0$, such that $B \in \rho_R^{k_i}(\emptyset)$. Since the rest of the symbols in the conjunct are nullable nonterminals,

$$\alpha_i \in \text{NULLABLE}^* \cdot \rho_R^{k_i}(\emptyset) \cdot \text{NULLABLE}^* \tag{20}$$

We have proved that for any conjunct $A \to \alpha_i$ of the rule (19) either there exists a number k_i, such that there exists a representation (20), or $A \to \alpha_i \cdot \in R$ (in this case let us define k_i as 0).

Since ρ_R is monotonous, we can consider the maximum of these k_i and conclude that for any i either $A \to \alpha_i \cdot \in R$, or $\alpha_i \in \text{NULLABLE}^* \cdot \rho_R^{\max(k_1,\ldots,k_l)}(\emptyset) \cdot \text{NULLABLE}^*$, and therefore, by the definition of ρ_R, $A \in \rho_R^{\max(k_1,\ldots,k_l)+1}(\emptyset) \subseteq \overline{\mathit{finished}}(R)$. □

Let us also mention the complexity of these operations as a function of the size of the grammar:

- For a given set of dotted conjuncts R, the set $\mathit{finished}(R)$ can be computed in time $O(|G|)$ by considering all the rules and for each of them looking for the appropriate dotted conjuncts in R.
- The set $\overline{\mathit{finished}}(R)$ can be computed in time $O(|G|^2)$ by a straightforward least fixed point algorithm, which will converge in at most $O(|G|)$ steps.

3.4. Operations with dotted conjuncts

All operations with sets of symbols and sets of dotted rules that were used in the original context-free recognition algorithm will now be redefined.

The following "product" operation is used to advance the dot in dotted conjuncts; it is actually a far-going generalization of the cartesian product of sets of nonterminals used in the Cocke–Kasami–Younger context-free algorithm.

Definition 12. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let Q be a set of dotted conjuncts. Let $R \subseteq V$.

$$Q \times R = \{A \to \alpha B \beta \cdot \gamma \mid A \to \alpha \cdot B\beta\gamma \in Q, \beta \Longrightarrow^* \varepsilon, B \in R\}. \tag{21}$$

If Q is an (i,k)-consistent set of dotted conjuncts and all symbols in R are known to derive some substring $a_{k+1} \dots a_j$, then $Q \times R$ will be an (i,j)-consistent set.

The set $Q \times R$ can be computed in time $O(|G|)$ by considering all dotted conjuncts $A \to \alpha \cdot x s_1 \dots s_k \in Q$ (where $\alpha \in V^*$, $x \in R$, $s_i \in V$), for each adding all dotted conjuncts $A \to \alpha x s_1 \dots s_i \cdot s_{i+1} \dots s_k$, such that $0 \leqslant i \leqslant k$ and $s_1, \dots, s_i \in \text{NULLABLE}(G)$, to the result.

Now we apply the mappings $\mathit{finished}(R)$ and $\overline{\mathit{finished}}(R)$ to define two different products of a set of dotted conjuncts by a set of dotted conjuncts:

Definition 13. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let Q and R be sets of dotted conjuncts. Define

$$Q \times R = Q \times \mathit{finished}(R), \tag{22a}$$

$$Q * R = Q \times \overline{\mathit{finished}}(R), \tag{22b}$$

The complexity results obtained above imply that $Q \times R$ can be computed in time $O(|G|)$ and $Q * R$ can be computed in time $O(|G|^2)$ for any sets of dotted conjuncts Q and R.

While it is obvious that $Q \times R$ is a subset of $Q * R$ for all Q and R (since, as mentioned above, $\mathit{finished}(R) \subseteq \overline{\mathit{finished}}(R)$ for any set of dotted conjuncts R), the following sufficient condition of their coincidence is worth being established:

Lemma 5. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $w = a_1 \dots a_n$ be a string. Let $0 \leqslant i \leqslant k \leqslant j \leqslant n$ and let Q and R be sets of dotted conjuncts, such that Q is (i,k)-consistent, while R is (k,j)-complete. Then $Q \times R = Q * R$.

Proof. By the definition of product, every dotted conjunct in $Q * R$ is of the form $A \to \alpha B \beta \cdot \gamma$, where

$$A \to \alpha \cdot B\beta\gamma \in Q, \tag{23a}$$

$$B \in \overline{\mathit{finished}}(R) \tag{23b}$$

and β derives the empty string. By (23a) and (i,k)-consistency of Q, the dotted conjunct $A \to \alpha \cdot B\beta\gamma$ is (i,k)-consistent, and therefore $S \overset{a_1 \ldots a_i}{\rightsquigarrow} A$, $\alpha \Longrightarrow^* a_{i+1} \ldots a_k$ and thus $A \overset{a_{i+1} \ldots a_k}{\rightsquigarrow} B$. Now, using Lemma 1, we obtain that $S \overset{a_1 \ldots a_k}{\rightsquigarrow} B$.

By (23b) and Lemma 2, $B \Longrightarrow^* a_{k+1} \ldots a_j$. Since R is (k,j)-complete, we can now use Lemma 1, to obtain that B is in $\mathit{finished}(R)$. Therefore, $A \to \alpha B\beta \cdot \gamma \in Q \times R$. □

Let us also observe that Lemma 5 *does not* imply that $\mathit{finished}(R) = \overline{\mathit{finished}}(R)$ for any (i,j)-consistent and -complete set R, because the set $\overline{\mathit{finished}}(R)$ may contain nonterminals that are not actually reachable from S by $a_1 \ldots a_i$ and thus are not relevant in this context.

A predictor operation is used to construct the (j,j)-consistent and complete set of dotted conjuncts from the set of all nonterminals that are reachable from S by the prefix $a_1 \ldots a_j$; this turns out to be a fairly easy task:

Definition 14 (*predict*). Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $R \subseteq V$. A dotted conjunct $B \to \alpha \cdot \beta$ is in $\mathit{predict}(R)$ if and only if

$$A \overset{\varepsilon}{\rightsquigarrow} B \quad (\text{for some } A \in R), \tag{24a}$$

$$\alpha \Longrightarrow^* \varepsilon \tag{24b}$$

The following method of computing the set $\mathit{predict}(R)$ for a given set of symbols $R \subseteq V$ is suggested: first, we precompute the relation $\overset{\varepsilon}{\rightsquigarrow} \subseteq N \times N$ by constructing the matrix of this relation; this involves taking a transitive closure of the matrix and therefore results in $O(|G|^3 \cdot \log |G|)$ complexity.

Then, for any given set $R \subseteq V$, we find all nonterminals $\{B\}$, such that for some $A \in R$ it holds that $A \overset{\varepsilon}{\rightsquigarrow} B$. For every such B and for every conjunct $B \to s_1 \ldots s_k \in \mathit{conjuncts}(G)$ $(k \geqslant 0,\ s_i \in V)$, we add all dotted conjuncts $B \to s_1 \ldots s_i \cdot s_{i+1} \ldots s_k$ (where $i \geqslant 0$, $s_1, \ldots, s_i \in \text{NULLABLE}(G)$) to the result of the operation. This takes $O(|G|^2)$ time.

This operation is also defined for the sets of dotted conjuncts:

Definition 15. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let R be a set of dotted conjuncts. Define

$$\mathit{predict}(R) = \mathit{predict}(\{A \mid \textit{some dotted conjunct } C \to \alpha \cdot A\beta \textit{ is in } R\}). \tag{25}$$

3.5. The algorithm

Now, after all operations have been properly redefined, the text of the algorithm exactly repeats the context-free case:

Algorithm 1. Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $w = a_1 \ldots a_n \in \Sigma^*$ $(n \geqslant 0)$ be a string. Construct $\{t_{ij}\}$, an upper-triangular $(n+1) \times (n+1)$ matrix of

sets of dotted conjuncts.

$t_{00} = predict(\{S\})$
for $j = 1$ to n
{
    // Scanner
    for $i = 0$ to $j - 1$
        $t_{ij} = t_{i,j-1} \times \{a_j\}$

    // Completer
    for $k = j - 1$ to 0
    {
        $t_{kj} = t_{kj} \cup t_{kk} * t_{kj}$
        for $i = k - 1$ to 0
            $t_{ij} = t_{ij} \cup t_{ik} \times t_{kj}$
    }

    // Predictor
    $t_{jj} = predict(\bigcup_{i=0}^{j-1} t_{ij})$
}

The algorithm gives an answer "yes" if and only if $S \in finished(t_{0n})$.

Every jth iteration of the outer loop of the algorithm is devoted to constructing the jth column of the matrix: the Scanner starts the construction by moving the dots over a_j in all the dotted conjuncts of the previous $(j-1)$th column, the Completer moves the dots over nonterminals in the dotted conjuncts from various elements of the matrix, while the Predictor constructs the diagonal element of the jth column on the basis of the $(j-1)$ nondiagonal elements of this column.

The "flow" of dotted conjuncts is started by the Predictor or by the initial assignment, which create dotted conjuncts with dots at the beginning, and is then maintained by the Scanner and the Completer that add new dotted conjuncts to new entries of the table by moving the dot forward in the earlier created dotted conjuncts. This process is aimed at eventually producing dotted conjuncts with dot at the end, which are then used by the functions *finished* and $\overline{finished}$ to justify shifting the dot over nonterminals done by the Completer, and, at the end of the computation, to determine whether $w \in L(G)$.

We shall use the following grammar to illustrate the operation of the algorithm:

Example 1. A conjunctive grammar for the language $L = \{wcw \mid w \in \{a,b\}^*\}$:

$$S \to C\&K$$
$$C \to XCX \mid c$$
$$K \to Aa\&Ka \mid Bb\&Kb \mid Rc$$
$$A \to XAX \mid aRc$$

Table 1
Prefix reachability by ε for the grammar from Example 1

$\overset{\varepsilon}{\rightsquigarrow}$	S	C	K	A	B	R	X
S	+	+	+	+	+	+	+
C	−	+	−	−	−	−	+
K	−	−	+	+	+	+	+
A	−	−	−	+	−	−	+
B	−	−	−	−	+	−	+
R	−	−	−	−	−	+	+
X	−	−	−	−	−	−	+

$$B \to XBX \mid bRc$$
$$R \to RX \mid \varepsilon$$
$$X \to a \mid b$$

This grammar is a slightly modified version of the grammar for the same language given in [7]; the main idea is that the nonterminal K takes symbols from the right part of the string one by one and uses nonterminals A and B to compare each of them to the corresponding symbol on the left, at the same time calling itself to proceed to the next symbol in the same way. The nonterminal C makes sure that both parts of the string are of equal length.

For this grammar, the set NULLABLE equals $\{R\}$, while the prefix reachability of the nonterminals from each other by the empty string ε is given in Table 1.

Now let us briefly consider algorithm's execution on the input string $w = abcab$; the final matrix constructed by the algorithm is shown in Table 2.

t_{00} equals $predict(\{S\})$. t_{01} is initially made by the Scanner by shifting the dot over a in the dotted conjuncts from t_{00}, which results in $t_{01} = \{A \to a\cdot Rc, A \to aR\cdot c, X \to a\cdot\}$; at this point $\overline{finished}(t_{01}) = \{R, X\}$, and then the product $t_{00} * t_{01}$ gives the rest of the dotted conjuncts in t_{01} by moving the dot over X and R in the dotted conjuncts from t_{00}. The element t_{11} equals $predict(\{C, A, B, R, X\})$, since these are the nonterminals that appear right after the dot in the dotted conjuncts from t_{01}.

So far the algorithm has worked as if the grammar were context-free, with the set of conjuncts acting as the set of productions. Let us discuss how the element t_{04} is constructed, where the algorithm's behaviour essentially differs from the context-free case. Initially, the Scanner assigns $t_{04} = \{K \to Aa\cdot, K \to Ka\cdot\}$. Then the dotted conjunct $B \to XBX\cdot$ is added to t_{04} from $t_{03} \times t_{34}$; another dotted conjunct $C \to XC\cdot X$ is added from $t_{01} \times t_{14}$. When the statement $t_{04} = t_{04} \cup t_{00} * t_{04}$ is executed, the expression $\overline{finished}(t_{04})$ evaluates to $\{B, K\}$; B is there due to the dotted conjunct $B \to XBX\cdot$, while $K \in \overline{finished}(t_{04})$ since *both* $K \to Aa\cdot$ and $K \to Ka\cdot$ are in t_{04}. Now, $t_{00} \times \{B, K\}$ completes the set t_{04}, adding, among others, the dotted conjunct $K \to K \cdot b$.

When the element t_{05} is being completed by the final assignment $t_{05} = t_{05} \cup t_{00} * t_{05}$, the dotted conjuncts $K \to Bb\cdot$, $K \to Kb\cdot$ and $C \to XCX\cdot$ are already in t_{05} (the first

Table 2
A recognition matrix for the string $w = abcab$

	0	1	2	3	4	5
0	$S \to \cdot C$ $S \to \cdot K$ $C \to \cdot XCX$ $C \to \cdot c$ $K \to \cdot Aa$ $K \to \cdot Ka$ $K \to \cdot Bb$ $K \to \cdot Kb$ $K \to \cdot Rc$ $K \to R \cdot c$ $A \to \cdot XAX$ $A \to \cdot aRc$ $B \to \cdot XBX$ $B \to \cdot bRc$ $R \to \cdot RX$ $R \to R \cdot X$ $R \to \cdot$ $X \to \cdot a$ $X \to \cdot b$	$C \to X \cdot CX$ $K \to R \cdot c$ $A \to X \cdot AX$ $A \to a \cdot Rc$ $A \to aR \cdot c$ $B \to X \cdot BX$ $R \to R \cdot X$ $R \to RX \cdot$ $X \to a \cdot$	$K \to R \cdot c$ $A \to aR \cdot c$ $R \to R \cdot X$ $R \to RX \cdot$	$S \to K \cdot$ $K \to A \cdot a$ $K \to K \cdot a$ $K \to K \cdot b$ $K \to Rc \cdot$ $A \to aRc \cdot$ $B \to XB \cdot X$	$S \to K \cdot$ $C \to XC \cdot X$ $K \to Aa \cdot$ $K \to K \cdot a$ $K \to Ka \cdot$ $K \to B \cdot b$ $K \to K \cdot b$ $B \to XBX \cdot$	$S \to C \cdot$ $S \to K \cdot$ $C \to XCX \cdot$ $K \to K \cdot a$ $K \to Bb \cdot$ $K \to K \cdot b$ $K \to Kb \cdot$
1		$C \to \cdot XCX$ $C \to \cdot c$ $A \to \cdot XAX$ $A \to \cdot aRc$ $B \to \cdot XBX$ $B \to \cdot bRc$ $R \to \cdot RX$ $R \to R \cdot X$ $R \to \cdot$ $X \to \cdot a$ $X \to \cdot b$	$C \to X \cdot CX$ $A \to X \cdot AX$ $B \to X \cdot BX$ $B \to b \cdot Rc$ $B \to bR \cdot c$ $R \to R \cdot X$ $R \to RX \cdot$ $X \to b \cdot$	$C \to XC \cdot X$ $B \to bRc \cdot$	$C \to XCX \cdot$	$\emptyset$
2			$C \to \cdot XCX$ $C \to \cdot c$ $A \to \cdot XAX$ $A \to \cdot aRc$ $B \to \cdot XBX$ $B \to \cdot bRc$ $R \to \cdot RX$ $R \to R \cdot X$ $R \to \cdot$ $X \to \cdot a$ $X \to \cdot b$	$C \to c \cdot$	$\emptyset$	$\emptyset$
3				$X \to \cdot a$ $X \to \cdot b$	$X \to a \cdot$	$\emptyset$
4					$X \to \cdot a$ $X \to \cdot b$	$X \to b \cdot$
5						$\emptyset$

two have been added by Scanner, while $C \to XCX\cdot$ came from $t_{04} \times t_{45}$). The function $\overline{finished}$ produces the set $\{S, K, C\}$, where C is there because of $C \to XCX\cdot$, K came from the mentioned $K \to Bb\cdot$ and $K \to Kb\cdot$, and therefore S belongs to $\overline{finished}(t_{05})$ due to the rule $S \to C\&K$. The dotted conjuncts $S \to C\cdot$ and $S \to K\cdot$ are then added to t_{05}, and thus $S \in finished(t_{05})$.

If, for instance, the string were *bbcab*, then A would not be in $finished(t_{03})$, $K \to A\cdot a$ would not be in t_{03}, $K \to Aa\cdot$ would not be in t_{04}, K would not be in $\overline{finished}(t_{04})$ (since the dotted conjunct $K \to Ka\cdot$ alone is not sufficient to apply the rule $K \to Aa\&Ka$), $K \to K \cdot b$ would not be added to t_{04} and consequently the dotted conjunct $K \to Kb\cdot$ would not make its way to t_{05}, thus keeping K and S out of $\overline{finished}(t_{05})$. If the string were *aacab*, then $K \to Kb\cdot$ *would* be added to t_{05} by the Scanner, but instead the dotted conjunct $K \to Bb\cdot$ would be missing, and hence the conjunction again would not be satisfied. If the string were *abbcab*, then both $K \to Ka\cdot$ and $K \to Bb\cdot$ would be in the top right element t_{06} and K would be in $\overline{finished}(t_{06})$, but this time the condition enforced by C is false (since $|abb| \neq |ab|$), and thus C and S would not be in $\overline{finished}(t_{06})$, again showing that the string is not generated by the grammar.

3.6. Proof of the algorithm's correctness

In this section we shall prove the correctness of the algorithm. Our proof generally follows the one from the context-free case, but, especially in the case of the algorithm's completeness, it is somewhat more complicated than the original proof.

Lemma 6 (Consistency of the algorithm). Let $\{t_{ij}\}$ be the matrix constructed by the algorithm. Then, for all i, j $(0 \leqslant i \leqslant j \leqslant n)$ t_{ij} is (i,j)-consistent.

Proof. We need to prove that every statement in the algorithm preserves the consistency of the matrix, i.e. that the (i,j)-consistency of every t_{ij} before the execution of each statement implies the consistency of all the elements of the matrix after the statement is executed.

The following cases have to be considered:

(i) (*Initial assignment*) $predict(\{S\})$ *is* $(0,0)$-*consistent.*
It is easily seen that the definition of $(0,0)$-consistency actually coincides with the definition of $predict(\{S\})$, since (14a) becomes the same as (24a) and (14b) turns out to be the same as (24b).

(ii) (*Scanner*) *For any* $i < j$, *if* $t_{i,j-1}$ *is* $(i, j-1)$-*consistent, then* $t_{i,j-1} \times \{a_j\}$ is (i,j)-*consistent.*
By the definition of product (Definition 12), every dotted conjunct in $t_{i,j-1} \times \{a_j\}$ is of the form $A \to \alpha a_j \beta \cdot \gamma$, where $A \to \alpha \cdot a_j \beta\gamma \in t_{i,j-1}$ and β is a nullable string. Since $A \to \alpha \cdot a_j \beta\gamma$ is $(i, j-1)$-consistent by the assumption,

$$S \overset{a_1 \ldots a_i}{\rightsquigarrow} A, \tag{26a}$$

$$\alpha \Longrightarrow^* a_{i+1} \ldots a_{j-1}. \tag{26b}$$

Using (26b) and the nullability of β, we can construct a derivation $\alpha a_j\beta \Longrightarrow^* \alpha a_j \Longrightarrow^* a_{i+1}\dots a_{j-1}a_j$, which, together with (26a), proves the (i,j)-consistency of $A \to \alpha a_j\beta \cdot \gamma$. Due to the arbitrary choice of the dotted conjunct in $t_{i,j-1} \times \{a_j\}$, this means that the whole set is (i,j)-consistent.

(iii) (*Completer*) Since there are two assignment statements in the Completer, we have two cases to consider: (i) For any k and j $(0 \leqslant k < j \leqslant n)$, if t_{kk} is (k,k)-consistent and t_{kj} is (k,j)-consistent, then $t_{kk} * t_{kj}$ is (k,j)-consistent; (ii) For any $i < k < j$, if t_{ik} is (i,k)-consistent and t_{kj} is (k,j)-consistent, then $t_{ik} \times t_{kj}$ is (i,j)-consistent.

Taking into consideration that $Q \times R \subseteq Q * R$ for arbitrary Q and R, let us prove a more general statement covering both cases: *For any i, k and j, such that $0 \leqslant i \leqslant k < j \leqslant n$, if t_{ik} is (i,k)-consistent and t_{kj} is (k,j)-consistent, then $t_{ik} * t_{kj}$ is (i,j)-consistent.*

Every dotted conjunct in $t_{ik} * t_{kj}$ is of the form $A \to \alpha B\beta\cdot\gamma$, where $A \to \alpha\cdot B\beta\gamma \in t_{ik}$, $B \in \overline{finished}(t_{kj})$ and $\beta \Longrightarrow^* \varepsilon$. By (i,k)-consistency of $A \to \alpha \cdot B\beta\gamma$,

$$S \overset{a_1\dots a_i}{\rightsquigarrow} A, \tag{27a}$$

$$\alpha \Longrightarrow^* a_{i+1}\dots a_k \tag{27b}$$

By Lemma 2 and the (k,j)-consistency of t_{kj}, $B \Longrightarrow^* a_{k+1}\dots a_j$. This allows to construct a derivation

$$\alpha B\beta \Longrightarrow^* \alpha B \Longrightarrow^* a_{i+1}\dots a_k B \Longrightarrow^* a_{i+1}\dots a_k a_{k+1}\dots a_j \tag{28}$$

which, together with (27a), means that $A \to \alpha B\beta \cdot \gamma$ is (i,j)-consistent.

(iv) (*Predictor*) *For any $j > 0$, if for all i $(0 \leqslant i < j)$ t_{ij} is (i,j)-consistent, then $predict(\bigcup_{i=0}^{j-1} t_{ij})$ is (j,j)-consistent.*

Let $A \to \alpha\cdot\beta \in predict(\bigcup_{i=0}^{j-1} t_{ij})$. Then there exists a number i $(0 \leqslant i < j)$, such that for some $D \to \gamma\cdot E\delta \in t_{ij}$ it holds that $A \to \alpha\cdot\beta \in predict(\{E\})$. By definition of *predict*,

$$E \overset{\varepsilon}{\rightsquigarrow} A, \tag{29a}$$

$$\alpha \Longrightarrow^* \varepsilon. \tag{29b}$$

By (i,j)-consistency of $D \to \gamma \cdot E\delta$,

$$S \overset{a_1\dots a_i}{\rightsquigarrow} D, \tag{30a}$$

$$\gamma \Longrightarrow^* a_{i+1}\dots a_j. \tag{30b}$$

The existence of the conjunct $D \to \gamma E\delta$ and (30b) imply that $D \overset{a_{i+1}\dots a_j}{\rightsquigarrow} E$, which, in conjunction with (30a) and (29a), leads to $S \overset{a_1\dots a_j}{\rightsquigarrow} A$. Taking (29b) into consideration, that proves the (j,j)-consistency of $A \to \alpha\cdot\beta$. □

Lemma 7 (Completeness of the algorithm). Let $\{t_{ij}\}$ be the matrix constructed by the algorithm. Then, for all i, j $(0 \leqslant i \leqslant j \leqslant n)$, t_{ij} is (i,j)-complete.

Proof. Let us prove that for all i and j ($i \leqslant j$), t_{ij} becomes (i,j)-complete after the last assignment to it: for the element t_{00} that is the first statement of the algorithm, for the rest of the diagonal elements that is the statement in the Predictor, and the elements t_{ij} ($i < j$) have their final values assigned to them by the statement

$$t_{kj} = t_{kj} \cup t_{kk} * t_{kj} \tag{31}$$

with $k = i$. The relative order of execution of these final assignments is as follows:

$$\begin{aligned}
&t_{00},\\
&t_{10}, t_{11},\\
&t_{12}, t_{02}, t_{22},\\
&t_{23}, t_{13}, t_{03}, t_{33},\\
&\quad\vdots\\
&t_{n-2,n-1}, t_{n-3,n-1}, \ldots, t_{0,n-1}, t_{n-1,n-1},\\
&t_{n-1,n}, t_{n-2,n}, t_{n-3,n}, \ldots, t_{1,n}, t_{0,n}, t_{n,n}
\end{aligned} \tag{32}$$

Our claim in proved by induction on the length of the computation.

Basis: As already established in the proof of Lemma 6 (part (i)), t_{00} becomes $(0,0)$-complete after the execution of the first statement of the algorithm.

Induction step: Consider an arbitrary (i,j)-consistent dotted conjunct $A \to \alpha \cdot \beta$, where $0 < i \leqslant j$.

Let us first consider the case $i < j$. Since $a_{i+1} \ldots a_j \neq \varepsilon$, there is a partition $\alpha = \alpha' X \alpha''$ (where $\alpha', \alpha'' \in V^*$; $X \in V$), such that for some k ($i \leqslant k < j$)

$$\alpha' \Longrightarrow^* a_{i+1} \ldots a_k, \tag{33a}$$

$$X \Longrightarrow^* a_{k+1} \ldots a_j, \tag{33b}$$

$$\alpha'' \Longrightarrow^* \varepsilon. \tag{33c}$$

The dotted conjunct $A \to \alpha' \cdot X\alpha''\beta$ is (i,k)-consistent by $S \overset{a_1 \ldots a_i}{\rightsquigarrow} A$ (which comes from the (i,j)-consistency of $A \to \alpha \cdot \beta$) and (33a). By the order of the computation (32), the last assignment to t_{ik} is carried out before the algorithm proceeds to constructing the jth column of the matrix, and therefore, by the induction hypothesis, $A \to \alpha' \cdot X\alpha''\beta$ is added to t_{ik} prior to the jth iteration of the outer loop. $S \overset{a_1 \ldots a_i}{\rightsquigarrow} A$ and (33a) also imply that

$$S \overset{a_1 \ldots a_k}{\rightsquigarrow} X \tag{34}$$

Depending on X and k, we have several cases:

(i) $X \in \Sigma$, i.e. $X = a_j$ and $k = j - 1$. Then $A \to \alpha' a_j \alpha'' \cdot \beta$ is in $t_{i,j-1} \times \{a_j\}$ and hence is added to t_{ij} by the Scanner in the jth iteration of the outer loop.

(ii) $X \in N$, $i < k$. By (33b), there is a rule $X \to \xi_1 \& \ldots \& \xi_t \in P$, such that $\xi_s \Longrightarrow^* a_{k+1} \ldots a_j$ for all s $(1 \leqslant s \leqslant t)$. Together with (34), that indicates the (k,j)-consistency of all $X \to \xi_s \cdot$.
For the given j, k and i, consider the execution of the statement

$$t_{ij} = t_{ij} \cup t_{ik} \times t_{kj} \tag{35}$$

of the Completer in the iteration (j,k,i) of the three nested loops of the algorithm. The set t_{ik}, as argued above, is already (i,k)-complete at this point. The final assignment (31) to the element t_{kj} was made at the beginning of the iteration (j,k) of the Completer, right before entering the third nested loop, in which the statement (35) is executed. Therefore, by the induction hypothesis, all dotted conjuncts $X \to \xi_s \cdot$ $(1 \leqslant s \leqslant t)$ have been added to t_{kj} prior to the execution of (35), which implies $X \in finished(t_{kj})$.
Then the dotted conjunct $A \to \alpha' X \alpha'' \cdot \beta$ is in $t_{ik} \times t_{kj}$ when the assignment (35) is being performed, and is accordingly added to t_{ij} by this assignment.

(iii) $X \in N$ and $i = k$, i.e. $\alpha' \Longrightarrow^* \varepsilon$, $\alpha'' \Longrightarrow^* \varepsilon$ and there exists a derivation

$$X \Longrightarrow \ldots \Longrightarrow a_{i+1} \ldots a_j. \tag{36}$$

Consider the derivation tree corresponding to (36). Let $\mathcal{N}$ be the set of all those internal vertices of this tree, which are least common ancestors of $a_{i+1}, \ldots, a_j$; by Proposition 1, these leaves have at least one least common ancestor, but it is not necessarily unique. Let $\mathcal{N}' \subseteq \mathcal{N}$ be those among these vertices that have at least one group of descendants which does not contain any vertices from $\mathcal{N}$ (the construction of this set is explained in Section 3.3 in the discussion of Lemma 4).

Let us take some vertex v_Y from $\mathcal{N}'$, labelled with some rule $Y \to \delta_1 \& \ldots \& \delta_t$, and some conjunct $Y \to \delta_s$ $(1 \leqslant s \leqslant t)$, such that the corresponding group of descendants of v_Y does not contain any vertices from $\mathcal{N}$ (such conjunct exists by the definition of $\mathcal{N}'$).

Consider the dotted conjunct $Y \to \delta_s \cdot$. It follows from the position of v_Y in the derivation tree that $\delta_s \Longrightarrow^* a_{i+1} \ldots a_j$ and $X \overset{\varepsilon}{\rightsquigarrow} Y$ – i.e., $Y \to \delta_s \cdot$ is (i,j)-consistent. We shall now prove that it must have been added to t_{ij} *before* the final assignment to t_{ij}, which is the statement (31) for $k = i$.

There exists a position p $(i \leqslant p < j)$ in the input string and a partition $\delta_s = \delta' Z \delta''$ (where $\delta', \delta'' \in V^*$; $Z \in V$), such that

$$\delta' \Longrightarrow^* a_{i+1} \ldots a_p, \tag{37a}$$

$$Z \Longrightarrow^* a_{p+1} \ldots a_j, \tag{37b}$$

$$\delta'' \Longrightarrow^* \varepsilon. \tag{37c}$$

The dotted conjunct $Y \to \delta' \cdot Z \delta''$ is (i,p)-consistent, and thus was added to t_{ip} before the jth iteration of the outer loop. Depending on Z, there are two cases to

consider:

(a) $Z \in \Sigma$. Then $p = j-1$, $Z = a_j$, the dotted conjunct $Y \to \delta' \cdot a_j \delta''$ is in $t_{i,j-1}$, and thus $Y \to \delta' a_j \delta'' \cdot$ is added to t_{ij} by the Scanner before the final assignment (31) to t_{ij}.

(b) $Z \in N$. Let us first show that this can take place only if $p > i$. Supposing that $Z \in N$ and $i = p$, we obtain $Z \Longrightarrow^{+} a_{i+1} \dots a_j$. Let v_Z be the descendant of v_Y corresponding to this instance of Z. By the definition of the set $\mathcal{N}$, it follows that $v_Z \in \mathcal{N}$, and thus the group of descendants of the vertex v_Y corresponding to the conjunct $Y \to \delta_s$ contains at least one vertex from $\mathcal{N}$, which contradicts our initial choice of the vertex and the group of descendants.

Then, as in case (ii) above, it is proved that the dotted conjunct $Y \to \delta' Z \delta'' \cdot$ is added to t_{ij} by statement (35) in the third nested loop of the algorithm (iteration $k = p$), and therefore it will be in t_{ij} at the time of the execution of the final assignment (31) to the element t_{ij}.

By the arbitrariness of our choice of the vertex v_Y, we obtain that for every such vertex and for every group of descendants that does not contain vertices from $\mathcal{N}$, the corresponding dotted conjunct $Y \to \delta_s \cdot$ is added to t_{ij} before the execution of the statement (31).

Therefore, by Lemma 4, the condition $X \in \overline{finished}(t_{ij})$ holds prior to the execution of (31). Since $A \to \alpha' \cdot X \alpha'' \beta$ is in t_{ii}, the dotted conjunct $A \to \alpha' X \alpha'' \cdot \beta$ is in $t_{ii} * t_{ij}$ and will be added to t_{ij} by the statement (31).

Now let us turn to the case of diagonal elements ($i = j > 0$). The (j,j)-consistency of $A \to \alpha \cdot \beta$ means that $S \overset{a_1 \dots a_j}{\rightsquigarrow} A$ and $\alpha \Longrightarrow^{*} \varepsilon$. By the definition of "$\rightsquigarrow$", there exists a partition $a_1 \dots a_j = u_1 \dots u_p$ and a sequence of nonterminals $C_0 = S, C_1, \dots, C_{p-1}, C_p = A$, such that for every r ($1 \leqslant r \leqslant p$) there is a rule $C_{r-1} \to \eta_1 \& \dots \& \eta_m$, and a representation of one of the conjuncts of this rule as $\eta_l = \gamma_r C_r \delta_r$ ($\gamma_r, \delta_r \in V^*$), in which $\gamma_r \Longrightarrow^{*} u_r$.

Since $a_1 \dots a_j \neq \varepsilon$, at least one of the strings $u_1, \dots, u_p$ is not empty. Consider the maximum r, such that $u_r \neq \varepsilon$. Then $u_r = a_{k+1} \dots a_j$ for some k ($0 \leqslant k < j$), and therefore there exists a rule $C_{r-1} \to \eta_1 \& \dots \& \eta_m$, such that for some l-th ($1 \leqslant l \leqslant m$) conjunct of this rule it holds that

$$\eta_l = \gamma_r C_r \delta_r, \tag{38a}$$

$$\gamma_r \Longrightarrow^{*} a_{k+1} \dots a_j \tag{38b}$$

Fig. 5 illustrates the prefix reachability $S \overset{a_1 \dots a_j}{\rightsquigarrow} A$ and the choice of r. As in the earlier Fig. 3, the unused groups of descendants of the vertices $C_1, \dots, C_p$ are not shown in Fig. 5; they are of the same form as the descendants of the vertex C_0, which are shown in both figures. This reachability easily implies $S \overset{a_1 \dots a_k}{\rightsquigarrow} C_{r-1}$ and $C_r \overset{\varepsilon}{\rightsquigarrow} A$.

Now the prefix reachability $S \overset{a_1 \dots a_k}{\rightsquigarrow} C_{r-1}$, together with (38b), means (k,j)-consistency of $C_{r-1} \to \gamma_r \cdot C_r \delta_r$. By the induction hypothesis, $C_{r-1} \to \gamma_r \cdot C_r \delta_r$ is added to t_{kj} by the Completer in the jth iteration of the outer loop of the algorighm and thus prior to the execution of the Predictor statement in the same iteration.

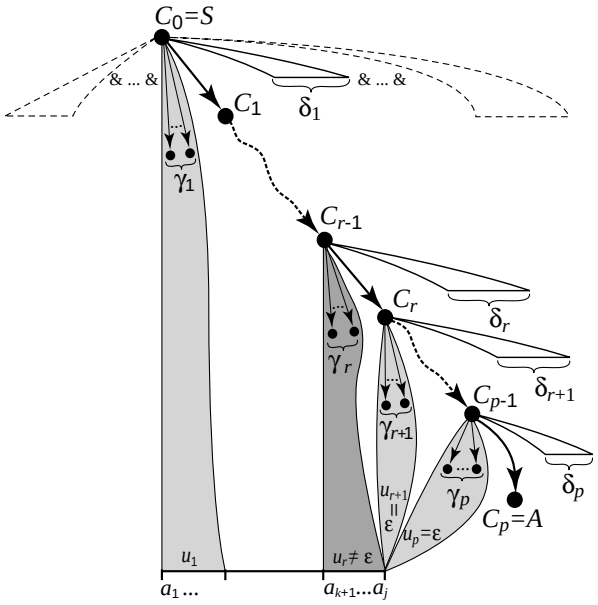

Fig. 5. Prefix reachability $S \overset{a_1 \ldots a_j}{\rightsquigarrow} A$ in the case of diagonal elements.

Since the nonterminal C_r appears right after the dot in one of the dotted conjuncts in the i-th column of the matrix, $C_r \overset{\varepsilon}{\rightsquigarrow} A$ and α is a nullable string, we conclude that

$$A \to \alpha \cdot \beta \in predict(\bigcup_{i=0}^{j-1} t_{ij}) \tag{39}$$

i.e., $A \to \alpha \cdot \beta$ is added to the set t_{jj} by the Predictor. □

The following theorem states the algorithm's correctness; its proof easily follows from the consistency and completeness lemmata:

Theorem 2. *Let $G = (\Sigma, N, P, S)$ be an arbitrary conjunctive grammar, let $w \in \Sigma^*$. Given w as an input, the algorithm for G always terminates and returns "yes" if and only if $w \in L(G)$, "no" otherwise.*

The algorithm has been implemented in the parser generator [12]; it could also be noted that Tables 1 and 2 included in this paper were produced by a generated parser.

4. Construction of derivation trees

So far we have been using the algorithm as a recognizer, which determines whether the input string is in the language. Now we turn to parsing, where the aim is to produce some derivation tree of the string.

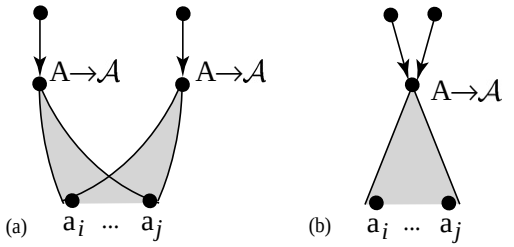

Fig. 6. (a) Two subtrees in a conventional derivation tree; (b) One shared subtree in the corresponding condensed derivation tree.

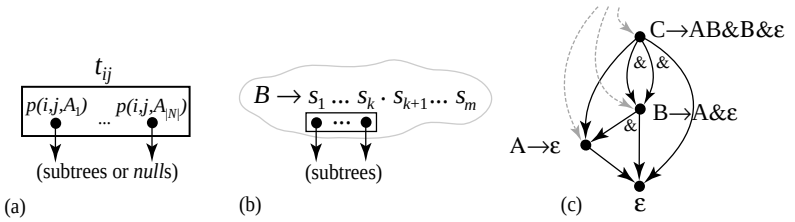

Fig. 7. (a) Subtrees referenced from elements of the matrix; (b) Subtrees referenced from dotted conjuncts; (c) Sample subtrees for nullable nonterminals.

4.1. Bottom-up tree construction

Let us first give a method of constructing so-called *condensed derivation trees*, in which not only terminal leaves, but any identical subtrees could be shared, provided that they have the same terminal descendants (see Fig. 6). Condensed trees turn out to be easier to construct and analyze than ordinary derivation trees, and their size (measured as a number of vertices) is linearly bounded by the length of the input string.

A plain derivation tree (as defined in Section 2) can be made out of a condensed tree by simply duplicating shared subtrees; in the worst case this involves exponential blowup in size (an example of exponential-length derivation in a conjunctive grammar is given in [9]).

A condensed derivation tree can be constructed simultaneously with the recognition matrix. We shall employ the following additional data structures:

(i) For every t_{ij} and for every nonterminal $A \in N$ we store a pointer $p(i,j,A)$, which is either *null*, or points to the derivation tree of some derivation $A \Longrightarrow \ldots \Longrightarrow a_{i+1} \ldots a_j$, as shown in Fig. 7(a).

(ii) Together with every dotted conjunct

$$A \to s_1 s_2 \ldots s_k \cdot s_{k+1} \ldots s_l \tag{40}$$

in t_{ij} we also store k pointers to the trees of $s_1, \ldots, s_k$ (see Fig. 7(b)).

(iii) For every nullable nonterminal A, a derivation tree of $A \Longrightarrow \cdots \Longrightarrow \varepsilon$ is pre-computed and stored for future use. These trees could share subtrees and use a single ε leaf, as illustrated in the example in Fig. 7(c), where the dotted grey arcs suggest this "future use".

Initially, all pointers $p(i,j,A)$ are set to *null* and there are no dotted conjuncts in the matrix.

For every dotted conjunct of the form $A \to \alpha a\beta \cdot \gamma$ ($a \in \Sigma$, $\beta \in \text{NULLABLE}^*$) created by the Scanner by moving the dot in the dotted conjunct $A \to \alpha \cdot a\beta\gamma$, the first $|\alpha|$ descendants are inherited from the source dotted conjunct, a is connected to the corresponding terminal leaf, and symbols from β are connected to precomputed epsilon derivation trees.

For every fixed i and j, the sets $\langle p(i,j,A)\rangle_{A \in N}$ are constructed at the same time as the mapping $\overline{finished}(t_{ij})$ is computed. Each time the operator $\rho_{t_{ij}}$ is applied to a temporary variable $Q \subseteq N$, and for each nonterminal A added to Q, according to Definition 11, there must be a a rule $A \to \alpha_1 \& \ldots \& \alpha_m$, such that for every kth conjunct one of the following holds: (i) $A \to \alpha_k \cdot \in t_{ij}$, or (ii) $\alpha_k = \beta_1 B \beta_2$ for some $\beta_i \in \text{NULLABLE}^*$ and $B \in Q$. A new vertex labeled with this rule is created and connected to the following subtrees:

- For each conjunct of type (i), all $|\alpha_k|$ descendants of the dotted conjunct $A \to \alpha_k \cdot$ are transferred to the newly created vertex. If $|\alpha_k| = 0$, then the vertex is connected to the earlier created "ε" leaf.
- For each conjunct of type (ii), we add $|\beta_1\beta_2|$ arcs leading to the precomputed derivation trees of the empty string from nullable nonterminals, and a single arc to the vertex pointed by $p(i,j,B)$.

A pointer to the newly created vertex is stored in $p(i,j,A)$.

For each dotted conjunct of the form $A \to \alpha B\beta \cdot \gamma$ ($B \in \overline{finished}(t_{ij})$ for some $i \leqslant j$; $\beta \in \text{NULLABLE}^*$) created by the Completer from the source dotted conjunct $A \to \alpha \cdot B\beta\gamma$, the first $|\alpha|$ descendants are inherited, B is connected to the subtree $p(i,j,B)$, and symbols from β are connected to the previously created derivation trees of ε.

For every dotted conjunct of the form $A \to \alpha \cdot \beta$ created by the initial assignment or by the Predictor, it holds that $\alpha \in \text{NULLABLE}^*$; these $|\alpha|$ vertices are being connected to the precomputed derivation trees of ε.

When a single dotted conjunct can be created in several ways (for instance, if $A \to \alpha \cdot B\gamma \in t_{ik} \cap t_{il}$ and $B \in finished(t_{kj}) \cap finished(t_{lj})$, then $A \to \alpha B \cdot \gamma$ is both in $t_{ik} \times t_{kj} \subseteq t_{ij}$ and in $t_{il} \times t_{lj} \subseteq t_{ij}$), then the algorithm could ignore all but the first representation it encounters, thus arbitrarily choosing one of the several possible derivation trees.

Finally, when the set $finished(t_{0n})$ is computed, if S is found to be in this set, then a pointer to the derivation tree of the whole input string will appear in $p(0,n,S)$.

This augmented version of Algorithm 1 is constant times slower than the original recognizer and uses constant times more space (where both constants depend upon the grammar), but the order of complexity is still cubic with respect to time and square with respect to space. This method of parse tree construction is applicable to any grammar and input string, but it has a drawback that it involves constructing all possible subtrees, many of which will never be used. While garbage collection can be implemented quite straightforwardly, it will not compensate the time already wasted. The other method we shall now discuss does not have this disadvantage at the cost of somewhat limited applicability.

4.2. Top-down tree construction

Another option is first to construct a plain recognition matrix using the unmodified recognition algorithm, and then use the constructed matrix to build a derivation tree. This approach to tree construction is suggested in [5], where it is shown to be applicable for cycle-free context-free grammars and proved to work in no more than quadratic time.

The following algorithm is a direct generalization of the context-free case, but, as we shall consequently demonstrate, its complexity may sometimes be considerably higher.

Algorithm 2 (Top-down tree construction). Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar. Let $w = a_1 \dots a_n \in \Sigma^*$ $(n \geqslant 0)$ be a string in $L(G)$. Let $\{t_{ij}\}$ be the upper-triangular matrix of (i,j)-consistent and complete sets created by Algorithm 1.

Define the following procedure:

parse(int i, int j, $A \in N$)
    precondition $0 \leqslant i \leqslant j \leqslant n$, $A \in \mathit{finished}(t_{ij})$
{
    Choose some rule $A \to \alpha_1 \& \dots \& \alpha_m$, such that $A \to \alpha_k \cdot \in t_{ij}$ for all k.
    Create a new vertex v labeled with $A \to \alpha_1 \& \dots \& \alpha_m$.
    /* Now we add descendants for v one by one, from right to left */
    for $k = m$ to 1
    {
        if $\alpha_k \neq \varepsilon$
        {
            Denote $\alpha_k = s_1 \dots s_{|\alpha_k|}$, where $s_1, \dots, s_{|\alpha_k|} \in \Sigma \cup N$.
            Let $l = j$.
            for $t = |\alpha_k|$ to 1 (loop invariant: $A \to s_1 \dots s_t \cdot s_{t+1} \dots s_{|\alpha_k|} \in t_{il}$)
                if $s_t \in \Sigma$
                {
                    Add a new leaf labeled with s_t.
                    $l = l - 1$
                }
                else
                {
                    Let $r = l$
                    while $A \to s_1 \dots s_{t-1} \cdot s_t \dots s_{|\alpha_k|} \notin t_{il}$ or $s_t \notin \mathit{finished}(t_{lr})$
                        $l = l - 1$
                    Add subtree $\mathit{parse}(l, r, s_t)$
                }
        }
        else
            Add a new leaf labeled ε.
    }
    return v
}

This procedure does not necessarily terminate, but it case it does, it returns a pointer to a derivation tree of the string $a_{i+1}\dots a_j$ from the nonterminal A. The derivation tree of the whole string w is given by $parse(0,n,S)$.

Note that the tree constructed by Algorithm 2 is not a condensed derivation tree, but a regular derivation tree, in which only terminal leaves can be shared.

Lemma 8. *Let the i, j and A satisfy the precondition of the procedure parse in Algorithm 2. Then, if the procedure $parse(i,j,A)$ terminates, it returns a pointer to a derivation tree of $a_{i+1}\dots a_j$ from A.*

Proof. The proof is carried out inductively on the structure of the tree of recursive calls. For a particular call $parse(i,j,A)$, let us consider the computation inside the loop "for $t=|\alpha_k|$ to 1". Obviously, the loop invariant holds prior to the first iteration of the loop, because $A\to\alpha_k\cdot$ is in t_{ij}.

For each tth iteration, if $s_t\in\Sigma$, then $s_t=a_l$ and the dotted conjunct $A\to s_1\dots s_{t-1}\cdot s_t\dots s_{\alpha_k}$ is $(i,l-1)$-consistent. Thus, after both variables t and l get decremented by one, the loop invariant will still hold.

Consider the second case, when $A\to s_1\dots s_t\cdot s_{t+1}\dots s_{\alpha_k}\in t_{ir}$ and $s_t\in N$. Let us show that then there exists a number l ($i\leqslant l\leqslant r$), such that $A\to s_1\dots s_{t-1}\cdot s_t\dots s_{\alpha_k}$ is in t_{il} and $s_t\in finished(t_{lr})$. Indeed, by (i,r)-consistency of $A\to s_1\dots s_t\cdot s_{t+1}\dots s_{\alpha_k}$ it holds that

$$s_1\dots s_{t-1}s_t \Longrightarrow^* a_{i+1}\dots a_r \tag{41}$$

By Theorem 1, $a_{i+1}\dots a_r\in L_G(s_1\dots s_{t-1})\cdot L_G(s_t)$, and therefore there exists a factorization of $a_{i+1}\dots a_r$ into two substrings,

$$a_{i+1}\dots a_l\in L_G(s_1\dots s_{t-1}) \tag{42a}$$

and

$$a_{l+1}\dots a_r\in L_G(s_t), \tag{42b}$$

for some l ($i\leqslant l\leqslant r$). Together with the reachability $S\overset{a_1\dots a_i}{\rightsquigarrow}A$, (42a) implies that the dotted conjunct $A\to s_1\dots s_{t-1}\cdot s_t\dots s_{\alpha_k}$ is (i,l)-consistent and thus is in t_{il}. On the other hand, the nonterminal s_t is obviously reachable from A by $a_{i+1}\dots a_l$ and thus, by Lemma 1, $S\overset{a_1\dots a_l}{\rightsquigarrow}s_t$. By Lemma 3 and (l,r)-completeness of t_{lr}, $s_t\in finished(t_{lr})$. This means that the inner `while` loop is certain to terminate before l gets less than i. Now $i\leqslant l\leqslant r\leqslant j$.

When the call to $parse(l,r,s_t)$ is being made, the precondition for the procedure *parse* — i.e., $s_t\in finished(t_{lr})$ — is met. By the induction hypothesis, if $parse(l,r,s_t)$ returns, then it will return a derivation tree of $a_{l+1}\dots a_r$ from s_t. After t gets decremented by one, the exit condition of the recently completed `while` loop will imply the loop invariant of "for $t=|\alpha_k|$ to 1".

Therefore, if all recursive calls to *parse* succeed, then $parse(i,j,A)$ will construct a derivation tree of the string $a_{i+1}\dots a_j$ from the nonterminal A. If at least one of the

recursive calls does not terminate, then $parse(i,j,A)$ does not terminate as well, which completes the proof of the lemma. □

The lower bound for the complexity of the algorithm is a constant by the length of the shortest derivation of w, because the construction actually simulates some derivation. In case the number of possible derivations of the string w is finite, the algorithm's complexity does not exceed a constant by $|w|$ by the length of the longest derivation, because for each application of each rule the two outer `for` loops perform a bounded number of iterations, while the internal `while` loop does no more than $|w|$ iterations.

However, there exist conjunctive grammars for which the derivation of any string of length n consists of $O(2^n)$ steps [9], such as the grammar $G = (\{a\}, \{S\}, \{S \to aS\&Sa,\ S \to a\}, S)$ for the language a^+, and therefore the worst-case complexity of this algorithm is exponential even if the choice or rules is assumed to be optimal. But still in some favourable cases one can expect quadratic or even linear computation time.

Let us also provide a sufficient condition of the algorithm's applicability to a grammar.

Lemma 9. *Let $G = (\Sigma, N, P, S)$ be a conjunctive grammar, such that the context-free grammar $G' = (\Sigma, N, conjuncts(P), S)$ is cycle-free, i.e. there is no nonterminal $A \in N$, such that $A \overset{G'}{\Longrightarrow}{}^{+} A$. Then for any string $w \in \Sigma^*$ Algorithm 2 always terminates.*

Proof. Let us state two easily proved facts:

- *If $parse(i,j,A)$ makes a recursive call to $parse(k,l,B)$, then $i \leqslant k \leqslant l \leqslant j$. This was established in Lemma 8.*
- *If $parse(i,j,A)$ makes a recursive call to $parse(i,j,B)$, then $A \overset{G'}{\Longrightarrow}{}^{+} B$.*

  This is so, because a recursive call to $parse(i,j,B)$ can only take place if one of the conjuncts of the selected rule is of the form $A \to \beta_1 B \beta_2$, where β_i are nullable strings, which implies that $A \overset{G'}{\Longrightarrow}{}^{+} B$.

Now, if Algorithm 2 does not terminate, then the tree of recursive calls of *parse* is infinite. Since it is finite-branching, then, by König's lemma, there is an infinite path in this tree. Let $\{(i_m, j_m, A_m)\}_{m=1}^{\infty}$ be the sequence of formal arguments to *parse* in the consecutive calls forming this path. Since the domain of possible triples (i,j,A) is finite, there exists a pair of equal elements $(i_{m_1}, j_{m_1}, A_{m_1}) = (i_{m_2}, j_{m_2}, A_{m_2})$, where $0 \leqslant m_1 < m_2$. This implies that $i_{m_1} = i_{m_1+1} = \cdots = i_{m_2}$ and $j_{m_1} = j_{m_1+1} = \cdots = j_{m_2}$, and thus yields a derivation of the nonterminal $A_{m_1} = A_{m_2}$ from itself in the context-free grammar of conjuncts G'. □

5. The case of linear conjunctive grammars

Linear context-free languages are known to be recognizable in quadratic time and linear space, and the same complexity upper bound holds in respect to linear conjunctive languages.

It turns out that our new parsing algorithm can be easily modified to work in quadratic time and linear space for arbitrary linear conjunctive grammars.

Let us consider the innermost loop in Algorithm 1:

$$\begin{aligned} &\text{for } i = k-1 \text{ to } 0\\ &\quad t_{ij} = t_{ij} \cup (t_{ik} \times t_{kj}). \end{aligned} \tag{43}$$

Note that the only possible effect of this assignment is the addition of one or more dotted conjuncts of the form $A \to uB \cdot v$ to t_{ij}, if $A \to u \cdot Bv \in t_{ik}$ and t_{kj} satisfies some other condition, which is not relevant in this context. This can take place only if the dotted conjunct $A \to u \cdot Bv$ is (i,k)-consistent, i.e.

$$S \overset{a_1 \ldots a_i}{\rightsquigarrow} A, \tag{44a}$$

$$u \Longrightarrow^* a_{i+1} \ldots a_j. \tag{44b}$$

The condition (44b) is satisfied if and only if $u = a_{i+1} \ldots a_k$, i.e. only if $|u|$ is equal to $k-i$. Hence, the dotted conjunct $A \to uB \cdot v$ can possibly be added to t_{ij} only on the $|u|$th iteration of the innermost loop. Since there is only a finite number of conjuncts in the grammar, there is an upper bound for the length of u for all conjuncts, and therefore it is possible to limit the number of iterations in the loop (43) by a constant depending only on the grammar.

Definition 16. Let G be a linear conjunctive grammar. The number

$$d = \max_{A \to uBv \in conjuncts(G)} |u|, \tag{45}$$

associated with the grammar, will be called the width of the grammar.

Now we can modify the original algorithm by replacing the loop (43) with

$$\begin{aligned} &\text{for } i = k-1 \text{ to } \max(0, k-d),\\ &\quad t_{ij} = t_{ij} \cup (t_{ik} \times t_{kj}). \end{aligned} \tag{46}$$

The resulting algorithm creates a recognition matrix identical to the one constructed by Algorithm 1; therefore, this algorithm is correct by the same Theorem 2.

So far we have obtained an $O(n^2)$-time and $O(n^2)$-space algorithm. Let us now show that the space requirements can be reduced to linear. Consider a single jth iteration of the outer loop of the algorithm, in which the j-th column of the matrix is constructed:

- The Scanner uses the previous $(j-1)$th column.
- The statement $t_{kj} = t_{kj} \cup t_{kk} * t_{kj}$ from the Completer is executed for each k ($0 \leqslant k < j$), i.e. the computed part of the main diagonal of the matrix is used.
- The statement $t_{ij} = t_{ij} \cup t_{ik} \times t_{kj}$ from the Completer is executed for all i and k, such that $\max(0, k-d) \leqslant i < k$, and therefore uses the band consisting of d diagonals of the matrix located right above the main diagonal.
- The Predictor does not use any elements not from the current column.

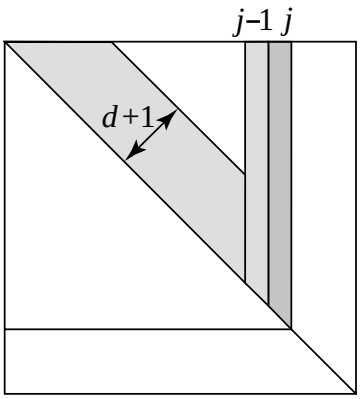

Fig. 8. Elements used at jth iteration of the algorithm.

The parts of the matrix used at the jth iteration of the algorithm are shown in Fig. 8. It is easily seen that it suffices to store a $(d+1)$-wide band of diagonals and two last columns of the matrix — i.e. $O(n)$ of elements.

It is interesting to note that the result of this section also holds in respect to the original GHR algorithm, which can be modified in exactly the same way to work in quadratic time and linear space on arbitrary linear context-free grammars.

6. Membership problem

The membership problem for conjunctive grammars is defined as follows: given a conjunctive grammar $G = (\Sigma, N, P, S)$ and a string $w \in \Sigma^*$, determine whether $w \in L(G)$ or not. In light of the results obtained in [7], the decidability of this problem is clear, since it is possible to transform the given grammar to binary normal form and then use the recognition algorithm for conjunctive grammars in binary normal form.

However, unlike the similar transformation in the context-free case, in the case of conjunctive grammars there is no known polymial upper bound for the complexity of the transformation to the normal form, and thus the algorithm created in this way might have exponential time complexity.

In this section we apply the algorithm given in Section 3 to solve the membership problem in polynomial time and consequently establish its **P**-completeness, as well as the **P**-completeness of the membership problem for linear conjunctive grammars.

6.1. Polynomial solution for the membership problem

Let us consider the complexity of the algorithm as a function of both the size of the grammar and the length of the string being recognized.

- The precomputation of the relation $\overset{\varepsilon}{\rightsquigarrow}$ on the set N, which is needed for efficient computation of *predict*, can be done in $\sim |G|^3 \cdot \log |G|$ steps by taking the reflexive and transitive closure of a certain Boolean matrix.

- The initial assignment to t_{00} is executed once and involves computing one *predict*: $\sim |G|^2$.
- The statement in the Scanner is executed $\sim |w|^2$ times, each time computing one product in $\sim |G|$ steps. Total: $\sim |w|^2 \cdot |G|$.
- The statement $t_{kj} = t_{kj} \cup t_{kk} * t_{kj}$ from the Completer is executed $\sim |w|^2$ times; each execution consists of computing one $\overline{\textit{finished}}$ ($\sim |G|^2$) and one product ($\sim |G|$). Total: $\sim |w|^2 \cdot |G|^2$.
- The statement $t_{ij} = t_{ij} \cup t_{ik} \times t_{kj}$ is executed $\sim |w|^3$ times, each time computing one *finished* ($\sim |G|$) and one product ($\sim |G|$). Total: $\sim |w|^3 \cdot |G|$.
- The statement in the Predictor is executed $\sim |w|$ times, and each time one needs to compute a union of less than $|w|$ sets of dotted conjuncts ($\sim |G| \cdot |w|$) and to compute one *predict* ($\sim |G|^2$). Total: $\sim (|G| \cdot |w|^2 + |G|^2 \cdot |w|)$.

We sum up these results to get the following upper bound for the algorithm's complexity:

$$C_1 \cdot |G|^3 \cdot \log |G| + C_2 \cdot |G|^2 \cdot |w|^2 + C_3 \cdot |G| \cdot |w|^3, \tag{47}$$

i.e. $O(n^4)$, where $n = |G| + |w|$ is the total length of an instance of the membership problem.

6.2. *P-completeness of the membership problem*

Now, having a polynomial solution for the membership problem, we shall show its **P**-hardness and hence **P**-completeness.

The membership problem for conjunctive grammars is **P**-hard, because its particular case, the membership problem for context-free grammars, is known to be **P**-complete. However, let us choose a different way of proving the problem's **P**-hardness — namely, by reducing the Monotone Circuit Value Problem (MCVP), which is known to be **P**-complete [2,11], to our problem. This will later allow to extend this proof to the case of the membership problem for linear conjunctive grammars.

Let $(C_1, \ldots, C_n)$ be a monotone circuit with inputs $x_1, \ldots, x_m$, where each gate C_i is one of the following: (i) some input x_j, (ii) conjunction of two preceding gates: $C_k \wedge C_l$ $(k, l < i)$, (iii) disjunction $C_k \vee C_l$ $(k, l < i)$. The gate C_n is called the output of the circuit.

The MCVP is stated as follows: given a monotone circuit and a Boolean vector of input values $(\sigma_1, \ldots, \sigma_m)$, determine, whether the circuit's output evaluates to true.

We reduce MCVP to the membership problem for conjunctive grammars by constructing a grammar $G = (\{0,1\}, \{A_1, \ldots, A_n, X\}, P, A_n)$, where each nonterminal A_i corresponds to the gate C_i and has one rule $A_i \to A_k \& A_l$ if the gate is a conjunction of C_k and C_l, two rules, $A_i \to A_k$ and $A_i \to A_l$, if it is a disjunction gate, and a rule $A_i \to X^{j-1} 1 X^{m-j}$ if it is an input x_j. There are also the rules $X \to 0$ and $X \to 1$ for the nonterminal X.

By a straightforward structural induction it can be proved that the circuit $(C_1, \ldots C_n)$ evaluates to 1 on input $(\sigma_1, \sigma_2, \ldots, \sigma_m)$ if and only if the string $\sigma_1 \sigma_2 \ldots \sigma_m$ is in $L(G)$.

Undoubtedly, this reduction can be carried out in logarithmic space, which proves the **P**-hardness of the membership problem for conjunctive grammars. Together with the results of Section 6.1, this leads us to the following statement:

Theorem 3. *The membership problem for conjunctive grammars is **P**-complete.*

6.3. Membership problem for linear conjunctive grammars

The membership problem for linear context-free grammars is known to be **NLOGSPACE**-complete [10], i.e., it is computationally as easy as the membership problem for NFAs [6,13]. We shall now demonstrate that a similar problem for linear conjunctive grammars turns out to be harder.

Let us modify the construction of conjunctive grammar given in Section 6.2 to obtain the following result:

Theorem 4. *The membership problem for linear conjunctive grammars is **P**-complete.*

Proof. We prove **P**-hardness of the problem by reduction from MCVP. For a given monotone circuit $\{C_1,\ldots,C_n\}$ with inputs $\{x_1,\ldots x_m\}$, construct a grammar $G=(\{0,1\}, \{A_1,\ldots,A_n\}\cup\{B_{jk}\mid 1\leqslant k\leqslant j\leqslant m\}\cup\{D_0,\ldots,D_{m-1}\},P,A_n)$, where each nonterminal A_i corresponds to the gate C_i and has one rule $A_i\to A_k\&A_l$ if the gate is a conjunction of C_k and C_l, two rules, $A_i\to A_k$ and $A_i\to A_l$, if it is a disjunction gate, and a rule $A_i\to B_{j1}$ if it is an input x_j.

For the nonterminals B_{jk} and D_k the grammar contains the following rules:

$$B_{jk}\to 0B_{j,k+1}\mid 1B_{j,k+1}\quad (1\leqslant k<j\leqslant m), \tag{48a}$$

$$B_{jj}\to 1D_{m-j}\quad (1\leqslant j\leqslant m), \tag{48b}$$

$$D_k\to 0D_{k-1}\mid 1D_{k-1}\quad (1\leqslant k<m), \tag{48c}$$

$$D_0\to\varepsilon \tag{48d}$$

The resulting grammar has $O(m^2+n)$ nonterminals, which, taking into account the constant number and length of rules for each nonterminal, limits the size of the grammar to the constant times square of the size of the instance of MCVP. This reduction certainly can be done by a deterministic logspace Turing machine.

Since the problem clearly is in **P**, we conclude that it is **P**-complete. □

Acknowledgements

I would like to express my gratitude to Kai Salomaa, who read the manuscript several times and provided numerous helpful suggestions on its improvement. I would also like to thank Vladimir A. Zakharov for his valuable comments on the earlier versions of this paper. I am grateful to the anonymous referee for a number of important remarks.

A summary of the results of this paper was presented in the conference talk [8].

References

[1] J. Earley, An efficient context-free parsing algorithm, Commun. ACM 13 (2) (1970) 94–102.
[2] L.M. Goldschlager, The monotone and planar cirquit value problems are log space complete for P, SIGACT News 9 (2) (1977) 25–29.
[3] S.L. Graham, M.A. Harrison, Parsing of general context-free languages, Adv. in Com. 14 (1976) 77–185.
[4] S.L. Graham, M.A. Harrison, W.L. Ruzzo, An improved context-free recognizer, ACM Trans. Program. Languages Systems 2 (3) (1980) 415–462.
[5] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.
[6] N. Jones, Space-bounded reducibility among combinatorial problems, J. Comput System Sci. 11 (1975) 68–85.
[7] A. Okhotin, Conjunctive grammars, J. Autom. Languages Combin 6 (4) (2001) 519–535.
[8] A. Okhotin, On P-completeness of the membership problem for conjunctive grammars, an abstract of a conference talk (in Russian), Diskretnaya matematika i matematicheskaya kibernetika: trudy mezhdunarodnoi shkoly-seminara, Ratmino, 31 May–3 June 2001, (Proceedings of the International School-Seminar on Discrete Mathematics and Mathematical Cybernetics).
[9] A. Okhotin, Top-down parsing of conjunctive languages, Grammars 5 (1) (2002) 21–40.
[10] I.H. Sudborough, A note on tape-bounded complexity classes and linear context-free languages, J. Assoc. Comput. Mach. 22 (4) (1975) 499–500.
[11] H. Vollmer, Introduction to Circuit Complexity: A Uniform Approach, Springer-Verlag, Berlin, Heidelberg, 1999.
[12] Whale Calf, a parser generator for conjunctive grammars, available at http://www.cs.queensu.ca/home/okhotin/whalecalf/.
[13] S. Yu, Regular Languages, in: Handbook of Formal Languages, Vol. 1, Springer-Verlag, Berlin, 1997, p. 41–110.