Excel@FIT 2015
http://excel.fit.vutbr.cz

Grammar Systems Applied to Parsing

Jakub Martiško*

Abstract
Most commonly used parsers are based on the model of context free languages and grammars. This type of grammar is relatively simple to describe and design, however its descriptive power is quite limited. One of the researched approaches, which deals with this problem are grammar systems. Using the combination of more simple grammars (usually context free), grammar systems are able to describe even some context sensitive languages while maintaining relative simplicity of context free grammars. This paper proposes modification of one of the variants of grammar systems. Parser, based on this modification, is also described in the paper. This proposed parser is able to parse same set of languages that are parseable by commonly used types of parsers and even some more complex languages.

Keywords: Grammar Systems — Parsing — Formal Languages

Supplementary Material: *N/A*

*xmarti52@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Formal languages are used in many scientific disciplines, like bioinformatics, linguistics and theoretical informatics. There are many variants of models describing formal languages, that mainly differ in the complexity of languages that they are able to describe. One of the most used of those models, in the field of compiler construction, are context free grammars and pushdown automata. One of the advantages of these models is its relative simplicity and ease of use. On the other hand, there are some quite simple languages, that can not be described by these grammars.

One of the approaches, how to increase to descriptive power of context free grammars while maintaining its simplicity, are grammar systems. Grammar systems consist of set of grammars, that somehow cooperate and generate language, that can be more complex than a language, generated by the grammars on their own. On the other hand, the rewriting rules of those grammars, are no more complex than the rules of the standard grammars. There are two main types of grammar systems, one of which is briefly described in section 2.

New variant of parsing is introduced in this paper. The proposed parser is based on the grammar systems, specifically on modification of CD grammar systems that is also introduced in this paper. This new parser consists of set of context free grammars which cooperate on parsing of the input string. These grammars are able to exchange some simple information and influence the computation according to some limitations on number of rules which can be used by each of the grammars. Section 3 describes this new modification of grammar systems, while section 4 deals with the proposed method of parsing.

While a bit more complex than standard parsers, the proposed method still works in deterministic way without bactracking or any kind of guessing. On the other hand, the descriptive power of the parser is increased compared to LR parsing and the parser is able to process even some context sensitive languages.

2. What are grammar systems?

Grammar systems (abrev. as GS) consist of several formal grammars (usually context free grammars), called components, which somehow cooperate in the output string generation. Even when using only context free grammars, grammar systems are able to generate languages, that are more complex than context free languages. There are two main types of grammar systems – parallel communicating (PC) grammar systems

and cooperating distributed (CD) grammar systems . Both of those system, together with some modifications are described in [1].

2.1 CD grammar systems

CD grammar systems consist of k grammars, each defined by its own set of rules. In this type of GS, the components work in sequential order, one at each given time. The whole grammar system contains only one starting nonterminal and all components rewrite same sentential form. During the computation, one of the components is chosen (in the standard definition of CDGS, this choice is nondeterministic) and activated. This component applies some of its rewriting rules on the sentential form, deactivates and new component is chosen.

Formally, CDGS of degree n, is $(n+3)$-tupple $\Gamma = (N, T, S, P_1, \ldots, P_n)$ where:

- **N** Is an alphabet of nonterminals of the GS.
- **T** Is an alphabet of terminals of the GS, where $N \cap T = \emptyset$
- **S** $S \in N$ is the starting nonterminal of the GS.
- P_i is the set of rewriting rules of the component i for each component of the system.

Alternatively, each component can be defined as a whole grammar instead of set of rewriting rules.

How long each activated component works, is determined by so called mode of derivation. There are three main modes of derivation:

- ***-mode.** Component working in this mode can work as long as it "wants", there are no limitations on the number of rules the component can use during its activation.
- **t-mode.** Component working in this mode must rewrite every nonterminal in the sentential from which appears on the left hand side of some of the active components rules.
- **k-mode.** There are three subtypes of this mode – $\leq k, \geq k, = k$. Component working in this mode has to make at most, at least or precisely k rewritings respectively, where k is some fixed constant.

Derivation in *-mode (also called normal mode) by component P_i (denoted as $\Rightarrow^*_{P_i}$) is defined as: $x \Rightarrow^*_{P_i} y \Leftrightarrow$ there is a sequence of rules $p_1, \ldots p_n \in P_i$ such that $x \Rightarrow_{p_1} x_1 \Rightarrow_{p_2} \cdots \Rightarrow_{p_i} y$, where x and y are strings over $N \cup T$.

Derivation in t-mode (also called termination mode) by component P_i (denoted as $\Rightarrow^t_{P_i}$) is defined as: $x \Rightarrow^t_{P_i} y \Leftrightarrow x \Rightarrow^*_{P_i} y$ and there is no string z and rule $p \in P_i$ such that $y \Rightarrow_p z$.

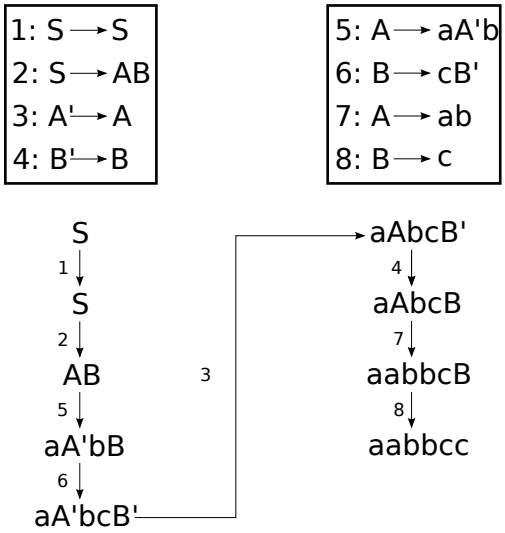

Figure 1. Example of CDGS with two components working in $= k$ mode with $k = 2$.

Derivation in $= k$-mode by component P_i (denoted as $\Rightarrow^{=k}_{P_i}$) is defined as: $x \Rightarrow^{=k}_{P_i} y \Leftrightarrow$ there is a sequence $x_1, \ldots, x_{k+1} \in (N \cup T)^*$ such that $x = x_1, y = x_{k+1}$ and for each $1 \leq j \leq k$ following holds $x_j \Rightarrow_{P_i} x_{j+1}$.

Derivation in $\leq k$-mode by component P_i (denoted as $\Rightarrow^{\leq k}_{P_i}$) is defined as: $x \Rightarrow^{\leq k}_{P_i} y \Leftrightarrow x \Rightarrow^{=k'}_{P_i} y$ for some $k' \leq k$. Similarly for $\geq k$-mode.

In the standard definition of CDGS, the derivation mode is defined for the whole system, which means, that every component works in the same derivation mode. Variant of those systems are so called Hybrid CDGS, which allow specification of unique derivation mode for each of the components of the system. Other variant, described for example in [2], are internally hybrid CDGS. Those systems allow combination of more derivation modes denoted for example as $(t \wedge = k)$. Component working in such mode must rewrite all nonterminals that appear as left hand side of any of its rules and must do so in exactly k steps.

Figure 1 presents some simple variant of CDGS consisting of two components which both work in $= 2$ variant of k mode. Only the first component can rewrite starting noterminal, so it is activated first. Because it has to apply two rules, it first rewrites S to S and then S to AB and deactivates. The second component is now activated applies rules 5 and 6. The first component is activated again and uses rules 3 and 4. The second components computation must always end with either 0 or 2 nonterminals in the sentential form, otherwise, the first component would not be able to apply exactly two rules and the whole system would get stuck. In this way, the computation continues and the GS generates string from language $\{a^n b^n c^n | n \geq 1\}$.

3. Proposed grammar system

Grammar system proposed in this paper is a variant of CDGS, more specifically their internally hybrid variant. The order, in which the components are activated is determined by one of the grammars. This grammar will be denoted as G_0 and called controlling component. Terminal symbols of this controlling component, correspond to the starting nonterminals of all other grammars, that will be called controlled components. Controlling component will thus generate sentence (from the point of view of the whole system, it will be only sentential form), which determines which components will be activated during the computation.

Formally, proposed GS is defined as:

$$\Gamma = (N, T, S, (N_0, T_0, S, P_0), (P_1, S_1), (P_2, S_2), \ldots, (P_n, S_n)),$$

where:

- N_0 is the alphabet of nonterminals of the controlling component.
- T_0 is the alphabet of terminals of the controlling component. $T_0 = \bigcup_{i=1}^{n} \{S_i\}$, $N_0 \cap T_0 = \emptyset$.
- S is the starting symbol of the whole system and the controlling component.
- P_0 is the set of rewriting rules of the form $P_0 \subseteq N_0 \times (N_0 \cup T_0)^*$.
- $N \cap T = \emptyset$.
- P_i for all $i, 1 \leq i \leq n$ is a set of rewriting rules of form $P_i \subseteq N \times (N \cup T)^*$. Those are the rewriting rules of the controlled components.
- $S_i \in N$ for all $i, 1 \leq i \leq n$ are the starting szmbols of the conrtrolled components.
- Following holds for the alphabets: $(N \cup T) \cap N_0 = \emptyset$.

The second modification deals with modes of derivation. In the proposed grammar system, the mode of derivation is not tied to the component for whole computation. Instead, each starting nonterminal, that appears on the right hand side of *controlling* components rules are assigned some mode of derivation. Component starting from such nonterminal then must work in the mode assigned to the corresponding nonterminal. In a case when come of the controlled components contains rule with starting nonterminal of some component, such starting nonterminal inherits the mode of derivation of the component that generated it.

This can be descibed by a map (a function to be more specific) d, of the form $d \subset P_0 \times D^*$, where D is set of all possible modes of derivation. This map assignes an n-tuple to each rewriting rule of the controlling component, where n is the number of occurances of starting symbols of other components at the right hand side of the rule.

In addition to the modes of derivation described in the previous section, new mode is introduced. Similarly to k modes (from now on, by k-mode, the internally hybrid variant $k \wedge t$ will always be meant), this new mode, assigns specific number of rules that must be used. This new mode, denoted as $C(l)$ does not use some constant value, instead it references some other component and component working in the C mode must then use same (at least, at most) number of rewriting rules as the referenced component. This referenced components starting nonterminal must appear in the same rule of the controlling grammar as the starting nonterminal of the component working in this mode. Symbol l in $C(l)$ then denotes, that the component references component, that starts its computation from the l-th starting symbol on the right hand side of the same rule. Only components, that work in some form of the t-mode (standard or internally hybrid) may be referenced. When the referenced component finishes its computation, all components that reference it, change their mode of derivation to appropriate variant of k-mode, where k is equal to the number of rewriting rules the referenced component used before it finished its computation.

Configuration of the system can be described as a two-tuple of the form (χ, δ), where:

- $\chi \in (N \cup T \cup N_0 \cup T_0)^*$
- $\delta = (f_1, \ldots f_k)$, where $f_j \in D, 1 \leq j \leq k$ and $k = |\chi|_{(N-N_0)}$, where $|\chi|_{(N-N_0)}$ is the number of symbols from $N - N_0$ in χ and D is set of possible modes of derivation.

Derivation step then consists of two parts, rewriting the sentential form χ and updating the tuple δ containing derivation modes. Rewriting of the sentential form is similar as a derivation steps of CDGS described in section 2.1. Updating of the tuple $\delta = (f_1, f_2, \ldots, f_k)$ consists of following steps:

- All derivation modes, corresponding to nonterminals that were not rewriten and that do not work in C-mode are left unchanged.
- All derivation modes, corresponding to nonterminals that were not rewriten and that do work in C-mode that referenced currently rewriten symbol are changed to corresponding k-mode.
- Derivation mode of currently rewriten symbol is replaced by n new derivation modes of the same type as the original derivation mode, where n is the number of new nonterminals introduced to χ.

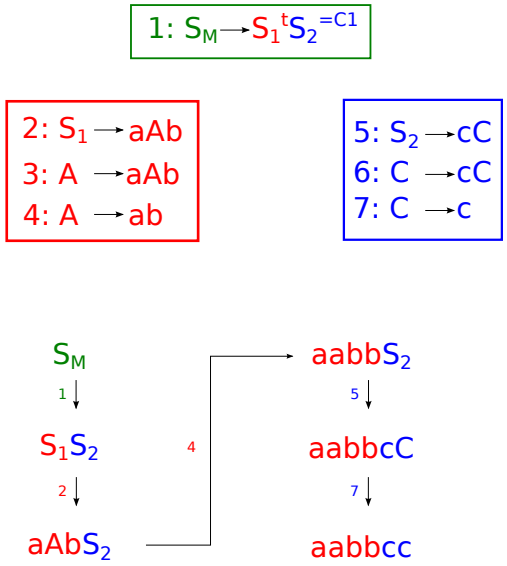

Figure 2. Example of proposed GS with two controlled components. Upper indices represent associated derivation modes.

- All derivation modes, corresponding to nonterminals that were not rewriten and that do work in $C(l)$-mode that referenced some symbol, that appeared right of the rewriten symbol, have their index l increased by $n-1$, where n is same as in previous step.

Picture 2, shows simple GS, that consists of two controlled components and one controlling component. Function d is displayed as upper indices of the starting nonterminals S_1 and S_2. First, the controlling component generates string using its only rule. Next, the controlling component with starting symbol S_1 is started. Since it works in t-mode, it uses rule 2 to rewrite nonteterminal S_1 and then it has to continue its computation, so it uses rule number 4. Both of those rewritings are considered as a single derivation step. Since there are no more symbols, that could be rewriten by this component, next one is activated. There is only one nonterminal in the sentential form, so rule number 5 must be used. Now the rules 6 or 7 can be used, however component is working in C-mode which means, that it can make only two rewritings so rule 7 must be used.

4. Modified grammar systems applied to parsing

Proposed parser combines two methods of parsing. Controlling component works in the top down approach while controlled components work in bottom up way.

Probably the most often used method of parsing is LR parsing which works in bottom up way. Main strengths of this methods are, that it is possible to parse any language that can be described by deterministic push down automaton and the parser itself works in deterministic way (without any kind of backtracking or ”guessing”). Using the proposed grammar system, this type of parser may be modified to be able to parse even some languages that are more complex than context free languages. LR parsing will be used for all of the controlled components.

The simplest variant of LR parser is SLR parser. SLR parser parses its input according to parsing table, that is constructed according to the context free grammar describing the input language. LR parsing table consists of two parts – action part and GOTO part. Except stack, LR parser ”remembers” read input using set of inner states of the parser. Each time some symbol is pushed onto the stack, it is pushed there together with state. The action part of the table tells the parser, whether it should shift input symbol or reduce the current top symbols on the stack to left hand side of some of the rule. This is determined by the current state of the parser and the input symbol. GOTO part determines the state of the parser after reduction. Detailed information about LR parsers, including the creation of the parsing table can be found in [3].

In the modification, only systems, where no controlled grammar can activate any other grammar, will be considered.

First part of the modification deals with ”remembering” of how many steps the parser made. This can be achieved by modifying the stack symbols. Instead of pushing just symbol of the alphabet (plus the state, however this part is not important for this modification and is left unchanged), the stack symbols are now two-tuples, where first element is the symbol itself and the second part is the number of reduction rules that lead to it. This second part is equal to zero for each terminal symbol. For each nonterminal, that can appear on the stack only when using some reduction rule, this value is equal to the sum of those values of each symbol on the right hand side of the rule used plus one. In this way, each component can return the number of reduction rules used during its computation to controlling grammar. Other than that, this part of parser works in the same way as standard LR parser. The parsing table is also constructed in the same way as the standard LR table.

Unlike controlled grammar, controlling grammar must choose which rule to use ”in advance”. This is important difference, because controlling grammar must activate the correct component which is achieved through rules. Controlling grammar must thus work

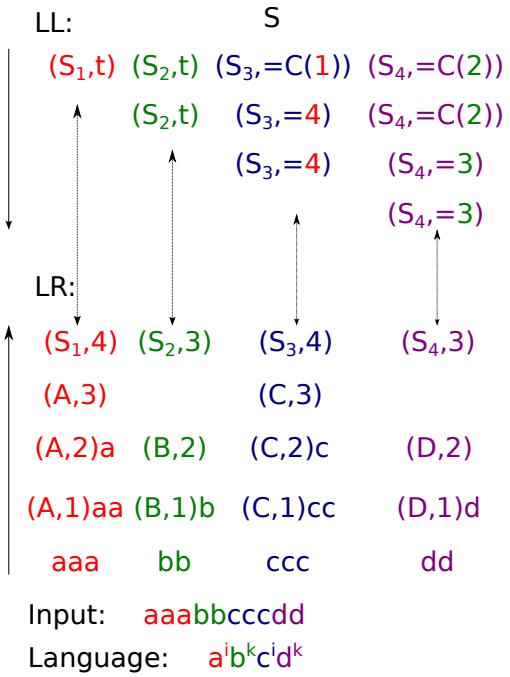

Figure 3. Example of parsing using the proposed GS.

in top down way. Typical method of top down parsing is LL parsing which is used for the controlling grammar. Each starting nonterminal is again transformed into twotuple, containing symbol itself and information about derivation mode. Instead of popping rules, LR parser for appropriate component is started. When the LR parser reduces its substring to starting nonterminal, this nonterminal is compared with the starting nonterminal on the top of the stack of LL parser and if the number of rules used is correct, symbol is popped from the stack and all starting symbols, that work in C-mode referencing this component are updated.

Since the input string is parsed as a sequence of substrings without specified length, for each activation of the component, set of symbols which represent the end of parsed substring must be specified. This set can be defined as set follow of the current starting noterminal.

Parser, based on proposed modification is shown in the figure 3. This parser consists of four controlled grammars, that work as LR parsers, and one controlling grammar, tha works as LL parser. Each of the controlled grammars is a regular grammar. The whole system is created as a concatenation of the controlled grammars, where each of those grammars generates substring consisting of the same terminal symbols. Additionaly, derivation mode is assigned to each of the grammars. Standard algorithms for parser constrution based on CFG models can be thus used, with modifications described above.

Parsing starts in the top down way. Controlling grammar generates string, consisting of starting symbols S_1, S_2, S_3 and S_4 with their corresponding derivatin modes. When using LL parsing, each terminal symbol (Starting symbols S_i behave as terminals for controling grammar) is checked whether it appears as next input symbol and popped from the stack. In the propposed parser, new parser is started instead. This parser works in bottom up way, using LR parsing. This LR parser is based on the first controlled component and ultimately reduces the input substring *aaa* to its starting nonterminal S_1. This is achieved using 4 reductions. Number of used reductions is now passed to the third component, which references just finished component and will from now on work in $=$ 4-mode, and the starting symbol of the first component is popped from the stack. Second component is started, and in LR way parses string *bb*. This is achieved by using 3 reductions. This information is passed to the fourth component and starting symbol S_2 is popped. The third component is started next, again, it works as LR parser and reduces its input string *ccc* to symbol S_3. This is completed in 4 reductions. Since the component works in $=$ 4-mode, parsing of this substring is accepted and the symbol S_3 is popped. In a similar way, the final substring *dd* is parsed by the final component. The stack of the LL parser is now empty and there are no more input symbols and the whole string is accepted and the sequence of used reductions is returned.

5. Conclusions

New modification of CD grammar systems was described in this paper. This system uses only context free grammars while the family of languages generated by the system is proper superset of family of context free languages. Parser based on this system is also described in the paper. This parser works using combination of top down and bottom up approach and is able to process even some context sensitive languages, for example language $\{a^i b^j c^i d^j | i \geq 1, j \geq 1\}$.

References

[1] Rozenberg G, Salomaa A. *Hadbook of Formal Languages, vol.2*. Springer Science & Business Media, 1997. ISBN 3-540-60648-3.

[2] Fernau H, Holzer M, Freund R. Hybrid modes in cooperating distributed grammar systems: internal versus external hybridization. *Theoretical Computer Science*, 259, 2001.

[3] A.V. Aho. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley series in computer science. Pearson/Addison Wesley, 2007.