Grammar systems and Boolean grammars

Master Thesis

By Bálint Bertalan

Master in mathematics

Supervisor: Erzsébet Csuhaj-Varjú, Full professor

Department of Algorithms And Their Applications, Faculty of Informatics

Eötvös Loránd University
Faculty of Sciences
2016

Acknowledgement

Firstly, I would like to thank my thesis advisor Erzsébet Csuhaj-Varjú introducing me this suggestive part of formal language theory. Without her patient, help and immense knowledge this thesis could not have been written.

I would also like to acknowledge my best friend Ádám Vida his incentive. I always wonder how enthusiastic materials scientist he is.

Finally, I need to express my gratitude to my family for their support and encouragement throughout the last couple of years. Thank you all.

Contents

Introduction

It is well-known that context-free grammars are considered as objects prescribing the syntax of a given language. Presumably the syntactical properties of a particular string do not depend on the actual context. This means, that we are able to build strings inductively using substrings: e.g. the concatenation of an article with a noun is a noun phrase, the concatenation of a noun phrase with a verb phrase is a sentence. A verb phrase is a..., and so on. Symbolically: $Sentence \to NounphraseVerbphrase; Nounphrase \to NounArticle; \ldots$. Thus $A \to \alpha$ can be read as the string has the property A iff it is representable as α. The non-compliance of the context makes context-free grammars of an obvious formalisation of the syntax. Despite their closeness to perfection regarding questions arised by decidability and complexity, their nature turned out to be inappropriate to characterize languages presented in practice. For example it was proved by Floyd in 1960s that the programming language Algol60 is not context-free, and it is not hard to prove, that modern languages such as C++ or Java are neither context-free. In order to go further and describe such a languages, more powerful models, in terms of the generative power, has been examined, the class of context-sensitive grammars. However they are much powerful, equivalent to **NSPACE**.

To avoid problems, like the use of a variable before declaration in a programming language, more strict rules should be required to represent given languages [4]. Alexander Okhotin introduced the ability to use additional Boolean operations, conjunction and negation explicitly in the right hand side of any rule [1]. So, rules have the form $A \to \alpha_1 \& \alpha_2 \& \ldots \& \alpha_m \& \neg\beta_1 \& \neg\beta_2 \& \ldots \& \neg\beta_n$. This means, that the string has the condition A iff it representable as $\alpha_1, \alpha_2, \ldots, \alpha_m$ and does not representable as any of $\beta_1, \beta_2, \ldots, \beta_n$ at the same time. This extension leads to the concept of Boolean grammars, which we demonstrate in the first part of the thesis, following the works [1, 2]. After these substantial properties we present the parsing algorithms concerning Boolean grammars, which were elaborated likewise by Okhotin in [1, 3]. We give examples also to help the reader in better understanding.

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to achieve its delegated objectives."[17]

Assume several agents, like the above description, are functioning in the same place, following a certain communication protocol and trying to develop a common symbolic description of some given task. An individual component of this community is a language determining device, a

language processor. This means, it is equipped with some set of rewriting rules of a grammar in particular type (regular, context-free, context-sensitive etc.) and communicates with other components through a common sentential form, which is derived by time according to the communication protocol. This syntactic model describes multiagent systems at the symbolic level, moreover provides a distributed model of language [5]. This concept is called Grammar system, which were introduced by Erzsébet Csuhaj-Varjú and Jürgen Dassow in 1988, the article appeared in 1990 [8], and developed further by her and Jozef Kelemen [10]. A monograph was published on the area in 1994, by Erzsébet Csuhaj-Varjú, Jürgen Dassow, Jozef Kelemen and Gheorge Păun [9].

Formally a grammar system Γ is an $n+3$ tuple: $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$, where N denotes a nonterminal alphabet, T the terminal alphabet, such that $N \cap T = \varnothing$ and $S \in N$ is the starting symbol. The components $P_1, P_2, \ldots, P_n$ are the language processors which own the rewriting rules of a particular type. They are able to perform the derivation in a cooperating distributed (CD grammar system)-, and in a parallel communicating (PC grammar system) manner (the previous ones realize steps sequentially). Another interesting notion is the concept of the team [12], where simultaneous actions of certain grammars i.e. teams are performed on a sentential form in parallel.

In the second part we describe the basic definitions and theorems of this field emphasizing the type CD grammar systems, following then papers cited above. Several examples are provided, in addition we examine the so-called final competence based derivation in more details, which leads us to a proof which shows, that this model in the k-step derivation mode, for any $k \geq 3$ is Turing-equivalent [6]. After that, we give a brief introduction to the concept of teams. Teams are installed between CD and PC grammar systems: the team attribution seizes the concept of a cooperation, and the manner, how the derivation is performed, refers to PC systems. A multiagent system is composed of several agents, which cooperate in some manner, and if it is enabled, the task could be processed in parallel. These two concepts resemble to each other.

The last part tries to reflect the question of a consensus arised by negotiation among agents. The negation in boolean rules somehow captures disagreement among agents. Accordingly, we consider the grammar systems with boolean rules, and introduce an idea to link together the field of Grammar systems and Boolean grammars. One of the most interesting question is, how to decide whether a given system is able to reach a consensus on a given string, or not. The idea is, that the protocol, which would determine how to from teams, will be the generalized LR parsing algorithm for Boolean grammars.

We assume that the reader is familiar with the notions of formal language theory, for further details consult [16]. Despite that, let us introduce some basic notations used in the thesis. The cardinality of a finite set X is denoted by $|X|$, as the length of a word w: $|w|$. The set of all words over the alphabet X is denoted by X^*, without the empty word X^+. The empty word is denoted by ε.

Budapest, May 2016

Bálint Bertalan

Chapter 1

Boolean grammars

Boolean grammars were introduced by Okhotin [1] as an augmentation of context-free grammars. The set of rules is equipped with all propositional connectives, thus with conjunction and negation also, not only with implicit disjunction which is allowed for a single nonterminal in the rules of context-free grammars as well. Basically context-free grammars reflects the fact, that a syntax does not depend on the context of a given string, thus the rule $A \to \alpha \mid \beta$ describe that a string has property A iff it is representable as α or β. But in this case we want specify all strings that are representable as α and β and not representable as γ at the same time: $A \to \alpha\&\beta\&\neg\gamma$. This leads us to a new model. Even though we represent context-free grammars as systems of language equations, we can not express such languages generated by this new model. It is well known that the intersection of two context-free language, and the complement of a context-free language may not be necessarily context-free, so in this "poor" logic of context-free grammars the conjunction and negation can't be represented with disjunction only.

1.1 Basic properties

In this section we recall the ideas concerning the rigorous formalization of the notions presented above, following [1, 2] and the introductory part of [3]. We give a strict definition of Boolean grammars and the language generated by them, although it is a bit complicated in a sense, because a contradiction expressed by a such rules $A \to \neg A$. This states that a string has a property A iff it does not have it.

1.1.1 Definition. (see [3]) A Boolean grammar is a quadruple $G = (N, T, P, S)$ where S is a starting symbol (axiom), N is a finite set of nonterminals and T is a finite of set terminal symbols such $N \cap T = \varnothing$, $S \in N$ and P is a finite set of rules of the form:

$$A \to \alpha_1\&\alpha_2\&\ldots\&\alpha_m\&\neg\beta_1\&\neg\beta_2\&\ldots\&\neg\beta_n, \quad m+n \geq 1, \alpha_i, \beta_i \in (N \cup T)^*.$$

The "chunks" of a rule of the form $A \to \alpha_i$ and $A \to \neg\beta_j$ are called conjuncts (for all i, j), positive and negative respectively. The set of all conjuncts is denoted by $\mathrm{Conj}(P)$. The conjunct with

unknown "sign" is denoted by $A \to \pm\alpha$, which means the presence of the negation is not known. The set of conjuncts of this kind is $\mathrm{Uconj}(P) = \{A \to \alpha \mid A \to \pm\alpha \in \mathrm{Conj}(P)\}$.

We can step "backward" one step if we do not allow negations ($n = 0$) for every rule. It reduces the model to the so-called conjunctive grammars, and one more step backward if even conjunctions are also prohibited ($m = 1$), this degrades the model to the standard context-free case. Assume that $m \geq 1$ and $n \geq 1$ for a particular rule, thus this rule intuitively means that if a string satisfies the syntactical conditions $\alpha_1, \alpha_2, \ldots, \alpha_m$ and does not satisfy any of the syntactical conditions $\beta_1, \beta_2, \ldots \beta_n$ at the same time, then this string satisfies the condition represented by A.

1.1 Example. (see [2]) Consider two Boolean grammars G_1, G_2 with rules:

$$\begin{array}{lll} G_1: & S \to AB\&\neg DC & \qquad G_2: \quad S \to C\&\neg AB\&\neg BA \\ & A \to aA \mid \varepsilon & \qquad\qquad A \to XAX \mid a \\ & B \to bBc \mid \varepsilon & \qquad\qquad B \to XBX \mid b \\ & C \to cC \mid \varepsilon & \qquad\qquad C \to XXC \mid \varepsilon \\ & D \to aDb \mid \varepsilon & \qquad\qquad X \to a \mid b \end{array}$$

In the case of G_1 it is easy to see that AB and CD generates the following languages $L(AB) = \{a^n b^m c^m\}$ and $L(DC) = \{a^n b^n c^m\}$. Thus, S specifies the combination of these two by its rule: $L(S) = L(AB) \cap \overline{L(DC)} = \{a^i b^j c^j\} \cap \overline{\{a^i b^i c^j\}} = \{a^m b^n c^n \mid m \neq n\}$.

G_2 generates the non-contex-free language $\{ww \mid w \in \{a,b\}^*\}$. Since $L(A) = \{uav \mid u, v \in \{a,b\}^*, |u| = |v|\}$ and $L(B)$ is the same, only the middle of the string replaced with b, we get, that the concatenation $L(AB)$ contains all strings of even length with a mismatched a on the left and b on the right, so $L(AB) = \{uavxby \mid u, v, x, y \in \{a,b\}^*, |u| = |x|, |v| = |y|\}$ and $L(BA) = \{ubvxay \mid u, v, x, y \in \{a,b\}^*, |u| = |x|, |v| = |y|\}$. Thus, S specifies that the generated strings have even length and no mismatches occur: $L(S) = \{aa, ab, ba, bb\}^* \cap \overline{L(AB)} \cap \overline{L(BA)} = \{ww \mid w \in \{a,b\}^*\}$.

A natural way is to represent grammars as language equations. For Boolean grammars such a system is presented wih the use of concatenation, union, intersection and complementation. This is apparent, but the semantics of these systems will be much more complicated, as we will see.

1.1.2 Definition. (based on [3]) Let $G = (N, T, P, S)$ be a Boolean grammar. The system of language equations associated with G is a system over T, with elements of N as variables. The equation have the following form for every nonterminal $A_k \in N$:

$$A_k = \bigcup_{A_k \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left\{ \bigcap_{i=1}^{m} \alpha_i \cap \bigcap_{j=1}^{n} \overline{\beta_j} \right\}$$

The solution is a vector of languages: $L = (L_1, L_2, \ldots, L_{|N|})$ if for each k the substitution of L_k for A_k turns each equation into an equality.

In some sense we can say that the generated language of the grammar is the first component of the unique solution vector. It is well-known that languages characterized in this manner coincide with the class of recursive sets [3]. To limit this amount of generative power we have to introduce

appropriate semantics of language equations, which will determine what solutions can be considered as unique solutions in some manner, and which comply for the basic requirements in terms of deduction. We notice that the languages $L_1, L_2 \subseteq T^*$ are considered to be equal modulo $M \subseteq T^*$ if $L_1 \cap M = L_2 \cap M$.

The presence of negation is may lead to logical contradictions. Grammars with such rules are able to generate non well-formed languages or their representation as language equations may have multiple solution [2]. To illustrate this, consider a grammar with two rules: $S \to \neg A$ and $A \to A$. The associated system is:

$$\begin{cases} S = \overline{A} \\ A = A \end{cases}$$

All solutions of this system are of the form $S = \overline{L}, A = L$ for some language $L \subseteq T^*$, but this is not a unique solution, which is an essential requirement. The uniqueness does not guarantee that a membership of some string is decidable on the membership of its substrings, which should be essential for grammars to represent induction. Although the following system has unique solution, it still leads to a contradiction:

$$\begin{cases} S = \{a\}A \cap \overline{S} \\ A = A \end{cases}$$

Assume that $A = L$ for some L, thus there is some word $w \in A$. From this we obtain a paradox $aw \in S$ iff $aw \notin S$. To determine whether $w \in A$, firstly we have to look the string aw, which cannot happen in models represent the inductive principle. So the unique solution for this system is $S = A = \varnothing$.

To avoid such problems we need additional restrictions. This will lead us to a sound mathematical definition of the language generated by a Boolean grammar.

1.1.3 Definition. (based on [2]) Consider a Boolean grammar $G = (N, T, P, S)$, and define the associated system of language equations as in Definition 1.1.2. Assume further that for every $l \geq 0$ there exist a unique vector of languages $L = (L_{A_1}, \ldots, L_{A_{|N|}})$ for every $A_i \in N$, such that $L_{A_i} \subseteq T^{\leq l}$ and a substitution of L_{A_i} for A_i turns every equation into an equality modulo $T^{\leq l}$. In this case we say that the system has a strongly unique solution and for every nonterminal A_i the generated language $L(A_i)$ is identical with L_{A_i}. Thus, the language generated by G is L_S.

Then the grammar $S \to aA\&\neg S$ and $A \to A$ which associates to the aforementioned system has two solutions modulo $T^{\leq l}$, clearly $S = A = \varnothing$ and $S = \varnothing, A = \{a^l\}$. Therefore we can consider this grammar ill-formed according to the definition of the strong unique solution.

Consider now the grammar with rules $S \to S, A \to \neg A\&\neg S$ and the associated system with it. It has unique solution $(T^*, \varnothing)$ according to the semantics of the strongly unique solution. Despite that, it is hard to explain why some particular string w belongs to the language. The interpretation could be that the second equation would form simply a contradiction if w would not be contained.

To rather comply for the intuition about the syntactical structure of strings, another semantics was introduced, which is based on an iterative method. If it terminates, it converges to one of the possible solutions of the system, by definition.

1.1.4 Definition. (based on [3]) We call a vector of languages $L = (L_1, L_2, \ldots, L_{|N|})$ a naturally reachable solution of systems associated with Boolean grammars if for every finite $M \subseteq T^*$, such that M is closed under the substring property, and for every string $w \notin M$ every sequence of vectors

$$L^0, L^1, \ldots, L^i, \ldots,$$

converges to

$$(L_1 \cap (M \cup \{w\}), L_2 \cap (M \cup \{w\}), \ldots, L_{|N|} \cap (M \cup \{w\}))$$

in finitely many steps and regardless of the choice of the components.
The initial element of this sequence is $L^0 = (L_1 \cap M, L_2 \cap M, \ldots, L_{|N|} \cap M)$. One next vector $L^{i+1} \neq L^i$ is obtained from the previous one by substituting some k-th component with $A_k(L^i) \cap (M \cup \{w\})$.

If we consider G_2 from example 1.1 as a system of language equations, we get the following system, clearly:

$$\begin{cases} S = C \cap \overline{AB} \cap \overline{BA} \\ A = XAX \cup \{a\} \\ B = XBX \cup \{b\} \\ C = XXC \cup \{\varepsilon\} \\ X = \{a\} \cup \{b\} \end{cases}$$

Let now illustrate how the iteration works to achieve a naturally reachable solution of this system. Let $M = \varnothing$ and $w = \{\varepsilon\}$. The initial vector is $L^0 = (\varnothing, \varnothing, \varnothing, \varnothing, \varnothing)$. At the first step we can choose C, thus we obtain the respective component of the vector L^1:

$$\begin{aligned} L^1_C = C(L^0) \cap (M \cup \{\varepsilon\}) = (XXC \cup \{\varepsilon\})(L^0) \cap \{\varepsilon\} = \\ \big(X(L^0)X(L^0)C(L^0) \cup \{\varepsilon\}(L^0)\big) \cap \{\varepsilon\} = \\ \varnothing \cup \{\varepsilon\} \cap \{\varepsilon\} = \{\varepsilon\}. \end{aligned}$$

Everything else stays the same, so nothing else can be substitute, $L^1 = (\varnothing, \varnothing, \varnothing, \{\varepsilon\}, \varnothing)$. For the next step S can be chosen already, thus

$$\begin{aligned} L^2_S = S(L^1) \cap (M \cup \{\varepsilon\}) = (C \cap \overline{AB} \cap \overline{BA})(L^1) \cap \{\varepsilon\} = \\ \big(C(L^1) \cap (T^* \setminus (AB))(L^1) \cap (T^* \setminus (BA))(L^1)\big) \cap \{\varepsilon\} = \\ (\{\varepsilon\} \cap T^* \cap T^*) \cap \{\varepsilon\} = \{\varepsilon\}. \end{aligned}$$

In two steps we have converged to the unique solution modulo $\{\varepsilon\}$: $(\{\varepsilon\}, \varnothing, \varnothing, \{\varepsilon\}, \varnothing)$.

Extend now M to $\{\varepsilon, a\}$ and let $w = \{aa\}$. The iteration takes the following way with the initial vector $L^0 = (\{\varepsilon\}, \{a\}, \varnothing, \{\varepsilon\}, \{a\})$. In the first step only C can be chosen, substitution with other nonterminals does not change the corresponding elements of the vector.

$$L^1_C = C(L^0) \cap (\{\varepsilon, a\} \cup \{aa\}) = (XXC \cup \{\varepsilon\})(L^0) \cap \{\varepsilon, a, aa\} = \{\varepsilon, aa\}.$$

Thus $L^1 = (\{\varepsilon\}, \{a\}, \varnothing, \{\varepsilon, aa\}, \{a\})$. For the next step S can be chosen only, again

$$L_S^2 = S(L^1) \cap (\{\varepsilon, a\} \cup \{aa\}) = \left(C \cap \overline{AB} \cap \overline{BA}\right)(L^1) \cap \{\varepsilon, a, aa\} = \{\varepsilon, aa\}.$$

Consequently the unique solution modulo $\{\varepsilon, a, aa\}$ is $(\{\varepsilon, aa\}, \{a\}, \varnothing, \{\varepsilon, aa\}, \{a\})$.

For more, extended moduli the iteration converges to a naturally reachable solution, for which the language corresponding to the first element of this vector consist all strings with even length, in addition which capture the duplicates, hence the generated language by the associated grammar, according to the semantics of the naturally reachable solution (also for the semantics of the strongly unique solution) is the well formed language $\{ww \mid w \in \{a, b\}^*\}$.

Now we have two feasible semantics for the solution of the system of language equations associated with some Boolean grammar. Therefore the vector of languages L generated by a grammar G is defined using either one of these semantics. Thus $L(G) = L_S$ (L_S denotes the first component of L).

We have to remark that it cannot be effectively decide whether a given Boolean grammar complies any of the two semantics has been introduced.

1.1.5 Theorem. *[1] The set of systems compliant to the semantics of the strongly unique solution or the semantics of the naturally reachable solution is co-**RE**-complete.*

Proof. [1] The proof concerns the strongly unique solution, for the other semantics it works same. That is, a nondeterministic Turing machine recognizes a complement of a problem and guesses an appropriate finite modulus M, and accepts iff a given system has none or more than one solution modulo M. This proves the membership in the class co-**RE**. To prove the co-**RE**-hardness we are going to reduce from the complement of the Post Correspondence Problem. Assume, an instance of PCP $\{(x_i, y_i)_{i=1}^n \mid x_i, y_i \in \Sigma^*\}$ is given, where Σ is some finite alphabet. Extend Σ to $\Sigma' = \Sigma \cup \{z_1, z_2, \ldots, z_k\}$. The system

$$\begin{cases} S = A \cap B \cap \overline{S} \\ A = z_1 A x_1 \cup \ldots \cup z_k A x_k \cup \{z_1 x_1\} \cup \ldots \cup \{z_k x_k\} \\ B = z_1 B y_1 \cup \ldots \cup z_k B y_k \cup \{z_1 y_1\} \cup \ldots \cup \{z_k y_k\} \end{cases}$$

has a strongly unique solution modulo $\Sigma'^{\leq l}$ iff the given instance is a no-instance, since if the PCP has a solution, then $L_A \cap L_B \neq \varnothing$. This will lead to a contradiction: $w \in L_S$ iff $w \notin L_S$. Therefore in case of the no-instance the unique vector will be $(\varnothing, L_A, L_B)$, where $L_A, L_B \subseteq \Sigma'^{\leq l}$. □

These semantics characterize the same class of languages. So, the languages generated by a Boolean grammar which complies to the semantics of the strongly unique solution, and the languages which are generated by a Boolean grammar which complies to the semantics of a naturally reachable solution coincides [1].

The next virtue which is maintained also relates to the terminology of the normal form. This is the generalization of the Chomsky normal form of the context-free grammars. This kind of representation is necessary e.g. for the basic parsing algorithm CYK, as we will see.

1.1.6 Definition. (based on [1]) We say that a given Boolean grammar $G = (N, T, P, S)$ is in binary normal form if every rule from P has one of the following forms:

$A \to B_1C_1\& \ldots \&B_mC_m\&\neg D_1E_1\& \ldots \&\neg D_nE_n\&\neg\varepsilon$, where $B_i, C_i, D_i, E_i \in N, m \geq 1, n \geq 0$.

$A \to a$, $a \in T$.

$S \to \varepsilon$ only if S does not appear on the right hand side of another rule.

(We will refer to $A \to B_1C_1\& \ldots \&B_mC_m\&\neg D_1E_1\& \ldots \&\neg D_nE_n\&\neg\varepsilon$ as long rule later.)

1.1.7 Theorem. *[1] Assume that an arbitrary Boolean grammar $G = (N, T, P, S)$ is given, which generates a language L under one of the two proposed semantics. Then exists and can be effectively constructed a Boolean grammar $G' = (S', N', T, P')$ in the binary normal form which generates the same language L under both semantics.*

In the next subsection we will see that languages generated by Boolean grammars can be recognized in time $\mathcal{O}(n^3)$. Before that, we describe a construction of a term rewriting system which allows the recognition in $\mathcal{O}(n)$ space. This demonstrates that the class $\mathcal{L}(Boolean)$ is involved by the class of deterministic context-sensitive languages $\mathcal{L}(dCS)$.

We have to introduce the notion of term, true term and the set of rewriting rules. Assume $G = (N, T, P, S)$ is a Boolean grammar in the binary normal form and P is linearly ordered. Denote the set of long rules with marked conjuncts by P', i.e., $P' = \{p_k \mid p$ is a long rule in P and has conjunct $p_k, 1 \leq k \leq m+n\}$. Label the nonterminals of G with signs $^+$ and $^-$, and let $N^+ = \{A^+ \mid A \in N\}$ and $N^- = \{A^- \mid A \in N\}$. Thus terms are defined over the alphabet $T \cup N \cup N^+ \cup N^- \cup P' \cup \{'(', ')'\}$.

1.1.8 Definition. (see [1]) Let us have the following definitions:

- The concept of a term:
  - For every $w \in T^+$ and $A \in N$ the objects $A(w), A^+(w)$ and $A^-(w)$ are terms for A.
  - If p_k is conjunct for A, i.e. $p_k = A \to \pm BC$, and t_1, t_2 are terms for B and C, respectively, then $p_k(t_1t_2)$ is a term for A.
- The concept of string value $\sigma(t)$ for all string $w \in T^+$ and the term t:
  - $\sigma(A(w)) = \sigma(A^+(w)) = \sigma(A^-(w)) = w$.
  - $\sigma(p_k(t_1t_2)) = \sigma(t_1) \cdot \sigma(t_2)$.
- The concept of a true term:
  - The term $A(w)$ is always true, the term $A^+(w)$ is true iff $w \in L(A)$ and the term $A^-(w)$ is true iff $w \notin L(A)$.
  - $t = p_k(t_1t_2)$ is true iff all of the following conditions hold:
    - (i) The subterms t_1 and t_2 are true.
    - (ii) If the long rule r precedes p in the linear ordering (p contains the conjunct p_k), then $\sigma(t) \notin L(r)$, where $L(r) = L(\alpha)$ for $r = A \to \alpha \in P$.

(*iii*) For every conjunct p_i which precedes p_k (both are conjunct of the rule p), $\sigma(t) \in L(p_i)$, where $L(p_i) = L(\psi)$ for $p_i = A \to \psi$.

(*iv*) For an arbitrary factorization uv of $\sigma(t_1) \cdot \sigma(t_2)$, such that $0 < |u| < |\sigma(t_1)|$ it holds that $u \notin L(B)$ and $v \notin L(C)$, where the conjunct $p_k = A \to \pm BC$.

- The set of rewriting rules:
  - Assume $a \in T$. Then the term $A(a)$ is rewritten with $A^+(a)$ if $A \to a \in P$, with $A^-(a)$ otherwise.
  - Let $T^+ \ni w = a_1 a_2 \dots a_n$. Then $A(w)$ is rewritten with $p_1(B(a_1)C(a_2 \dots a_n))$, where p is the first rule of A, and its first conjunct is $p_1 = A \to \pm BC$.
  - The term $p_k(B^+(u)C^+(v)$ is rewritten with:
    - If p_k is positive then rewrite with $A^+(uv)$, if this is the last conjunct.
    - If p_k is positive and there are more conjuncts of the rule p, then rewrite with $p_{k+1}(D(a_1)E(a_2 \dots a_n))$, where $uv = a_1 a_2 \dots a_n$ and $p_{k+1} = A \to DE$.
    - If p_k is negative then rewrite with $A^-(uv)$, if p is the last rule for A.
    - If p_k is negative and the next rule for A is r, then rewrite with $r_1(D(a_1)E(a_2 \dots a_n))$, where $uv = a_1 a_2 \dots a_n$ and $r_1 = A \to \pm DE$.
  - The rewriting of the other terms $p_k(B^+(u)C^-(v)), p_k(B^-(u)C^+(v)), p_k(B^-(u)C^-(v))$ depends on the length of the string v, as follows:
    - If $|v| > 1$, let $v = ax, a \in T, x \in T^+$. In this case rewrite with $p_k(B(ua)C(x))$.
    - If $|v| = 1$ and p_k is positive (negative) then do the same as in above, according to the positive (negative) case.

After this long definition we can draw some consequences, which will work as lemmas for the theorem proving the inclusion in $\mathcal{L}(dCS)$.

1.1.9 Lemma. *[1]*

- *True term is rewritten with a true term.*
- *Every term with string value w contains $\mathcal{O}(|w|)$ symbols.*
- *For every nonterminal $A \in N$ the term $A(w)$ is eventually transformed to $A^+(w)$ or $A^-(w)$.*

1.1.10 Theorem. *[1] Languages generated by Boolean grammars can be recognized in* ***DSPACE****(n).*

Proof. Let $G = (N, T, P, S)$ be some Boolean grammar in binary normal form. The following Turing machine can recognize $L(G)$: for a given string w write the term $S(w)$ to tape of the machine, and while is possible, perform transformations according to the rewriting rules. By the above lemma the resulting term is either $S^+(w)$ or $S^-(w)$. Hence the initial term is always true, and truth is kept (also from the above lemma), the resulting term must be also true. Thus we can accept if the result is $S^+(w)$, and reject if it is $S^-(w)$. This justifies the conclusion on whether $w \in L(G)$ or not. All terms contains $\mathcal{O}(|w|)$ symbols, thus we can convert the constructed Turing machine into a deterministic linear bounded automaton by compressing some symbols into one. □

1.2 Parsing

In this section we are going to describe two elaborated parsing algorithms for Boolean grammars, following [1] and mainly [3], henceforward.

Parsing is a process when for a given string we verify that it is well-formed or not according to a given grammar. For this we heavily utilize its substrings. The consequence also means that we can determine whether the considered string belongs to the language generated by this particular grammar. The theoretical framework, the so-called parsing schemata provides such concepts as items, which are elementary propositions, and a set of inference rules, which are used to deduce the truth of these items. In addition, parsing is strongly related with the concept parse tree.

In the following two methods, which we are going to describe now, these agreements are not fully maintained. For example the classical Cocke–Kasami–Younger (CYK) algorithm does not generate a parse tree, so in strict sense it is not a parser, rather a recognizer (but from the resulted items we can easily construct its parse tree). In the case of Boolean LR parsing the items, the arcs of a graph structured stack can be erased and set back again. This manner also does not fit into the deductive paradigm.

1.2.1 Basic parsing

An essential algorithm for recognizing ordinary context-free languages is the CYK algorithm. This requires the grammar to be in the Chomsky normal form and works in dynamic programming manner. Although the requirement of the normal form reduce its value, it gives a starting-point of more sophisticated algorithms. It creates an $n \times n$ table T for which every item $T_{i,j}$ contains certain nonterminals. This idea can be smoothly extended to Boolean grammars, as we will see.

Suppose that a given $G = (N, T, P, S)$ Boolean grammar is in the binary normal form. Then the items are the followings:

$$T_{i,j} = \{A \in N \mid a_{i+1} \dots a_j \in L(A)\}, \quad \text{for every } 0 \leq i < j \leq n.$$

For a given input string $w \in T^*, w = a_1 a_2 \dots a_n$ the initial items are $T_{i-1,i} = \{A \in N \mid A \to a_i \in P\}$ for every $1 \leq i \leq n$. To start from these sets and calculate the items correspond to longer substrings we use a function $f : 2^{N \times N} \to 2^N$. This determines the inductive step in the following way:

$$T_{i,j} = f(\bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}).$$

Assume there is a rule $A \to B_1 C_1 \& \dots \& B_m C_m \& \neg D_1 E_1 \& \dots \& \neg D_n E_n \& \neg\varepsilon$. Then A belongs to $f(R), R \subseteq N \times N$ iff $(B_s, C_s) \in R$ and $(D_t, E_t) \notin R$ for every $1 \leq s \leq m$ and $1 \leq t \leq n$. To decide whether the given input string is well-formed we have to determine $T_{0,n}$. Thus if $S \in T_{0,n}$ the input belongs to the language generated by G.

It is easy to see that the induction step is correct. This means that for all i and j, where $j - i > 1$ the substring $a_{i+1} \dots a_j \in L(B)L(C)$ iff

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

Since $\varepsilon \notin L(B)$ for every nonterminal $B \in N$ and the substring $a_{i+1} \dots a_j \in L(B)L(C)$ iff the whole string is concatenation of substrings belong to $L(B)$ and $L(C)$ respectively, but this is equivalent with $(B, C) \in T_{i,k} \times T_{k,j}$. The pseudo code is the following:

Data: input string $w = a_1 a_2 \dots a_n$
Result: sets $T_{i,j}$ of nonterminals
for $i = 1 \dots n$ **do**
    initialization: $T_{i-1,i} = \{A \in N \mid A \to a_i \in P\}$
end
for $l = 2 \dots n$ **do**
    for $i = 0 \dots n - d$ **do**
        let $j = i + l$
        let $R = \varnothing$
        for $k = i + 1 \dots j - 1$ **do**
            collecting nonterminals: $R = R \cup (T_{i,k} \times T_{k,j})$
        end
        filtering by f: $T_{i,j} = f(R)$
    end
end

Algorithm 1: CYK for Boolean grammars (based on [1])

The algorithm works in time $\mathcal{O}(n^3)$. The number of items $T_{i,j}$ is $\mathcal{O}(n^2)$ and to compute the function f takes constant time furthermore there are $\mathcal{O}(n)$ operations in the outer loop. This gives the result. In addition it obviously gives a correct result. Indeed the terminal symbol belongs to $L(A)$ iff there is a rule $A \to a$ and since $\varepsilon \notin L(B)$ for every $B \in N$ no strings of length 1 are in $L(BC)$ so none of longer rules can produce a. The form of language equations gives that $a_{i+1} \dots a_j \in L(A)$ iff there is a rule that for every $1 \leq s \leq m$ the current string is in $L(B_s)L(C_s)$ and for every $1 \leq t \leq n$ is not in $L(D_t)L(E_t)$. We have already seen this is equivalent with $(B_s, C_s) \in \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$ and $(D_t, E_t) \notin \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$. Since the algorithm computes these unions the definition of the function f gives the proper set of nonterminals for which $a_{i+1} \dots a_j \in L(A)$. The correctness of this algorithm gives that the languages generated by Boolean grammars are in **P**.

We have to remark that there are other more elaborated variants of the above algorithm which uses different data structures for which the algorithm works in $\mathcal{O}(|G| \cdot n^\omega)$ time for some $\omega < 2.376$. The most time-consuming step is to determine the set R. Namely, if for every i, j this union is determined independently not utilizing available partial information the spending linear time is unavoidable. But we can reduce the work to Boolean matrix multiplication indicate the membership of A in set $T_{i,j}$ by a product of these appropriate matrices. Thus the time required for the whole process is closely related to matrix multiplication algorithms. For more details please see e.g. [2].

1.2.2 Generalized LR parsing

The LR parser family doing bottom-up parsing as well, but guided by a parsing table constructed by a given grammar. For a certain input string the table provides information about to shift the next symbol or reduce by a particular rule in every current state. The members of the family differs only in the manner how the parsing table is constructed. To generalize this kind of parsing to Boolean grammars, the SLR(1) type was adapted [3]. We recall the essential notations concerned to LR parsing.

Firstly, we are going to construct a finite deterministic automaton. This called the LR(0) automaton which recognizes the bodies of the grammar rules. The following component is the reduction function, which tells the conjuncts recognized in a given state, depending on whether the unread portion of the input starts with a given string.

1.2.1 Definition. Assume that a Boolean grammar G is given with the set of rules P. Then the object $A \to \alpha \cdot \beta$ is called a dotted conjunct if $A \to \alpha\beta \in \mathrm{Uconj}(P)$. The set of all dotted conjuncts is denoted by $\mathrm{Dconj}(P)$.

The states of the automaton will be the subsets of $\mathrm{Dconj}(P)$, so $Q = 2^{\mathrm{Dconj}(P)} \cup \{q_{\mathrm{acc}}\}$, where q_{acc} is a distinguished state, capturing acceptance. The $\varnothing$ state is an error state, we will denote it with Err. Additional auxiliary operations, *goto* and *closure*, are required to define the initial state and the transitions between states. Let $X \subseteq \mathrm{Dconj}(P)$ and $s \in T \cup N$, then

$$goto(X, s) = \{A \to \alpha s \cdot \beta \mid A \to \alpha \cdot s\beta \in X\}.$$

$closure(X)$ is a least set of dotted conjuncts that contains X, and if $A \to \alpha \cdot B\gamma \in closure(X)$, then for each $B \to \beta \in \mathrm{Uconj}(P)$ the dotted conjunct $B \to \cdot\beta \in closure(X)$ also holds. The initial state of the automaton is

$$q_0 = closure\left(\{S \to \cdot\sigma \mid S \to \sigma \in \mathrm{Uconj}(P)\}\right).$$

The transition function $\delta : \mathrm{Dconj}(P) \times (T \cup N) \to Q$ is defined as follows:

$$\delta(q, s) = closure(goto(q, s)).$$

If $closure(goto(q_0, S)) = \varnothing$ we simply assign $\delta(q_0, S)$ with q_{acc}.

To define the reduction function, firstly we have to determine some additional objects. For an arbitrary string w define a starting substring as follows:

$$First_k(w) = \begin{cases} w, & \text{if } |w| \le k \\ \text{first } k \text{ symbols of } w, & \text{if } |w| > k \end{cases}$$

The extension to some language L is pretty straightforward: $First_k(L) = \{First_k(w) \mid w \in L\}$. And for arbitrary nonterminal A let $\mathrm{FIRST}_k(A) = First_k(L(A))$.

To determine the reduction function another terminology for a string is required. This is the set of all following sstrings for a given nonterminal, on some manner. Let us define it precisely:

1.2.2 Definition. Let $B \in N$ and assume there exists a finite sequence of conjuncts, all of the form:

$$A_0 \to \pm\alpha_1 A_1 \beta_1, A_1 \to \pm\alpha_2 A_2 \beta_2, \ldots, A_{l-1} \to \pm\alpha_{l-1} A_l \beta_{l-1},$$

such that $A_0 = S$ and $A_l = B$. We say that the string $u \in T^*$ follows B if $u \in L(\beta_l \ldots \beta_1)$. Thus for every nonterminal A define the set $\mathrm{FOLLOW}_k(A)$ as the set of proper slice strings which follow A: $\mathrm{FOLLOW}_k(A) = \{First_k(u) \mid u \text{ follows } A\}$.

To determine these sets we are going to compute there supersets $\textsc{Pfirst}_k(A)$ and $\textsc{Pfollow}_k(A)$ respectively, such that $\mathrm{FIRST}_k(A) \subseteq \textsc{Pfirst}_k(A)$ and $\mathrm{FOLLOW}_k(A) \subseteq \textsc{Pfollow}_k(A)$.
Assume that the Boolean grammar $G = (N, T, P, S)$ is compliant to the semantics of naturally reachable solution. Let $k > 0$ and $s \in T \cup N$ arbitrary. $\textsc{Pfirst}_k(s)$ is the following: if $u \in L(s)$ then $First_k(s) \in \textsc{Pfirst}_k(s)$. Then for every nonterminal A we compute $\textsc{Pfirst}_k(A)$ as follows:

For every $A \in N$ let $\textsc{Pfirst}_k(A) = \varnothing$
For every $a \in T$ let $\textsc{Pfirst}_k(a) = \{a\}$
while *new strings can be added to* $\langle \textsc{Pfirst}_k(A) \rangle$ **do**
  foreach $A \to s_{11} \ldots s_{1l_1} \& \ldots \& s_{m1} \ldots \ s_{ml_m} \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P$ **do**
    $\textsc{Pfirst}_k(A) = \textsc{Pfirst}_k(A) \cup \bigcap_{i=1}^{m} First_k(\textsc{Pfirst}_k(s_{i1}) \cdot \ldots \cdot \textsc{Pfirst}_k(s_{il_i}))$
  end
end

Algorithm 2: $\textsc{Pfirst}_k(A)$ for Boolean LR [3]

The negative conjuncts do not play any role in the result, they can be omitted. To compute $\textsc{Pfollow}_k(A)$ we are going to utilize the outcome of the this algorithm.

Assume again that the Boolean grammar $G = (N, T, P, S)$ is compliant to the semantics of naturally reachable solution, and let $k > 0$. The set $\textsc{Pfollow}_k(A)$ is the of all strings, for which if u follows the nonterminal A, then $First_k(u) \in \textsc{Pfollow}_k(A)$.

For the axiom S let $\textsc{Pfollow}_k(S) = \{\varepsilon\}$
For every nonterminal $A \in N \setminus \{S\}$ let $\textsc{Pfollow}_k(A) = \varnothing$
while *new strings can be added to* $\langle \textsc{Pfollow}_k(A) \rangle$ **do**
  foreach $B \to \beta \in Uconj(P)$ **do**
    foreach *partition* $\beta = \gamma A \delta$, *where* $\gamma, \delta \in (T \cup N)^*$ *and* $A \in N$ **do**
      $\textsc{Pfollow}_k(A) = \textsc{Pfollow}_k(A) \cup First_k(\textsc{Pfirst}_k(\delta) \cdot \textsc{Pfollow}_k(B))$
    end
  end
end

Algorithm 3: $\textsc{Pfollow}_k(A)$ for Boolean LR [3]

Now we can define the reduction function $R : Q \times T^{\leq k} \to 2^{\mathrm{Conj}(P)}$. This is

$$R(q, w) = \{A \to \alpha \mid A \to \alpha\cdot \in q, w \in \textsc{Pfollow}_k(A)\}.$$

For every rule in P, if the rule $A \to \alpha \in R(q,u)$ we can reduce with in the actual state, with lookahead w.

These constructions complete the whole SLR(k) table. The next step is to define the generalized LR algorithm for Boolean grammars.

The Knuth's LR algorithm uses a linear stack as memory. In Tomita's generalized LR algorithm, which provides the principle of the Boolean case, the memory is represented with different data structure, called graph structured stack. This is a directed graph with a distinguished node, called the source node. With this we are able to represent all possible branches which can occur in the nondeterministic computation. The nodes are labelled with states of the previously created LR automaton and a number corresponding the position in the input, the arcs with symbols from $T \cup N$. One particular set of nodes is designated, called the top layer of the stack. Every label of the top layer's nodes must be pairwise distinct, and arcs leaving any node from this set must go to another node in the next top layer. Precisely:

1.2.3 Definition. Assume that a Boolean grammar $G = (N, T, P, S)$ is given with its LR automaton (Q, q_0, δ, R), and some input string $w = a_1 a_2 \dots a_{|w|}$. Then the graph structured stack is a directed graph with vertices $V \subseteq Q \times \{0, 1, \dots, |w|\}$ and with arcs labelled with $T \cup N$. The following condition must hold: for every arc from (p, i) to (q, j) labelled with $s \in T \cup N$, $i \leq j$ and $\delta(p, s) = q$. The set of vertices of the form (q, k), where $q \in Q$ and $0 \leq k \leq |w|$ is called the k-th layer of the graph. A nonempty layer with the actual greatest k is called the top layer.

Firstly the stack only contains a single source node, at this time it forms the top layer. The whole algorithm is an alternation of reduction phases and shift phases. In the reduction phase those arcs are manipulated which goes to the actual top layer, and in this time no input is consumed. The shift phase reads a single input symbol and forms the new top layer. For each top layer node (p, i) the algorithm looks up the entry $\delta(p, a)$ in the SLR(k) table and creates a new node $(q, i+1)$ if $\delta(p, a) = q \in Q$. Then connect this two node with the arc a. This called Shift q. It might be, that $\delta(p, a) = Err$, then no node is created. Those branches which are not extended are removed. This can cause that the whole graph is deleted. This reports syntax error.

The reduction in context-free case is performed as the following: assume that the current lookahead string is $u = First_k(a_j \dots a_{|w|})$ for a given input string w. Suppose further, there exist a top layer node q, a node p and some rule $A \to \alpha$, such that $A \to \alpha \in R(q, u)$ and p is connected to q with path α. In this case we can do the reduction operation at p, Reduce $A \to \alpha$ at p. The new arc labelled with A, which goes from p to the top layer node $q' = \delta(p, A)$ is placed. If q' does not exist currently, it is created.

In the Boolean case this is slightly complicated, because the conjunction, and especially the negation. In the reduction phase are two operations, obviously the reduction and the so-called invalidation, which means removing an existing arc, placed by an earlier reduction. The reduction from some node q with the rule $A \to \alpha_1 \& \dots \& \alpha_m \& \neg\beta_1 \& \dots \& \neg\beta_n$ requires the following condition: q must be connected with distinct nodes in the top layer by all paths $\alpha_1, \dots, \alpha_m$ and must not be connected any of the paths $\beta_1, \dots, \beta_n$. If this holds, we can perform the Reduce

$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n$ at q, adding an arc A from q to $\delta(q, A)$. Invalidation takes place if any of the above conditions don't met. This means that if there exist an arc labelled with A from q, but for some i there is no path α_i from q to some top layer node, or there is some j, for which exists a path β_j from q to some top layer node. Then the earlier reduction has to be invalidated by removing the arc labelled with A from q to $\delta(q, A)$.

The nondeterministic choice of reductions and invalidations can lead to different results, can construct different graphs. To avoid this annoying behaviour, we have to formalize this subroutine and give an algorithmic description of feasible choices. We are going to apply reduction and invalidation sequentially on the basis of collected conjuncts, which reflects the actual state of the graph. We will denote the predecessors of length l of the vertex v by $\mathrm{Pred}_l(v)$. These are simply the nodes which are connected to v by an l long path. Thus the reduction algorithm is the following:

while *the graph changes* **do**
  % Collect conjuncts
  let x be an array of sets of vertices, indexed by conjuncts $A \to \alpha$
  foreach *node* $v = (q, i_{top})$ *in the top layer* **do**
    foreach $A \to \alpha \in R(q, u)$, *where* u *is the lookahead string* **do**
      $x[A \to \alpha] = x[A \to \alpha] \cup \mathrm{Pred}_{|\alpha|}(v)$
    end
  end
  % Reduction
  let Valid $= \varnothing$, a set of arcs
  foreach *rule* $A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P$ **do**
    foreach *node* $v \in \bigcap_{i=1}^{m} x[A \to \alpha_i] \setminus \bigcup_{j=1}^{n} x[A \to \beta_j]$ **do**
      if v *is not connected to the top layer labelled with* A **then**
        place arc from v to top layer, label with A
      end
      Valid = Valid $\cup$ {arc from v to the top layer, labelled with A}
    end
  end
  % Invalidation
  foreach *arc* $a = (v, v')$ *ending in the top layer, labelled with* A **do**
    if $a \notin$ *Valid* **then**
      remove a from the graph
    end
  end
end

Algorithm 4: Reduction subroutine [3]

Now we can perform the reduction and the invalidation independently. The whole algorithm is an alternation of the shift phase and the reduction phase, thus we will embed this reduction subroutine into the whole process.

Data: input string $w = a_1 a_2 \dots a_{|w|}$
initially the stack contains a single node v, labelled with q_0, the top layer is $\{v\}$
call Reduction using lookahead $First_k(w)$
for $i = 1 \dots n$ **do**
    Shift a_i
    if *the top layer is empty* **then**
        Reject
    end
    call Reduction using lookahead $First_k(a_{i+1} \dots a_{|w|})$
    remove all nodes which can't be reached from v
end
if *exists an arc from* v *to* $\delta(q_0, S)$ *in the top layer, labelled with* S **then**
    Accept
else
    Reject
end

Algorithm 5: Generalized LR for Boolean grammars [3]

The domain of applicability depends on an essential property, the lack of circular dependence of a nonterminal upon itself, combined with a negated dependence of this nonterminal on something else. The following definition formalizes this statement:

1.2.4 Definition. (based on [3]) Assume that a Boolean grammar $G = (N, T, P, S)$ is given and let $G^+ = (N, T, P^+, S)$ be the conjunctive grammar created from G by the exclusion of negative rules. So:

$$P^+ = \{A \to \alpha_1 \& \dots \& \alpha_m \mid A \to \alpha_1 \& \dots \& \alpha_m \& \neg\beta_1 \& \dots \& \neg\beta_n \in P\}.$$

Denote the set of nonterminals of G^+ which generate the empty string by $Null(G^+)$, thus $Null(G^+) = \{A \in N \mid \varepsilon \in L_{G^+}(A)\}$. We will call the sequence of conjuncts a chain from A_1 to A_{l+1} if:

$$A_1 \to \gamma_1 A_2 \delta_1, A_2 \to \gamma_2 A_3 \delta_2, \dots, A_l \to \gamma_l A_{l+1} \delta_l, \quad l \geq 1 \text{ and } \gamma_i, \delta_i \in Null(G^+)^*.$$

If the chain ends in the same nonterminal as the start, then it is a cycle. A right-chain from A_1 to A_{l+1} is a chain from A_1 to A_{l+1} without the condition $\gamma_i \in Null(G^+)^*$.

Assume there is a cycle for the nonterminal A. We say that this cycle is negatively fed by a right chain, if there exists a chain from A to some nonterminal B such that B has a conjunct $B \to \neg\beta$.

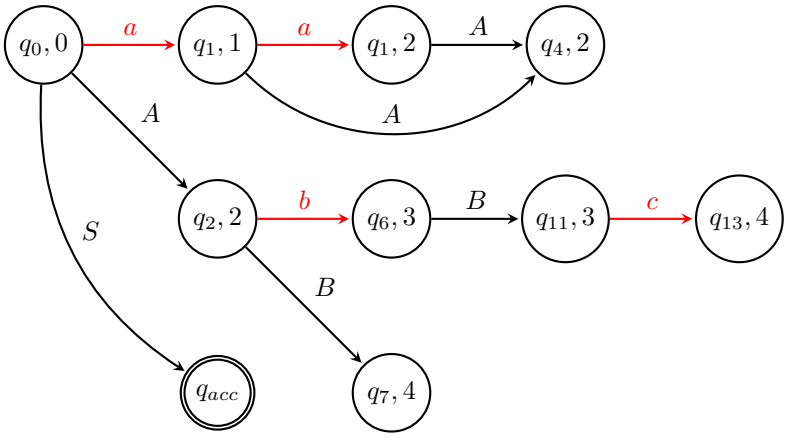
For example the simple grammar $S \to S \mid aS \mid \neg A, A \to aA \mid a$ contains a negatively fed cycle. There is a trivial cycle $S \to S$ which is fed by a negative conjunct $S \to \neg A$.

Boolean grammars which do not contain a negatively fed cycle can be recognized by the generalized LR algorithm properly:

1.2.5 Theorem. *(see [1]) Let be given a Boolean grammar* $G = (N, T, P, S)$ *without negatively fed cycles, with its SLR*(k) *table and the input string* $w \in T^*$*. Then the generalized LR algorithm respect to the given SLR*(k) *table terminates on* w *and accepts iff* $w \in L(G)$*. The upper bound of operations of the whole algorithm is* $\mathcal{O}(|w|^4)$*.*

Now we are going to illustrate how the algorithm works, using the grammar G_1 from the Example 1.1. Remember, it has rules: $S \to AB\&\neg DC, A \to aA \mid \varepsilon, B \to bBc \mid \varepsilon, C \to cC \mid \varepsilon, D \to aDb \mid \varepsilon$, and generates the language $\{a^m b^n c^n \mid m \neq n\}$ (see the SLR(1) table on the next page). Assume that the string $aabc$ is given.

Initially the graph structured stack contains only the state q_0, no arc is added. The table shows us that no reduction is applicable, since $R(q, a) = \varnothing$, thus the shifting phase is performed twice. After these shifts, the graph has three nodes and two edges labelled with the terminal symbol a, the top layer is $q_1, 2$. In the current reduction phase three new arcs are added, labelled with the nonterminal A: firstly the rule $A \to \varepsilon$ is applied, this creates the new top layer $q_4, 2$. From the node $q_1, 1$ there is a path aA, so the second A arc is presented from $q_1, 1$ to $q_4, 2$, consequence of the rule $A \to aA$. From the root, the actual top layer node is also reachable on the same path, this creates the new top layer $q_2, 2$ and an arc from $q_0, 0$ to this one likewise labelled with A, from rule $A \to aA$. The actual lookahead is b, thus we shift b, create the new top layer $q_6, 3$. In the reduction phase on the lookahead c we can reduce by the rule $B \to \varepsilon$, the new node $q_{11}, 3$ is presented. After that, we shift c. The current lookahead is ε, so in the reduction phase we reduce by the rule $B \to bBc$ from the node $q_2, 2$ and add the new top layer $q_7, 4$. This allows us to reduce by $S \to AB$ from the root. Thus we can accept the string $aabc$.



The final content of the stack

The SLR(1) table of the grammar G_1 from the Example 1.1

States of the automaton, augmented with q_{acc}		δ								R			
		a	b	c	S	A	B	C	D	ε	a	b	c
$\{S \to \cdot AB, S \to \cdot DC, A \to \cdot aA, A \to \cdot, D \to \cdot aDb, D \to \cdot\}$	0	1			q_{acc}	2			3	$A \to \varepsilon, D \to \varepsilon$		$A \to \varepsilon$	$D \to \varepsilon$
$\{A \to a \cdot A, A \to \cdot aA, A \to \cdot, D \to a \cdot Db, D \to \cdot aDb, D \to \cdot\}$	1	1				4			5	$A \to \varepsilon, D \to \varepsilon$		$A \to \varepsilon$	$D \to \varepsilon$
$\{S \to A \cdot B, B \to \cdot bBc, B \to \cdot\}$	2		6				7			$B \to \varepsilon$			$B \to \varepsilon$
$\{S \to D \cdot C, C \to \cdot cC, C \to \cdot\}$	3			8				9		$C \to \varepsilon$			
$\{A \to aA \cdot\}$	4									$A \to aA$		$A \to aA$	
$\{D \to aD \cdot b\}$	5		10										
$\{B \to b \cdot Bc, B \to \cdot bBc, B \to \cdot\}$	6		6				11			$B \to \varepsilon$			$B \to \varepsilon$
$\{S \to AB \cdot\}$	7									$S \to AB$			
$\{C \to c \cdot C, C \to \cdot cC, C \to \cdot\}$	8			8				12		$C \to \varepsilon$			
$\{S \to DC \cdot\}$	9									$S \to \neg DC$			
$\{D \to aDb \cdot\}$	10									$D \to aDb$			$D \to aDb$
$\{B \to bB \cdot c\}$	11			13									
$\{C \to cC \cdot\}$	12									$C \to cC$			
$\{B \to bBc \cdot\}$	13									$B \to bBc$			$B \to bBc$
q_{acc}	14												

Chapter 2

Grammar systems

The three main types of grammar systems which were influenced by several areas of science, mainly distributed artificial intelligence, artificial life, biology and distributed and parallel computing provide a formal language theory based framework to examine multiagent systems at symbolic level. These models has their own elaborated, far-reaching theory with objectives to answer the questions arised by communication, distribution and generative power. Grammar systems were introduced by Csuhaj-Varjú and Dassow [8] for modelling the blackboard model of problem solving at syntactic level. In the blackboard system a common knowledge base is updated from time to time, iteratively by specialized knowledge sources, starting with some specification and ending with an exact solution. They provided a system of grammars working in cooperating distributed manner, which was elaborated further by Csuhaj-Varjú and Kelemen [10]. The term cooperating grammars was first introduced by Meersman and Rozenberg in the 1970s to denote the extension of the two-level substitution mechanism of grammars to a multi-level concept. In cooperating distributed grammar systems the components of the system can be regarded as individual problem solving agents - since the nonterminals corresponds to unsolved subproblems of the whole task - who jointly solve a problem by modifying the content of the blackboard.

Other model is the so-called eco-grammar system which captures life-like phenomena among agents [11]. This language theoretic framework describes interaction with the evolving environment, dynamical changes like birth, death, overpopulation.

Another important area in the theory of grammar systems is the so-called parallel communicating grammar system (PC grammar system) introduced by Păun and Sânten [15]. In this approach grammars are located in nodes of a graph. Every component works on its own sentential form. This representation is the so-called classroom model of problem solving. Unlike the blackboard model, here every agent has its own notebook containing the description of the particular subproblem of the given task and one distinguished agent operates on the blackboard, the so-called master, who knows the whole description of the task and has the ability to decide whether the task is completed or not.

2.1 Cooperating distributed grammar systems

In this section we are going to describe the concept of cooperating distributed grammar systems, following [5] and [6]. To facilitate the easier reading, we slightly deviate from the notations used in the referred articles.

A cooperating distributed grammar system (a CD grammar system) is a finite set of generative grammars which cooperate with each other in deriving (accepting) words of a language. According to a given communication protocol, they communicate by a common sentential form which describes the actual state of the system. At any time there is exactly one such a string and the grammars work on this string in turn. The formal definition is the following:

2.1.1 Definition. (based on [6]) A CD grammar system is an $n+3$ tuple $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$, where S is a starting symbol (axiom), N is a finite set of nonterminals and T is a finite set of terminals such that $N \cap T = \varnothing$, $S \in N$ and for each $1 \leq i \leq n$ the component P_i is a set of context-free rules over $(N \cup T)$ (the context-free property means that every rule has the form $A \to \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$).

The cooperation protocol defines a strategy which leads to a language containing the words generated by the system according to this particular derivation mode. The direct derivation step of a component P_i is denoted by $x \Longrightarrow_{P_i} y$, where $x, y \in (N \cup T)^*$. The transitive, transitive and reflexive closure of the direct derivation is denoted by $\Longrightarrow^+_{P_i}$ and $\Longrightarrow^*_{P_i}$ respectively. Now we list the basic derivation modes:

2.1.2 Definition. (based on [5]) Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system $(n \geq 1)$ and assume that $x, y \in (N \cup T)^*$ are two sentential forms of Γ. Then we say that y is derived from x in Γ by one of the listed modes:

- t-mode derivation or terminal mode, denoted by $x \overset{t}{\Longrightarrow}_{P_i} y$, iff $x \Longrightarrow^*_{P_i} y$ for some component P_i and there is no $z \in (N \cup T)^*$ such that $y \Longrightarrow_{P_i} z$.
- $= k$-mode or k steps derivation for any $k \geq 1$, denoted by $x \overset{=k}{\Longrightarrow}_{P_i} y$, iff there is a sequence of strings $x_1, x_2, \ldots, x_k$ such that $x = x_1$ and $y = x_k$, and there is a component P_i such that $x_j \Longrightarrow_{P_i} x_{j+1}$ for each $1 \leq j \leq k$.
- $\geq k$-mode ($\leq k$-mode) of derivation or at least (at most) k steps derivation, denoted by $x \overset{\geq k}{\Longrightarrow}_{P_i} y$ ($x \overset{\leq k}{\Longrightarrow}_{P_i} y$), iff $x \overset{=l}{\Longrightarrow}_{P_i} y$ for some $l \geq k$ $(1 \leq l \leq k)$.
- $*$-mode of derivation or arbitrary many derivation steps, denoted by $x \overset{*}{\Longrightarrow}_{P_i} y$, iff $x \overset{=l}{\Longrightarrow}_{P_i} y$ for some $1 \leq l$.

The t-mode derivation naturally describes the competence of the current component, since it has to work on the sentential form as it is able to perform at least one derivation step. The $=, \geq, \leq k$-modes prescribe the number or a bound of the numbers of steps that a grammar has to perform in succession. Denote the basic modes by the set B, where $B = \{t, *\} \cup \{\geq, \leq, = k \mid k \geq 1\}$. Now we define the language generated by a CD grammar system in a particular mode.

2.1.3 Definition. (based on [6]) Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system ($n \geq 1$) and $m \in B$ a basic derivation mode. The language generated by Γ is the following:

$$L(\Gamma, m) = \{w \in T^* \mid S \overset{m}{\Longrightarrow}_{P_{i_1}} w_1 \overset{m}{\Longrightarrow}_{P_{i_2}} \ldots \overset{m}{\Longrightarrow}_{P_{i_k}} w_k = w, k \geq 1 \text{ and } 1 \leq i_j \leq n\}.$$

Let us now see an example of a whole system and the derivation mode.

2.1 Example. [5] Consider a CD grammar system $\Gamma = (N, S, T, P_1, P_2, P_3)$ system, where the nonterminals are $N = \{S, S', A, B, A', B'\}$, the terminal alphabet consists of two letters, so $T = \{a, b\}$ and the components have the following rules:

$$P_1 = \{S \to S', S' \to AB, A' \to A, B' \to B\}$$
$$P_2 = \{A \to aA'b, B \to cB'\}$$
$$P_3 = \{A \to ab, B \to c\}$$

Assume that the system is either in the termination, exactly 2 steps or at least 2 steps derivation mode: $m \in \{t, = 2, \geq 2\}$. Firstly, P_1 is able to perform a derivation, than we have to choose between two components which can continue. If we start with $S \overset{m}{\Longrightarrow}_{P_1} AB$ it is obvious that we obtain a string abc after the participation of the third component: $AB \overset{m}{\Longrightarrow}_{P_3} abc$. After we have picked out the second component, the sequence of the sentential form has the form: $AB \overset{m}{\Longrightarrow}_{P_2} aA'bcB' \overset{m}{\Longrightarrow}_{P_1} \ldots \overset{m}{\Longrightarrow}_{P_1} a^{n-1}Ab^{n-1}c^{n-1}B$ and is terminated by the third component again $\overset{m}{\Longrightarrow}_{P_3} a^n b^n c^n$. We obtain the language $L(\Gamma, m) = \{a^n b^n c^n \mid n \geq 1\}$ The result is a non-context-free, context-sensitive language.

Assume that now we have some different mode: $m \in \{*, = 1\} \cup \{\leq k \mid k \geq 2\}$. In this way, we obtain a different language $L(\Gamma, m) = \{a^n b^n c^m \mid n, m \geq 1\}$, where n and m are not necessarily equal. Consider e.g. the sentential forms above, and now alternate between the first and second component, pumping the substring $a^k b^k$ and leave the number of c's in the suffix $c^l B'$ less then k. This means that if we apply only one step per component we get a simple set of context-free grammars, thus the obtained language is context-free. This shows how the derivation mode influences the result.

2.2 Example. Consider now Γ int t-mode with 4 components which has all 7 rules:

$$P_1 = \{S \to A, S \to D, D \to D'D'\}$$
$$P_2 = \{D' \to D\}$$
$$P_3 = \{D \to a\}$$
$$P_4 = \{A \to a^{2^n}A, A \to a^{2^n}\}, n \geq 0$$

It easy to see that if we choose the rule $S \to D$ of the first component, we follow by doubling of D's and at end, terminate with a^{2^r}, where $r \geq 0$:

$$S \overset{t}{\Longrightarrow}_{P_1} D \overset{t}{\Longrightarrow}_{P_1} D'D' \overset{m}{\Longrightarrow}_{P_2} DD \overset{t}{\Longrightarrow}_{P_1} D'D'D'D' \overset{t}{\Longrightarrow}_{P_2} \ldots \overset{t}{\Longrightarrow}_{P_3} a^{2^r}$$

If we firstly choose the first rule of P_1 we end up with $S \overset{m}{\Longrightarrow}_{P_1} A \overset{m}{\Longrightarrow}_{P_4} a^{2^n}a^{j \cdot 2^n}$, where $j \geq 0$. So the whole systems generates $L(\Gamma, t) = \{a^{2^r} \mid r \geq 0\} \cup \{a^{2^n}a^{j \cdot 2^n} \mid j \geq 0\}$ which is a product of two

languages $\{a^{2^i} \mid 0 \leq i \leq n-1\}\{a^{j \cdot 2^n} \mid j \geq 1\}$. To generate this language with a grammar system we need only 7 rules, although that is known that any context-free grammar which generates the corresponding language must have at least $n+1$ production rules.

2.1.1 Generative power of CD grammar systems

In this section we present some well known results answering the question arised by the generative power of the model. As previously, to help the easier reading, we simplified some notations from the cited articles.

Denote the class of languages generated by a cooperating grammar systems with n context-free components, where each has at most m rules and performing the derivation in some $b \in B$ basic mode by $\mathcal{L}(CD^b_{n,m})$. It is known that whenever we change the type of the rules we won't rise the generative power of the grammar class [5]: $\mathcal{L}(CD^b_{\infty,\infty}(r)) = \mathcal{L}(r)$, where r can be any regular, linear or context-sensitive grammar. This is different when the components have context-free rules. Assume that the mode b is chosen now from $\{*, = 1, \geq 1\} \cup \{\leq k \mid k \geq 1\}$, furthermore the grammars are all context-free. The following holds [5]:

2.1.4 Theorem.

(1) $\mathcal{L}(CF) = \mathcal{L}(CD^b_{1,\infty}) = \mathcal{L}(CD^b_{2,\infty}) = \mathcal{L}(CD^b_{n,\infty})$.

(2) $\mathcal{L}(CF) = \mathcal{L}(CD^t_{1,\infty}) = \mathcal{L}(CD^t_{2,\infty}) \subset \mathcal{L}(CD^t_{3,\infty}) = \mathcal{L}(CD^t_{n,\infty}) = \mathcal{L}(ET0L)$, *for every* $n \geq 3$.

Here t means the terminal mode derivation and $ET0L$ denotes the class of extended tabled interactionless L (or Lindenmayer) systems.

An extended tabled interactionless L system is a quadruple $G = (\Sigma, T, \mathcal{P}, w)$, where Σ is an alphabet $T \subseteq \Sigma, w \in \Sigma^+$ is the axiom and $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ for some positive n. Ps are sets of context-free rules. Precisely every P_i is a finite subset of $\Sigma \times \Sigma^*$ such that for every $s \in \Sigma$ there exist at least one element in P_i in the form $s \to \sigma$. We say that $\Sigma^+ \ni x = x_1 x_2 \ldots x_n$ directly derives $\Sigma^* \ni y = y_1 y_2 \ldots y_n$ in G (denoted by $x \Longrightarrow_G y$) if there exist a component P_j such that for every $1 \leq i \leq n$ this P_j has a rule $x_i \to y_i$). The language generated by G is $L(G) = \{z \in T^* \mid w \Longrightarrow^*_G z\}$.

Theorem 2.1.4 demonstrates that systems with two components are less powerful than components with three, and the hierarchy based on the number of components collapses to three which is as powerful as the $ET0L$ systems.

If there are more components which can be activated at the same time how should we choose one of them to increase the generative power? Is there any strategy or leave this step completely nondeterministic? Csuhaj-Varjú, Dassow and Vaszil proposed a strategy how to choose between the components. These are variants of the so-called competence-based derivation [6].

As we mentioned before, the nonterminals in the sentential form can be considered as unsolved subproblems of the whole task, and the model as a multiagent system. In every step one particular component applies some rule, if there is any which is applicable. This ability can be considered as the competence of a current grammar. The measure of the competence is a number of different

nonterminals which can be rewritten by the component, thus the competence is high if this number is large [6]. Two well-known competence based derivations, which we have presented also are the t-mode and the $= k$ mode derivations. This concept is based on the common sentential form, which is a straightforward idea. But there is an other measure, the so-called efficiency [6]. This measures the number of nonterminals in the sentential form obtained after the grammar finished the work. The component is as efficient as greater extent decreases the number of different nonterminals in the string after finishing the derivation. This is called final competence.

2.1.5 Definition. (based on [6]) Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system $x \in (N \cup T)^+$ a sentential form and $m \in \{t\} \cup \{= k \mid k \geq 1\}$ a derivation mode. Denote the final competence on x at j steps ($j \geq 1$) with $FC^j(x)$, namely:

$$FC^j(x) = \min\{|\text{alph}(y) \cap N| : x \Longrightarrow_{P_i}^{j} y, \text{alph}(y) \neq \text{alph}(x), 1 \leq i \leq n\},$$

where $\text{alph}(x)$ is the minimal alphabet Σ such that $x \in \Sigma^+$, so the minimal number of distinct symbols which can generate x. We say that the component P_i is maximally efficient in j steps if there is a j step derivation performed by P_i obtain the string y such that $|\text{alph}(y) \cap N| = FC^j(x)$. This also holds for the mode $m \in B$, and P_i is a maximally efficient component on x in the m-mode derivation if it is able to perform a derivation such that for the obtained string y: $|\text{alph}(y) \cap N| = FC^m(x)$. In the case of t-mode derivation $FC^t(x)$ is defined in the obvious way.

How can we determine whether a component is a maximally efficient one? Since this measure depends on the number of nonterminals only in the derived string, we have to notice that $FC^m(x) = FC^m(A_1 A_2 \ldots A_n)$, where $N \supseteq U = \{A_1, A_2, \ldots, A_k\} = \text{alph}(x) \cap N$ thus we can augment the definition to subsets $U \subseteq N$, denoted by $FC^m(U)$. For the component P_i and the subset U define the set $M_{U,i} = \{M_1, M_2, \ldots, M_l\}$, where each M_j is a subset of N, as follows: for all words w_j which can be generated by P_i in mode m from x ($\text{alph}(x) \cap N = U$) $M_{U,i} \ni M_j = \text{alph}(w_j) \cap N$. The minimal cardinality among these sets $M_{U,i}$ will assign the final competence. Obviously we can effectively compute these kind of sets. It has only to be checked that from $A_1 A_2 \ldots A_n$ the component P_i in the m-mode is able to generate the word over $(T \cup M)$ for arbitrary subset $M \subseteq N$. Thus for every string for which $\text{alph}(x) \cap N = U$, $FC^m(U)$ and the set of all components with this maximal final competence is simply computable.

The derivations can be categorized in a Γ system by these measures.

2.1.6 Definition. (based on [6]) Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system and $m \in \{t\} \cup \{= k \mid k \geq 1\}$ the derivation mode, and let

$$D : S = w_0 \overset{m}{\Longrightarrow}_{P_{i_1}} w_1 \overset{m}{\Longrightarrow}_{P_{i_2}} \ldots \overset{m}{\Longrightarrow}_{P_{i_l}} w_l \in T^*$$

for some $l \geq 1$ be an m-mode derivation in Γ.

- We say that the derivation D was performed in FC^m-mode if P_{i_r} is a maximally efficient component in the m-mode derivation on w_{r-1} for every $1 \leq r \leq l$.

- We say that the derivation D was performed in SFC^m-mode (strongly maximal final competence) if D is an FC^m-mode derivation and for all $1 \leq r \leq l$ $|\text{alph}(w_r) \cap N| = FC^m(w_{r-1})$.

These two types of derivation modes are not identical since in the second manner the activated component has to produce a word w_r with $|\text{alph}(w_r) \cap N| = FC^m(w_{r-1})$, while this is not required in the first case.

$L(\Gamma, C^m)$ will denote the language generated by a grammar system Γ in C^m-mode, where $C \in \{FC, SFC\}$ and $m \in \{t\} \cup \{= k \mid k \geq 1\}$. The family of languages generated by this kind of systems is denoted by $\mathcal{L}(CD, C^m)$.

2.3 Example. Consider now the system $\Gamma = \{S, \{A, B, A', B'\}, \{a, b\}, P_1, P_2, P_3\}$, where the three grammars have the following production rules:

$$P_1 = \{S \to AB, A \to aA', A \to a, B \to bB'\}$$
$$P_2 = \{S \to AB, A \to aA', B \to b, B \to bB'\}$$
$$P_3 = \{A' \to A, B' \to B\}$$

Assume we are using the SFC^t-mode derivation. Since $FC^t(S) = 1$, we have to derive a word with only one nonterminal, thus after application of P_1 we obtain the string abB', after P_2 the string is going to be $aA'b$. After using P_3 the derivation has to terminate, P_1 produces aab and P_2 produces abb. We get the finite language $L(\Gamma, SFC^t) = \{aab, abb\}$.
If we are in the FC^t-mode, the situation is completely different. From the axiom S by P_1 alternated by P_3 or P_2 alternated by P_3 we can produce strings of the form $w_1 = a^n b^n B$ and $w_2 = a^n A b^n$. Since $FC^t(w_1) = FC^t(w_2) = 0$ in the first case we can only use P_2 and the second case P_1. The obtained language is $L(\Gamma, FC^t) = \{a^i b^j, a^j b^i \mid i > j \geq 1\}$.

The following holds [6]:

2.1.7 Theorem.

(1) $\mathcal{L}(CD, SFC^t) \subseteq \mathcal{L}(CD, FC^t) = \mathcal{L}(RCET0L)$

(2) $\mathcal{L}(CD, FC^{=k}) = \mathcal{L}(RE) = \mathcal{L}(CD, SFC^{=k})$ *for every* $k \geq 3$

Here $RCET0L$ denotes the class of random context $ET0L$ systems, and RE the class of recursively enumerable languages. For the proof of (1) see e.g. [6].

A random context $ET0L$ system is a quadruple $G = (\Sigma, T, \mathcal{P}, w)$, where Σ, T, w are the same as in the previously presented $ET0L$ system. $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ for some positive n such that for every $1 \leq i \leq n$, P_i is a triplet, namely $P_i = (P_i', R_i, Q_i)$, where $R_i, Q_i \subseteq \Sigma$ and $G' = (\Sigma, T, \{P_1', P_2', \ldots, P_n'\}, w)$ is an ordinary $ET0L$ system. We say that $x \in \Sigma^+$ directly derives $y \in \Sigma^*$ if $x \Longrightarrow_{G'} y$ and x contains every letter from R_i but does not from Q_i. The language generated by G is $L(G) = \{z \in T^* \mid w \Longrightarrow_G^* z\}$.

To prove (2) we need other description, in this case of the programmed grammar with appearance checking, and one auxiliary, well known fact, namely, that the class of languages generated by programmed grammars with appearance checking coincides with $\mathcal{L}(RE)$.

So the context-free programmed grammar with appearance checking is an augmented simple context-free grammar $G = (N, T, P, S)$, where N is the set of nonterminals and T is the set of terminal symbols, of course $N \cap T = \varnothing$, $S \in N$ is the axiom. P is a finite set of rules, where every rule r has the form $(r : A \to \alpha, \mathrm{Succ}_r, \mathrm{Fail}_r)$, where $A \in N$ and $\alpha \in (N \cup T)^*$ and $\mathrm{Succ}_r, \mathrm{Fail}_r \subseteq \{r_1, r_2, \ldots, r_{|P|}\}$, the success and failure sets of the rule r. These last two sets will control the derivation process. A derivation is as follows:

$$S = y_0 \Longrightarrow_{r_1} y_1 \Longrightarrow_{r_2} \ldots \Longrightarrow_{r_k} y_k = w.$$

For every $1 \leq i \leq k$ either $y_{i-1} = x_{i-1,1}A_i x_{i-1,2}, y_i = x_{i-1,1}\alpha_i x_{i-1,2}$ and if $i < k$ then $r_{i+1} \in \mathrm{Succ}_{r_i}$, or $y_i \cap \{A_i\} = \varnothing, y_{i-1} = y_i$ and if $i < k$ then $r_{i+1} \in \mathrm{Fail}_{r_i}$. We define the language generated by G as $L(G) = \{w \in T^* \mid S \Longrightarrow^*_{r_{i_j}} w\}$. The class of languages generated by context-free programmed grammars with appearance checking and ε-rules is denoted by $\mathcal{L}(PR, ac, \varepsilon)$. We note, that this notation slightly differs from the usual one.

Proof. (of the Theorem 2.1.7 (2)) Firstly we prove that $\mathcal{L}(CD, FC^{=k}) \subseteq \mathcal{L}(RE)$. Let $L' = L(\Gamma, FC^{=k}) \in \mathcal{L}(CD, FC^{=k})$. We know that a language is recursively enumerable iff there is an enumeration procedure that enumerates all strings from the language. Since the system is in FC-mode in a given time of the derivation process we can list all components from $I(U)$ which have the maximal final competence on a sentential form x such that $\mathrm{alph}(x) \cap N = U$. Thus the "enumerator machine" for a string $str \in \Sigma^*$ (Σ is the alphabet of Γ) can easily check the membership for L' while starting from S and always choose components from $I(U)$. The checking always halts and the machine outputs str if there is a derivation sequence which derives this given string, and outputs nothing if there isn't. Moreover this works for $\mathcal{L}(CD, SFC^{=k})$ as well.

[6] Now, only it has left to show the inclusion $\mathcal{L}(PR, AC, \varepsilon) \subseteq \mathcal{L}(CD, FC^{=k})$ (this is sufficient, since the previous statement). Assume that a programmed grammar $G = (N, T, P, S)$ is given, and $k = 3$. We are going to construct a CD grammar system Γ which will simulate G. Assume that G has rules of the form $A \to \alpha$, where $\alpha = BC$ or $\alpha \in (T \cup \{\varepsilon\})$, $B, C \in N$ and ε denotes the empty string. Then Γ will be the following:

$$\Gamma = (N', T, P_0, P_{r,i}, \mathcal{P}_r, P_{\bar{r},1}, P_{\bar{r},2}, P_A, P_{t,1}, P_{t,2}, \bar{S}).$$

$P_{r,i}, \mathcal{P}_r, P_{\bar{r},1}, P_{\bar{r},2}, P_A$ are sets of rule sets, where $r \in \{r_1, r_2, \ldots, r_{|P|}\}, 1 \leq i \leq 5$ and $A \in N$. The set of nonterminals N' is the augmentation of N by other auxiliary symbols:

$$N' = \{A, A_r, W_r, W'_r, [r], [r]', [r]'', [\bar{r}], [\bar{r}]', [\bar{r}]''\} \cup \{[t], [t]', [t]'', X, X', X'', Y, Y', Z, W, F, S', S'', \bar{S}\},$$

where $A \in N$ and $r \in \{r_1, r_2, \ldots, r_{|P|}\}$. The components have the following rules:

$$P_0 = \{\bar{S} \to S', S' \to S''\} \cup \{S'' \to [r]SXY, S'' \to [\bar{r}]SXY \mid r \in \{r_1, r_2, \ldots, r_{|P|}\}\}$$

For all rules $(r : A \to BC, \mathrm{Succ}_r, \mathrm{Fail}_r)$ of G we define 5 different $P_{r,i}$ components as follows:

$$P_{r,1} = \{A \to B_r C_r W[r]', X \to X', Y \to Y'\}$$

$$P_{r,2} = \{[r] \to [r]'', [r]'' \to Z, [r]' \to [t]\} \cup \{[r]' \to [s], [r]' \to [\bar{s}] \mid s \in \mathrm{Succ}_r\}$$

$$P_{r,3} = \{Z \to \varepsilon, X' \to X, Y' \to Y\}$$

$$P_{r,4} = \{B_r \to B, C_r \to C, W \to \varepsilon\}$$

$$P_{r,5} = \{[r] \to [r]'', [r]'' \to F\} \cup \{[s]' \to F \mid s \neq r\}$$

For the rules $(r : A \to \alpha, \mathrm{Succ}_r, \mathrm{Fail}_r)$, where $\alpha \in T \cup \{\varepsilon\}$ the rules $P_{r,2}, P_{r,3}, P_{r,5}$ remains the same, $P_{r,1}$ and $P_{r,4}$ change:

$$P_{r,1} = \{A \to W_r W_r' W[r]', X \to X', Y \to Y'\}, P_{r,4} = \{W_r \to \alpha, W_r' \to \varepsilon, W \to \varepsilon\}$$

The set $\mathcal{P}_r$ has three members: $\mathcal{P}_r = \{P_1^r, P_2^r, P_3^r\}$, where for every r:

$$P_1^r = \{[r]' \to F, X \to F, Y \to F\}, P_2^r = \{[r]' \to F, X' \to F, Y \to F\},$$
$$P_3^r = \{[r]' \to F, X \to F, Y' \to F\}$$

It "just" remains to define the last 5 types:

$$P_{\bar{r},1} = \{[\bar{r}] \to [\bar{r}]', [\bar{r}]' \to [\bar{r}]'', [\bar{r}]'' \to [s], [\bar{r}]'' \to [\bar{s}], [\bar{r}]'' \to [t] \mid s \in \mathrm{Fail}_r\}$$

$$P_{\bar{r},1} = \{X \to X', X' \to F, [\bar{r}] \to A\}$$

For every nonterminal $A \in N$ let P_A be the following: $P_A = \{[t] \to A, X \to X', X' \to F\}$ and $P_{t,i}$ be:

$$P_{t,1} = \{[t] \to [t]', X \to X', X' \to X''\}, P_{t,2} = \{[t]' \to \varepsilon, Y \to \varepsilon, X'' \to \varepsilon\}.$$

Assume that the sentential form $w = w_1 w_2$, which belongs to G is given. Then the corresponding sentential form of Γ has the form $w_1[r]w_2XY$ or $w_1[\bar{r}]w_2XY$, where $[r]$ or $[\bar{r}]$ captures that the next rule r performed in G is from Succ or Fail set, respectively, and X, Y are marker nonterminals. Assume the current sentential form of Γ has the form $w_1[r_i]w_2XY$ for some $r_i \in \{r_1, r_2, \ldots, r_{|P|}\}$. For some r (not necessarily $r = r_i$) there is an applicable component $P_{r,1}$. Consider one of these components and suppose that the rule r has form $(r : A \to BC, \mathrm{Succ}_r, \mathrm{Fail}_r)$.

We are going to show that to continue the derivation it is required to rewrite only one A and choose the component for which $r = r_i$. Assume these do not hold. Depending on the occurrences of A in the sentential form, at least one of X or Y remains unprimed (since we are in $FC^{=3}$-mode). Thus the components in $\mathcal{P}_r$ and the components $P_{r,4}, P_{r,5}$ would be able to continue the derivation. Because A was rewritten more then once, the symbol $[r]'$ is present more then one place, thus due to the assumption $r \neq r_i$, the component $P_{r,5}$ cannot decrease the number of nonterminals, therefore in some order one component from $\mathcal{P}_r$ and the $P_{r,4}$ must be applied. These two lead to the trap symbol F, from the procedure can't be continued. We obtain that the derivation of Γ is successful only when $P_{r,1}$ rewrites only one A to $B_r C_r W[r]'$ and the XY to $X'Y'$. From this sentential form $P_{r,4}$ produces (at least once) BC from $B_r C_r$ and erases W. If $r \neq r_i$ the trap symbol F can be present again by $P_{r,5}$. So $r = r_i$ is also necessary. This means that the component $P_{r_i,1}$ corresponding to the rule r_i of G was applied. Then the derivation can be continued with $P_{r,2}$ which only changes $[r_i]'$ to $[s]$, $[\bar{s}]$ or $[t]$. $P_{r_i,3}$ is the following which can

decrease the number of nonterminals, replacing Z with the empty string and change back $X'Y'$ to XY. The result sentential form is $w_1'[s]w_2'XY, w_1'[\bar{s}]w_2'XY$ or $w_1'[t]w_2'XY$ which fit to the form $w_1'w_2'$ obtained in G applying rule r_i and $s \in \mathrm{Succ}_{r_i}$. The sentential form $w_1'[s]w_2'XY$ is exactly the same as mentioned above, thus the same application of rules leads to a successful simulation of the application of rule s. The components $P_{\bar{s},1}$ and $P_{\bar{s},2}$ are responsible for simulating the failure of the application of rule s which is the case when we have the sentential form $w_1'[\bar{s}]w_2'XY$. Both take three steps and do not increase the number of nonterminals. If there is no A we cannot apply the rule $(s : A \to BC, \mathrm{Succ}_s, \mathrm{Fail}_s)$, in this case the component $P_{\bar{s},1}$ become active which introduces a symbol corresponding to a rule in Fail_s. If there is a nonterminal A then the derivation has to be continued with $P_{\bar{s},1}$ since this component is able to decrease the number of nonterminals. Since we want to simulate the failure of rule s we present the trap symbol F. If we have $w_1'[t]w_2'XY$ then the simulation by Γ should be finished. In this case we use the components P_A and $P_{t,1}, P_{t,2}$ which terminate the derivation. If the rules have the form $(r : A \to \alpha, \mathrm{Succ}_r, \mathrm{Fail}_r)$, where $\alpha \in (T \cup \{\varepsilon\})$ then the same thread is adaptable.

In the general case, for any $k \geq 3$ only simple modifications are needed. We might simply add more $k-3$ marker nonterminals and additional $k-3$ rules to the components which either prime or unprime these symbols in $k-3$ additional steps. This alternation between these primed and unprimed versions does not change the number of different nonterminals in the sentential form, thus everything also hold for this system. It is easy to see that Γ does not generate any terminal word which is not in $L(G)$.

To prove that $\mathcal{L}(CD, SFC^{=k}) = \mathcal{L}(RE)$ we will follow the same thread as above, although only some modifications are required in the components of Γ. $\mathcal{P}_r$ is not needed anymore and $P_{r,1}$ should has the form $P_{r,1} = \{A \to B_rC_rW[r]'X'X'', X' \to \varepsilon, X'' \to \varepsilon\}$ for the rule $(r : A \to BC, \mathrm{Succ}_r, \mathrm{Fail}_r)$ and $P_{r,1} = \{A \to W_rW_r'W[r]'X'X'', X' \to \varepsilon, X'' \to \varepsilon\}$ for the rule $(r : A \to \alpha, \mathrm{Succ}_r, \mathrm{Fail}_r)$, where $\alpha \in (T \cup \{\varepsilon\})$. □

2.1.2 Teams

In this part we are going to present the idea of a team, forming from components of a grammar system. We follow [12] mainly, and a result from [13]. As in the previous sections, we slightly deviate from the standard notations.

On the blackboard model the knowledge sources, the agents, here the corresponding grammars contribute to the problem solving by changing the content of the blackboard, in our case to rewrite the sentential form. In the sense what we have discussed so far only one component is active in a current time. What happens if we allow more than one component to be active at the same time and work parallel on the common string. A team is such a subset of all components, the number of components is prescribed in some manner and the members are chosen non-deterministically. This mixed system (sequentialism and a certain degree of parallelism) is proved to be efficient enough in the t-mode derivation. Also this object can be described as an intermediate model between the CD grammar systems and the parallel communicating grammar systems, which is the grammatical model of parallel computing. The interested reader is referred to e.g. [15].

2.1.8 Definition. (based on [12]) Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, w)$ be a context-free grammar system, which is anologous to the previous definition, except that the axiom is not a simple nonterminal (S), but a string $w \in (N \cup T)^*$. Simply an s-team $(s \geq 1)$ is a subset τ_s of $\{P_1, P_2, \ldots, P_n\}$, namely $\tau_s = \{P_{i_1}, P_{i_2}, \ldots, P_{i_s}\}$. For an s-team τ_s and for $x, y \in (N \cup T)^*$ we say that x directly derives y $(x \Longrightarrow_{\tau_s} y)$ iff

$$x = x_1 A_1 x_2 \ldots x_s A_s x_{s+1}, y = x_1 y_1 x_2 \ldots x_s y_s x_{s+1}, \text{ where } x_j \in (N \cup T)^*, 1 \leq j \leq s+1$$
$$\text{and } A_j \to y_j \in P_{i_j}, 1 \leq j \leq s.$$

So every component in the team uses one of its rules in parallel. The derivation modes, what we defined in 2.1.2 are also valid in terms of the teams. The derivation in t-mode is successful only when no more rules of any of the team components are applicable. Say we have a team τ_2 with two components, with rules $A \to a$ and $B \to b$ respectively. Then the string $AABB$ is rewritable in 2 steps, but the string AAB cannot be rewritten since $AAB \Longrightarrow_{\tau_2} aAb$ and A will still present, however B won't.

The language generated by Γ with teams τ_s in mode m is $L(\Gamma, m, \tau_s) = \{z \in T^* \mid w \Longrightarrow^*_{\tau_s} z\}$. The class of all languages generated by such a system is denoted by $\mathcal{L}(TCD^m)$, where $\mathcal{L}(TCD^m) = \bigcup_{s \geq 1} \mathcal{L}(TCD^m, \tau_s)$.

2.4 Example. Assume we have a system $\Gamma = (AB, \{A, B, A', B'\}, \{a, b, c\}, P_1, P_2, P_3, P_4)$ with components:

$$P_1 = \{A \to aA'b, A \to ab\} \quad P_2 = \{B \to cB', B \to c\}$$
$$P_3 = \{A' \to A\} \quad P_4 = \{B' \to B\}$$

The axiom string is AB, thus at any time in the derivation process if a sentential form has only one nonterminal, then no team can work. The rules $A \to ab$ and $B \to c$ can be used in the last step of a process. The only teams what we can form are $\tau_2^1 = \{P_1, P_2\}$ and $\tau_2^2 = \{P_3, P_4\}$. Along the process these two teams alternate, the possible derivations has the form:

$$AB \overset{m}{\Longrightarrow}_{\tau_2^1} aA'bcB' \overset{m}{\Longrightarrow}_{\tau_2^2} aAbcB \overset{m}{\Longrightarrow}_{\tau_2^1} \ldots \overset{m}{\Longrightarrow}_{\tau_2^2} a^{n-1}Ab^{n-1}c^{n-1}B \overset{m}{\Longrightarrow}_{\tau_2^1} a^n b^n c^n.$$

Here $m \in \{t, =1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$. The obtained language already has been featured in example 2.1.

2.5 Example. Let now examine a bit more "complicated" system. Let Γ be a same system as above, but with components:

$$P_1 = \{A \to A'A'\} \quad P_2 = \{B \to aB', B \to bB'\}$$
$$P_3 = \{A' \to AA\} \quad P_4 = \{B' \to aB, B' \to bB\}$$
$$P_5 = \{A \to c, B \to B, B' \to B'\} \quad P_6 = \{B \to cB, B \to c, A \to A, A' \to A'\}$$

It is easy to see that components P_5 and P_6 can't form two element teams with components P_1, P_2, P_3, P_4 due to that a t-mode derivation can't be correctly finished caused by the rules $A \to$

$A, B \to B, A' \to A', B' \to B'$. Starting from AB only the team $\tau_2^1 = \{P_1, P_2\}$ can be activated leads to the sentential form $A'A'aB'$ and $A'A'bB'$ according to which rule will P_2 choose. From this point the team $\tau_2^2 = \{P_3, P_4\}$ can continue and leads to A^4wB, where $w \in \{a, b\}^*$ and $|w| = 3$. After $i \geq 0$ steps we get a sentential form $A^{2^{2i}}wB$, with $w \in \{a, b\}^*$ and $|w| = 2^{2i} - 1$. The team $\tau_2^3 = \{P_5, P_6\}$ should finish the derivation by synchronizing the choice of the rules among these two members, namely P_5 will apply its rule $A \to c$ while P_6 the rule $B \to cB$ and the last $B \to c$. Thus the language generated by Γ is: $L(\Gamma, t, \tau_2) = \{c^{2^{2^n}}wc^{2^{2^n}} \mid w \in \{a, b\}^*, |w| = 2^{2n} - 1, n \geq 0\}$.

We are now interested whether this whole construction of teams leads to a greater generative capacity or not. The following theorems verify that teams only with two members in t-mode are strong enough to generate all languages what can be generated by s-teams ($s \geq 2$), moreover the class $\mathcal{L}(TCD^t, \tau_2)$ properly contains the language family $ET0L$.

2.1.9 Theorem. *[12]* $\mathcal{L}(CD^m) = \mathcal{L}(TCD^m, \tau_1) \subset \mathcal{L}(TCD^m, \tau_2)$, *where the mode* $m \in \{= 1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$.

Proof. [12] The first equality is obviously following from the definition of the teams. To prove the inclusion we are going to utilize the fact that $\mathcal{L}(CF) = \mathcal{L}(CD^m)$ (see 2.1.4 (1)). Since from example 2.4 we see that $\mathcal{L}(TCD^m, \tau_2) - \mathcal{L}(CF) \neq \varnothing$. Therefore only remains to show that $\mathcal{L}(CF) \subseteq \mathcal{L}(TCD^m, \tau_s)$ for every $s \geq 2$. Assume that an arbitrary context-free grammar $G = (N, T, P, S)$ is given. We will construct a grammar system Γ_s with teams as follow:

$$\Gamma_s = (N \cup \{A_i \mid 1 \leq i \leq s-1\}, T, P_1, P_2, \ldots, P_{s-1}, SA_1A_2 \ldots A_{s-1}),$$

where for every $1 \leq i \leq s-1$ the component P_i has rules $A_i \to A_i$ and $A_i \to \varepsilon$, moreover $P_s = P$. We are going to have only one team, with all components. At any derivation step which is not a termination, the components $P_1, P_2, \ldots, P_{s-1}$ must use the rule $A_i \to A_i$, respectively and the component P_s any rule of P. If P terminates the derivation $P_1, P_2, \ldots, P_{s-1}$ cancels out the nonterminal A_i by using $A_i \to \varepsilon$. Then it is obvious that $L(G) \subseteq L(\Gamma_s, m)$. □

Consider a system with $2, 3, 4, \ldots$-teams in t-mode derivation. Do these systems generate an ascending hierarchy of languages or this chain stabilizes for a particular value of s? Csuhaj-Varjú and Păun showed that languages generated by teams only with 2 members surprisingly includes all languages generated by teams with more members. The following holds:

2.1.10 Theorem.

(1) $\mathcal{L}(TCD^t, \tau_s) \subseteq \mathcal{L}(TCD^t, \tau_{s+1})$, *for every* $s \geq 1$ *[12].*

(2) $\mathcal{L}(TCD^t, \tau_s) \subseteq \mathcal{L}(TCD^t, \tau_2)$, *for every* $s \geq 2$ *[13].*

We want to prove now that $\mathcal{L}(TCD^t, \tau_1)$ is a proper subset of $\mathcal{L}(TCD^t, \tau_2)$ also. For this, we need two lemmas what describe some sufficient condition to be an $EDT0L$ language, which denotes the deterministic variant of $ET0L$ languages.

2.1.11 Lemma. *[12] Assume that* Σ_1 *and* Σ_2 *are two disjoint alphabets, and* $L_1 \subseteq \Sigma_1^+$ *and* $L_2 \subseteq \Sigma_2^+$, *moreover let* $f : L_1 \to L_2$ *be a surjective function. Be* $L = \{wf(w) \mid w \in L_1\}$ *a language. Then the following two predicates hold:*

(1) *If L is an $ET0L$ language, then L_2 is an $EDT0L$ language.*

(2) *If L is an $ET0L$ language and f is a bijection, then L_1 is also an $EDT0L$ language.*

2.1.12 Lemma. *[12] Assume that Σ is an alphabet which has at least 2 elements and let $k \geq 1$ given. Let $L_1 = \{w \in \Sigma^* \mid |w| = k^n, n \geq 0\}$ and $L_2 = \{w \in \Sigma^* \mid |w| = n^k, n \geq 0\}$ be two languages. Then neither L_1, nor L_2 are $EDT0L$ languages.*

2.1.13 Theorem. *[12] $\mathcal{L}(TCD^t, \tau_1) \subset \mathcal{L}(TCD^t, \tau_2)$.*

Proof. [12] It remains to show the properness since then inclusion follows from the aforementioned theorem 2.1.10. That's why we are going to create a system which generates the language does not belong to the class $ET0L$. Let Γ be the following:

$$\Gamma = (\{A, B, A', B'\}, \{a, b, c, d, e\}, P_1, P_2, \ldots, P_6, AB),$$

where the corresponding components are:

$$P_1 = \{A \to A'A'\} \quad P_2 = \{B \to aBc, B \to bBd, B \to aB'c, B \to bB'd\}$$
$$P_3 = \{A' \to AA\} \quad P_4 = \{B' \to aB'c, B' \to bB'd, B' \to aBc, B' \to bBd\}$$
$$P_5 = \{A \to e, A' \to e, B \to B, B' \to B'\}$$
$$P_6 = \{B \to B, B \to ac, B \to bd, B' \to B', B' \to ac, B' \to bd, A \to A, A' \to A'\}$$

It is easy to see that the teams we can only form are $\tau_2^1 = \{P_1, P_2\}, \tau_2^2 = \{P_3, P_4\}$ and $\tau_2^3 = \{P_5, P_6\}$. Starting from the initial string AB only τ_2^1 is applicable, leading to sentential form which contains nonterminals A' and B'. Thus τ_2^2 is on turn and so on, this alternation proceeds until τ_2^3 does not terminate the derivation. The obtained string has length $3 \cdot 2^n$, where n is the number of alternation between τ_2^1 and τ_2^2. τ_2^3 will terminate the process such the first third part of the string will have only terminals e and the remaining part will has the form $wh(w^R)$, where $w \in \{a, b\}^+$ h is a morphism such that $h(a) = c$ and $h(b) = d$ and w^R denotes the reverse of the string w. This morphism is required to describe the generated language more compactly. Thus $L(\Gamma, t, \tau_2) = \{e^{2^n} wh(w^R) \mid w \in \{a, b\}^+, |w| = 2^n\}$. Now we have to show that $L(\Gamma, t, \tau_2) \notin \mathcal{L}(ET0L) = \mathcal{L}(CD^t) = \mathcal{L}(TCD^t, \tau_1)$ (this last equality follows from definitions). Assume this is not true, $L(\Gamma, t, \tau_2)$ is an $ET0L$ language. So if we simply erase all terminal symbols e and keep everything else the same, we obtain the language $L = \{wh(w^R) \mid w \in \{a, b\}^+, |w| = 2^n\}$ which also belongs to the family of the $ET0L$ languages. Therefore from Lemma 2.1.11 (2) we get that the language $L_1 = \{w \mid w \in \{a, b\}^+, |w| = 2^n\}$ must be an $EDT0L$ languages since the morphism h is a bijection, but this contradicts to the Lemma 2.1.12. □

It is proved that grammar systems with teams of consisting of two components in the t-mode characterize the family of recursively enumerable languages [14]. Since the proof is quiet complicated, we omit the details.

2.2 Grammar systems with Boolean grammars

So far we have examined grammar systems with context-free rules. In this section we are going to provide an absolutely natural way to change the type of rules of the components to boolean ones, and look at how the system can generate strings, regard to its current set of rules, which can be considered as a kind of knowledge-base, or how it is applicable to judge, that a description of a certain given problem is correct, or it does not fit any demand. We can regard this section as **an attempt, an idea which needs further elaboration**, how to link the field of Boolean grammars to Grammar systems, i.e. utilize the negative conjuncts in rules to express disagreement among agents, during the process of negotiation.

We consider the components of the system as problem solving agents, which have some partial knowledge about their perceptible environment, or are able to give complete or particular solution of a given task. For example, assume that there are trading bots, softwares which sell/buy products, shares on the stock exchange. A component can be considered such a bot, its set of rules as a representation of its knowledge, and the task is to decide whether to make an action is admissible or not. These single bots e.g. crawl the web and collect information which is then used to make decisions. From their individual knowledge we can prepare their distributed knowledge-base. This intriguing notion, a distributed knowledge is a kind of knowledge which is implicitly presented in a group. It is possible that nobody knows some statement denoted by ψ exactly, but it can be derived from more different ones, i.e. somebody knows φ and somebody knows that $\varphi \to \psi$ is true. We will make a set of rules which we will use to decide a string, but first of all we want to see how generate a language with such a system. To present this idea, we are going to exploit the notion of the team.

Consider a CD grammar system $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$, where $n \geq 1, N \cap T = \varnothing$ and $S \in N$ as before, but the rules have the form: $A \to \alpha_1 \& \alpha_2 \& \ldots \& \alpha_m \& \neg\beta_1 \& \neg\beta_2 \& \ldots \& \neg\beta_n$, where $m + n \geq 1$ and $\alpha_i, \beta_i \in (N \cup T)^*$. We have seen, there is no definition of Boolean grammars by rewriting, but with system of language equations. In our case we want to mix the kind of rewriting with this language equation paradigm.

Among the components there is a distinguished one, P_1, which owns the starting symbol S. Thus, if we have the starting rule $S \to \alpha_1 \& \alpha_2 \& \ldots \& \alpha_m \& \neg\beta_1 \& \neg\beta_2 \& \ldots \& \neg\beta_n$, the generated language will be $L(S) = L(\alpha_1) \cap \ldots \cap L(\alpha_m) \cap \overline{L(\beta_1)} \cap \ldots \cap \overline{L(\beta_m)}$. If α_i, β_j for $1 \leq i, j \leq m + n$ contains nonterminals $A_1, A_2, \ldots, A_k$, then those components will be activated, which are able to provide the languages $L(A_1), L(A_2), \ldots L(A_k)$. They will form a team, and perform a substitution in parallel. Of course, it is possible, that they can only answer partially. In this case they simply substitute a boolean expression according their rules. After in such substitution we perform the operations of the intersections and present complementations. If more components are able to answer $L(A_i)$, we choose nondeterministically. The derivation is a sequence of forming such teams, according to the arised subproblems represented by certain nonterminals.

The key question about negotiation is the existence of the consensus. Is there a way, how the

system, the deliberating agents can achieve an agreement? Assume some string *str* is given, and the task is to decide whether this string complies to the distributed knowledge. To judge it, we are going to apply the generalized LR algorithm 5 for Boolean grammars. The protocol, how to gather components into one team, is determined by reductions and invalidations occurring in the course of the algorithm. We are going to make up a grammar $G = (N, T, D, S)$, which will represent the distributed knowledge of the system, actually some set of the rules of the component's, denoted by D. Firstly, let D be a multiset of all rules of all components, and assume there is a linear ordering on the right-hand side of the rules. In the next step, we omit some rules in a following way: for the rules with the same nonterminal on the left we pick up that, which is more strict, gives more restriction. This means, if the right-hand side of the other is the continuation of this one, we leave out the shorter: if $r_1 : A \to \pm\alpha_1 \& \ldots \& \pm \alpha_k$ and $r_2 : A \to \pm\alpha_1 \& \ldots \& \pm \alpha_k \& \pm \alpha_{k+1} \& \ldots$ are given, we choose r_2. If there are two rules $A \to \alpha$ and $A \to \neg\alpha$ we choose nondeterministically. If G consists negatively fed cycles (see the Definition 1.2.4) we say, that the knowledge is inconsistent and a consensus is not reachable. If it does not hold, we can perform the algorithm. Assume a string *str* is given. According to the SLR(1) table the reductions make the proper arcs labelled with the associated nonterminals along the process. We say, those components agree, which own a rule applied in the current reduction/invalidation step. These form a team. The invalidations react to if we have to make corrections, this seems, those teams which caused this mistake were wrong before. We call the set of components respective to the invalidated rules corrector grammars. The consensus is reachable if there is an arc labelled with S to the final top layer node, as in the algorithm termination.

Bibliography

[1] A. Okhotin: *Boolean grammars*, Information and Computation 194(1): 19-48 (2004)

[2] A. Okhotin: *Conjunctive and Boolean grammars: the true general case of the context-free grammars*, Computer Science Review 9: 27-59 (2013)

[3] A. Okhotin: *Generalized LR parsing algorithm for Boolean grammars*, International Journal of Foundations of Computer Science 17(3): 629-664 (2006)

[4] A. Okhotin: *Describing the syntax of programming languages using conjunctive and Boolean grammars*, submitted

[5] E. Csuhaj-Varjú: *Grammar Systems*, Formal Languages and Applications (Studies in Fuzziness and Soft Computing 148). Ed. by C. Martín-Vide, V. Mitrana and Gh. Păun. Springer, Berlin: 275-310 (2004)

[6] E. Csuhaj-Varjú, J. Dassow and Gy. Vaszil: *Variants of competence-based derivations in CD grammar systems*, International Journal of Foundations of Computeer Science 21(4): 549-569 (2010)

[7] H. Bordihn and B. Reichel: *On Descriptions of Context-Free Languages by CD Grammar Systems*, Journal of Automata, Languages and Combinatorics 7(4): 447-454 (2002)

[8] E. Csuhaj-Varjú and J. Dassow: *On cooperating/distributed grammar systems*, Journal of Information Processing and Cybernetics EIK 26(1-2): 49-63 (1990)

[9] E. Csuhaj-Varjú, J. Dassow, J. Kelemen and Gh. Păun: *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach Scienece Publishers, 1994

[10] E. Csuhaj-Varjú and J. Kelemen: *Cooperating grammar systems - A syntactical framework for blackboard model of problem solving*, Proceedings of the Conference on Artificial Intelligence and Information Control System of Robots, 1989, November 6-10.

[11] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová and Gh. Păun: *Eco-Grammar Systems: A Grammatical Framework for Studying Life-Like Interaction*, Artificial Life 3(1): 1-28 (1997)

[12] L. Kari, A. Mateescu, Gh. Păun and A. Salomaa: *Teams in cooperating grammar systems*, Journal of Experimental and Theoretical Artificial Intelligence 7(4): 347-359 (1995)

[13] E. Csuhaj-Varjú and Gh. Păun: *Limiting the team size in cooperating grammar systems*, Bulletin of the EATCS 49: 198-201 (1993)

[14] Gh. Păun and G. Rozenberg: *Prescribed teams of grammars*, Acta Informatica 31: 525-537 (1994)

[15] Gh. Păun and L. Sânten: *Parallel communicating grammar systems: the regular case*, Annals of the University of Bucharest, Mathematics-Informatics Series 38: 55-63 (1989)

[16] G. Rozenberg and A. Salomaa, eds.: *Handbook of Formal Languages, Vol. I-III.*, Springer (1997)

[17] M. Wooldridge: *Intelligent Agents*, Multiagent Systems, second edition, Ed. by G. Weiss. The MIT Press: 3-50 (2013)