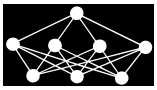

REGULAR GRAMMAR TRANSFORMATION INSPIRED BY THE GRAPH DISTANCE USING GA

*Vít Fábera, Jan Zelenka, Vlastimil Jáneš, Mária Jánešová**

Abstract: This paper introduces a method how to transform one regular grammar to the second one. The transformation is based on regular grammar distance computation. Regular grammars are equivalent to finite states machines and they are represented by oriented graphs or by transition matrices, respectively. Thus, the regular grammar distance is defined analogously to the distance between two graphs. The distance is measured as the minimal count of elementary operations over the grammar which transform the first grammar to the second one. The distance is computed by searching an optimal mapping of non-terminal symbols of both grammars. The computation itself is done by the genetic algorithm because the exhaustive evaluation of mapping leads to combinatorial explosion. Transformation steps are derived from differences in matrices. Differences are identified during the computation of the distance.

Key words: *Finite state machine, automata, graph, graph distance, regular grammar distance, genetic algorithm*

Received: April 26, 2011
Revised and accepted: July 28, 2011

1. Introduction

A concept of the multilingual model of system alliance was presented e.g. in [1]. The alliance is understood as a heterogenous entity consisting of elements that are coupled strongly or weakly. Interfaces are placed between two elements and between elements and neighboring area (supersystem). One way how to describe the communication on interfaces is to use languages which can be generated by grammars. There are several variants of this model and many tasks to define and to solve. Languages and grammars were classified by Chomsky (unrestricted, context-sensitive, context-free, regular grammars). The selection of the grammar to model the communication determines the language and the complexity of the language

*Vít Fábera, Jan Zelenka, Vlastimil Jáneš, Mária Jánešová
Czech Tech. Univ. in Prague, Faculty of Transportation Sciences, E-mail: fabera@fd.cvut.cz, zelenka@fd.cvut.cz, janes@fd.cvut.cz, janesova@fd.cvut.cz

analysis. The regular grammars generate regular languages and the regular languages are accepted (recognized) by the (nondeterministic) finite state machines, the context-free languages are generated by the context-free grammars and they are recognized by the (nondeterministic) pushdown automata. Context-sensitive and unlimited languages generated by the appropriate grammars are more complex and their analysis is an NP problem and it can be done by Turing machines.

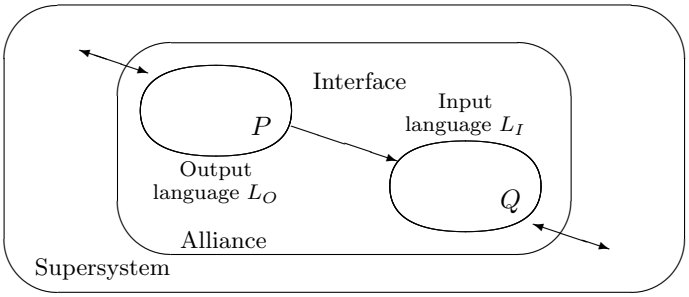

Fig. 1 *Model of the alliance.*

We suppose an alliance containing members P, Q, Fig. 1. Member Q is characterized by an input language L_I (occurring on the input interface) and member P is characterized by an output language L_O (occurring on the output interface). The P member sends messages to the Q member. The special case is when input and output languages are the same, $L_I = L_O$, or the output language is a subset of the input language, $L_O \subset L_I$. The interface is regular from the point of system analysis view. No conversion or translation is needed. If languages differ the interface is irregular and the regularization process has to be done. The following two methods are used very often in system analysis – to modify the element (its output function) or to insert a conversion element between members. These methods can be realized by specified steps in the multilingual model:

1. Transformation (modification) one of two languages L_O, L_I (grammars respectively) toward to the second language (grammar respectively).

2. Searching a translation grammar between two languages and assign this grammar to the conversion element.

The decision depends on several aspects which alternative to choose. Especially, it depends on "distance" of languages. We can measure the cost of searching translation grammar and implementing it to the conversion element and the cost of transformation of the one language to the second one. The both costs are then compared. If the transformations mentioned above are considered, the regard for two language components should be taken into account: syntax and semantics. Consequently, the correctness of translation (transformation) has two components: syntactic and semantic correctness [8]. The translation will be syntactically correct if the recipient accepts the sentence, semantically correct if the sentence has the same or similar meaning which was meant by the first member. Syntactically

correct translation should be done when the sentences will be created according to correct rules by e.g. finite state machine (FSM), for semantic correctness there should be real-time acknowledgement of the meaning, at least for the very first time of communication. Semantically correct translation is much more complicated (two semantically corresponding terminals may figure in more rules that are neither corresponding nor similar). We focused on the first task. We try to find an optimal transformation of one regular grammar to the second one. Our transformation is syntactically correct.

2. The Transformation of Regular Grammars

Let us assume two regular grammars: $G_1 = \langle N_1, T_1, P_1, S_1 \rangle$, $G_2 = \langle N_2, T_2, P_2, S_2 \rangle$, where N_1, N_2 are sets of non-terminal symbols, T_1, T_2 are sets of terminal symbols, P_1, P_2 are sets of rules, $S_1 \in N_1, S_2 \in N_2$ are start symbols of the grammars. Sets of non-terminal symbols are generally different, $N_1 \neq N_2$, the count of non-terminal symbols $q_1 =\mid N_1 \mid$, $q_2 =\mid N_2 \mid$ can be equal or not, $q_1 = q_2$ or $q_1 \neq q_2$. The forms of regular grammar rules are: $A \to aB$ or $A \to a$, where A, B are non-terminal symbols, a is a terminal symbol.

The ideal case is if $T_1 = T_2$. Let us suppose $T_1 \subset T_2$ or $T_2 \subset T_1$. Then, the smaller set can be simply completed so that $T_1 = T_2$. If $T_1 \cap T_2 \neq \{\}$ and $T_1 \not\subset T_2$ neither $T_2 \not\subset T_1$ missing symbols are added to the both sets. If $T_1 \cap T_2 = \{\}$ mapping of terminal symbols must be constructed, $M : T_1 \to T_2$. Moreover, if $\mid T_1 \mid \neq \mid T_2 \mid$ some symbols must be added to the set with less cardinality. We only focus on the case when $T_1 = T_2$ or $T_1 \subset T_2$ or $T_2 \subset T_1$. Accordingly, if $q_1 \neq q_2$, $q_1 > q_2$ without a loss of generality, the set N_2 is expanded by adding new non-terminal symbols so that the cardinality of the both sets are the same.

The task is to transform the generative regular grammar G_1 to the generative regular grammar G_2. The simple algorithm is described briefly below:

1. If $T_1 \neq T_2$ insert missing terminal symbols into the appropriate set.
2. If $q_1 \neq q_2$ add new nonterminal symbols to the smaller set to be $q_1 = q_2$.
3. Search the optimal mapping of non-terminal symbols such a number of steps of transformation is minimal. The count of steps means the distance between grammars.
4. Apply transformation steps that are derived from the optimal mapping.

2.1 The distance between regular grammars

A regular grammar is equivalent to a finite state machine (FSM) which is generally nondeterministic and is defined without output function. The distance between grammars can be defined analogously to the distance of the finite state machines. The finite state machine is represented by the transition graph. Hence, the grammar and FSM distance measurement is derived from the measuring of the distance between two graphs [2].

2.1.1 The distance between graphs

There are several approaches how to define the distance between two graphs. The authors in [3], [4] define the distance between two unlabeled un-oriented graphs as the length of the shortest sequence of elementary operations e_i that transform the graph GR_1 to the graph GR_2. The e_i operation is the *edge rotation* or the *edge deletion*. The edge deletion operation solves the situation when two graphs differ in the number of edges. Authors also take into account graphs of an arbitrary order (they differ in the number of vertices). They define that the graphs are equivalent if they differ only by isolated vertices and their distance is equal to zero in this case.

The distance calculation is an NP-complete problem. In [3], the authors considered only planar graphs. Bounds of distance are known and they are derived from the different count of edges of the both graphs. Therefore, the authors delete some edges and rotate edges and check for isomorphism using the linear time algorithm for planar graphs introduced by Hopcroft and Wong [7]. In [5], the authors define "edit distance" between labeled un-oriented graphs. They consider the following operations: edge and vertex addition/deletion and substitution. The authors take into account labeling of vertices (the mapping of vertices of GR_1 to GR_2 is partially given) and they define the cost of each operation. The edit distance measurement is applied in more fields (fingerprints, chemistry). The edit distance is calculated by A* algorithm and graph matching [5].

2.1.2 The distance between regular grammars

We have already mentioned that a regular grammar is equivalent to a finite state machine and this is represented by the oriented transition graph or the transition matrix.

We remind firstly how rules of the regular grammar are transformed to the transitions of the corresponding finite state machine. We consider a regular grammar $G_1 = \langle N, T, P, S\rangle$ and the corresponding finite state machine $FSM = \langle X, Q, \lambda, S\rangle$, where X is an input symbol set, Q is a set of internal states, λ is a *transition relation* $\lambda \subset Q \times X \times Q$ because the automaton can be nondeterministic.

The input symbol set is equal to the terminal symbol set, $X = T$. The set of internal state is equivalent to the set of non-terminal symbols plus extra terminal state TS, $Q = N \cup \{TS\}, TS \notin N$. The initial states is the state equivalent to the start symbol of the grammar S. The rule $A \to aB$ is transformed to the transition from the A state to the B state initiated by the input symbol a, so $(A, a, B) \in \lambda$. The rule $A \to a$ is transformed to the transition from the A state to the terminal state TS initiated by the input symbol a, $(A, a, TS) \in \lambda$.

We use operations over the grammar in correspondence to [3] due to the equivalence regular grammar = finite state machine = transition graph: *edge rotation* and *edge addition/deletion*. It is clear that *edge rotation* covers three operations over an existing rule in the grammar: *the change of the non-terminal symbol* on the right side of the existing rule ($A \to aB$ to $A \to aC$), *the removing of the non-terminal symbol* from the right side of the existing rule ($A \to aB$ to $A \to a$) (if the target state is changed to TS) and *the adding the non-terminal symbol* to the right side of the existing rule ($A \to a$ to $A \to aB$) (if the previous state is TS) . The

edge addition/deletion means *removing an existing/adding a new rule* to the set of rules. One step of using of the edge rotation compensates two necessary steps: edge deletion followed by edge addition. The operations are depicted in Fig. 2.

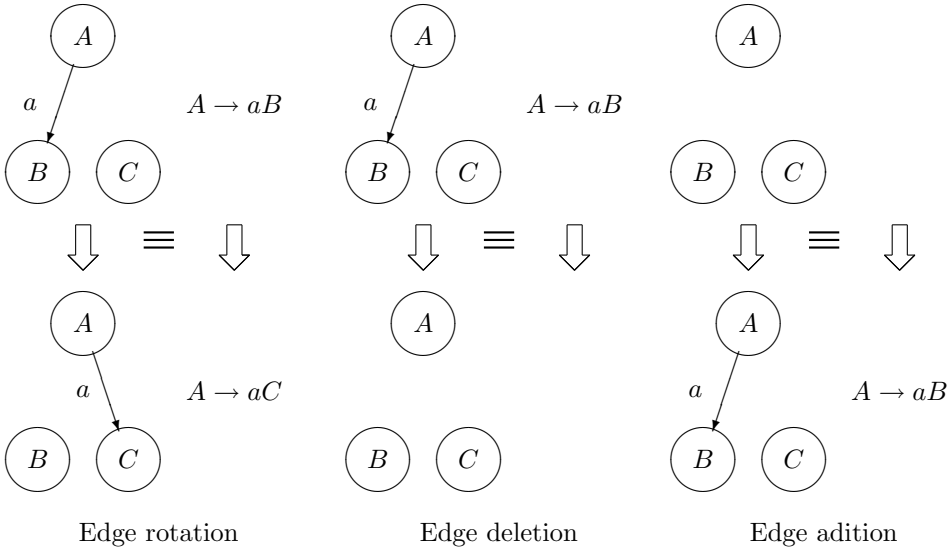

Fig. 2 *Transformation operations.*

Let us consider two regular grammars G_A, G_B with the same count of non-terminal symbols and the same sets of terminal symbols. A sequence of grammars exists:

$$G_A = G_1, G_2, \ldots, G_i, G_{i+1}, \ldots, G_n = G_B, \tag{1}$$

where the grammar G_i is transformed to the G_{i+1} grammar by one of the following operations e_i: *the change of the non-terminal symbol on the right side* of the existing rule, *the removing of the non-terminal symbol on the right side* from the existing rule, *the adding of the non-terminal symbol to the right side* of the existing rule, *removing an existing/adding a new rule* to the set of rules. The distance between grammars is the smallest positive n over all sequences.

If grammars are identical the distance is zero. Identical grammars could have different labeling of non-terminal symbols. For example, G_1 and G_2 are identical on Tab. I.

Additionally, the distance between grammars that differ only in isolated non-terminal symbols is zero. Let us suppose a grammar that contains no rule having the non-terminal symbol D on the right side of this rule and contains some rule $D \to bE$ (or $D \to b$). The rule $D \to bE$ (or $D \to b$) cannot be used because the D non-terminal symbol does not occur in derivation and it cannot be rewritten. Such rule $D \to bE$ (or $D \to b$) is useless and it can be removed from the grammar. The

G_1:			G_2:		
N_1	=	$\{S, A, B\}$	N_2	=	$\{R, C, D\}$
T_1	=	$\{a, b\}$	T_2	=	$\{a, b\}$
P_1	=	$\{S \to aA, A \to aA, A \to a,$	P_2	=	$\{R \to aC, C \to aC, C \to a,$
		$A \to bB, B \to bB, B \to b,$			$C \to bD, D \to bD, D \to b,$
		$B \to aA\}$			$D \to aC\}$
S is a start symbol			R is a start symbol		

Tab. I *Identical grammars.*

grammar with useless rules is not acceptable. We only consider grammars in the sequence (1) which are acceptable.

The distance is a measure and it must satisfy three criteria:

1. to be positive or zero
2. to be symmetrical
3. triangle inequality

$$d(G_A, G_B) \leq d(G_A, G_C) + d(G_C, G_B). \tag{2}$$

The positivity is implied from the definition. The symmetry is implied from the definition and from the equation (1) as well – the sequence of grammars in (1) is turned.

The triangle inequality can be proven with the proof by contradiction. If there is only one sequence from G_A to G_B including G_C, $G_A, \ldots, G_C, \ldots, G_B$, then the inequality (2) changes to the equality, $d(G_A, G_B) = d(G_A, G_C) + d(G_C, G_B)$. If a sequence or more sequences from G_A to G_B exist that do not contain G_C these sequences can be of the same length, longer or shorter one than the sequence containing G_C. Because the distance is defined as the smallest positive n, the shortest sequence is chosen and the distance $d(G_A, G_B)$ is always less than or equal to $d(G_A, G_C) + d(G_C, G_B)$.

2.2 The Distance computation

Regular grammars can be represented like equivalent finite state machines – by transition matrices. The matrix element is not an isolated non-terminal symbol but the characteristic vector due to the possibility of the nondeterministic machine.

The distance can be computed by searching an optimal mapping non-terminal symbols of the grammar G_1 to G_2. The mapping should assign start symbols and TS reciprocally. For each mapping, the count of transformation steps is computed from matrices of grammars. The distance between grammars is given as a minimum of counts over all mappings.

The computational complexity is factorial due to check of all permutations. It can be computed using brute force only for small grammars. One way how to search the optimal mapping is to use a genetic algorithm. The linear chromosome codes mapping of G_1 non-terminal symbols to G_2 non-terminal symbols. We use one

point *order crossover* with a randomly generated crossover point. The *mutation* swaps the assignment of two symbols.

The complete algorithm is shown in flowchart in Fig. 3.

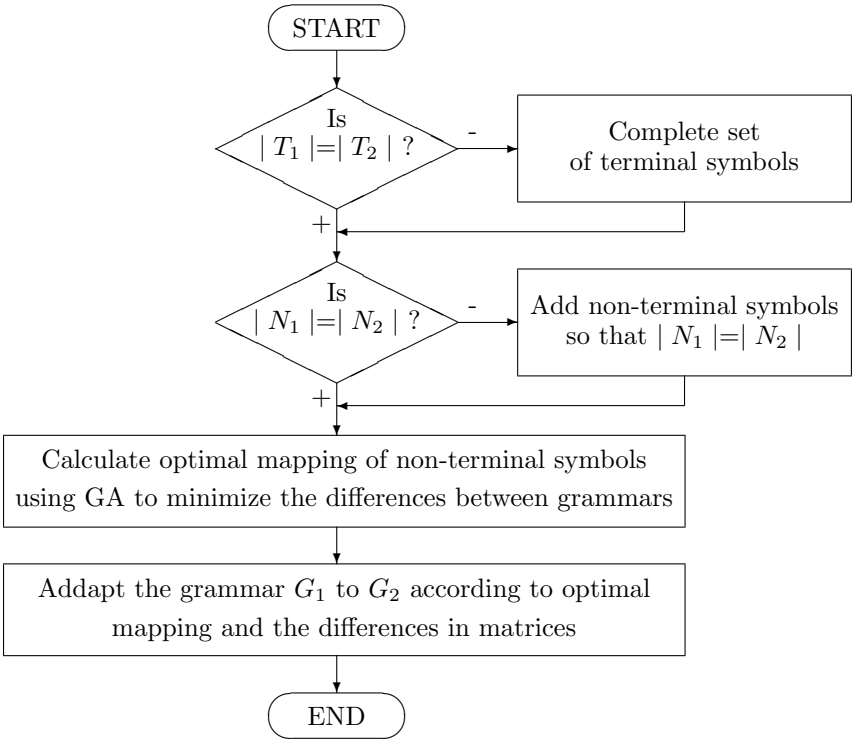

Fig. 3 *Flowchart of the algorithm.*

2.2.1 Transformation steps computation from transition matrices

The set of rules of the regular grammar is represented by a transition matrix equivalent to FSM (Fig. 4(a)). The elements of the transition matrix are characteristic vectors of the subset of non-terminal symbols. TS non-terminal symbol is added

	$\cdots$	a	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
A	$\cdots$	$\bar{v}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

(a)

N	$=$	$\{$	S	A	B	C	TS	$\}$
$\vec{v}$	$=$	$($	0	1	0	1	1	$)$

(b)

Fig. 4 *Grammar representation.*

to be able to represent rules without non-terminal on their right side, see vector $\bar{v}$ in matrix on Fig. 4(b). Let us suppose the set of non-terminal symbols is $N = \{S, A, B, C, TS\}$. If the characteristic vector $\bar{v}$ is $(0, 1, 0, 1, 1)$ it means the set of rules contains $A \to aA$, $A \to aC$, $A \to a$, Fig. 4(b).

The algorithm that finds count of transformation steps for an actual mapping is described below in pseudo C. The grammar G_1 (G_2 respectively) is represented by the matrix M_1 (M_2 respectively). The characteristic vector in the matrix M_1 is denoted with index 1 etc.

```
sum = 0;
for each element in matrix M1
{
   Find an appropriate element v2 in M2 that corresponds with v1 of
M1    according to the actual mapping;
   Transform vector v1 of M1 to v1' according to the actual mapping;
   /* Calculate the difference in number of rules */
   diff = cout_of_ones(v1') - cout_of_ones(v2);
   /* Add or delete non-terminals to/from v1' so that the count of
      non-terminals is the same as in v2 ≡ edge (rule) addition/deletion */
   if (diff < 0) Add_non_terminals_to(v1',-diff);
   if (diff > 0) Delete_non_terminals_from(v1',diff);
   /* Count number of edge rotation as Hamming distance between v1'
      and v2 divided by 2 */
   rot = Hamm_dist(v1',v2)/2;
   sum = sum + abs(diff)+rot;
}
```

It is clear that "right" non-terminals are added/deleted to minimize the count od next steps – edge rotation.

Example:

Two grammars $G_1 =< N_1, T_1, P_1, S >$, $G_2 =< N_2, T_2, P_2, R >$ are given. The set of non-terminals are $N_1 = \{S, A, B\}$, $N_2 = \{R, C, D\}$, the sets of terminals are the same $T_1 = T_2 = \{a, b\}$.
The first grammar contains six rules:
$P_1 = \{S \to aA, S \to bB, A \to aA, B \to bB, A \to a, B \to b\}$.
The second grammar has five rules:
$P_2 = \{R \to aC, R \to bD, C \to aC, D \to bC, C \to a\}$.
We suppose the following mapping of non-terminals: $S \to R, A \to C, B \to D$.

Fraction of matrices representing both grammars and characteristic vectors are in Fig. 5. We show one step of the computation for $\bar{v}_1$ and $\bar{v}_2$ elements and we calculate the contribution to the total count of differences.

1. *Find an appropriate element $\bar{v}_2$ in M_2 that corresponds with $\bar{v}_1$ of M_1.*

The vector $\bar{v}_1$ of M_1 represents right sides of rules $B \to bX$ of G_1. Because the non-terminal B is mapped to D non-terminal the appropriate vector is one of those $\bar{v}_2$ of M_2 which represents the right sides of rules $D \to bX$.

2. *Transform vector $\bar{v}_1$ of M_1 to $\bar{v}_1^{'}$ according to actual mapping.*

   The vector $\bar{v}_1$ does not change in this case due to $B \to D$ mapping and B and D non-terminals are in the same position. TS is assigned without changing. Thus, $\bar{v}_1^{'} = (0, 0, 1, 1)$ and $B \to bB$, $B \to b$ have images $D \to bD$, $D \to b$ in G_2.

3. *Calculate the difference in number of rules.*

   diff = cout_of_ones($\bar{v}_1^{'}$) - cout_of_ones($\bar{v}_2$) = 2 - 1 = 1

4. *Delete non-terminals from $\bar{v}_1^{'}$.*

   The difference is 1 so one non-terminal from $\bar{v}_1^{'}$ is deleted, for example D (B from $\bar{v}_1$ originally). The modified vector $\bar{v}_1^{'} = (0, 0, 0, 1)$.

5. *Calculate the contribution to the total count of differences.*

   rot = Hamm_dist((0,0,0,1),(0,1,0,0))/2 = 2/2 = 1

   The contribution is abs(diff)+rot = 1 + 1 = 2 (the total sum is increased by 2).

If we make a transformation of G_1 towards to G_2 we derive transformation steps from matrices and vectors simply. There are two steps in our example:

1. The non-terminal D was deleted in step 4. If we return back to the G_1 the rule $B \to bB$ is deleted from G_1 (edge deletion).

2. We have to do one edge rotation according to step 5 (rot = 1), $\bar{v}_1^{'} = (0, 0, 0, 1)$ $\to \bar{v}_2 = (0, 1, 0, 0)$. It means the non-terminal A is added to the right side of the rule $B \to b$, thus the rule $B \to b$ is changed to $B \to bA$.

M_1:

	a	b
S	$\cdots$	$\cdots$
A	$\cdots$	$\cdots$
B	$\cdots$	$\bar{v}_1$

M_2:

	a	b
R	$\cdots$	$\cdots$
C	$\cdots$	$\cdots$
D	$\cdots$	$\bar{v}_2$

		$\{$	S	A	B	TS	$\}$
$\bar{v}_1$	=	(	0	0	1	1	)
		$\{$	R	C	D	TS	$\}$
$\bar{v}_2$	=	(	0	1	0	0	)

Fig. 5 *Matrix representation of grammars and characteristic vectors.*

3. Examples

The test program is written as a console application in C++ language. Two input grammars are stored in text files and they are described by a list of non-terminals, terminals, by a set of rules and a start terminal is defined. Parameters are the probability of crossover P_c, the probability of mutation P_m, the size of generation N and the maximal count if iterations I.

We tested the algorithm on 6 simple examples with recommended values for genetic algorithms: $P_c = 0.8$, $P_m = 0.1$. The size of generation was $N = 100$ and the maximal count of iteration was $I = 100$. The grammars had up to 10 rules and

the rules were constructed by man so that the start distance was up to 5. The GA always found a correct solution.

The example of different grammars and output of the program is shown:

```
Grammar G1:                Grammar G2:
Non-terminals = {S,A,B}    Non-terminals = {R,C,D}
Terminals = {a,b}          Terminals = {a,b}
Rules =                    Rules =
{                          {
S -> aA                    R -> aC
S -> bB                    R -> aD
A -> aA                    C -> aC
A -> a                     C -> a
B -> bB                    D -> bC
B -> b                     }
}
Start = S                  Start = R
```

The output of the program is shown:

```
The distance : 2
The best founded mapping
G1: S   A   B
------------------
G2: R   C   D

Deletion of rule B -> bB
Change of rule B -> b to B -> bA

Transformed grammar G1
----------------------
Non-terminals = {S,A,B}
Terminals = {a,b}
Rules =
{
S -> aA
S -> bB
A -> aA
A -> a
B -> bA
}
Start = S
```

4. Conclusion

The distance betwen two regular grammars was defined and the algorithm was presented which calculates the distance between two regular grammars using genetic

algorithm. The algorithm is based on searching optimal mapping of non-terminal symbols and minimizing differences in matrices that represent both grammars. Transformation steps how to transform the first regular grammar to the second one are derived from differences in matrices related to the optimal mapping of non-terminal symbols. The algorithm can be used to adjust the communication on interfaces between elements if the communication is described by regular languages (or finite state machines) and the languages differ.

Further work will be focused on searching the translation regular grammar in case two regular grammars are given with the consideration of semantic aspects.

References

[1] Votruba Z., Novák M., Brandejský T., Fábera V., Bouchner P., Zelenka J., Vysoký P., Bělinová Z., Sadil J.: Theory of System Alliances in Transportation Science, Neural Network World, Institute of Computer Science, Academy of Sciences of the Czech Republic, Faculty of Transportation Sciences, Czech Technical University, 2009.

[2] Fábera V., Jáneš V., Jánešová M.: The Distance between FSMs and its Computing Using Genetic Algorithm, Proceedings of CSE 2010 International Scientific Conference on Computer Science and Engineering, Slovakia, 2010, pp. 295-301.

[3] Kubicka E., Kubicki G., Vakalis I.: Using Graph Distance in Object Recognition, In: Proc. 1990 ACM Ann. Conf. on Cooperation, ACM Press, 1990, pp. 43-48.

[4] Chartrand G., Saba F., Zou H. B.: Edge Rotation and Distance Between Graphs, Cas. Pest. Mat., 110, 1985, pp. 87-91.

[5] Riesen K., Bunke H.: Approximate graph edit distance computation by means of bipartite graph matching, Image and Vision Computing, 27, 2009.

[6] Riesen K., Bunke H.: Graph Classification by Means of Lipschitz Embedding, IEEE Transaction on Systems, Man and Cybernetics, **39**, 6, 2009, pp. 1472-1483.

[7] Hopcroft J. E., Wong J. K.: Linear Time Algorithm for Isomorphism of Planar Graphs, Proceedings of Sixth Anual ACM Symposium on Theory of Computing, Seattle, Washington, 1974.

[8] Zelenka J.: Evolutionary Operators in Multiagent Space Conception, MENDEL 2009, Brno University of Technology, 2009, pp. 62-66.

[9] Votruba Z., Novák M.: Alliance approach to the modeling of interfaces in complex heterogeneous objects, Neural Network World, 20, 5, pp. 609-619.

[10] Brandejský T.: Suggestion for Evolutionary Strategy Implementation in CUDA, MENDEL 2010, Brno University of Technology, 2010, pp. 35-40.