Triangle: Language, Literature, Computation, n. 8, 2012
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

Derivation "Trees" and Parallelism in Chomsky-Type Grammars

Benedek Nagy

Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Tarragona, Spain

Faculty of Informatics
University of Debrecen
Debrecen, Hungary
E-mail: nbenedek@inf.unideb.hu

Summary. In this paper we discuss parallel derivations for context-free, context-sensitive and phrase-structure grammars. For regular and linear grammars only sequential derivation can be applied, but a kind of "parallelism" is present in linear grammars. We show that finite languages can be generated by a recursion-free rule-set. It is well-known that in context-free grammars the derivation can be in maximal (independent) parallel way. We show that in cases of context-sensitive and recursively enumerable languages the parallel "branches" of the derivation have some synchronization points. In the case of context-sensitive grammars this synchronization can only be local, but in a derivation of an arbitrary grammar we cannot make this restriction. We present a framework to show how the concept of parallelism can be fit to the derivations in formal language theory using tokens.

1 Introduction

Chomsky type grammars and generated language families are one of the most basic and important fields of theoretical computer science. The field is fairly old, the basic concept and results are from the middle of the last century. On other hand, at the end of the twentieth century parallel computing played an increasingly greater role. In this paper we analyze the derivations of Chomsky type grammars and their relation to the concept of parallelism. We will use some variations of the well-known derivation trees and tokens on graphs in their Petri-net forms.

Note that parallelism is not new in formal language theory. The so-called Indian and Russian grammars were investigated. These grammars can be related to the context-free case. The Lindenmayer systems are also parallel rewriting systems, but the language family they produce is orthogonal to the Chomsky-type language family (for more about this topic see [14]). In this paper we would like to analyse how the concept of parallelism can be present in Chomsky-type grammars.

In the next section we recall some basic definitions that we will need later on. In Section 3 we show that in finite languages a derivation cannot be recursive independently of the forms of the rules used. In Sections 4 and 5 we show that the derivations must be sequential in linear and regular grammars, but a kind of "parallelism" is needed in derivations for linear languages. After this we show that in the case of context-free grammars the derivation can be maximal parallel without any restriction or communication. In Section 7, which is based on Penttonen's old result ([11]), we build the derivation graphs for context-sensitive grammars in a tree-like form. With this approach we show that in these cases the derivation can be parallel to some synchronization points of the neighboring branches. Finally in type 0 grammars, using an appropriate normal form we show that synchronization can happen between branches at a distance, when all mediate branches terminate by the empty word.

Using our approach we find that the generating power increases when the possibility of parallelism is present. Moreover with local communication (synchronization) it is more powerful. Finally when the synchronization is not merely local we get the whole recursively enumerable language class. We also present some interesting further branches of research based on these results.

2 Preliminaries

In this section we recall some basic concepts and facts about the field of formal languages and Petri-nets. First the definitions of the Chomsky-type grammars and the Chomsky hierarchy are shown.

2.1 Chomsky-type grammars

A grammar is a construct $G = \langle N, T, S, H \rangle$, where N, T are the non-terminal and terminal alphabets, with $N \cap T = \emptyset$; they are finite sets. $S \in N$ is a special symbol, called initial letter. H is a finite set of pairs, where a pair is usually written in the form $v \rightarrow w$ with $v \in (N \cup T)^* N (N \cup T)^*$ and $w \in (N \cup T)^*$. (We used the well-known Kleene-star notation.) H is the set of derivation rules.

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 $v_1, v_2, v', w' \in (N \cup T)^*$ such that $v = v_1 v' v_2$, $w = v_1 w' v_2$ and $v' \rightarrow w' \in H$. A derivation $v \Rightarrow^* u$ holds if and only if either $v = u$ or there is a finite sequence of sequential forms connecting them as $v = v_0, v_1, ... v_m = u$ in which $v_i \Rightarrow v_{i+1}$ is a direct derivation for each $0 \leq i < m$. A sequence of letters $v \in (N \cup T)^*$ derived from S is called a sentential form, while we refer to $u \in T^*$ as a (terminal) word. We sign the empty word by λ.

The language generated by a grammar G is the set of terminal words that can be derived from the initial letter: $L(G) = \{w | S \Rightarrow^* w \wedge w \in T^*\}$.

Two grammars are (weakly) equivalent if they generate the same language (modulo the empty word).

Depending on the possible structures of the derivation rules the following classes of grammars/languages are considered:

- Type 0, or phrase-structure grammars: there is no further restriction on the possible derivation rules.
- Type 1, or context-sensitive grammars: all derivation rules are in the form $v_1 A v_2 \rightarrow v_1 w v_2$, with $v_1, v_2 \in (N \cup T)^*$, $A \in N$ and $w \in (N \cup T)^* \setminus \{\lambda\}$ (except possibly for the rule $S \rightarrow \lambda$, in which case S does not occur on any right hand side of a rule).
- Type 2, or context-free grammars: for every rule the next scheme holds: $A \rightarrow v$ with $A \in N$ and $v \in (N \cup T)^*$.
- Linear grammars: each rule is one of the next forms: $A \rightarrow v$, $A \rightarrow vBw$; where $A, B \in N$ and $v, w \in T^*$.
- Type 3, or regular grammars: each derivation rule is one of the following forms: $A \rightarrow w$, $A \rightarrow wB$; where $A, B \in N$ and $w \in T^*$.
- Finite languages: in this case, the restriction is not actually for the rules, but the number of the words in the language: it must be finite.

The generated language is regular/ linear/ context-free/ context-sensitive/ recursively enumerable if it is generated by a regular/ linear/ context-free /context-sensitive/ phrase-structure grammar, respectively. For these families of languages we use the notations $L_{reg}, L_{lin}, L_{CF}, L_{CS}, L_{RE}$, respectively, and L_{fin} denotes the finite languages.

The Chomsky-type grammars and language families are well known. The generating powers of these grammars are in the following hierarchy.

$L_{fin} \subsetneq L_{reg} \subsetneq L_{lin} \subsetneq L_{CF} \subsetneq L_{CS} \subsetneq L_{RE}$.

2.2 Petri nets

In this subsection we recall some concepts about Petri-nets based on [12, 2]. Formally the structure of a Petri-net is a directed bipartite graph: (V, E) with two types of nodes: $V = S \cup T$ $(S \cap T = \emptyset)$ and edges $E \subseteq (S \times T) \cup (T \times S)$. The first node types (represented by ellipses in the figures) are places, and the other types are the transitions (rectangles). The connections are represented by arrows. Each arrow has different types of end nodes. In Figure 1 an example is shown. We indicated the places of the net by numbers. At each place there can be a token, so the state of the system is a binary vector with the dimension of the number of places of the net. A transition can switch if every place has a token from which an arrow goes to the transition (i.e. the transition is active, iff all its input places have tokens). After a transition has switched the tokens will move through on the transition. This means that a token will be at each output place and no token will appear at the input places of the switched transition.

In the figure starting with a token at 1 (i.e. system-state (10000000)), only 1 transition is active and the result will be the vector (01100000) after it switches. After the next step we have (00111100). Now three of the transitions are active. Suppose that the one needed tokens at 3 and 6 is going resulted (00011110). Here a token at 6 is used, but it is at the same place after the transition. We will call this type of connection a context-connection. Note that we allow transitions without outgoing edges which allows us to delete a token from the system. This happens with the token at 4 after the next transition switches. Finally, after the last possible transition the system is in state (00001011). There are no more active transitions, and the system halts. Note that the last two transitions are independent, so they can go at the same time in parallel.

We will simulate the possible derivation processes at various types of grammars with the help of Petri-nets. The derivations have a widely used graphical representation (basically for context-free grammars), which we will use in the sections below. In this paper we will use Petri-nets that start with only 1 token and finish the process with exactly the same number of tokens as the number of letters in the derived word. In our nets the states will be labelled with letters. The tokens at non-terminal labelled places are "living" tokens and at the terminal labelled places the tokens are not "living". The transitions will be the derivation rules used.

To read the derived word an order of the leaf-places will be used. A derivation is successful if and only if the leaves have tokens and therefore the systems halts. As we will see, a system may halt without a successful derivation.

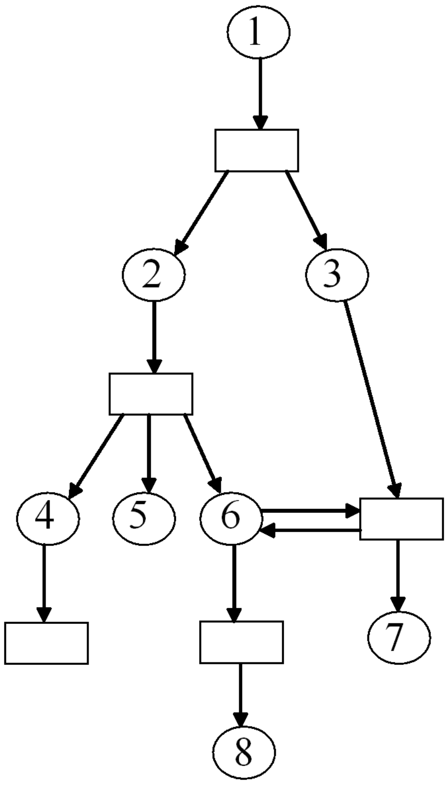

Fig. 1. An example for a Petri-net

3 Parallelism and Finite Languages

From this section we use the Chomsky hierarchy in reverse order. We start from the smallest family (the finite one), and we go in the direction of the more effective grammars.

For the Chomsky-type grammars there are so-called normal forms, in which the form of the derivation rules used are more restricted than in the original definition. It is also well-known that using only such restricted-form rules the generating power remains the same.

From now on we do not care whether the empty word is in language L or not. It is obvious that using most normal forms the resulting language is $L \setminus \{\lambda\}$. (If one wants to generate the original language including λ, then she/he can add the new rules $S_0 \rightarrow \lambda, S_0 \rightarrow S$ (with a new non-terminal S_0, where S was the initial letter in the old grammar, and S_0 is the initial letter in the new grammar) to the set of rules in any cases.)

Let us review the case of finite languages. It is a very important subfamily of regular languages (recently, for instance the so-called cover automata are used to describe them).

First, there is a normal form for these languages. Let L be a finite language. One can use the normal form containing rules of form $S \rightarrow w$, where $w \in T^*$ ($S \rightarrow w_i$ for each $w_i \in L$).

In many cases the normal form above is not an efficient way of generating languages. For this reason, we usually have other types of rules. To increase the efficiency (description of the grammar, generation speed, i.e. to get system with small number of derivations; and small cardinality of the rule set H) of the derivations we can allow stronger generating (for instance CF, or CS) rules as well (as they can be used in coding, for instance).

Finite languages form a special class of regular languages that have no cycle in the automaton. When a non-terminal has been replaced in a derivation, it can never again be in the sentential form. (We call this the anti-pumping property of finite languages.)

Apart from using a special regular grammar to generate a finite language we can use grammar in which there is no restriction on the form of the rules, so parallel derivation can used. For this effective generating method, we need a restriction on the non-terminals used: namely, they must have at least a partially ordering relation. We can formulate this in the following way.

Theorem 1. *Let G be a grammar with an ordering relation among the non-terminals such that*

- S is the smallest,

- if there is rule $u \rightarrow v$ in which the non-terminal A is rewritten, then each non-terminal appearing in v in the place of A is strictly greater than A.

Then G generates a finite language.

Proof. The second condition clearly implies that for context-free rules after a step used for the non-terminals A it disappears and only greater non-terminals appears. The number of non-terminals is finite which implies that the derivation finish in a limited number of steps. The same holds for context-sensitive and 0-type rules as well. Starting from the initial letter the sentential form only has non-terminals with higher values. (In CS-rules the context remains, but in the substituted part the value of non-terminals increases. In arbitrary rules each of non-terminals on the right-hand-side has greater values than the highest on the left-hand side. Therefore there is no way of pumping a word.) □

How the “efficiency” of a grammar generating, for example, a finite language (using for instance, context-free rules) can be measured is an interesting question.

4 Derivations in Regular Grammars

First, normal forms for generating regular languages are recalled. There are several alternative forms of these types of grammars.

A grammar is called right-(left-)linear if all derivation rules are in the forms $A \to uB, A \to u$ ($A \to Bu, A \to u$) with $A, B \in N$ and $u \in T^*$. Each regular language can be generated by either a left-linear or a right-linear grammar. The more restricted regular forms of the grammar can also be used. Each regular grammar has equivalent grammars that use only rules in the form $A \to aB, A \to a$ (or $A \to Ba, A \to a$) with $A, B \in N$ and $a \in T$.

Now we show examples for regular derivations both in the usual “derivation-tree” and the Petri-net forms.

Example 1. Let $G_1 = \langle \{S, A, B, C\}, \{a, b\}, S, \{S \to aS, S \to baA, A \to aA, A \to baaB, B \to aB, B \to b, B \to baaaC, C \to aC, C \to a\} \rangle$ be a regular grammar. Figure 2 shows a derivation in this system.

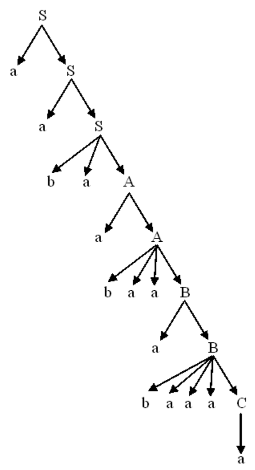

Fig. 2. Derivation in a regular grammar

Example 2. Let $G_2 = \langle \{S, A\}, \{a, b\}, S, \{S \to aA, A \to bS, A \to b,\} \rangle$ be a regular grammar. Figure 3 shows a derivation in this system by a Petri-net that initially has 1 token in the top position.

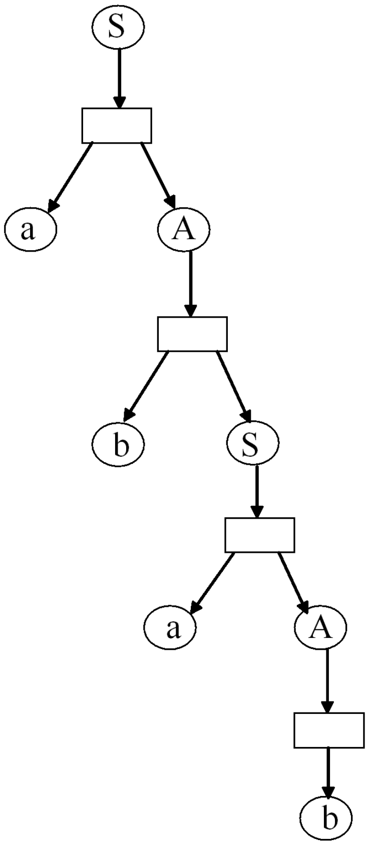

Fig. 3. Petri-net representing a derivation of a regular grammar

In Figure 2 a so-called derivation-tree is shown. In these graphs all leaves are labelled by a terminal symbol (or sometimes by the empty word λ). All other nodes (they are labelled by non-terminals) must have some (at least one) successors.

Using the Petri-net representation of a derivation we can assume that the net starts with only one token at the start-node. When the tree branches a living token that goes through on a transition, it must be multiplied in the following way: it disappears from its original place and there will be a token at each successive place. The derivation is (successfully) finished when all leaves (terminal labelled places) have a token, and there are no more tokens in the graph. In this case there is no living token in the net any more. In regular cases the process can only go in one order.

Using only rules of types $A \to aB, A \to a$ the derivation goes letterwise. Starting from the first letter of the word to the last one the derivation gives one letter in each step. This is a totally sequential derivation. In all regular languages every word can be built letter to letter from the beginning.

We discuss some of our results and comments about regular grammars in the section below in comparison ti the next class, the linear one.

5 Derivations in Linear Grammars

As we can see in regular and linear grammars each rule has a non-terminal on the left hand side and at most one non-terminal on the right hand side. (Linear grammars are special type 2 grammars with at most 1 non-terminal on the right hand side of each rule.)

Therefore we have the following statement.

Proposition 1. *The derivations in a linear (or regular) grammar can only be in sequential mode.*

Proof. Starting from the initial letter the sentential form contains at most 1 non-terminal in each step. When the sentential form has a non-terminal we must replace it by a rule in the next step. Without a non-terminal the derivation cannot continue. It is terminated with the word containing only terminals. □

The only difference between the regular and the linear case is the following. In a derivation of a regular grammar the same side is always used for further derivation; all the terminals appear on the other side of the non-terminal. These derivations are totally sequential ones. In linear grammars the terminals can appear on both sides of the derivations graph. It is a kind of parallelism which appears at these derivations. Both sides can/must be built parallel to the words. In the general case a word cannot be obtained from the beginning to the end.

Now, a normal form for linear grammars is presented.

Lemma 1. *Every linear grammar has an equivalent grammar in which all rules are in forms $A \to aB, A \to Ba, A \to a$ $(A, B \in N, a \in T)$.*

Proof. Introducing new non-terminals each linear rule can be replaced by a sequence of rules in the desired forms. □

Now we show examples for derivations in linear grammars.

Example 3. Let $G_3 = \langle\{S, A, B\}, \{a, b\}, S, H_3\rangle$ be a linear grammar, with rule set $H_3 = \{S \to aA, A \to Sa, A \to a, S \to bB, B \to Sb, B \to b, S \to a, S \to b\}$. Figure 4 shows a derivation in this grammar.

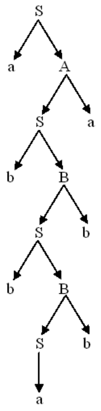

Fig. 4. Derivation in a linear grammar with rules in normal form

Example 4. Let $G_4 = \langle\{S\}, \{a, b\}, S, \{S \to ab, S \to aSb\}\rangle$ be a linear grammar. Figure 5 shows a Petri-net representation of a derivation in this system.

Remark 1. When a grammar only has rules in types $A \to aB, A \to Ba$, $A \to a$ the derivation graph is a binary tree. All non-terminal nodes except the last one has two successor nodes and exactly one of them is a non-terminal node.

Based on Proposition 1 the following statement is true.

Corollary 1. *The derivation trees and the derivations in linear (or especially regular) grammars have one-to-one correspondence.*

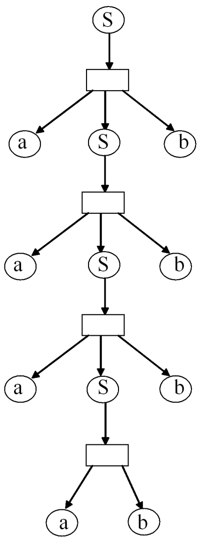

Fig. 5. Petri-net for derivation in a linear grammar

The derivation processes that use the Petri-net representation in linear and regular cases can only go in one order: the order of switching of the transitions is determined by the net. The regular and the linear cases only differ in the ordering of the output places and leaves. Based on these facts linear languages can be recognized by finite automata equipped by two heads [10, 8].

The concept of derivation-trees is more complete in the context-free grammars, as the next section will present.

6 Derivations in Context-Free Grammars

CF grammars are very popular because the concept of derivation trees fits very well in these derivations. CF grammars are more powerful than previous linear and regular grammars. The left hand side of each rule contains only one non-terminal as in the linear or regular case, but there is no restriction for on the right hand side.

Well-known the following fact about the possible 'divide to smaller (easier) parts (problems)' type parallelism.

Proposition 2. *In a context-free grammar the derivation can go in a maximal parallel way. The derivation tree can be built by levels: i.e., every non-terminal of the sentential form can be rewritten using a derivation rule at the same time.*

Proof. It follows from the structure of the derivation rules. In each rule-using a non-terminal will be replaced independently of the other parts of the sentential form. For each derivation (knowing the replaced non-terminal and the applied rule) there is a unique derivation tree. But we get the same result as the original derivation, if all non-terminals of the sentential form is replaced by each step. In this way we build a complete level of the derivation tree in each step. □

Corollary 2. *The possible sequential derivations in context free grammars form equivalent classes. Each class can be represented by a derivation tree. Each derivation tree can be represented by a unique sequential derivation, the so-called left-most derivation.*

In this way, one can assume that the nodes of the derivation tree are problems, the child nodes are the subproblems and the terminal-labeled nodes are the easily-solved (or trivial) problems. The derivation is "more parallel" in the context-free case than in the linear one. The word can be built in many places at the same time independently.

Example 5. Let $G_5 = \langle \{S, A\}, \{a, b, c, d\}, S, H_5 \rangle$ be a context-free grammar, with rule set $H_5 = \{S \rightarrow AbA, S \rightarrow cA, A \rightarrow a, A \rightarrow dSd\}$. Figure 6 shows a derivation-tree in this system.

Now, we recall possible normal-forms for context-free grammars.

Fact 1 *For each context-free grammar there is an equivalent grammar in which all derivation rules are in one of the forms $A \rightarrow BC, A \rightarrow a$ $(A, B, C \in N, a \in T)$. A grammar that only has these kinds of rules is in the so-called Chomsky normal form.*

Using the Chomsky normal form the tree has a special binary tree form. Each node labeled by a non-terminal has two successor nodes labeled by non-terminals or only one successor node labeled by a terminal (leaf-node).

In the next example we generate the Dyck language.

Example 6. Let $G_6 = \langle \{S, A, B, C\}, \{a, b\}, S, H_6 \rangle$ be a context-free grammar in Chomsky normal form, with rule set $H_6 = \{S \rightarrow SS, S \rightarrow AB, S \rightarrow AC, C \rightarrow SB, A \rightarrow a, B \rightarrow b\}$. Figure 7 shows a derivation-tree in Petri-net form in this system.

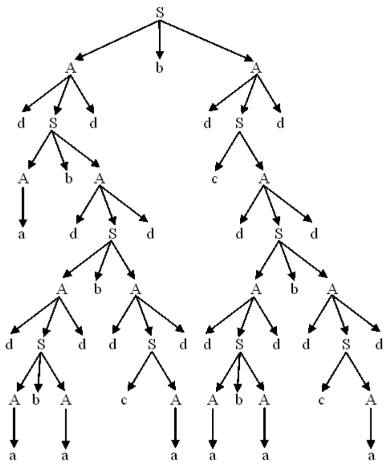

Fig. 6. Derivation in a context-fee grammar

As the figures show the parallel branches of the trees are independent of each other. The branches of the derivation use their tokens independently. In the case of CF (and also regular and linear) grammars the sequence of switching of the Petri-net of a derivation(-tree) must be finished by a successful derivation, since there is at least 1 active transition until the system-state equals the state with tokens exactly on the leaves.

In the sections below we show that the concept of derivation trees and the parallel derivations can be used in context sensitive and phrase structured grammars as well (for more details, see [6, 9]).

7 Derivations in Context-Sensitive Grammars

Normally we can use sequential derivations with the sentential forms in CS case. The concept of derivation trees does not work in pure form. The neighborhood of a non-terminal can also be important using a replacing rule. In the “old days” of formal language theory various attempts were made to describe the derivations of context-sensitive grammars by tree-like structures. In general, the results were not satisfactory. In this section we present a new kind of derivation structure, which may be useful.

We use two kinds of edges in these derivation graphs. The original, derivation edges come from the replaced non-terminal and go to the new parts of the string

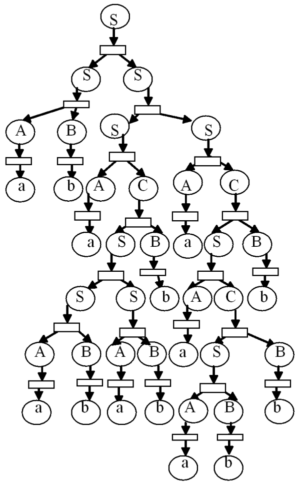

Fig. 7. Petri-net form of a derivation in a Chomsky normal form grammar

given in the right hand side of the derivation rule used. The new type of edges (represented by boxes and dotted arrows) show the neighborhood of the non-terminals replaced as it is requested by the applied rule. We will use the names context-box and context-edge.

Example 7. Let $G_7 = \langle\{S, A, B, C, D, E, F, G, I, J, K, L, M, O, P\}, \{a, b, c\}, S, H_7\rangle$ be a context sensitive grammar, with rule set $H_7 = \{S \to aSA, S \to bSB, abS \to abCE, baSA \to baDFA, EA \to EG, EG \to IG, IG \to IE, IE \to AE, EB \to EJ, EJ \to KJ, KJ \to KE, KE \to BE, FA \to FL, FL \to ML, ML \to MF, MF \to AF, FB \to FO, FO \to PO, PO \to PF, PF \to BF, CA \to CE, CB \to CF, DA \to DE, DB \to DF, C \to a, D \to b, E \to a, F \to b\}$. Figure 8 shows a possible derivation-graph in this system.

The derivation here can have parallel branches, but the solutions of the subproblems are not necessarily independent. Sometimes in order to continue the work on the solution of a subproblem results are needed from other (neighboring) subproblem-solutions. (Communication is used in this way among neighboring branches.)

Now we use Penttonen's result:

Fact 2 *Every context-sensitive language can be generated by a grammar whose derivation rules are of the form $A \to BC$, $AB \to AC$, $A \to a$, where A, B and*

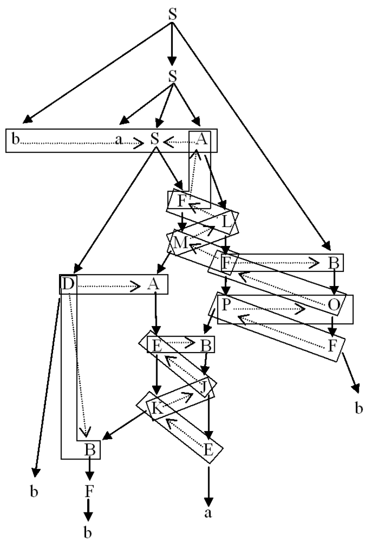

Fig. 8. "Derivation-tree" in a context sensitive grammar with context-boxes

C are nonterminals and a is a terminal. This normal form is from ([11]), where it was called the one-sided normal form.

Using this normal form the derivation 'tree' will be simpler. Each context-box contains only a left-neighbor non-terminal. Using this special grammar form the derivation graphs will have simpler structures.

Example 8. Let $G_8 = \langle\{S, A, B, C, D, E, F, G, I, J, K, L, M, O\}, \{a, b, c\}, S, H_8\rangle$ be a context sensitive grammar in Penttonen normal form, with rule set $H_8 = \{S \to AG, G \to BC, A \to IJ, J \to DE, EB \to EE, EC \to EK, K \to FL, D \to IM, M \to AB, BE \to BB, BF \to BO, O \to CL, A \to a, B \to b, C \to c, D \to a, E \to b, F \to c, I \to a, L \to c\}$. Figure 9 shows the Petri-net of a possible derivation-graph in this system.

Remark 2. In a context-sensitive grammar the derivation can be parallel, but when the context is important a synchronization point is needed. Therefore, the derivation is not maximal parallel; the 'speeds' of the branches usually differ.

There is a token for each non-terminal in the graph which is present at the same time (in the actual sentential form). Then using a rule in which there are more

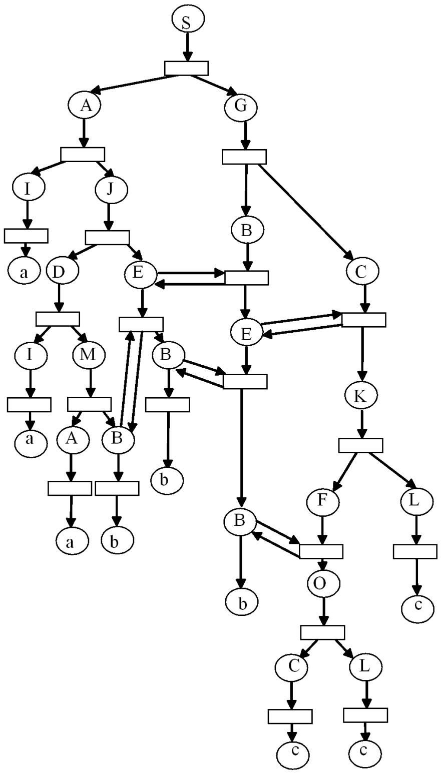

Fig. 9. Petri-net of a "derivation-tree" in a Penttonen normal form grammar

non-terminals on the right hand side than the left hand side the graph will have more tokens than before. For example at rule $S \to AG$ a new token will be born as in previous (i.e. CF) cases. The rules that need context can be used only if all the letters which are in their context have tokens. For instance, in the rule $EC \to EK$ the non-terminal C can be replaced by K only if there are tokens at C and at the left-most neighbor in the derivation, which is a node labelled by an E. Then a token moves from the node labelled by C to node K on a normal graph edge. The derivation is successfully terminated if all the leaves have tokens, and there are no other tokens in the graph. As we can see, we cannot generally use the concept 'level' in these graphs.

A derivation from a non-terminal can be continued when all branches are after the points where this non-terminal (as a place in the net) is needed as a (part of a) context. This means that the non-terminal has been used at all the context-edges which contain it; it satisfies the other branches of the derivation at the meeting points.

Using Penttonen's result, the derivation graph has a simpler form. The derivation can go left to right. The leftmost branch does not depend on other branches. The next branch may need context somewhere, and it can be found on the finished left neighbor branch. It is important that these context edges cannot cross each-other. Of course a derivation graph usually represents more than one sequential derivation. From the graph the "left-most" derivation of a word in a CS grammar can be obtained in the following recursive way.

Use the left-most branch up to the first place at which a context edge starts. Then use the next branch till this point and use the context edge as well. (When a context edge starts in this branch, then the right neighboring branch must be derived to this point as well, and so on.) When a branch terminates, the next one is its right-hand neighbor (at the lowest branching point). When all branches have terminal letters on their end, the derivation is finished.

In the Petri-net form of the derivations the "appearance check" appeared. In these transitions more tokens are needed, but only one of them "develops" after the transition (i.e. it is cancelled from its original place and appears in new place(s)). All others will be at the same place as before this step.

The derivation process in context-free (and specially in linear and regular) cases can halt only without a "living token", finishing the derivation of a word. In the case of context-sensitive derivations the net can halt without terminating the derivation, so the applications of the rules has a new restriction. This restriction is presented by the context edges and the context-connections of the net.

As we can see, if a kind of synchronization (communication, or appearance check) is used among the parallel branches, the generating power of the grammar increases. If Penttonen normal form is used our notation is redundant. The context arrows or context boxes can be deleted, because the context used must be a 'one letter left-context'. Using this kind of approach of context-sensitive derivations an algorithm based on the Cocke-Younger-Kasami algorithm can be presented for CS-parsing. (This could be the subject of future study.)

8 Derivations in RE (Arbitrary) Grammars

With the original form of these grammars (see Section 2.1) we cannot say anything about parallelism, because we cannot be sure about what kind of context will be needed and how it will change in a future derivation step. Using sentential form the derivation can be in sequentially, but we would like to say something more.

A normal form of the grammars can again be of help. There are many normal forms for phrase-structured grammar. We will use the following facts (they can be found in [15, 4]).

Fact 3 *Each phrase-structure grammar is equivalent to a grammar with only context-sensitive rules, and a single additional rule $A \to \lambda$.*

Every recursively enumerable language can be generated by a grammar containing rules only in forms $A \to BC$, $AB \to AC$, $A \to a$ and $A \to \lambda$ (where $A, B, C \in N, a \in T$).

Using the normal form given in Fact 3 our case looks like the context sensitive case but the rule(s) $A \to \lambda$, (where $A \in N$). Let us check what difference appears for the reason of these eliminating rules.

Example 9. Let $G_5 = \langle \{S, A, B, C\}, \{a, b, c\}, S, \{S \to AB, B \to BC, B \to \lambda, AC \to AA, A \to a, B \to b, C \to c\} \rangle$ be a phrase structured grammar. Figure 10 shows a possible derivation-graph in this system.

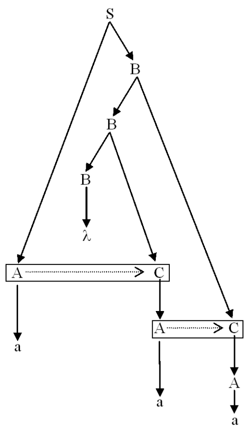

Fig. 10. "Derivation-tree" in a phrase structured normal form grammar

As we can see, the context edges can connect nodes which are far from each other when the empty word (λ) is derived from all the branches between. In these cases the node which needs to be a context must wait till the derivation is finished by the empty word in other branches, and then it can be used as the context of a further node.

A new phenomenon appears in these cases: The derivation process looses the tokens at the leaves containing the empty-word. (In all previous sections the derivation processes were without loosing tokens.) In the Petri-net representation the transitions without outgoing edges represent these λ-rules.

9 Conclusions

An approach to generate a finite language efficiently is shown (allowed any kind of derivation rules). In the case of regular and linear grammars the derivation must be sequential; the sequence of switching transitions of the corresponding Petri-net is unique. The generating power of a grammar is larger if the possibility of parallelism exists. In linear grammars the word is built parallely in two places: i.e. the order of leaves varies. In context free grammars the derivation can be maximal parallel, the word can be built in several places at the same time and the branches of the derivations are independent of each other. The Petri-net forms a tree, the number of possible switching sequences of transitions are usually high and they all represent successful derivations. As we have seen the generating power increases to allow synchronization (communication or appearance checking) between the parallel branches. In the context-sensitive case it is enough to use only the left-neighboring token in this synchronization. A sequence of switching transitions may not wait for these synchronization points, so the net can halt without a successful derivation. Moreover, if λ-rules are allowed this context communication can connect nodes which are far from each other so the efficiency is similar to that of the Turing Machines. In our paper we have mixed the concept of the derivation in a grammar and the theory of Petri-nets. Using grammars in normal forms at each token-multiplying step only one new token may appear. The parallelism of context-free derivations is our basis. With other kinds of restriction (for instance ordering the non-terminals) parallelism can be used in derivations of finite (regular/linear) grammars as well. To analyze what effectiveness mean involving the parallelism is a topic of further research. On the basis of the derivation-graph presented for the context-sensitive case we are working on a CYK-like parsing algorithm for grammars in Penttonen normal form.

As the first version of this paper is written in 2004, and it appears in 2010, some ideas presented here are further developed in, for instance, [7, 9, 6, 8].

Acknowledgements

The research is partly supported by funds of the Hungarian Ministry of Education (OTKA F043090 and OTKA T049409).

References

1. Hopcroft, J. and J.D. Ullmann (1979). *Introduction to automata theory, languages, and computation.* Reading: Addison-Wesley.
2. Kleijn, J. (2003). Concurrency and formal language theory. In *Lecture motes in the 2nd International PhD School on Formal Languages and Applications.* Tarragona: Universitat Rovira i Virgili.
3. Martín-Vide, C. (2003). Formal language theory: classical and non-classical machineries. In *Lecture notes in the 2nd International PhD School on Formal Languages and Applications.* Tarragona: Universitat Rovira i Virgili.
4. Mateescu, A. (2004). On context-sensitive grammars. In C. Martín-Vide, V. Mitrana and Gh. Păun (eds.), *Formal Languages and Applications*, pp. 139–162. Berlin: Springer.
5. Nagy, B. (2004). Derivations in Chomsky-type grammars in mirror of parallelism (extended abstract). In *IS-TCS'04, Theoretical Computer Science – Information Society (ACM conference)*, pp. 181-184. Ljubljana.
6. Nagy, B. (2006). Left-most derivation and shadow-pushdown automata for context-sensitive languages. In *Proceedings of the 10th WSEAS International Conference on Computers*, pp. 962-967. Athens.
7. Nagy, B. (2006). On the notion of parallelism in artificial and computational intelligence. In *Proceedings of the 7th International Symposium of Hungarian Researchers on Computational Intelligence*, pp. 533–541. Budapest.
8. Nagy, B. (2008). On $5' \rightarrow 3'$ sensing Watson-Crick finite automata. *Lecture Notes in Computer Science*, 4848: 256–262.
9. Nagy, B. (2010). Derivation trees for context-sensitive grammars. In M. Ito, Y. Kobayashi and K. Shoji (eds.), *Automata, Formal Languages and Algebraic Systems.* Singapore: World Scientific.
10. Nagy, B. (2012). A class of 2-head finite automata for linear languages. *Triangle*, 8: 89–99 (this volume).
11. Penttonen, M. (1974). One-sided and two-sided context in formal grammars. *Information and Control*, 25: 371–392.
12. Reisig, W. and G. Rozenberg (eds.) (1998). *Lectures on Petri Nets I: Basic models.* Berlin: Springer.
13. Révész, G. (1983). *Introduction to formal languages.* New York: McGraw-Hill.
14. Rozenberg, G. and A. Salomaa (eds.) (1997). *Handbook of formal languages.* Berlin: Springer.
15. Salomaa, A. (1973). *Formal languages.* New York: Academic Press.