Computer Science Vol. 4. 2002 81'32:512.53/.54

*Zbigniew Skolicki**

SEMIGROUPS, GROUPS AND GRAMMAR INFERENCE PROBLEM

In the paper we analyse a problem of inferring a grammar from a given sample of a language. We try to present an algebraic formalism capable of describing the issue. We consider two cases: a case of inferring canonical finite-state grammars, and a case of inferring general grammars. In both cases we define a semigroup structure. Finally we look at the possibility of getting a structure of a group.
Keyword: *grammar inference, semigroup, group*

PÓŁGRUPY, GRUPY I ZAGADNIENIE WNIOSKOWANIA GRAMATYCZNEGO
Praca omawia problem wnioskowania gramatycznego na podstawie próbki języka. Problem przedstawiony jest w kontekście algebraicznym, poprzez próbę stworzenia adekwatnego formalizmu opisującego to zagadnienie. W pracy rozważone są dwa przypadki – kanonicznej gramatyki regularnej oraz ogólnej gramatyki. Dla obu podproblemów stworzony został opis używający półgrup. W końcowej części pracy rozważamy możliwość opisania wnioskowania gramatycznego przy użyciu grup.
Słowa kluczowe: *wnioskowanie gramatyczne, półgrupy, grupy*

1. Introduction

Grammars and formal automata are the basic objects of formal linguistics theory. Research, classification and trials of mathematical description leads to a later practical usage. There is a lot of applications. From a wide range of fields, where they are used, let us only say about the high energy physics, satellite photography analysis, applications in medicine (trials of automatic preliminary diagnosis based on the ECG and EEG charts), compilators generating, speech recognition and artificial intelligence [1, 2]. When we start from having a grammar, we can analyse the corresponding language, we can bulid the automata for this grammar and we can describe it in a formal way to mathematically test its properties [5]. These problems are very well known in computer science [3, 4]. However, much more intriguing concept is the one of reversing the problem. In reality we often come across

* Institute of Computer Scence, Jagiellonian University, Cracow, e-mail: skolicki@ii.uj.edu.pl

a situation when we have a language which we want to describe. We need to bulid the proper grammar which would be able to generate the language. The process of creating the grammar is the main issue here [1]. The basic requirement on the inferred grammar is that it was compatible with the sample of the language presented. Moreover, it should be relatively easy to construct and of a reasonable size. (By size we understand the number of nonterminals and productions). Optimally, we would have an automata that would get a language sample on the input and produce the proper grammar on the output. Unfortunately, construction of such an automata in the general case seems to be very difficult and therefore there is a lot of algorithms saying what to do in specific cases (we can put restrictions on the form of the inferred grammar, for example we can require it to be regular). We hope that transferring the problem into the field of mathematics would give us a different view on the issue and would enable us to get results of notable implications in formal linguistics. This would definitely improve our practical skills in automatic data transforming. In this paper we will present a trial of a general, mathematical formulation of the problem.

2. Problem definition

Let us have the language L, for which we want to bulid the grammar. At first we will introduce some definitions.

Definition 1

$S^+(L)$ **(a positive sample)** *of the language L is a set of words of which we know that they belong to L. We omit* (L) *if it is known which language we mean.*

$$S^+ = \{+y_1, ..., +y_n\}, \forall_i y_i \in V_T^*.$$

$S^-(L)$ **(a negative sample)** *of the language L is a set of words of which we know that they* ***do*** *not belong to L. We omit* (L) *if it is known which language we mean.*

$$S^- = \{-y_1, ..., -y_t\}, \forall_i y_i \in V_T^*.$$

$S(L)$ **(a sample)** *of the language L is a pair of the positive and the negative sample. We omit* (L) *if it is known which language we mean.*

$$S = (S^+, S^-).$$

Definition 2

The grammar G is ***compatible*** *with the sample S if the following inclusions hold:*

$$S^+ \subset L(G),$$

$$S^- \subset V_T^* \setminus L(G).$$

In the definition 2 by V_T^* we denote all the words over the terminal alphabet V_T, which consists of all the terminals found in S.

Let us show on a simple diagram the relationships between the sample S, the grammar G and the language L (Fig. 1).

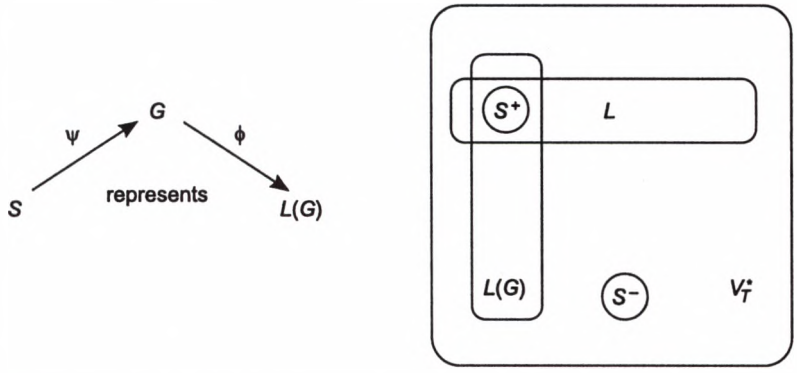

Fig. 1. Relationships between the sample S, the grammar G and the language L

Naturally we would like to generate such a grammar G that would produce a language compatible with the sample S. To describe these processes we have introduced two functions: ψ that leads from the set of all possible samples into the set of all possible grammars, and ϕ that leads from the set of grammars into the set of all possible languages. Obviously, we have $\phi(G) = L(G)$ and therefore this function is precisely defined. The problem is to find the adequate function ψ.

3. Canonical finite-state grammars and semigroups

Let us first present the simplest algorithm of getting **canonical definite finite-state grammar** $G_c = (V_N, V_T, P, S_0)$ from the positive sample

$$S^+ = \{x_1, ..., x_n\}.$$

Algorithm 1a [1]

Step 1. Scan all the words in S^+, indentify all the used signs and the set of them ascribe to V_T.

Step 2. For each word $x_i = a_i^1, ..., a_i^n \in S^+$ create $(n - 1)$ non-terminals (let us call them $Z_i^1, ..., Z_i^{n-1}$) and create a distinctive set of new productions following the pattern:

$$S \to a_i^1 Z_i^1,$$

$$Z_i^1 \to a_i^2 Z_i^2,$$

...

$$Z_i^{n-1} \to a_i^n.$$

Step 3. Assign:

V_N – the set of all nonterminals created in the step 2.
P – the set of all productions created in the step 2.
S – the start symbol.

It is not difficult to notice that a canonical grammar for each word has a separate sequence of productions, which applied one after another derive the word.The language generated by such a grammar is equal to S^+. Hence, the first condition of compatibility is fulfilled. Because S^- is empty in this case, automatically also the second condition is true and therefore the algorithm is sound.

In the above example we thought about the sample as of something that we are given before we start to construct the language. However this is not always the case and, in fact, it is more often that we learn sequentially about the consecutive words. We have to build the grammar on-line, and modify it according to the new information about each word.

Definition 3

An information sequence ω *for the language L is a series of words of which we know whether they belong or not to L. It can be infinite.*

The words in information sequence are presented one by one and the process is extended in time. Therefore we have to modify slightly the Algorithm 1a. to operate on-line.

Algorithm 1b

Step 1. Assign:

$V_N = \varnothing$,
$V_T = \varnothing$,
$P = \varnothing$,
S – the start symbol.

Step 2. For each word $x_i = a_i^1, ..., a_i^n$ from the information sequence do:

Step 2.1. Scan the word x_i and identify all the signs used. Add the new ones to V_T

$$V_T = V_T \cup \{a_i^1, ..., a_i^n\};$$

Step 2.2. Create $(n-1)$ non-terminals (let us call them $Z_i^1, ..., Z_i^{n-1}$) and create a distinctive set of new productions following the pattern:

$$P_i^1 : S \rightarrow a_i^1 Z_i^1,$$

$$P_i^2 : Z_i^1 \rightarrow a_i^2 Z_i^2,$$

...

$$P_i^n : Z_i^{n-1} \rightarrow a_i^n;$$

Step 2.3. Update the non-terminals and productions sets

$$V_N = V_N \cup \{Z_i^1, ..., Z_i^{n-1}\},$$

$$P = P \cup \{P_i^1, ..., P_i^n\}.$$

Definition 4

We denote by $S_t(L)$ *the sample of the language L that is created by taking the words from information sequence* ω *presented until time t.*

If we assume that we get information about one word in each time unit then $S_t(L)$ is built from the first t words of the information sequence for the language L. The difference between G_t and G_{t+1} corresponds to the modification made upon receiving the $(t+1)$-th word on the input.

Let us assume that we have a universal set $\bar{V}$ of all terminals and nonterminal symbols from grammars, and a set $\bar{D}$ of all possible grammars (we have the inclusion $\bar{D} \subset 2^{\bar{V}} \times 2^{\bar{V}} \times \times 2^{\bar{V}^* \times \bar{V}} \times \bar{V}$, because each grammar can be presented as a quadruple (V_N, V_T, P, S) and for each component the proper inclusion holds.)

These additional sets are needed to introduce a transition function δ, which describes the change of the grammar that must be done when we get information on the next word from the information sequence. The δ function takes a grammar and a word from the language and returns the new grammar that generates both the words accepted by the grammar on the input as well as the new word. It is worth noticing that we don't have to built the new grammar from the beginning. It is enough to modify the already existing one. The definition of such a function for the process of building a canonical finite-state grammar is given below.

Definition 5

The δ function is defined in the following way:

$$\delta : \bar{D} \times \bar{V}^* \to \bar{D},$$

$$\delta((V_N, V_T, P, S_0), a^1_{t+1} \ldots a^{|x_{t+1}|}_{t+1}) = (V'_N, V'_T, P', S_0),$$

$$\delta((V_N, V_T, P, S_0), \lambda) = (V_N, V_T, P \cup \{S_0 \to \lambda\}, S_0),$$

where:

$$V'_N = V_N \cup \bigcup_{j=1}^{|x_{t+1}|-1} \{Z^j_{t+1}\},$$

$$V'_T = V_T \cup \bigcup_{j=1}^{|x_{t+1}|-1} \{a^j_{t+1}\},$$

$$P' = P \cup \{S_0 \to a^1_{t+1} Z^1_{t+1}\} \cup \bigcup_{j=1}^{|x_{t+1}|-1} \{Z^j_{t+1} \to a^{j+1}_{t+1} Z^{j+1}_{t+1}\} \cup Z^{|x_{t+1}|-1}_{t+1} \to a^{|x_{t+1}|}_{t+1}.$$

The following statement holds

$$\delta(G_t, x_{t+1}) = G_{t+1},$$

x_{t+1} – the new word presented in time $t+1$,
G_t – the grammar generated in time t, it accepts the words $x_1, \ldots, x_t$,
G_{t+1} – the grammar generated in time $t+1$, it accepts the words $x_1, \ldots, x_{t+1}$.

It is not difficult to expand this function to the function δ_2, which takes on its input a grammar and a sequence of words (instead of one word) and returns the grammar that accepts both previous as well as new words. We can give a recursive

Definition 6

The function $\delta_2 : \bar{D} \times (\bar{V}^*)^* \to \bar{D}$, *is defined in the following way*:

$$\delta_2(G, x_1) = \delta(G, x_1),$$

$$\delta_2(G, x_1 \ldots x_n) = \delta(\delta_2(G, x_1 \ldots x_{n-1}), x_n),$$

where $x_1 \ldots x_n$ *is the input sequence of words.*

Further, we can construct a set of functions δ_ω describing the change of the grammar upon receiving the word sequence ω.

Definition 7

For each sequence of words $\omega = x_1 \ldots x_n$ *the function* $\delta_\omega : \overline{D} \rightarrow \overline{D}$ *is defined in the following way*

$$\delta_\omega(G) = \delta_2(G, \omega).$$

The following statement holds for the catenation of these functions with every grammar G as an argument

$$(\delta_{\omega_2} \circ \delta_{\omega_1})(G) = \delta_{\omega_2}(\delta_{\omega_1}(G)) = \delta_2(\delta_2(G, \omega_1), \omega_2) = \delta_2(\delta_2(G, \omega_1), x_2^1 \ldots x_2^{|\omega_2|}) =$$

$$= \delta(\delta_2(\delta_2(G, \omega_1), x_2^1 \ldots x_2^{|\omega_2|-1}), x_2^{|\omega_2|}) = \ldots = \delta(\ldots\delta(\delta_2(G, \omega_1), x_2^1)\ldots, x_2^{|\omega_2|}) =$$

$$= \delta(\ldots \delta_2(G, \omega_1 x_2^1)\ldots, x_2^{|\omega_2|}) = \ldots = \delta_2(G, \omega_1 x_2^1 \ldots x_2^{|\omega_2|}) = \delta_2(G, \omega_1\omega_2) = \delta_{\omega_1\omega_2}(G).$$

And therefore the catenation is associative

$$\delta_{\omega_3} \circ (\delta_{\omega_2} \circ \delta_{\omega_1}) = \delta_{\omega_3} \circ \delta_{\omega_1\omega_2} = \delta_{\omega_1\omega_2\omega_3} = \delta_{\omega_2\omega_3} \circ \delta_{\omega_1} = (\delta_{\omega_3} \circ \delta_{\omega_2}) \circ \delta_{\omega_1}.$$

Putting all of the above together we notice that functions δ_ω with catenation constitute a semigroup. What's more, this is a commutative semigroup, because the resulting grammar built for an information sequence ω is always the same, indepedently of the order in which the words appear in ω (we allow permutations on nonterminals). This is a result of the fact that for each word there are separate productions created.

$$(\delta_{\omega_2} \circ \delta_{\omega_1})\,(G) = (\delta_{\omega_1} \circ \delta_{\omega_2})\,(G).$$

Of course, we have to find the neutral element for this semigroup. However, contrary to what we might think, δ_λ is not the one, because if the input grammar doesn't produce the empty word, then it will be modified on applying δ_λ so that it generates it. Therefore we have to put $\delta_\varnothing$ (this corresponds to the situation when in time unit t there is no word presented). It holds that $\delta_\varnothing = id \mid_{\overline{D}}$. The seeming inconsistency results from the fact that $\delta_\varnothing$ corresponds to the empty word of the monoid $(\overline{V}^*)^*$ and λ is the empty word of the monoid $\overline{V}^*$.

Let us define the starting grammar for the inference process to be

$$G_0 = (\{S_0\}, \varnothing, \varnothing, S_0).$$

Different information sequences may result in equivalent grammars. We can use this for creating equivalence classes on information sequences. For example, we can have one of the following definition of equivalence relation:

- $\omega_1 \Re \omega_2 \Leftrightarrow \delta_{\omega_1}(G_0) = \delta_{\omega_2}(G_0)$ ($\equiv$ if grammars equal).
- $\omega_1 \Re \omega_2 \Leftrightarrow L(\delta_{\omega_1}(G_0)) = L(\delta_{\omega_2}(G_0))$ ($\equiv$ if languages equal).

If we think about how a canonical finite-state grammar is constructed we quickly notice that in both cases we have the same relation $\Re$ (allowing for permutation of nonterminals in productions of the grammars). The following condition holds

$$\omega_1 \Re \omega_2 \Leftrightarrow \forall\, \tau_1, \tau_2 \in (\overline{V}^*)^* \; \tau_1\omega_1\tau_2 \Re \tau_1\omega_2\tau_2$$

and therefore the relation $\Re$ is a congruence. Instead of using the information sequences we could use the equivalence classes of them. Because two canonical finite-state grammars produce exactly the words that were given in the information sequence they are equal if and only if the set or words presented are the same. Therefore an equivalence class of a sequence ω consists of all the sequences that possess the same words, maybe in different order. Simply saying, the order of words is unimportant and therefore we can use sets of input words (samples) instead of information sequences. The δ_ω functions can be redefined to δ_A functions, where A is a set of words.

Now we can define the function ψ in the following way

$$\psi(S^+) = \delta_{S^+}(G_0).$$

It seems that constructing the analogous functions for the so-called *derived grammars* should not be much harder. A derived grammar is created from a canonical grammar by groupng the non-terminals into equivalence classes and taking these classes as new non-terminals (and, obviously, exchanging the non-terminals in all the productions and in the start state). One could probably create these equivalence classes already in the process of the grammar constructing. When analysing a word one wouldn't necessary introduce as many new non-terminals as the length of the word. Instead we could use some of the already existing non-terminals. The equivalence class to which a new non-terminal would belong, if created, would be the right substitution.

Canonical finite-state grammars are very 'reliable' structures. They depend very closely and strictly on samples and therefore we have the implications:

$$S_1^+ \subset S_2^+ \Rightarrow \psi(S_1^+) = G_1 \subset G_2 = \psi(S_2^+),$$

$$S_1^+ \subset S_2^+ \Rightarrow L(G_1) \subset L(G_2).$$

4. General grammars and semigroups

Up to this point we have analysed the situation in which we have the empty negative sample. In other words, all the words on the input were supposed to be examples of the words constituing the language L. It is interesting to see what we can say in the situation when the grammar is affected by both a positive and a negative sample.

The algorithm of building a canonical finite-state grammar is a good example of grammar inference, but in reality it leads to huge and impractical grammars (it doesn't generalize the language it gets on the input). Therefore in practical applications we would definitely have to find a more effective tool for inferring a grammar. Because of that we will consider a general case in which we resign from the Algorithm 1 and assume having some more sophisticated one. The grammar produced by this algorithm would generate such a langu-

age, that the positive sample would be a proper subset of it. Therefore the negative sample can influence the grammar being created (if we were still using Algorithm 1 we could simply ignore the negative sample and the grammar wouldn't produce any word from it – because it wouldn't produce any word other than in the positive sample at all!). The negative sample can restrict the language being created.

Let us look if we can get the structure of a semigroup following the path analogous to the one that we have already seen. As previously we will try to construct the mappng δ that describes the changes made in the process of creating the grammar.

We cannot allow δ to have the domain $\overline{D} \times \overline{V}$, because in general case it is not enough to know the grammar itself. We have to keep somewhere the information about the words that have been declined (the words in the negative sample) and also about the words that have been assured to be from the language. We cannot say, knowing only the grammar, which words are not generated, because we were forbidden to generate them, and which ones are not generated because it happened so in the process of the grammar constructing. Therefore, introducing new productions we wouldn't be able to say if we did not start to generate some words that are disallowed. Similarly, we wouldn't be able to say which words are generated because we wanted them to be generated and which ones are generated by chance. Modifying the grammar to refuse generating some word from the negative sample, we could accidentaly stop generating some of the wanted words. To summarize, it seems necessary to remember all the words from the samples. The function δ must be dependent on them.

Hence the following general

Definition 8

The function δ *is defined in the follwing way*:

$$\delta : \overline{D} \times 2^{\overline{V}^*} \times 2^{\overline{V}^*} \times \{+, -\} \times \overline{V}^* \to \overline{D} \times 2^{\overline{V}^*} \times 2^{\overline{V}^*},$$

$$\delta\,(G, S^+, S^-, \xi, x) = (G', S'^+, S'^-),$$

where:

- G – *the input grammar,*
- S^+ – *the positive sample of the language L,*
- S^- – *the negative sample of the language L,*
- ξ – *the sign equal to '+' or '–' to denote whether the new word belongs or not to the language L,*
- x – *the new word,*
- G' – *the new grammar that generates the words from the positive sample,* S'^+ *and does not generate words from the negative sample* S'^-,
- S'^+ – *the new positive sample given by the following rule:*

$$S'^+ = \begin{cases} S^+ \cup \{x\} & \text{if } \xi = '+', \\ S^+ & \text{if } \xi = '-', \end{cases}$$

- S'^- – *the new negative sample given by the follwing rule:*

$$S'^{-} = \begin{cases} S^{-} & \text{if } \xi = '+', \\ S^{-} \cup \{x\} & \text{if } \xi = '-'. \end{cases}$$

We observe that the below-given statement holds

$$\delta(G_t, S_t^+, S_t^-, \xi_{t+1}, x_{t+1}) = (G_{t+1}, S_{t+1}^+, S_{t+1}^-).$$

One could ask what for to use G_t when creating G_{t+1} if we have all the information in (S_{t+1}^+, S_{t+1}^-) and don't need G_t. This is because we believe that modyfing the grammar is quicker and therefore more effective than building it anew. Secondly, the order in which words appear in the information sequence may influence the grammar created. We loose this information because we don't need it any more, however this information is 'hidden' in the structure of the grammar. Finally, this approach will enable us to give a neat theoretical description.

Now we will follow the same process of enhancing δ function as we did for the canonical finite-state grammars. At first we will make δ to accept sequences of words.

We give a recursive

Definition 9

The function δ_2 *is defined in the following way*:

$$\delta_2 : \bar{D} \times 2^{\bar{V}^*} \times 2^{\bar{V}^*} \times (\{+, -\} \times \bar{V}^*)^* \to \bar{D} \times 2^{\bar{V}^*} \times 2^{\bar{V}^*},$$

$$\delta_2(G, S^+, S^-, \xi_1 x_1) = \delta(G, S^+, S^-, \xi_1 x_1),$$

$$\delta_2(G, S^+, S^-, \xi_1 x_1 \ldots \xi_n x_n) = \delta(\delta_2(G, S^+, S^-, \xi_1 x_1 \ldots \xi_{n-1} x_{n-1}), \xi_n x_n).$$

Futher, we create a set of functions δ_ω

Definition 10

For each sequence of '+' / '–' *signs and words* (*interleaved*) $\omega = \xi_1 x_1 \ldots \xi_n x_n$ *the function* δ_ω *is defined in the following way*:

$$\delta_\omega : \bar{D} \times 2^{\bar{V}^*} \times 2^{\bar{V}^*} \to \bar{D} \times 2^{\bar{V}^*} \times 2^{\bar{V}^*},$$

$$\delta_\omega(G, S^+, S^-) = \delta_2(G, S^+, S^-, \omega).$$

The catenation of δ_ω functions is associative and hence again we get a structure of a semigroup with the neutral element $\delta_\varnothing$. This time, however, we have no information on whether this is a commutative semigroup because we don't know if the order in which we analyze words is important or not. This is dependent on the actual algorithm we use and nothing can be said in general.

We assume that in one time unit one new word comes, either into the positive sample or into the negative sample. We define a seqence ω_S by putting all this words into one series. If, in time t, we get a word x_t from the language L, then we append $+x_t$ to ω_S. If the word x_t does not belong to the language L, we append $-x_t$. Now, we can give the definition for the function ψ (G_0 was defined in section 3)

$$\psi(S) = \delta_{\omega_S}(G_0, \theta, \theta).$$

In the general case the congruence $\Re$ on the input information sequences cannot be defined as the equality of the languages generated by the proper grammars. It may happen by chance that two different stucturally grammars produce the same language. However, after adding productions for the next word to be generated these grammars become inconsistent. We may still have the relation

$$\omega_1 \Re \omega_2 \Leftrightarrow \delta_{\omega 1}(G_0) = \delta_{\omega 2}(G_0).$$

The size of the equivalence class depends on the inference algorithm used. However, it is very unlikely that such a class will consist of all the sequences with the same words.

Let us look at the following implications:

1) $S_1^+ \subset S_2^+ \Rightarrow G_1 \subset G_2$ (G_i are the grammars constructed),
2) $S_1^+ \subset S_2^+ \Rightarrow L(G_1) \subset L(G_2)$,
3) $S_1^- \subset S_2^- \Rightarrow G_1 \supset G_2$ (G_i are the grammars constructed),
4) $S_1^- \subset S_2^- \Rightarrow L(G_1) \supset L(G_2)$.

Although they may hold in some simple cases, in general they are not necessarily true. Let us see the following

Example 1

Let us consider the sets:

$$S_t^+ = \{ab, a^2b^2\},$$

$$S_t^- = \{aaaabc\},$$

$$G_t = (\{S\}, \{a, b, c\}, \{S \rightarrow aS, bS, a, b, \lambda\}, \{S\}).$$

The new word in the information sequence is $+abc$ what means that it should be added into the positive sample. If we only added some productions and thus enabled generating abc starting from S, we would simultanously enabled generating the word $aaaabc$ in the derivation shown below

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow^* aaaabc.$$

Therefore we have to remove some productions, so the inclusion $G_1 \subset G_2$ won't hold. Moreover, it is unlikely (although it depends on the algorithm) that the inclusion $L_1 \subset L_2$ was true. We would have to insert such new productions, that all the words that were generated before can be generated after the modifications. Despite the fact that it will surely complicate the grammar it may not agree with the idea of abstraction of the language given to the input.

Analogously, the bigger S^- set does not mean that the grammar and the language must be smaller. On the contrary, if we disallow some word this will probably lead to the enlargement of the grammar (it must be more complex). The changes to the set of production may cause the langauge to grow, too.

In the rare case when the implications 2) and 4) hold it is sensible to define a partial order on samples.

Definition 11

The relation π making the partial order on samples is defined in the following way

$$(S_1^+, S_1^-)\,\pi\,(S_2^+, S_2^-) \Rightarrow (S_1^+ \subset S_2^+) \wedge (S_1^- \supset S_2^-).$$

Directly from inclusions 2) and 4) we see that

$$S_1\,\pi\,S_2 \Rightarrow L(G_1) \subset L(G_2)$$

what can help us to reject some of possible samples, if we know that the smaller one (in the sense of π) already makes the language too big.

5. A word on grammars inference, δ (functions and groups)

One may ask if it is possible to get a structure of a group on the δ functions. The answer is yes, but only under very heavy restrictions. When we get a new word in the information sequence, we put this information into the grammar structure by modyfing its productions (and nonterminals and terminals). To create a structure of a group we need to fulfill the following requirements:

- For each operation δ we have to be able to find a reverse operation $-\delta$ that would remove the information introduced by δ and restore the previous state.
- We cannot expect that the last operation is the one to be reversed. Therefore the information put by different operations into the grammar structure must be kept separately, so that a piece of information corresponding to some operation δ can be removed without interferring the rest of information.
- Each mapping δ (except the neutral mapping) must either remove some information that has been introduced by the mapping $-\delta$ (because $(-\delta) \circ \delta = id$), or, if $-\delta$ did not happen, it must put some information (because $\delta \neq id$). This information can be later removed by $-\delta$ (because $\delta \circ (-\delta) = id$).
- Repeatitive applying of the same operation δ_x must each time modify the information kept in the grammar. Although it seems natural to ignore the second and consecutive application of the same operation (we already have information about the word x and it is stored in the grammar) and to put $\delta_x \circ \delta_x = \delta_x$ we cannot do it. There is no problem if we want to have a semigroup structure. However, if we aim at getting a structure of a group, we must obey the statement saying that $\delta \circ (-\delta) = id = (-\delta) \circ \delta$. If we allowed both of the above equations we would get a contradiction.

$$\delta = \delta \circ id = \delta \circ \delta \circ (-\delta) = \delta \circ (-\delta) = id.$$

Each operation would have to be the identity!

The algorithm of building a canonical finite-state grammar is a good candidate to be modified to get the structure of a group. It fulfills the above-given requirements. The "information" about each word generated is kept separately, namely in the form of a set of distinct productions. These productions can safely be removed in the process of the 'reverse' operation. Removing these productions will not cause the grammar to cease generating other words, because the sets of productions are disjunctive and also because, in the process

of generating a word we don't use any of the productions for any other word. The order in which the productions were introduced does not matter and thus we can safely remove ones created many steps ago.

Let us look at two possible implementations.

Example 2

As in the ordinary algorithm for a canonical grammar we will have just words in the information sequence – we won't divide them into positive and negative samples. However we will allow one word to appear many times. Every time the word comes, it will make the grammar either start or stop generating the word, depending on the actual state of the grammar. In this way the operation $-\delta_x$ is the same operation as the δ_x one. In practice we have only to change the step 2 of the algorithm. The updated algorithm is given below.

Algorithm 2a

Step 1. Assign:

$V_N = \emptyset$,
$V_T = \emptyset$,
$P = \emptyset$,
S – the start symbol.

Step 2. For each word $x_i = a_i^1, ..., a_i^n$ from the information sequence do:

Step 2.1. Check if there exist productions:

$$S \to a_i^1 Z_i^1,$$
$$Z_i^{j-1} \to a_i^j Z_i^j,$$
$$...$$
$$Z_i^{n-1} \to a_i^n.$$

with some nonterminals $Z_i^1, ..., Z_i^{n-1}$.

If yes, call these productions $P_i^1,..., P_i^n$. If not, go to step 2.3.

Step 2.2. It holds that $x_i \in L(G)$ and we want to remove it from $L(G)$. Remove the productions and nonterminals:

$$P = P \setminus \{P_i^1, ..., P_i^n\},$$
$$V_N = V_N \setminus \{Z_i^1, ..., Z_i^{n-1}\}.$$

Check all the productions in G to find out which signs from the set $\{a_i^1, ..., a_i^n\}$ are not used any more. Remove them from V_T (the last operation can be omitted).

Go to step 2 (take the next word).

Step 2.3. The word x_i is not generated by the grammar G. Add the appropriate terminals to V_T, nonterminals to V_N and productions to P so that it can be derived (copy steps 2.1 – 2.3 from the Algorithm 1b).

We see, that after applying the operation δ_x twice we return to the previous state of the grammar (with accurancy to leaving some more terminals if we don't care to remove them).

Hence, we have

$$\delta_x \circ \delta_x = id$$

and the structure $(\{\delta_x | x \in V_T^*\}, \circ)$ is a group.

However, it seems unnatural to assume that a word should be generated by the inferred grammar if it appears odd number of times on the input and should not be generated if it appears even number of times.

Example 3

It seems to be much more intuitive to assume that if a word gets repeated on the input then it is even more likely to be in the language analysed than in the case when it shows up only once. In this case the operation cannot be self-reverse. Thus, we need two kinds of operations – one for inserting information about the words to be generated and the second for removing the information. If any word appears in the positive sample, productions for deriving this word are added to the grammar. Therefore we cannot harness any from the words in the positive sample to do the reverse operation. It seems logical that such function should be performed by the words appearing in the negative sample. To see this let us remind what effect a negative sample should have on the grammar. If a grammar generates some word and this word appears in the negative sample, the grammar should stop deriving it. In the case of canonical finite-state grammars this is equivalent to removing the proper productions. And this is exactly what we need! The only 'trick' here is that we have to allow words to appear in both positive and negative samples. Moreover, if the word occurs more times in the negative sample we have to remember it, not to start deriving the word before it comes the same number of times in the positive sample (due to the property that says $-\delta \circ \delta = id$). If the word occurs more times in the positive sample, we should 'accumulate' this knowledge, either by creating separate sets of productions for each occurence, or by keepng the number of appearances in some other structure.

It seems sensible to keep two structures together with the grammar, to keep words from the positive and negative samples. With each word we must also keep the number saying how many times this word has appeared. This can be realised by keepng two sets S_G^+ and S_G^- . In the set S_G^+ we will keep pairs $(\omega, n(\omega))$, where ω is a word and $n(\omega)$ says how many times more the word ω has appeared in S^+. In the set S_G^- we will keep analogous information about the words that has occured more times in S^-.

Now the algorithm can be written as it is shown below:

Algorithm 2b

Step 1. Assign:

$$V_N = \varnothing,$$
$$V_T = \varnothing,$$
$$P = \varnothing,$$
$$S - \text{the start symbol},$$
$$S_G^+ = \varnothing,$$
$$S_G^- = \varnothing.$$

Step 2. For each word $x_i = a_i^1, ..., a_i^n$ from the information sequence do:

Step 2.1. If x_i is in the positive sample ($x_i \in S_i^+$) then:

Step 2.1.1. Check if there exist a pair (x_i, n) in $S_{\bar{G}}$.

If yes, go to step 2.1.2.
If no, go to step 2.1.3.

Step 2.1.2. If $n > 1$ then $S_{\bar{G}} = S_{\bar{G}}, \cup \{(x_i, n-1)\} \setminus \{(x_i, n)\}$
otherwise $(n = 1)$ $S_{\bar{G}} = S_{\bar{G}} \setminus \{(x_i, 1)\}$
go to step 2 (take the next word).

Step 2.1.3. Check if there exist a pair (x_i, n) in S_G^+.

If yes, go to step 2.1.4.
If no, go to step 2.1.5.

Step 2.1.4. $S_G^+ = S_G^+ \cup \{(x_i, n+1)\} \setminus \{(x_i, n)\}$
go to step 2 (take the next word).

Step 2.1.5. Add the possibility of deriving the word x_i by creating proper productions and adding terminals and nonterminals (copy the steps 2.1– 2.3 of the Algorithm 1b).

Step 2.2. If x_i is in the negative sample ($x_i \in S_i^-$) then:

Step 2.2.1. Check if there exist a pair (x_i, n) in S_G^+.

If yes, go to step 2.2.2.
If no, go to step 2.2.3.

Step 2.2.2. If $n > 1$ then $S_G^+ = S_G^+ \cup \{(x_i, n-1)\} \setminus \{(x_i, n)\}$ otherwise
$(n = 1)$ $S_G^+ = S_G^+ \setminus \{(x_i, 1)\}$
go to step 2. (take the next word).

Step 2.2.3. Check if there exist a pair $\{(x_i, n)\}$ in $S_{\bar{G}}$.

If yes, go to step 2.1.4.
If no, go to step 2.1.5.

Step 2.2.4. $S_{\bar{G}} = S_{\bar{G}} \cup \{(x_i, n+1)\} \setminus \{(x_i, n)\}$
go to step 2 (take the next word).

Step 2.2.5. Remove the possibility of deriving the word x_i by removing proper productions, terminals and nonterminals (copy the steps 2.1 – 2.2 of the Algorithm 2a)

The δ_x functions should operate on the three arguments: the grammar G, the S_G^+ set and the $S_{\bar{G}}$ set. We have:

$$\delta_x : \bar{D} \times 2^{V_T^* \times \mathcal{N}} \times 2^{V_T^* \times \mathcal{N}} \rightarrow \bar{D} \times 2^{V_T^* \times \mathcal{N}} \times 2^{V_T^* \times \mathcal{N}},$$

$$\delta_x(G, S_G^+, S_{\bar{G}}) = (G', S_G^{+\prime}, S_{\bar{G}}'),$$

where: G', $S_G^{+\prime}$ and $S_{\bar{G}}'$ are the input arguments modified by the Algorithm 2b.

Again we see that for every element δ_x we have the reverse element δ_{-x} and vice versa. Thus we have a group $(\bigcup_{x \in \mathcal{V}^*} \{\delta_x, \delta_{-x}\}, \circ)$.

Although this implentation is sligthly more complex, the interpretation is indeed much more intuitive. A word is generated only if it appeared more times in the positive sample than in the negative one. We can imagine a system that reads some data and this data is transformed into grammar inference automata. Of course read errors may happen and, as a consequence, the same word can be sent both into the positive sample (as belonging to the language) as well as into the negative one. However, these errors, if not to often, will be ignored. The grammar inference automata will look in which of the samples the word appeared more often.

Unfortunately it is even more complex to make the same reasoning with the general process of grammar inference. This is due to the fact, that the information that we put into the grammar may be 'diffused' and it may be impossible to take it out of the grammar not influencing derivation of the other words. The same productions can be used to generate different words and hence the problem. In the previous example we used the structure of S_G^+ and S_G^- because we wanted to keep information on how many times the word appeared in the sample. The word itself (in case of S^+) could be extracted directly from the grammar. Now, we will need a similiar structure because in general we cannot extract even the sample words from the grammar. When we get a word from the positive sample we create some productions that enable the grammar to derive the word. These productions may also enable to produce some other new words – this way by using the productions already existing, we diminish the number of productions and also we make a kind of generalization of the input. However, this results in the fact, that the information 'we know a word $x : x \in L$', which comes through the positive sample is transformed into weaker 'we know a set $A : \exists x \in A : x \in L$'.

As it was said in the section 4, the order of words is important.

Summarising all these arguments it seems that the only way to preserve the structure of a group would be to keep all the information sequence and make δ_x functions dependent on it. It we wanted to remove the possibility of deriving of some word from the grammar (when the word appears in the negative sample), we would have to remove this word from the information sequence (from the positive sample) and create the grammar anew.

6. Conclusions

From the theoretical point of view, the process of the algebraization of the concepts of grammar inference seems possible. It would be of a similiar formalisation level as the one accepted for the weaker grammars and automata [2, 3]. However, because it was done here in a very general way, we can suppose that in case of more specific applications – namely particular algorithms for grammar inferring – the theory may get much more complex. In practical applications we must very carefully watch how much time the proposed procedure takes. This is, in turn, a simple derivative of the number of possible solutions to analyse. Therefore a transformation of the described model into effective algorithm should be done in a context of well-known application (e.g. in pattern recognition). This is beyond the scope of this paper. Nevertheless, it seems, that the theoretical description of the issue may lead to new, interesting results, which would stimulate the research in grammar inference problem area.

Acknowledgments

I would like to thank my supervisor, Prof. Mariusz Flasiński, who has suggested the topic, supplied the bibliography and made comments on the final form of this paper. Valuable discussions with Prof. Flasiński help me to develop in science, for which I am very grateful.

References

[1] Fu K.S.: *Syntactic Pattern Recognition and Applications*. New Jersey, Prentice-Hall, Inc. 1982
[2] Aho A.V., Ullman J.D.: *The Theory of Parsing, Translation and Compling*. Englewood Cliffs, N.J., Prentice-Hall, Inc. 1972
[3] Chomsky N.: *Syntactic Structures*. The Hague, Mouton Publishers 1957
[4] Hopcroft J.E., Ullman J.D.: *Formal Languages and Their Relation to Automata*. Reading, MA, Addison-Wesley 1969
[5] Skolicki Z.: *Semigroups and automata*. ZN UJ (to be printed)