Graphs of Generative Grammars

Benedek Nagy

Faculty of Informatics, University of Debrecen, nbenedek@inf.unideb.hu

Abstract: In this paper we introduce the concept of the graph of a generative grammar. These special and-or graphs represent more information about the grammar than the dependency graphs. It generalizes the concept of finite automata (in regular case). We analyze how it is related to programmed grammars (using context-free rules). Moreover the graph of the grammar is more general; it can be defined for generative grammars to any recursive enumerable languages containing all information of the grammar.

Keywords: formal languages, graphs, grammars, derivations, automata, programmed grammars, dependency graphs

1 Introduction

The Chomsky type (generative) grammars and the generated language families are one of the most basic and most important fields of theoretical computer science. The graph theory is a well-known and widely used part of discrete mathematics. There are several connections between these fields, such as derivation trees, dependency graphs. Graphs can control the derivation process at programmed grammars. In this paper we are introducing a new relation, namely the graph of a grammar. These graphs give graphical representations of the grammars. This concept is very helpful to understand and model how the production rules of grammars work in derivation processes. We show that the graph of the grammar can be considered as a generalization of the finite automata and of the dependency graphs in the same time. Moreover, by using context edges, it is defined for all context sensitive and for all recursive enumerable languages as well.

2 Definitions

First we recall some definition about the generative grammars and formal languages ([3]). We are fixing our notations as well.

A grammar is a construct $G = (N,T,S,H)$, where N, T are the non-terminal and terminal alphabets. $S \in N$ is a special symbol, called initial letter (or start symbol). H is a finite set of pairs, where a pair uses to be written in the form $v \to w$ with $v \in (N \cup T)^* N (N \cup T)^*$ and $w \in (N \cup T)^*$. H is the set of derivation (or production) rules. The sign λ refers for the empty word. Let G be a grammar and $v,w \in (N \cup T)^*$. Then $v \Rightarrow w$ is a direct derivation if and only if there exist v_1,v_2, $v_0,w_0 \in (N \cup T)^*$ such that $v = v_1 v_0 v_2$, $w = v_1 w_0 v_2$ and $v_0 \to w_0 \in H$. The derivation $v \Rightarrow^* u$ holds if and only if either $v = u$ or a finite sequence connect them as $v = v_0, v_1, \ldots, v_m = u$ in which $v_i \Rightarrow v_{i+1}$ is a direct derivation for each $0 \le i < m$. A sequence of letters $v \in (N \cup T)^*$ is a sentential form if $S \Rightarrow^* v$. The language generated by a grammar G is the set of (terminal) words can be derived from the initial letter: $L(G)=\{w | S \Rightarrow^* w,\ w \in T^*\}$.

Two grammars G_1,G_2 are called equivalent if $L(G_1) \setminus \{\lambda\} = L(G_2) \setminus \{\lambda\}$.

Depending on the possible structures of the derivation rules we have the following classes of grammars and languages (for more details we refer for [3]):
Phrase-structure (type 0) grammars - recursive enumerable languages: arbitrary;
context-sensitive (type 1) grammars and languages: $v_1 A v_2 \to v_1 v v_2$ $(A \in N, v \neq \lambda)$;
context-free (type 2) grammars and languages: $A \to v$ $(A \in N)$;
linear grammars and languages: $A \to v$ $(A \in N,\ v \in T^*NT^* \cup T^*)$; and
regular (type 3) grammars and languages: $A \to v$ $(A \in N,\ v \in T^*N \cup T^*)$.

There are known subclasses of regular languages, such as the finite languages and the union-free languages [6]. The finite languages can be generated by rules of the form $S \to w,\ w \in T^*$. The union-free languages can be described by regular expression without union (only concatenation and Kleene-star can be used).

For our convenience we present some widely used normal forms of various types of grammars.

Fact 1. Each regular grammar has an equivalent one containing rules of forms $A \to aB$ and $A \to a$. Each linear language can be generated using only rules of type $A \to aB$, $A \to Ba$ and $A \to a$. Every context-free language can be generated by rules of the forms $A \to BC$, and $A \to a$ (Chomsky normal form). Every context-sensitive language can be generated by rules of the forms $A \to BC$, $AB \to AC$ and $A \to a$ (Penttonen's one-sided normal form, [8]). Each (phrase-structure) grammar can be simulated by rules of the following types ([5]): $A \to BC$, $AB \to AC$, $A \to a$, $A \to \lambda$, where $A,B,C \in N$ and $a \in T$.

Note that in this paper we generate recursive enumerable languages using only context-sensitive (or context-free) rules allowing deletion-rules (in which only the substituted non-terminal will be deleted).

Now we recall the concept of dependency graph of non-terminals of context-free grammars based on [4]. The nodes are the non-terminal symbols. There is an edge from the node labelled by the non-terminal symbol A to the node labelled by B if there is a rule in the grammar $A \rightarrow v_1Bv_2$ for some $v_1,v_2 \in (N \cup T)^*$. We will extend this definition in the following way.

We define the graph of a grammar for context-free, context-sensitive and phrase-structured (special formed) grammars. The nodes will be labelled by the letters of the alphabet (the non-terminal and terminal symbols) and by the empty word. The edges will represent the derivation rules. For each rule in H let directed edges be from the substituted non-terminal to all letters introduced in the rule (we use multiplicities as well); these edges are in an 'and'-relation which is represented by a directed arc gathering them in the correct order (see Figures 1 and 4). Moreover at non-context-free rules the context node(s) are connected to the bundle of the edges; these context-edges are shown by broken lines. Formally:

Definition 1. Let $G = (N,T,S,H)$ be a grammar. Let the graph $\Gamma(V,E,Ec)$ be defined in the following way: $V=T \cup N \cup \{\lambda\}$. There are two types of edges, E contains the directed bundles of 'derivation'-edges as follows. For each rule $uAv \rightarrow ua_1 \ldots a_nv$ ($u,v \in (N \cup T)^*, A \in N$, $a_i \in N \cup T$ for $i \in \{1,\ldots,n\}$) the bundle of (directed) edges is in the graph: $((A,a_1),\ldots,(A,a_n)) \in E$ in the appropriate order. In case of $n = 0$ (deletion-rule) the bundle contains the only edge (A,λ). The set Ec contains the 'context'-edges. They are between nodes and bundles of edges in E. For each rule with $uv \neq \lambda$, there are context edges connecting each element of uv (with multiplicities) to the bundle of edges representing this rule.

Note that the node λ will play only if there is a rule in which the right-hand-side is strictly shorter than the left-hand-side.

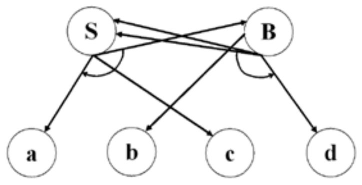

Figure 1
An example for the graph of a context-free grammar

The concept of the graph of a grammar can be used as a visualization of derivations. Put a token to the node S initially. The set of the existing tokens refers for the letters of the current sentential form at each time. When a derivation rule is applied, a token is deleted from the node labelled by the substituted non-terminal and the number of tokens is increasing at every end-node of the bundle of the applied rule by the multiplicities of the arrows (these new tokens are the children of the deleted one). A context-sensitive rule can be applied if there are (appropriate) tokens at the nodes which are connected to the bundle by context edges. Appropriate means here that those tokens must represent those letters of the sentential form which are neighbours of the replaced non-terminal as it is given by the production rule. A derivation is successfully finished when all the tokens are on the terminals (and maybe on the empty word).

Now we recall the concept of programmed grammars (without appearance checking) from [1]. Let a directed graph be given. At each node there is a context-free rule. The derivation starts at any rule in which the start symbol (*S*) is in the left-hand-side. The derivation can be continued only with a rule which is connected to the node containing the previous applied rule. So, the directed edges of the graph drive the derivation. The process is finished with a generated word if there are not any non-terminal symbols in the sentential form.

In this paper we will use Greek letters referring for the nodes of the graphs.

3 Results about Regular Grammars

In this section we show how the three different graph approaches represent the grammars, translation algorithms are given among them. Some grammars with special structured graphs are also detailed.

In the graph of a regular grammar there is at most 1 edge going to a non-terminal in each bundle of edges (and there are no context-edges in the graph).

Now let us see the special case, when the grammar is in normal from, i.e., each rule is one of the following type: $A \rightarrow a, A \rightarrow aB$. The graph of a regular grammar in normal form holds all information without signing the direction of the bundles of edges, since at each bundle having more than one edge the edge going to a terminal symbol precede the edge going to a non-terminal symbol. So in regular case we can have a very simple graph representation for every language. Moreover, using the normal form, the relation between the graph of the grammar and the automaton accepting the same language is straightforward.

So, there are three variations of graphs which represent regular languages. In the first ones the nodes represent the rules of the regular grammar and edges between them represent the possible ways of the derivation (i.e., which rule can be the next one). It is the programmed grammar graph form. There is an arrow from the node having rule $A \rightarrow aB$ to all rules having rules starting with *B*. (Other rules cannot continue the derivation.)

The second type of graphs are and-or graphs having bundles of edges. In them the non-terminals and terminals are the nodes, they are connected by 'and'-edges if a rule introduce more letters. It is our new form, called the graph of a grammar, moreover this graph includes the dependency graph in the following way. If from node γ there is an arrow to node δ, then the letter labelling the node δ is dependent by the non-terminal labelling γ. It means that the letter of δ can be derived from the non-terminal of γ.

The third graphs have only non-terminal nodes (and a node representing λ) and has only 'or'-edges. The terminals are written on the edges. This graph is the finite automaton accepting the language with initial state S and final state λ.

The graph of a grammar, the nondeterministic finite automaton of the language and the programmed grammar form can be converted to each-other in constructive ways. Now we detail these processes.

We start from the finite automata. Let the nodes of the graph of the grammar are the states of automata and the terminal labels of the transitions. If there is a transition from state A to a final state by terminal a, then the graph of the grammar has the edge (A, a) (it is a bundle with 1 edge). For all transitions: A to B by a the graph has the bundle $((A,a),(A,B))$. It is obvious that the graph representing the same grammar as the automaton. (We refer the non-terminals by A,B and the terminal symbols by a.)

Now we translate the graph of the grammar to the programmed grammar form. The rules at the nodes of the programmed grammar are representing the bundles of the graph of the grammar. So, for each bundle having only one edge in the form (A,a) a node is created with rule $A \to a$. For bundles having 2 edges as $((A,a),(A,B))$ the rule is created $A \to aB$. Now, let an edge be from each node having A in the right-hand-side of its rule to each node having A at the left-hand-side (so to each node representing the bundles of the edges from A in the graph of the grammar).

Finally, we translate the programmed grammar form to finite automaton. Let the non-terminals be the states and S be the initial state of the automata. Moreover we put an additional state to the automaton as final state. If there is a rule in the programmed grammar $A \to a$, then let a transition be from the state of A to the final state by a. For all rules of the form $A \to aB$ let a transition be from the state of A to the state of B by the terminal symbol a. (In this way the number of transitions from each node will be the same as the number of connections from the nodes of the programmed grammar graph which has A in the left-hand-side of the rules. Moreover the transitions are going to those non-terminal labelled states (or the final state) which occur on the right-hand-side of the rules by the appearing terminal symbols.)

Note that using only regular rules a programmed grammar cannot go beyond regular languages even if the edges are varied.

Fig. 2 shows examples for all the three graphs of the language $a^*b(bc)^*$ generated by grammar $(\{S,B,C\},\{a,b,c\},\mathrm{S},\{S \to aS, S \to b, S \to bB, B \to bC, C \to c, C \to cB\})$.

Now we define the concept of (directed) cycle for these graph-representations. In case of finite automata and programmed grammar forms the usual graph-theoretical definition works. In graph of a grammar there is a cycle if a descendant of a token will appear at the same node. In Figure 2 a direct cycle is shown at S and there is a 2-step cycle including nodes B and C.

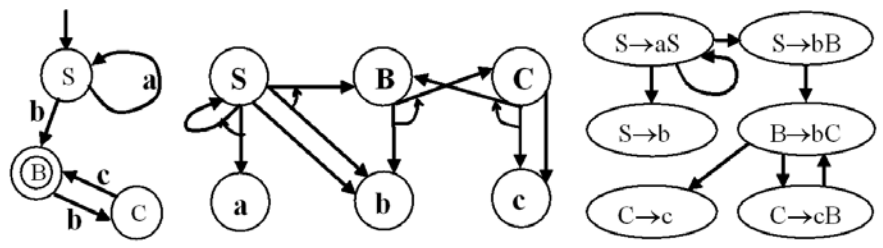

Figure 2
An example for various concepts of graphs of a regular grammar
(a. finite automaton, b. graph of grammar, c. programmed grammar)

Proposition 1. The minimal number of cycles is the same in finite automaton, in graph of grammar and in programmed grammar forms of the same grammar.

The grammars that can be represented by cycle-free graphs generate finite languages. All finite languages can be generated by such grammars. Moreover the generated language is finite even if any kinds of rules are allowed in a cycle-free graph of grammar.

With only regular rules the language is union-free if and only if a graph without alternating paths represents a grammar of the language. It means that there are not two cycle-free paths from a node γ to a node δ in automata and in programmed grammar forms. At automata and at the graph of the grammar form it means that there is at most one cycle free path from a non-terminal labelled node to any other non-terminal labelled node.

4 Results about Grammars with Context-Free Rules

The graph of the grammar is an extension of the well-known dependency graphs of context-free grammars. The extension goes in two steps. First the dependency relation can be extended to terminals (and if deletion rules are used, to the empty word) as well. In second step, the edges coming from the same rule are gathered to a bundle, representing that they are dependent (they generate at the same time in a derivation). Sometimes several copies of edges of the original dependency graphs are needed to pack them in bundles.

Note that in context-free case the graph of the grammar holds all information of the grammar. The order of the edges in a bundle is important and it can be seen on the figures as well (see Fig. 1).

4.1 Linear Languages

Now, let us analyse a subclass of context-free languages, namely the linear ones. This case is very similar to the previously described regular case. In linear grammars, such as in regular ones there is only one non-terminal in the sentential form in each step while the derivation procedure is going. By this reason the derivation can be continued only by a rule having the appearing non-terminal in its left hand side. Therefore the derivation process has a sequential order in the graph of the grammar by following it in non-terminal nodes.

Using only rules of the presented normal form all the three graphical representation of the grammars can easily be given. The programmed grammar and the graph of the grammar forms hold all information. The finite automata can be extended to carry all information [7], these automata have two heads, and the stepping head depends on the form of the used rule ($A \rightarrow aB$ and $A \rightarrow Ba$). Programmed grammars having only linear rules cannot generate more than linear languages.

4.2 Graphs of the Grammars and Programmed Grammars

Generally at context-free grammars the derivation cannot be followed by a sequence of non-terminal labelled nodes in the graph as it works in linear grammars. However, at the so-called programmed grammars (without appearance checking) a given graph controls the derivation. So the programmed grammars can be interpreted as an extension of finite automata in a certain sense. These graphs are also related to our graphs as we detail below. A programmed grammar can be seen on Fig. 3 left. To get the new concept of graphs of these (programmed) grammars we need to have some nodes with the same labels. Following the paths of the possible derivations one can obtain the graph shown on the right.

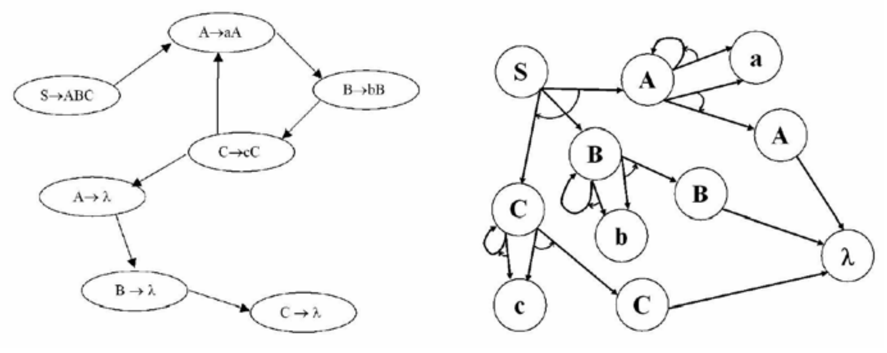

Figure 3
A programmed grammar generating a non context-free language and its graph-of-grammar presentation

Since the order of applications of rules are not arbitrary in a programmed grammar (one can have a complete directed graph – with n nodes and n^2 edges – to represent a context-free grammar), the graph of the grammar contains a non-terminal as many times as many role it has in the generation.

We need to gather all possibilities of the next applied rules of the given non-terminal (left-hand-side). If this set is not the same for the occurrences of a non-terminal at two nodes in the programmed grammar, then we need two nodes for the non-terminal representing these two rules in our new graph.

This new graph does not have all information about the programmed grammar. At programmed grammars the rules and the roles of the non-terminals are synchronized.

4.3 Regular Languages and Programmed Grammars

Now, assume the opposite way: given the graph of the programmed grammar, what can be the generated language.

We are dealing programmed grammars with some special structured graphs.

The simplest ones are the trees and the directed acyclic graphs.

First let us consider that directed acyclic graphs, i.e., cycle-free graphs direct the derivations.

Theorem 1. The class of languages defined by programmed grammars using directed acyclic graphs are exactly the finite languages.

Proof. First, we show that a programmed grammar with a cycle-free graph can generate only a finite language. In a cycle-free graph there are only finitely many paths (starting and finishing at arbitrary nodes), therefore only finitely many generation sequence exist. These finite sequences can generate finitely many words.
Now we prove the opposite direction. Let $L=\{w_1,\dots,w_n\}$ be a finite language. We construct a special programmed grammar with cycle-free graph to generate L. Let the structure be a tree with n nodes. The vertices have the rules $S\to w_i$ (for $1\le i\le n$). Clearly, it is cycle-free and generates L. □

The trees and the line-graphs (when each node has degree at most 2, but the two end-nodes having only degree 1) are special cases of the directed acyclic graphs.

Now let us see graphs with cycles (the used term 'ring' is coming form network theory). First we are analysing graphs having exactly 1 ring and some additional structure (these are the so-called ring-backbone graphs).

Lemma 1. Given a programmed grammar with a graph containing exactly 1 (directed) cycle. If the number of the non-terminals (as a multiset) are changing deriving through the cycle, then the generated language is finite.

Proof. There are finitely many paths into and from the cycle. Each pair of these paths coincides at most 1 multiset of non-terminals. When the multiset of the non-terminals of the sentential form is "growing" (i.e., at least one element is growing) in a derivation cycle, then after some cycles there will not be any terminal derivations. Therefore words can be generated with paths containing only a limited number of the cycle. It implies that only finitely many words can be generated. □

Consider the language $a^n b^m c^k$, where n,m,k are independent variables. This regular language (defined by the regular expression $a^*b^*c^*$) cannot be obtained by a programmed grammar having only 1 cycle in the graph.

In Figure 3 a programmed grammar shown which generates the language $a^n b^n c^n$, $n>0$. It is known that this language is not regular and not even context-free.

What does it mean that the programmed grammar have (or have not) cycle in terms of their graphs of grammars? We have the following result.

Theorem 2. The graph of a grammar form of the programmed grammar must have cycle if and only if the language cannot be generated by cycle-free programmed grammar.

Proof. We prove the theorem by directions. Assume that the language cannot be generated without cycle by programmed grammar (i.e., it is not finite). The programmed grammar has cycle and therefore there is a non-terminal symbol A such that its number does not change in a derivation-cycle, however there is a rule in the cycle with A in the left-hand-side. This fact means that one can go further in the derivation getting A in the same role, therefore it must be a cycle in the graph-of-grammar. (This cycle can be direct as Figure 3 shows at an occurrence of A, B and C, or it can be indirect as generated by for instance rules $A \rightarrow aB, B \rightarrow bA$.)
Other way, if the graph of the grammar has a cycle, then a rule can be applied in arbitrary many times. It means that must be a cycle in the programmed grammar.□

About the number of cycles in the programmed grammar we have the following more general theorem.

Theorem 3. The languages obtained by programmed grammar (without appearance checking) with a graph having (at most) n cycles (for any fixed natural number n) is incomparable with the regular languages.

Proof. The theorem is a consequence of the following facts. Only regular languages defined by regular expressions using at most n Kleene-stars can be described by these graphs. Some non-regular (and non-context-free) languages can be generated as well. □

Note that in [2] there are several other kinds of graphs are analysed in this point of view.

5 Graphs of Grammars with Non-Context-Free Rules

One can use the concept of the graph of a grammar for context-sensitive and phrase-structured grammars as well. The graph representation of the grammars exists for context sensitive case. For phrase structured grammars one should use rules in context-sensitive form and allow deletion rules as well. Since in pure form of the grammars their graphs are to complicated, we recommend to use the special normal forms.

In Figure 4 an example is shown. The graph represents the grammar ({*A,B,S*}, {*a,b,c*}, *S*, {*S→aAB*, *S→BB*, *AB→A*, *A→SB*, *A→abc*, *A→*λ, *B→BAB*, *B→c*}). A derivation in this grammar is shown: it starts with a token on *S*. *S→aAB*: children tokens will be on *a*, *A* and on *B*. Now the rule *AB→A* is used: the token deleted from *B* because there is a token on *A* (context-edge, and this token represents the left neighbour of *B* in the sentential form) and a token appears on λ. Now *A→abc* is used: the token of *A* is deleted and new tokens will appear on *a*, *b* and on *c*. There is no token on non-terminals, the derivation is finished, the derived word (can be formed by the tokens of the end-state): *aabc*.

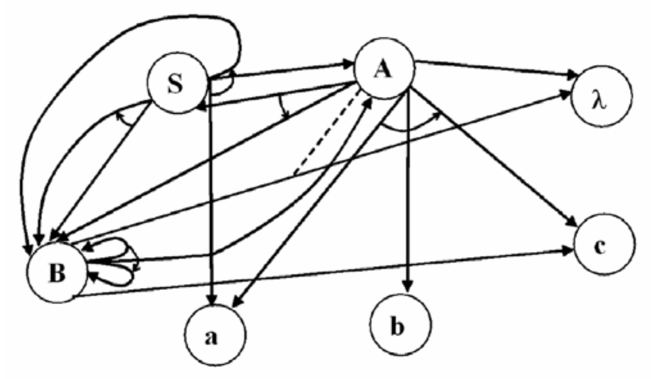

Figure 4
An example for the graph of a phrase-structured grammar

In these graphs the derivation cannot go independently at the nodes present in the sentential form, the context-condition plays an important role therefore the context-edges are needed. They represent the fact, that some rules can be applied only if the contexts are present. The graph represents only that there is/are context condition(s), but it does not refer to their order. Using the presented special normal forms the context must be 1-letter left-context, therefore the presentation is simple, the graph of the grammar has all information about the grammar.

Conclusions

We introduced the graphs of the generative grammars which generalize the concepts of finite automata (for regular case) and dependency graphs (for context-free case). The nodes are the terminal and non-terminal symbols. The bundles of the edges of the graph demonstrate the rules of grammar. For regular grammars in normal form the directions of the bundles of arrows are uniquely determined, therefore our notation is slightly redundant in this case. In linear (and regular) grammars there is at most one arrow for non-terminals in each bundle, while in context-free case there can be several. The new concept is more general, one can use it in the case of context-sensitive and eliminating rules as well. In case of context-sensitive grammars the so-called context-edges are needed, they connect nodes and bundles of edges. At non-context-sensitive grammars the node labelled by λ must be used. Using normal forms the graph has all information about the represented (phrase structured) grammar, and its structure is simple. By the help of tokens in our graph representation one can have a better understanding of the possible derivation processes.

References

[1] Jürgen Dassow, Gheorghe Paun: Regulated rewriting in formal language theory. EATCS Monographs on Theoretical Computer Science, 18. Springer-Verlag, Berlin, 1989

[2] Cristina Bibire, Madalina Barbaiani, Jürgen Dassow, Szilárd Fazekas, Aiden Delaney, Mihai Ionescu, Guangwu Liu, Atif Lodhi, Benedek Nagy: Languages generated by programmed grammars with graphs from various classes. J. Appl. Math. Comput., 22 (2006) 21-38

[3] John E. Hopcroft, Jeffrey D. Ullmann: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979

[4] Peter Linz: An Introduction to Formal Languages and Automata. Jones and Bartlett Publisher, 2001

[5] Alexandru Mateescu: On context-sensitive grammars, in Martin-Vide, C., Mitrana, V. and Paun, Gh. (eds.): Formal languages and applications. (Studies in Fuziness and Soft Computing 148), Springer-Verlag, Berlin, Heidelberg, 2004., pp. 139-161

[6] Benedek Nagy: Union-free languages and 1-cycle-free-path-automata, Publ. Math. 68 (2006), 183-197

[7] Benedek Nagy: On 5'→3' sensing Watson-Crick finite automata, In proceedings of DNA 13, LNCS 4848 (2008), 256-262

[8] Marti Penttonen: One-sided and two-sided context in formal grammars. Information and Control 25 (1974), 371-392