THE GENERATIVE POWER OF UNSYNCHRONIZED CONTEXT-FREE PARALLEL COMMUNICATING GRAMMAR SYSTEMS

by

Simin Li

A thesis submitted to the
Department of Computer Science
in conformity with the requirements for
the degree of Master of Science

Bishop's University
Canada
April 2022

Abstract

Parallel communicating grammar systems (PCGS for short) were introduced as a grammatical model of parallel computations. Communication plays a major role in parallel processing architectures and it makes the whole system more powerful in solving a common task than its grammar components. In this thesis we investigate the generative power of unsynchronized PCGS with context-free components. We show that all the languages generated by these systems can be accepted in linear space and so are context sensitive.

Acknowledgments

I would like to thank the Computer Science department at Bishop's University for giving me the opportunity to pursue a Master's degree.

I would like to underline the support, patience and guidance received from Dr. Stefan D. Bruda, without him none of this would have been possible.

I would like to thank all other professors in Department of Computer Science at Bishop's University, Dr. Russell Butler, Dr. Mohammed Ayoub Aloui Mhamd, from whom I learned a lot.

Last but not least, I would like to thank my parents, for their enduring love and support, and believing in me when I did not believe in myself.

Contents

Chapter 1

Introduction

Parallel communicating grammar systems (PCGS for short) have been introduced as a language-theoretic treatment of concurrent systems [15], with the aim of combining the concepts of parallelism and communication into a suitable model for theoretical studies of the properties of parallel processing systems.

A PC grammar system consists of several grammars (components) working in parallel, each of them having its own sentential form or string. They use a set of rewriting rules and start from their own axiom (or start symbol), repeatedly rewriting the corresponding sentential form using the given rules. During the process each component can point to other components of the system using query symbols. When the query symbol Q_j is introduced by a component i, the current sentential form of the component j will be sent to the component i, replacing the occurrences of Q_j. This whole process is a derivation that is, a sequence of component-wise rewriting and communication steps. The communication between components is based on the query symbol appearing in the current sentential forms generated by the grammars, and has priority over rewriting: once a query symbol appears it must be satisfied before any rewriting step can happen. One of these component grammars of a PCGS is distinguished as the master grammar. Exactly all the terminal strings generated by the master grammar constitute the generated language. Formal definitions will be provided in the next chapter.

Grammars communicate in one of two ways: returning or non-returning. In a returning system, once a communication request has been completed the queried component will erase its string, replace it with the respective axiom, and continue the derivation from there. If a system is in non-returning mode, then the component string remains unchanged after a communication event, and the subsequent derivation continues to rewrite that string.

PCGS can also be classified as synchronized or unsynchronized. In a synchronized PCGS, during a rewriting step each component must (synchronously) apply a rewriting rule. The only exception if when the respective string consists exclusively of terminals, case in which nothing happens with that string during a rewriting step.

Otherwise, if some component does not have any suitable rewriting rule, then the derivation will block. We do not have any such a concern for unsynchronized PCGS. In such a PCGS each grammar is allowed to either perform a rewriting step or wait.

Because of the synchronization and communication facilities, PCGS whose components are of a certain type are generally more powerful than a single Chomsky grammar of the same type [3, 15].

Synchronized PCGS have received sustained attention through the years. In particular, all flavors of synchronized PCGS with context-free components have been found to be Truing complete [4, 18]. The unsynchronized variant on the other hand has received almost no attention. This motivates our thesis, where we investigate the generative power of unsynchronized PCGS with context-free components. We find quite a sharp contrast between the synchronized and the unsynchronized case. Indeed, we establish that all the languages generated by unsynchronized context-free PCGS can be accepted in linear space. That is, all these languages are context-sensitive languages.

Chapter 2

Preliminaries

For an alphabet V, the free monoid generated by V under the operation of concatenation, is denoted by V^*. For $x \in V^*$, and a set $U \subseteq V$, $|x|$ is the length of x and $|x|_U$ stands for the number of occurrences of elements of U in x. The empty string (and only the empty string) is denoted by ε.

2.1 Grammars

A Chomsky grammar is denoted by $G = (N,\ T,\ S,\ P)$, where N is the set of nonterminal symbols, T is the set of terminal symbols, $S \in N$ is the axiom (or start symbol), and $P \subseteq ((N \cup T)^* N (N \cup T)^*) \times (N \cup T)^*$ is the set of rewriting rules (or just rules, for short). A rewriting rule $(\sigma,\ \sigma')$ is customarily written $\sigma \to \sigma'$. A rewriting step replaces the string $w = u\sigma v$ with $w' = u\sigma' v$ whenever $\sigma \to \sigma' \in P$ (written $w \Rightarrow_G w'$, though we often omit the subscript G whenever the grammar is understood from the context). The language generated by G is denoted by $L(G)$. *REG*, *LIN*, *CF*, *CS*, *RE* are the families of regular, linear, context-free, context-sensitive, recursively enumerable languages, respectively.

The Chomsky hierarchy defines four classes of grammars, depending on the form of the rewriting rules. Let $G = (\Sigma,\ V,\ S,\ P)$ be a grammar. We then have the following:

1. G is a type-0 or unrestricted grammar, if G does not have any restriction. This type of grammar can generate languages which can be semi-decided by Turing machines. Languages generated by a type-0 grammar are called recursively enumerable, or RE for short [11].

2. G is a type-1 or context-sensitive grammar, if $|\alpha| \leq |\beta|$ for every rewriting rule $\alpha \to \beta$ in P. This type of grammar can have a rewriting rule of the form $S \to \varepsilon$, but only if S is not on the right-hand side of any rewriting rule. Languages generated by a type-1 grammar are called context sensitive, or CS

for short [11]. It should be noted that all the context-sensitive languages can be accepted by linear space-bounded Turing machines [16].

3. G is a type-2 or context-free grammar, if $|\alpha| = 1$ for every rewriting rule $\alpha \to \beta$ in P (meaning that α is a single nonterminal symbol). Linear grammars are a special type of context-free grammars, in which no rewriting rule is allowed to have more than one nonterminal symbol in its right-hand side. Languages generated by a type-2 grammar are called context free, or CF for short and languages generated by the linear grammar sub-type are called Linear or LIN [9, 10].

4. G is a type-3 or regular grammar, if its rewriting rules have one of the following forms: $A \to cB,\ A \to c,\ A \to \varepsilon$, or $A \to B$, where $A,\ B$ are nonterminals and c is a terminal. Languages generated by a type-3 grammar are called regular, or REG for short.

   Note in passing that a language is called semilinear if and only if it is letter equivalent to a regular language. Two languages are called letter equivalent whenever the languages are indistinguishable from each other if we only look at the relative number of occurrences of symbols in their strings, without taking their order into consideration [11].

2.2 Parallel Communicating Grammar Systems

A parallel communicating grammar system (PCGS for short) provides a theoretical prototype that combines the concepts of grammars with parallelism and communication. The idea behind PCGS is the notion of multiple grammars that work together in parallel to generate strings and communicate in the process with each other. This concept supports the investigation of language-theoretic properties of parallel systems.

Definition 2.1. PARALLEL COMMUNICATING GRAMMAR SYSTEM [3]: *Let $n \geq 1$ be a natural number. A parallel communicating grammar system (or PCGS) of degree n is an $(n+3)$-tuple*

$$\Gamma = (N,\ T,\ K,\ G_1,\ \ldots,\ G_n)$$

where $N,\ T,\ K$ are pairwise disjoint alphabets, with $K = \{Q_1,\ \ldots,\ Q_n\}$. The elements of N are nonterminal symbols and those of T are terminal symbols. Each G_i, $1 \leq i \leq n$ is a usual Chomsky grammar:

$$G_i = (N \cup K,\ T,\ P_i,\ S_i)\,,\ 1 \leq i \leq n$$

The grammars G_i, $1 \leq i \leq n$, are called the components of the system. The elements of K are called query symbols; their indices $1,\ \ldots,\ n$ point to the components $G_1,\ \ldots,\ G_n$, respectively.

PCGS derivations consist of a series of rewriting and communication steps. The communication has priority over rewriting, meaning that a rewriting step is allowed only when no query symbol appears in the current configuration. Note that the query symbols are associated in a one-to-one manner with the components.

Definition 2.2. DERIVATION IN A PCGS [3]*: Given a PCGS Γ as above, for two n-tuples $(x_1,\ x_2,\ \ldots,\ x_n)$ and $(y_1,\ y_2,\ \ldots,\ y_n)$, with $x_i,\ y_i \in V_\Gamma^*,\ 1 \le i \le n$ (we call such an n-tuple a configuration), and $x_1 \notin T^*$, we write*

$$(x_1,\ x_2,\ \ldots,\ x_n) \Rightarrow (y_1,\ y_2,\ \ldots,\ y_n)$$

if one of the following two cases holds:

1. *$|x_i|_K = 0$ for all $1 \le i \le n$, then $x_i \Rightarrow_{G_i} y_i$ or $x_i = y_i \in T^*,\ 1 \le i \le n$.*
2. *There is $i,\ 1 \le i \le n$, such that $|x_i|_K > 0$. We write such a string x_i as*

   $$x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1},\ t \ge 1,$$

   for $t \ge 1, z_i \in (N \cup T)^*,\ 1 \le j \le t+1$. If $|x_{i_j}|_K = 0$ for all $1 \le j \le t$, then

   $$y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1},$$

   [and $y_{i_j} = S_{i_j},\ 1 \le j \le t$]. For all the indices i not specified above we have $y_i = x_i$.

   A PCGS is returning *if the derivation proceeds as above, and* non-returning *if the phrase "[and $y_{i_j} = S_{i_j},\ 1 \le j \le t$]" is removed from the definition*

In other words, an n-tuple $(x_1,\ \ldots,\ x_n)$ yields $(y_1,\ \ldots,\ y_n)$ if either of the two cases hold:

1. If there is no query symbol in $x_1,\ \ldots,\ x_n$, then we have a component-wise derivation $x_i \Rightarrow_{G_i} y_i,\ 1 \le i \le n$ (one rule is used in each component G_i), unless x_i is terminal ($x_i \in T^*$), case in which it remains unchanged ($y_i = x_i$).
2. If a query symbol appears somewhere in the configuration then a communication step is required: Each occurrence of Q_j in x_i is replaced by x_j, providing that x_j does not contain query symbols. In a communication step all the occurrences of query symbols are eventually replaced by strings containing no such symbols. Once the communication step is complete, the grammar G_j continues processing from its axiom or from x_j depending on whether the system is returning or non-returning.

We use $\Rightarrow$ to denote any derivation step (both component-wise rewriting and communication). Whenever not clear from the context we may use $\Rightarrow_r$ and $\Rightarrow_{nr}$ for the returning and non-returning modes, respectively. A sequence of rewriting and communication steps are denoted by $\Rightarrow^*$, the reflexive and transitive closure of $\Rightarrow$. Again we may occasionally qualify this operator by using $\Rightarrow^*_r$ or $\Rightarrow^*_{nr}$ (for the returning or non-returning modes).

A derivation in a PCGS is blocked (that is, cannot continue) in the following two cases [3, 13, 14, 17]:

1. The derivation cannot continue when a component x_i of the current n-tuple $(x_1, \ldots, x_n)$ contains nonterminals but does not contain any nonterminal that can be rewritten in G_i.

2. The derivation cannot continue when a circular query appears. This happens when G_{i_1} introduces Q_{i_2}, G_{i_2} introduces Q_{i_3}, and so on until $G_{i_{k-1}}$ introduces Q_{i_k} and G_{i_k} introduces Q_{i_1}. On one hand, communication has priority. On the other hand, only strings without query symbols can be communicated. It follows that no communication is possible in this cycle. In other words, no continuation of the derivation is possible.

Definition 2.3. LANGUAGES GENERATED BY PCGS [3]: *The language generated by a PCGS* Γ *is the language generated by its first component* (G_1 *above*), *when starting from the configuration* $(S_1, \ldots, S_n)$, *that is*

$$L_f(\Gamma) = \{w \in T^* \mid (S_1, \ldots, S_n) \Rightarrow^* (w, \alpha_1, \ldots, \alpha_n)$$

where $\alpha_i \in (N \cup T \cup K)^*$, $2 \leq i \leq n$.

The tuple of axioms $(S_1, \ldots, S_n)$ is where the derivation begins. Before G_1 produces a terminal string, a number of rewriting and/or communication steps are performed. At the end we will get a terminal string produced by the master grammar.

Definition 2.4. PCGS SEMANTICS [17]: *Let* $\Gamma = (N, T, K, G_1, \ldots, G_n)$ *be a PCGS.*

1. *If only* G_1 *is allowed to introduce query symbols, then we say* Γ *is a centralized PCGS. On other hand we say* Γ *is a non-centralized PCGS if there is no restriction imposed on the introduction of query symbols.*

2. *A PCGS is said to be returning if each component resumes working from its axiom after being communicated. When each component continues the processing of the current string after communication instead, the PCGS is said to be non-returning.*

3. *A system is synchronized when each component grammar uses exactly one rewriting rule in each component-wise derivation step (except when the component grammar is holding a terminal string, case in which it is allowed to wait). In a non-synchronized system, each component may choose to either rewrite or wait in any step which is not a communication step.*

Since the returning and the non-returning modes of derivation can be used for the same system, we may denote by $L_r(\Gamma)$ the language generated by Γ in the returning mode, and by $L_{nr}(\Gamma)$ the language generated by Γ in the non-returning mode. We will often omit the subscript r of nr whenever we want to refer to both modes, or the mode is understood from the context.

In the synchronized case we denote by $PC_n(X)$, $n \geq 1$, the family of languages generated in the returning mode by non-centralized PCGS with at most n components and with rules of type X (where X is an element of the Chomsky hierarchy). We add the symbol C if centralized systems are used, and the symbol N if the non-returning mode of derivation is used. We thus we obtain the classes $CPC_n(X)$, $NPC_n(X)$, $NCPC_n(X)$. When an arbitrary number of components is considered, we use $*$ in the subscript instead of n. If the number of components has no restriction, the subscript n may also be removed, thus obtaining $PC(X)$, $CPC(X)$, $NPC(X)$, and $NCPC(X)$.

For the unsynchronized case we add the prefix U, thus obtaining the classes $UPC(X)$, $UCPC(X)$, $UNPC(X)$, and $UNCPC(X)$ (and $UPC_n(X)$, $UCPC_n(X)$, $UNPC_n(X)$, $UNCPC_n(X)$ as well).

We assume that the reader is familiar with the basic structure and behaviour of a Turing machine [11], so we only provide a definition for the intuitive though not as frequently used notion of space bounds for these machines.

Definition 2.5. SPACE-BOUNDED TURING MACHINE [16]: *Given a Turing machine M and an input string $x \in T^*$, the working space of M on x is the length of all the work tapes of M to accept x. More generally, let S be any function from $\mathbb{N}$ to $\mathbb{N}$. Let $L \subseteq T^*$, we say that M semi-decides L in space S provided that M semi-decides (or accepts) L and uses at most $S(n)$ tape cells on any input of length n in T^*. Then M is a $S(n)$ space-bounded Turing machine.*

Space-bounded Turing Machines will be used to define the computational complexity of certain classes of PCGS with context-free components.

2.3 Examples

PCGS can be classified according to their grammar structure, behavior after satisfying a query, and timing. Now we give some examples to show the corresponding situations respectively.

Recall that a PCGS is centralized if there is only grammar authorized to introduce query symbols. If the number of grammars that can use the query to request a string is greater than two, a PCGS is non-centralized.

For example, given

$$\Gamma = (\{S_1,\ S_2,\ S_3\},\ K,\ \{a,\ b,\ c\},\ G_1,\ G_2,\ G_3).$$

The following is a centralized PCGS, since query symbols only appears in P_1.

$$\begin{aligned}&P_1 = \{S_1 \to aS_1,\ S_1 \to aQ_2,\ S_2 \to bQ_3,\ S_3 \to c\},\\&P_2 = \{S_2 \to bS_2\},\\&P_3 = \{S_3 \to cS_3\}.\end{aligned}$$

On the other hand, the following is a non-centralized PCGS, since query symbols appears in both P_1 and P_2.

$$\begin{aligned}&P_1 = \{S_1 \to aS_1,\ S_1 \to aQ_2,\ S_2 \to bQ_3,\ S_3 \to c\},\\&P_2 = \{S_2 \to bS_2,\ S_2 \to bQ_3\},\\&P_3 = \{S_3 \to cS_3\}.\end{aligned}$$

Consider now what happens after a component provides the string for the grammar which request it by issuing the corresponding query symbol. In a returning system, the grammar will resume working from its axiom. In a non-returning system, the grammar will continue the rewriting of the current string of the corresponding component.

Let us consider the following PCGS as an example:

$$\begin{aligned}&\Gamma = (\{S_1,\ S_2,\ S_3\},\ K,\ \{a,\ b,\ c\},\ G_1,\ G_2,\ G_3),\\&P_1 = \{S_1 \to aS_1,\ S_1 \to aQ_2,\ S_2 \to bQ_3,\ S_3 \to c\},\\&P_2 = \{S_2 \to bS_2\},\\&P_3 = \{S_3 \to cS_3\}.\end{aligned}$$

For the following derivation, the component P_1 and P_2 return to their corresponding axiom when Q_1 and Q_2 are satisfied. Thus, PCGS is in returning mode when the following derivation happens:

$$\begin{aligned}&(S_1,\ S_2,\ S_3) \Rightarrow (aS_1,\ bS_2,\ cS_3) \Rightarrow^{*n} (a^nS_1,\ b^nS_2,\ c^nS_3) \Rightarrow\\&(a^{n+1}Q_2,\ b^{n+1}S_2,\ c^{n+1}S_3) \Rightarrow (a^{n+1}b^{n+1}S_2,\ S_2,\ c^{n+1}S_3) \Rightarrow\\&(a^{n+1}b^{n+2}Q_3,\ bS_2,\ c^{n+2}S_3) \Rightarrow (a^{n+1}b^{n+2}c^{n+2}S_3,\ bS_2,\ S_3) \Rightarrow\\&(a^{n+1}b^{n+2}c^{n+3},\ b^2S_2,\ cS_3).\end{aligned}$$

By contrast, PCGS is in the non-returning mode if the following derivation occurs:

$$\begin{aligned}&(S_1,\ S_2,\ S_3) \Rightarrow (aS_1,\ bS_2,\ cS_3) \Rightarrow^{*n} (a^nS_1,\ b^nS_2,\ c^nS_3) \Rightarrow\\&(a^{n+1}Q_2,\ b^{n+1}S_2,\ c^{n+1}S_3) \Rightarrow (a^{n+1}b^{n+1}S_2,\ b^{n+1}S_2,\ c^{n+1}S_3) \Rightarrow\\&(a^{n+1}b^{n+2}Q_3,\ b^{n+2}S_2,\ c^{n+2}S_3) \Rightarrow (a^{n+1}b^{n+2}c^{n+2}S_3,\ b^{n+2}S_2,\ c^{n+2}S_3) \Rightarrow\\&(a^{n+1}b^{n+2}c^{n+3},\ b^{n+3}S_2,\ c^{n+3}S_3).\end{aligned}$$

A PCGS is called synchronized if each grammar uses exactly one rewriting rule in each component-wise derivation step (except if the component grammar is holding a terminal string). If each grammar can choose either rewrite or wait in any step which is not a communication step, then the system is called unsynchronized. Note that the way how a query symbol generates an immediate communication step is the same in both the synchronized and unsynchronized systems.

We provide an example by using the system Γ with $P_1,\ P_2,\ P_3$ as above. The following derivation assumes that the system is synchronized:

$$\begin{aligned}&(S_1,\ S_2,\ S_3) \Rightarrow (aS_1,\ bS_2,\ cS_3) \Rightarrow^{*n} (a^nS_1,\ b^nS_2,\ c^nS_3) \Rightarrow^{*k}\\ &(a^{n+k}S_1,\ b^{n+k}S_2,\ c^{n+k}S_3) \Rightarrow^{*m} (a^{n+k+m}S_1,\ b^{n+k+m}S_2,\ c^{n+k+m}S_3) \Rightarrow\\ &(a^{n+k+m+1}Q_2,\ b^{n+k+m+1}S_2,\ c^{n+k+m+1}S_3) \Rightarrow\ \ldots\end{aligned}$$

On the other hand, when the system is considered unsynchronized, the following is a possible derivation:

$$\begin{aligned}&(S_1,\ S_2,\ S_3) \Rightarrow (aS_1,\ bS_2,\ cS_3) \Rightarrow^{*n} (a^nS_1,\ bS_2,\ c^nS_3) \Rightarrow^{*k}\\ &(a^{n+k}S_1,\ b^{1+k}S_2,\ c^nS_3) \Rightarrow^{*m} (a^{n+k+m}S_1,\ b^{1+k+m}S_2,\ c^{n+m}S_3) \Rightarrow\\ &(a^{n+k+m}Q_2,\ b^{1+k+m}S_2,\ c^{n+m}S_3) \Rightarrow (a^{n+k+m}b^{1+k+m}S_2,\ S_2,\ c^{n+m}S_3) \Rightarrow\ \ldots\end{aligned}$$

Note that the derivation in an unsynchronized system is a special case of the derivation in the synchronized system. Indeed, any unsynchronized system can be converted into an equivalent synchronized system by adding the set $\{A \to A : A \in N\}$ to all the sets of rewriting rules in all the components.

Now that the semantics of the PCGS communication and synchronization has been illustrated, we proceed with a few more interesting examples.

Example 1. *Consider a centralized non-returning regular PCGS*

$$\Gamma_1 = (\{S_1,\ S_2\},\ K,\ \{a,\ b,\ c\},\ G_1,\ G_2)$$

with

$$\begin{aligned}P_1 &= \{S_1 \to aS_1,\ S_1 \to aQ_2,\ S_2 \to cQ_2,\ S_2 \to c\},\\ P_2 &= \{S_2 \to bS_2\}.\end{aligned}$$

A derivation in Γ_1 *has the following form:*

$$\begin{aligned}&(S_1,\ S_2) \Rightarrow^* (a^kS_1,\ b^kS_2) \Rightarrow (a^{k+1}Q_2,\ b^{k+1}S_2) \Rightarrow\\ &(a^{k+1}b^{k+1}S_2,\ b^{k+1}S_2) \Rightarrow (a^{k+1}b^{k+1}cQ_2,\ b^{k+2}S_2) \Rightarrow\\ &(a^{k+1}b^{k+1}cb^{k+2}S_2,\ b^{k+2}S_2) \Rightarrow (a^nb^ncb^{n+1}c\ldots cb^{k+j}c,\ b^{k+j}S_2),\end{aligned}$$

for $r \geq 2,\ k_i \geq 0,\ 1 \leq i \leq r$. *Therefore:*

$$L(\Gamma_1) = \{a^{k_1}b^{k_1}a^{k_2}b^{k_2}\ldots a^{k_r}b^{k_r}c \mid r \geq 2,\ k_1 \geq 1,\ k_i \geq 2,\ 1 \leq i \leq r\}.$$

Although $G_1,\ G_2$ are regular grammars, $L(\Gamma_1)$ is not linear. The result that $L(\Gamma_1)$ is not linear can be proved easily by using the following necessary condition for a language to be linear: If $L \subseteq V^*$ is a linear language, then two regular languages $L_1,\ L_2$ exist, such that $L \subseteq L_1L_2$ and for each $x \in L_1$ $(y \in L_2)$ there is an $y \in L_2$ $(x \in L_1)$, such that $xy \in L$.

Example 2. *Consider the following PCGS:*

$$\Gamma_2 = (\{S_1,\ S_1',\ S_2,\ S_3\},\ K,\ \{a,\ b\},\ G_1,\ G_2)$$

with

$$\begin{aligned}&P_1 = \{S_1 \to abc,\ S_1 \to a^2b^2c^2,\ S_1 \to aS_1',\ S_1 \to a^3Q_2,\ S_1' \to aS_1',\\ &\ S_1' \to a^3Q_2,\ S_2 \to b^2Q_3,\ S_3 \to c\},\\ &P_2 = \{S_2 \to bS_2\},\\ &P_3 = \{S_3 \to cS_3\}.\end{aligned}$$

We start with $(S_1,\ S_2,\ S_3)$. First using the rule $S_1 \to aS_1'$ and the rule $S_1' \to aS_1'$ in P_1 successively, then use the unique rules in $P_2,\ P_3$ for $k \geq 0$ times. We get

$$(S_1,\ S_2,\ S_3) \Rightarrow_r (aS_1',\ bS_2,\ cS_3) \Rightarrow_r^* (a^{k+1}S_1',\ b^{k+1}S_2,\ c^{k+1}S_3).$$

Eventually, the rule $S_1' \to a^3Q_2$ in P_1 will be used:

$$(a^{k+1}S_1',\ b^{k+1}S_2,\ c^{k+1}S_3) \Rightarrow_r (a^{k+4}Q_2,\ b^{k+2}S_2,\ c^{k+2}S_3).$$

Since the query symbol Q_2 is present, $b^{k+2}S_2$ is sent to the first component and replaces Q_2 in the following communication step:

$$(a^{k+4}Q_2,\ b^{k+2}S_2,\ c^{k+2}S_3) \Rightarrow_r (a^{k+4}b^{k+2}S_2,\ S_2,\ c^{k+2}S_3).$$

Now we perform the following steps:

$$\begin{aligned}&(a^{k+4}b^{k+2}S_2,\ S_2,\ c^{k+2}S_3) \Rightarrow_r (a^{k+4}b^{k+4}Q_3,\ bS_2,\ c^{k+3}S_3) \Rightarrow_r\\ &(a^{k+4}b^{k+4}c^{k+3}S_3,\ bS_2,\ S_3) \Rightarrow_r (a^{k+4}b^{k+4}c^{k+4},\ b^2S_2,\ cS_3).\end{aligned}$$

Therefore, all strings $a^nb^nc^n,\ n \geq 4$, can be produced in this way. We can use the following derivation to obtain the string $a^3b^3c^3$:

$$\begin{aligned}&(S_1,\ S_2,\ S_3) \Rightarrow_r (a^3Q_2,\ bS_2,\ cS_2) \Rightarrow_r (a^3bS_2,\ S_2,\ cS_3) \Rightarrow_r\\ &(a^3b^3Q_3,\ bS_2,\ c^2S_3) \Rightarrow_r (a^3b^3c^2S_3,\ bS_2,\ S_3) \Rightarrow (a^3b^3c^3,\ b^2S_2,\ cS_3).\end{aligned}$$

Finally, the strings $abc,\ a^2b^2c^2$ are produced directly by the master component P_1. Therefore the language generates by this system is

$$L_r(\Gamma_2) = L_{nr}(\Gamma_2) = \{a^nb^nc^n : n \geq 1\}.$$

Since there is only one query of P_1 to P_2 and only one to P_3, we obtain the same language in both returning and non-returning modes.

Note in passing that the above system is centralized. This demonstrates the increased power of a PCGS. Indeed, quite a simple PCGS with regular components can generate a classic examples of non-context-free languages.

Example 3. *Consider the following PCGS:*

$$\Gamma_3 = (\{S_1,\ S_2,\ S_3\},\ K,\ \{a,\ b,\ c,\ d\},\ G_1,\ G_2,\ G_3)$$

with

$$\begin{aligned}P_1 &= \{S_1 \to aS_1,\ S_1 \to aQ_2,\ S_3 \to d\},\\ P_2 &= \{S_2 \to bS_2,\ S_2 \to bQ_3\},\\ P_3 &= \{S_3 \to cS_3\}.\end{aligned}$$

Each derivation in Γ_3 *starts with*

$$(S_1,\ S_2,\ S_3) \Rightarrow^{*k} (a^kS_1,\ b^kS_2,\ c^kS_3),\ k \geq 0,$$

and then use the rules $S_1 \to aQ_2,\ S_2 \to bQ_3$ *in* $G_1,\ G_2$ *respectively. We obtain three cases:*

$$\begin{aligned}(a^kS_1,\ b^kS_2,\ c^kS_3) &\Rightarrow (a^{k+1}Q_2,\ b^{k+1}S_2,\ c^{k+1}S_3),\\ (a^kS_1,\ b^kS_2,\ c^kS_3) &\Rightarrow (a^{k+1}S_1,\ b^{k+1}Q_3,\ c^{k+1}S_3),\\ (a^kS_1,\ b^kS_2,\ c^kS_3) &\Rightarrow (a^{k+1}Q_2,\ b^{k+1}Q_3,\ c^{k+1}S_3).\end{aligned}$$

In the first case, after communicating $b^{k+1}S_2$ *to* G_1*, the derivation is blocked. In the second case the result is the same, after communicating* $c^{k+1}S_3$ *to* G_2*. In the third case, there are two query symbols in the configuration.* Q_2 *cannot be satisfied for the moment since it asks for a string containing query symbols. Hence, we first satisfy* Q_3*:*

$$(a^{k+1}Q_2,\ b^{k+1}Q_3,\ c^{k+1}S_3) \Rightarrow (a^{k+1}Q_2,\ b^{k+1}c^{k+1}S_3,\ v_3)$$

(the form of v_3 *depends on whether* Γ_3 *is considered as returning or non-returning).*

Now Q_2 *can be satisfied:*

$$(a^{k+1}Q_2,\ b^{k+1}c^{k+1}S_3,\ v_3) \Rightarrow (a^{k+1}b^{k+1}c^{k+1}S_3,\ v_2,\ v_3).$$

When Γ_3 *is non-returning,* $v_2 = b^{k+1}c^{k+1}S_3$ *and* S_3 *cannot be rewritten in* P_2*, so the derivation is blocked. Thus, the derivation can be concluded only in the returning case, case in which we have:*

$$(a^{k+1}b^{k+1}c^{k+1}S_3,\ S_2,\ S_3) \Rightarrow (a^{k+1}b^{k+1}c^{k+1}d,\ v_2',\ cS_3),$$

where $v_2' \in \{bS_2,\ bQ_3\}$*. In conclusion,*

$$L_r(\Gamma_3) = \{a^nb^nc^nd \mid n \geq 1\}.$$

The purpose of this example is to show a more complicated way of working with query symbols in the non-centralized case.

Chapter 3

Previous Work

Many results exist about the generative capacity of various types of PCGS. In this chapter, we focus on synchronized PCGS since no such results for the unsynchronized case exists.

The diagram in Figure 3.1 indicates some relations between the eight basic families of languages defined in the previous chapter, as well as their relationships with families in the Chomsky hierarchy. (MAT denotes the family of languages generated by matrix grammars with ε-free context-free rules and without appearance checking).

The two most powerful PCGS are Context sensitive (CS, for short) and recursively enumerable (RE, for short). Surprisingly their behavior is quite similar as well, though not identical. We first note the obvious fact that a recursively enumerable grammar is just as powerful as a PCGS with recursively enumerable components. Thus we have:

$$RE = Y_n(RE) = Y_*(RE),\ n \geq 1,$$

for all $Y \in \{PC,\ CPC,\ NPC,\ NCPC\}$ [3].

To a certain degree, the same holds for PCGS with context-sensitive components in relation with context-sensitive languages:

$$CS = Y_n(CS) = Y_*(CS),\ n \geq 1,$$

for all $Y \in \{CPC,\ NCPC\}$ [3]. Note however that this result describes the centralized case and does not hold for the non-centralized PCGS. Indeed, we note in Section 3.1 that non-centralized context-free PCGS are Turing complete, which also make non-centralized context-sensitive PCGS Turing complete (since context-free languages are also context sensitive).

It should be noted that PCGS with context-sensitive components rely on a computationally expensive model, which limits their usefulness. As with normal grammars, the most useful classes are the simple ones. Therefore, we are more interested in the PCGS with regular (REG, for short) or context-free (CF, for short) components.

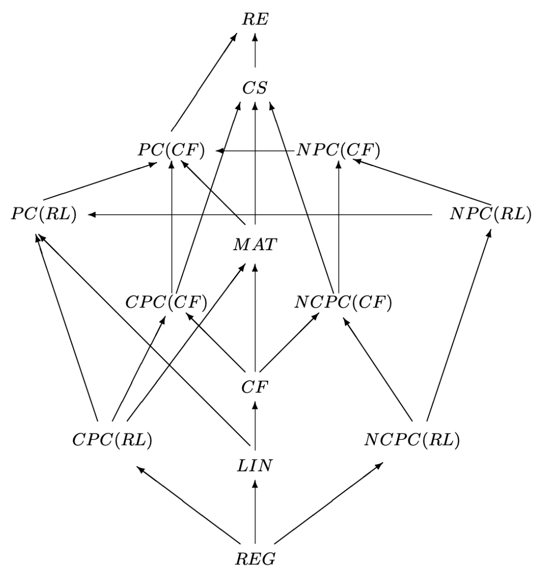

Figure 3.1: Hierarchies of PC families

Now recall the result below, which shows that the class of languages generated by a centralized returning PCGS with regular components is a subset of the class of languages generated by a non-centralized returning PCGS with regular components. In other words, a PCGS is generally more powerful than a single grammar component and the more communications a system has the more powerful the system is [17]:

$$CPC_n(REG) \subsetneq PC_n(REG), \ n > 1.$$

The idea also holds for PCGS with context-free components as following: [7]:

$$CPC_*(CF) \subseteq PC_*(CF).$$

However, the system may not be necessarily made more powerful only by increasing communication in this case.

Generally speaking, the centralized variant is a special case of a non-centralized PCGS. Indeed, the centralized qualifier limits the communication initiation to the first grammar in the system. Therefore, any languages generated by a centralized PCGS of any type can be generated by a non-centralized PCGS of the same type:

$$CPC_n(X) \subseteq PC_n(X), \ n \geq 1.$$

This indicates that the introduction of increasingly more powerful communication facilities is largely the reason why the generative power of a PCGS is greater than a single grammar component. Once these facilities are limited, the generative power is also limited.

The following two results further demonstrate that the generative power of PCGS is limited. When we have only two regular components, the languages generated by centralized or non-centralized PCGS are all context-free.

- $CPC_2(REG) \subsetneq CF$.
- $PC_2(REG) \subseteq CF$.

They has been proved in [3].

Another way to increase the generative power of a system is to increase the number of components in the system. We already mentioned that the generative capacity in the recursively enumerable case will not be changed and to some degree the same is true for the context-sensitive case. However if we investigate the classes lower in the hierarchy we can see that an increase in the number of components generally increases the generative capacity of the system [3]:

1. There exists a language generated by a PCGS with 2 or more regular components that cannot be generated by a linear grammar:
$$Y_n(REG) \setminus LIN \neq \emptyset,$$
for $n \geq 2,\ Y \in \{PC,\ CPC,\ NPC,\ NCPC\}$.

2. There exists a language generated by a PCGS with 3 or more regular components that cannot be generated by a context-free grammar:
$$Y_n(REG) \setminus CF \neq \emptyset,$$
for $n \geq 3,\ Y \in \{PC,\ CPC,\ NPC,\ NCPC\}$ (and $n \geq 2$ for non-returning PCGS).

3. There exists a language generated by a PCGS with 2 or more linear components that cannot be generated by a context-free grammar:
$$Y_n(REG) \setminus CF \neq \emptyset,$$
for $n \geq 2,\ Y \in \{PC,\ CPC,\ NPC,\ NCPC\}$.

4. There exists a language generated by a non-returning PCGS with 2 or more regular components that cannot be generated by a context-free grammar:
$$Y_n(REG) \setminus CF \neq \emptyset,$$
for $n \geq 2,\ Y \in \{NPC,\ NCPC\}$.

Clearly if the power of the components increases, then the power of the a PCGS will generally increase. This applies strictly to the centralized case for regular, linear and context-free components [3]:

$$CPC_n(REG) \subsetneq CPC_n(LIN) \subsetneq CPC_n(CF),\ n \geq 1.$$

For the non-centralized case, the same relationship would hold as well, but this needs to be investigated.

3.1 PCGS with Context-Free Components

We already mentioned the number of components as an important factor in the generative power of PCGS. It therefore makes sense to consider the hierarchies generated by this factor. Some of these hierarchies are in fact infinite, namely $CPC_n(REG)$ and $CPC_n(LIN)$, $n \geq 1$ [3]. Some other hierarchies however do collapse. A very notable example is the case of PCGS with context-free components, which was investigated in a series of papers, some times also related to size complexity. Collapsed to the class of recursively enumerable languages have been observed in all the synchronized, non-centralized cases.

It was also shown [6] that the set of recursively enumerable languages can be generated by non-returning PCGS with 6 context-free components through simulating a 2-counter machine. Further research on Turing completeness of non-returning PCGS with context-free components also exists [12]. Since the non-returning systems can be simulated by returning systems using grammars holding intermediate strings [8], the results mentioned above [6, 12] also work for returning systems, though not necessarily with the same maximum number of components.

Returning systems have also been considered directly. A well known result is that the non-centralized CF-PCGS hierarchy collapses at 11 components more specifically, $RE = PC_{11}CF = PC_*CF$ [4]. The PCGS developed in this paper is able to simulate an arbitrary 2-counter machine. It is shown in Figure 3.2. This upper bound on the number of components has subsequently been improved to 5 [5]. These bounds are substantially tighter that the ones that can be established using a non-returning to returning simulation [8].

However, all the results regarding the Turing completeness of returning context-free PCGS assumed what was later called a broadcast communication model [18]. In such a model a component that is being queried retains its original string throughout the communication step, that is, until all components requesting that string have received copies of it; only then is that component reset to its axiom. The original definition of a communication step (Definition 2.2) is to some degree ambiguous in this respect, but it appears nonetheless to imply a slightly different behaviour, that was named one-step communication. Under one-step communication the queried component returns to the axiom immediately after being communicated, regardless of the number of components that will subsequently request a copy of its string. The original results [4, 5] do not holds under the one-step communication model. It was shown however that the hierarchy still collapses, though not necessarily at such as low number of components as 11 or 5. In fact the Turing complete returning PCGS that was developed for the one-step communication model features 95 components [18]. It was developed as a direct simulation of the original, 11-component system used in the broadcast communication model [4]. It is possible (but not yet known) that a direct construction will reduce the number of components.

Regarding size complexity, we note that any recursively enumerable language can be generated by returning PCGS with context-free components if the number of nonterminals in the system is less than or equal to the natural number k [2].

3.2 Unsynchronized PCGS

As already mentioned earlier, little attention has been dedicated to unsynchronized PCGS. It is quite obvious that the family of languages generated by a PCGS family in unsynchronized mode is (not necessarily strictly) included in the family of languages generated by the same PCGS family in synchronized mode. Indeed, recall that any unsynchronized PCGS can be converted into an equivalent synchronized PCGS by the simple expedient of adding the set $\{A \to A : A \in N\}$ to the set of rewriting rules of each component. In other words, every unsynchronized PCGS can be simulated by a synchronized PCGS with components form the same grammar family.

Once the unsynchronized PCGS have been found weaker, they have been effectively ignored. Results on this matter are scarce and include the following [3]: The centralized case is very weak in some circumstances: $L(UCPC_*REG) = REG$, $L(UCPC_*LIN) = LIN$. On the other hand, $L(UNCPC_2REG)$ contains non-semilinear languages. It is also the case that $L(UPC_2REG) \setminus L(REG) \neq \emptyset$, but that $L(UPC_2REG) \subseteq L(CF)$, $L(UPC_2LIN) \setminus L(CF) \neq \emptyset$, and $L(UCPC_2CF) \setminus L(CF) \neq 0$.

Note that all the results mentioned above are very punctual and have not been extended in almost 20 years. Further note that to the best of our knowledge no results regarding non-centralized unsynchronized context-free PCGS exist.

$$
\begin{array}{lcl}
P_m & = & \{S \rightarrow [I], [I] \rightarrow C, C \rightarrow Q_{a_1}\} \cup \\
& & \{< I > \rightarrow [x, q, Z, Z, e_1, e_2] | (x, q_0, Z, Z, e_1, e_2, 0) \in R, x \in \Sigma\} \cup \\
& & \{< I > \rightarrow x[y, q, Z, Z, e_1, e_2] | (x, q_0, Z, Z, q, e_1, e_2, +1) \in R, x, y \in \Sigma\} \cup \\
& & \{< x, q, c_1', c_2', e_1', e_2' > \rightarrow [x, q', c_1, c_2, e_1, e_2] | (x, q, c_1, c_2, q', e_1, e_2, 0) \in R, \\
& & x \in \Sigma, c_1', c_2' \in \{Z, B\}, e_1', e_2' \in \{-1, 0, +1\}\} \cup \\
& & \{< x, q, c_1', c_2', e_1', e_2' > \rightarrow x[y, q', c_1, c_2, e_1, e_2], < x, q_F, c_1', c_2', e_1', e_2' > \rightarrow x | \\
& & (x, q, c_1, c_2, q', e_1, e_2, +1) \in R, c_1', c_2' \in \{Z, B\}, \\
& & e_1', e_2' \in \{-1, 0, +1\}, x, y \in \Sigma\}, \\
P_1^{c_1} & = & \{S_1 \rightarrow Q_m, S_1 \rightarrow Q_4^{c_1}, C \rightarrow Q_m\} \cup \\
& & \{[x, q, c_1, c_2, e_1, e_2] \rightarrow [e_1]', [+1]' \rightarrow AAC, [0]' \rightarrow AC, [-1]' \rightarrow C | \\
& & x \in \Sigma, q \in E, c_1, c_2 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \cup \\
& & \{[I] \rightarrow [I]', [I]' \rightarrow AC\}, \\
P_2^{c_1} & = & \{S_2 \rightarrow Q_m, S_2 \rightarrow Q_4^{c_1}, C \rightarrow Q_m, A \rightarrow A\} \cup \\
& & \{[x, q, Z, c_2, e_1, e_2] \rightarrow [x, q, Z, c_2, e_1, e_2], [I] \rightarrow [I] | x \in \Sigma, q \in E, \\
& & c_2 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \\
P_3^{c_1} & = & \{S_3 \rightarrow Q_m, S_3 \rightarrow Q_4^{c_1}, C \rightarrow Q_m\} \cup \\
& & \{[x, q, Z, c_2, e_1, e_2] \rightarrow a, [x, q, B, c_2, e_1, e_2] \rightarrow [x, q, B, c_2, e_1, e_2] \\
& & [I] \rightarrow [I] | x \in \Sigma, q \in E, c_2 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \\
P_4^{c_1} & = & \{S_4 \rightarrow S_4^{(1)}, S_4^{(1)} \rightarrow S_4^{(2)}, S_4^{(2)} \rightarrow Q_1^{c_1}, A \rightarrow a\} \\
P_1^{c_2} & = & \{S_1 \rightarrow Q_m, S_1 \rightarrow Q_4^{c_2}, C \rightarrow Q_m\} \cup \\
& & \{[x, q, c_1, c_2, e_1, e_2] \rightarrow [e_2]', [+1]' \rightarrow AAC, [0] \rightarrow AC, [-1] \rightarrow C | \\
& & x \in \Sigma, q \in E, c_1, c_2 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \cup \\
& & \{[I] \rightarrow [I]', [I]' \rightarrow AC\} \\
P_2^{c_2} & = & \{S_2 \rightarrow Q_m, S_2 \rightarrow Q_4^{c_2}, C \rightarrow Q_m, A \rightarrow A\} \cup \\
& & \{[x, q, c_1, Z, e_1, e_2] \rightarrow a, [x, q, c_1, B, e_1, e_2] \rightarrow [x, q, c_1, B, e_1, e_2], \\
& & [I] \rightarrow [I] | x \in \Sigma, q \in E, \\
& & c1 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \\
P_3^{c_2} & = & \{S_3 \rightarrow Q_m, S_3 \rightarrow Q_4^{C_2}, C \rightarrow Q_m\} \cup \\
& & \{[x, q, c_1, Z, e_1, e_2] \rightarrow a, [x, q, c_1, B, e_1, e_2] \rightarrow [x, q, c_1, B, e_1, e_2] \\
& & [I] \rightarrow [I] | x \in \Sigma, q \in E, c1 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \\
P_4^{c_2} & = & \{S_4 \rightarrow S_4^{(1)}, S_4^{(1)} \rightarrow S_4^{(2)}, S_4^{(2)} \rightarrow Q_1^{c_2}, A \rightarrow a\} \\
P_{a_1} & = & \{S \rightarrow Q_m, [I] \rightarrow < I >, [x, q, c_1, c_2, e_1, e_2] \rightarrow < x, q, c_1, c_2, e_1, e_2 >, \\
& & < x, q, c_1, c_2, e_1, e_2 > \rightarrow < x, q, c_1, c_2, e_1, e_2 >, , I > \rightarrow < I >, | x \in \Sigma, \\
& & q \in E, c_1, c_2 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \\
P_{a_2} & = & \{S \rightarrow S^3, S^{(1)} \rightarrow S^{(2)}, S^{(2)} \rightarrow S^{(3)}, , S^{(3)} \rightarrow S^{(4)}, \\
& & S^{(4)} \rightarrow Q_2^{c_1} Q_3^{c_1} Q_2^{c_2} Q_3^{c_2} S^{(1)}\}.
\end{array}
$$

Figure 3.2: A CF-PCGS that simulate a 2-counter machine

Chapter 4

All Languages Generated by Unsynchronized Context-Free PCGS Are Context Sensitive

In this chapter, we will study the computational complexity of unsynchronized context-free PCGS. We first show that in the absence of ε-rules (that is, rewriting rules of the form $A \to \varepsilon$) languages generated by unsynchronized context-free PCGS can be recognized by nondeterministic Turing machines using $O(|w|)$ tape cells for each input instance w. We adapt for this purpose an earlier construction used (unsuccessfully) for synchronized non-returning PCGS [1]. Later in the chapter we adapt this proof to handle ε-rules.

4.1 Unsynchronized PCGS with no ε-rules

Throughout this section we allow no ε-rules in any of our PCGS. We will let this kind of rules back in in the next section.

Definition 4.1. NON-DIRECT-SIGNIFICANT COMPONENTS: *During a derivation process in PCGS, a component x_i of the current configuration is called non-direct-significant for the generation of the string w if either:*

(i) $i \neq 1$ *and the respective component is not queried, or*

(ii) $i = 1$ *and the derivation from x_1 to w in G_1 cannot end successfully unless x_1 is reduced to the axiom sometime in the future, or*

(iii) $i \neq 1$ *and x_i is queried by x_j, $j \neq i$, and then x_j become non-direct-significant.*

All the others components are called direct-significant. Any component which is reduced to the axiom becomes direct-significant even if it was not so before.

In other words, a non-direct-significant component of a PCGS cannot directly participate at a successful derivation. It can only produce lateral effects (by queries which can modify other components) or block the derivation (by circular queries or by featuring nonterminals for which there are no applicable rewriting rules). Note however that none of these blocking scenarios apply to unsynchronized systems. Indeed, in unsynchronized systems each grammar can choose to either rewrite or wait in any step which is not a communication step. For example, if one grammar has no rules that can be used in a configuration, then this grammar can choose to wait and the overall derivation can still proceed successfully. Similarly, if a component introduces a query symbol that would generate a circular query (in conjunction with other query symbols in other components), then that component can equally choose not to apply the rule that introduces the problematic query symbol, so that no circular query happens.

Definition 4.1 introduces a class of components for which the structure is irrelevant for the derivation. Therefore, these components can be removed as long as the information relevant for lateral effects is kept. Note that in a returning system a non-direct-significant component may become direct-significant in the future, but that can only happen when the respective component is reduced to the axiom, which is done with no regard of the structure of that component.

Lemma 4.1. *Let $\Gamma = (N,\ K,\ T,\ G_1,\ \ldots,\ G_n)$ be an unsynchronized returning context-free PCGS without ε-rules, and let w be a string. Let also $(x_1,\ \ldots,\ x_n)$ be a configuration of the system. Then, if the length of a component x_i becomes greater than $|w|$, that component becomes non-direct-significant for the generation of w.*

Proof. We consider two situations:

(i) For $i = 1$. If $|x_1|_K = 0$, x_1 will be rewrited using the rules of G_1. In addition, since these are context-free rules and there are no ε-rules, the length of x_1 does not decrease. If $|x_1|_K \neq 0$, a communication step will be performed. Since there are not empty components to be queried (there are no ε-rules to generate them), the communication step does not reduce the length of the component. In all, the length of x_1 cannot decrease and therefore can never become w. The only way for x_1 to become w is for the first component to be queried and so reduced to the axiom. Therefore x_1 is non-direct-significant according to the Definition 4.1.

(ii) For $i \geq 2$ we have three situations:

   (a) The component x_i is never queried, therefore it is non-direct-significant.

   (b) x_i is queried by the first component, case in which the length of x_1 becomes greater than $|w|$. Then x_1 becomes non-direct-significant (according to point (i) above), and so x_i is non-direct-significant.

(c) x_i is queried by another component x_j, $j \neq i$, $j \neq 1$, which become in that way longer than w and also can not decrease unless it is reduced to the axiom.

□

Corollary 4.2. *Let* $\Gamma = (N,\ K,\ T,\ G_1,\ \ldots,\ G_n)$ *be an unsynchronized non-returning context-free PCGS without* ε*-rules, and let* w *be a string. Let* $(x_1,\ \ldots,\ x_n)$ *be a configuration of the system. Then, if the length of a component* x_i *becomes greater than* $|w|$*, that component becomes non-direct-significant for the generation of* w*.*

Proof. The same considerations as in the proof of Lemma 4.1 apply. The only difference is that no component is ever reduced to the axiom (since the system is non-returning). Removing any reference to the operation of returning a component to its axiom from the previous proof therefore results trivially in a valid proof for the non-returning case. □

Theorem 4.3. *Let* Γ *be an unsynchronized (returning or non-returning) PCGS with* n *context-free components* $(n \geq 1)$ *and no* ε*-rules. Then there is a Turing machine* M *that recognizes the language* $L(\Gamma)$ *using at most* $O(|w|)$ *amount of work tape space for each input instance* w*.*

Proof. Let $\Gamma = (N,\ K,\ T,\ G_1,\ \ldots,\ G_n)$ be an unsynchronized returning PCGS, where $G_i = (N \cup K,\ T,\ P_i,\ S_i)$, $1 \leq i \leq n$, are context-free grammars without ε-rules. We will construct a nondeterministic Turing machine M which recognizes $L(\Gamma)$.

M will be a standard Turing machine, with a work tape equipped with a read/write-head. Given an input string $w \in T^*$, M will simulate step by step the derivation of Γ that produces w. The alphabet of the tape of M is $N \cup K \cup T \cup \{@\}$, where $@ \notin N \cup K \cup T$.

In what follows we will refer to Lemma 4.1, and thus establish the proof for the returning case. This will also establish the proof for the non-returning case as long as all the references to Lemma 4.1 are replaced to references to Corollary 4.2.

According to the Definition 2.2, there are two types of derivation steps to simulate: the component-wise rewriting and the communication. M will keep on its tape the current configuration $(x_1, x_2, \ldots, x_n)$ and will work on it as follows:

(i) If $|x_i|_K = 0$ for all i, $1 \leq i \leq n$, M simulate rewriting for each component x_i, $1 \leq i \leq n$. If $|x_i|_N > 0$, M can choose to either leave x_i unchanged or rewrite x_i by using a rule selected from the rule set P_i. If there is some i for which $|x_i|_N = 0$ or no suitable rule exists to rewrite x_i, then that component remains unchanged.

If $|x_i| > |w|$ then, according with Lemma 4.1, x_i become non-direct-significant. Therefore its structure is irrelevant and it will be replaced by the string

$$@N_1 \ldots N_j Q_1 \ldots Q_k, \tag{4.1}$$

where @ is a special symbol ($@ \notin N \cup T \cup K$), $N_1, \ldots, N_j$ are exactly all the distinct nonterminals in x_i and $Q_1, \ldots, Q_k$ are exactly all (not necessarily distinct) query symbols in x_i.

Rewriting strings of form (4.1) works as follows: Let $k = 0$ (no query symbols in the string) and the rewriting rule to be applied be

$$A \to \alpha_1 A_1 \alpha_2 A_2 \ldots \alpha_m A_m \alpha_{m+1},$$

where $A \in N$, $A_1, \ldots, A_m \in N \cup K$ and $\alpha_1, \ldots, \alpha_{m+1} \in T^*$. Then, there is $N_r = A$, for some $1 \le r \le j$ (otherwise the rule is not applicable). For each k, $1 \le k \le j$, if A_k does not already exists in x_i then it is added to x_i. Since this is an unsynchronized system, each grammar can choose to wait for an arbitrary amount of time. In fact, the only addition to unsynchronized systems over a synchronized system is the possibility of waiting. Therefore, we can assume that each nonterminal appears infinitely many times without modifying the overall language generated by the system. A detailed argument on this matter can be found below.

(ii) If there are query symbols in the current configuration, then M simulates communication steps. If there are circular queries, M rejects the input and halts. Otherwise, M nondeterministicaly selects a component x_i for which Q_j, $1 \le j \le q$, are all the query symbols and $|x_j|_K = 0$, $1 \le j \le q$. M sequentially replaces Q_j by x_j. If either the current x_j is of the form (4.1) or x_i becomes longer than $|w|$ after replacement, then x_i becomes non-direct-significant, so it will be replaced by a string of the form (4.1).

The same mechanism also applies to non-direct-significant components, with the addition that after the completion of the communication step all terminals are removed, all duplicate nonterminals are also removed, and the nonterminals and query symbols are re-ordered so that the form (4.1) is maintained.

This communication step is repeatedly performed until there are no query symbols in the current configuration. The components that have been communicated are reduced to their respective axiom whenever applicable (that is, whenever the system is returning).

M repeats steps (i) and (ii) as applicable until w and x_1 are identical, case in which M accepts the input and halts.

In the case of a returning system when a component x_i is queried, M has to simulate the returning of x_i to the axiom. This is done by replacing x_i by the axiom of its grammar (S_i). Note that this replacement dose not depend of the form of x_i so the processing of non-direct-significant components is correct (note however that those components become direct-significant and so are no longer stored in the form (4.1)). Obviously this discussion does not apply to non-returning systems, where a non-direct-significant component can never become direct-significant again.

Note that nonterminal symbols are never removed from a non-direct-significant component. This might appear to pose problems in the following circumstances: The continuation of derivation from a nonterminal A being rewritten into another nonterminal B would be different from the derivation from A (such as by B blocking further rewriting if contrast with A that would allow the derivation to continue). In a direct-significant component, the occurrence of A is replaced by B and A disappears, but in non-direct-significant component, A and B will both remain in the string after the occurrence of A was "replaced" by B.

We argue that the continuing presence of A does not affect the possible derivations given that in an unsynchronized system A is not required to rewrite itself into B at any fixed time; we can just not apply the respective rule for an arbitrary number of rewriting steps. Additionally, the nature of the component grammars (that are all context free) ensures that there is no possible rewriting of A and B together that is not a combination of rewriting A and B individually. True, the derivation in the corresponding component would block if A were not present. However this does not have any influence in the overall derivation, since in this case the blocked component can also just wait as much as necessary for the other components to perform their derivations. In other words, the presence of a blocking nonterminal in a component does not cause blocking in the overall derivation. Therefore, the fact that A is still present is irrelevant. Overall, in non-direct-significant components, two nonterminals appearing sequentially one after the other has the same effect as the two nonterminals appearing simultaneously.

Now let us count the amount of work space used by M during the derivation. Each component is either direct-significant (case in which we need $O(|w|)$ space to store it), or it is not. We have a fixed, finite number t of nonterminals for any given PCGS and exactly n query symbols. Any query symbol that is introduced in a component must be immediately eliminated by performing a communication step, and so the maximum number of query symbols that can appear simultaneously in a string is again fixed for a given PCGS and is no larger than the size m of the longest rewriting rule in that PCGS. Therefore the length of a non-direct-significant component is independent of $|w|$ and is less than $1 + (t + m)$.

A communication step may use temporarily an amount of tape space double than the space used by a single component temporarily. For example, a string of length $|w|$ is queried by another string of length $|w|$; before the reduction to form (4.1) we have to use $2|w|$ tape cells. Therefore, the number of cells used by a component is less than $2 \times \max(|w|,\ 1 + (t + m))$. We also need some extra space on the tape to keep the rules and possibly other information about the PCGS being simulated, whose size we denote by ρ. In all, the space used by M is upper-bounded by $2 \times n \max(|w|,\ 1 + (t + m)) + \rho$. However, neither of t, m, or ρ are $|w|$-dependent, so the overall space used is linear in $|w|$, as desired. □

Corollary 4.4. *All languages generated by unsynchronized (returning or non-returning) PCGS with context-free components and no ε-rules are context sensitive.*

Proof. This result follows directly from Theorem 4.3 given that all context-sensitive languages can be accepted in linear space [16]. □

4.2 Handling ε-rules

Handling ε-rules in the same space constraints is easy once the following observation is made: Let G be a context-free grammar and let n be a positive integer. Suppose that we have a derivation in G that goes from a string of linear length to a string of a superlinear length and then back to linear length. Then there always exists an alternate derivation in G (starting and ending with the same string as the original derivation) that does not use any intermediate string of superlinear length. Formally:

Lemma 4.5. *Let G be context-free grammar and let $n \geq 0$. Suppose that there exists a derivation $x_s \Rightarrow_G^* x_b \Rightarrow_G^* x_d$, where $|x_s| = O(n)$, $|x_b|$ is superlinear in n, and $|x_d| = O(n)$. Then there always exists an alternate derivation $x_s \Rightarrow_G^* x_d$ such that the length of all string in that derivation is $O(n)$.*

Proof. For convenience we use "nonterminal" and "terminal" to denote not just the symbol itself but also an occurrence of that symbol (which is which should be clear from the context).

The string x_b is superlinear, because it contains either a superlinear number of terminals or a superlinear number of nonterminals (or both). In the former case the length of the string can never become sublinear later and then x_d does not exist, so the claim is vacuously true. Thus it must be the case that the number of terminals in x_b is linear (and so the number of nonterminals is superlinear).

Let then $|x_b|_T = f(n)$ and $|x_b|_N = g(n)$ with $f(n) = O(n)$ and $g(n)$ a superlinear function. The terminals are introduced by at most $g(n)$ rewritings (applications of some rewriting rule), which necessarily introduces $O(g(n))$ nonterminals. It follows that a superlinear number $f(n) - g(n)$ of nonterminals are introduced by rules that only introduce nonterminals. A superlinear number of these nonterminals are then erased (using ε-rules), so they do not play any role in the generation of x_d. If so, then we can simply not apply those rules (which in the end do not have any effect anyway). Omitting these rules results in a derivation that produces the same string (in a possibly lower number of steps) and never encounters any intermediate string of superlinear size. □

Obviously Lemma 4.5 is equally applicable to any component-wise derivation in some component of an unsynchronized context-free PCGS. The fact that the PCGS is unsynchronized allows the replacement of any derivation with any equivalent derivation without changing the overall string that is eventually generated by that PCGS.

Theorem 4.6. *Let Γ be an unsynchronized (returning or non-returning) PCGS with n context-free components ($n \geq 1$) possibly featuring ε-rules. Then there is a Turing machine M that recognized the language $L(\Gamma)$ using at most $O(|w|)$ amount of work tape space for each input instance w.*

Proof. As before let $\Gamma = (N,\ K,\ T,\ G_1,\ \ldots,\ G_n)$ be an unsynchronized non-returning PCGS, where $G_i = (N \cup K,\ T,\ P_i,\ S_i)$, $1 \leq i \leq n$, are context-free grammars.

M is once more a standard Turing machine, with a work tape equipped with a read/write-head. Given an input string w, M will simulate step by step the derivation of w by Γ. The alphabet of the tape of M is $N \cup K \cup T \cup \{@\}$, where $@ \notin N \cup K \cup T$.

We essentially reproduce the proof of Theorem 4.3, with all the changes that are needed to handle ε-rules (which is basically just a matter of using Lemma 4.5 in all the appropriated places). In particular, we continue to refer to Lemma 4.1 and thus establish the proof for the returning case, which will also establish the proof for the non-returning case as long as all the references to Lemma 4.1 are replaced to references to Corollary 4.2. For completeness, we reproduce the whole proof, though is an abbreviated manner.

The machine M keeps on its work tape the current configuration $(x_1, \ldots, x_n)$ of Γ and keeps changing it as before:

(i) If $|x_i|_K = 0$ for all i, $1 \leq i \leq n$, M simulate rewriting for each component x_i, $1 \leq i \leq n$. If $|x_i|_N > 0$, M can choose to either remains unchanged or rewrite x_i by using a rule selected from the rule set P_i. If there are some i for which $|x_i|_N = 0$ or a suitable rule does not exist, then M will remain unchanged. So far we have not placed any limitation on the use of ε-rules; they can be applied as desired (or needed).

If $|x_i| > |w|$ then, according with Lemma 4.5, x_i become non-direct-significant and so its representation will be replaced with a string of the form (4.1) as shown on page 21.

The rewriting of the strings of form (4.1) proceeds as in the previous proof. Specifically, suppose we apply the rule

$$A \rightarrow \alpha_1 A_1 \alpha_2 A_2 \ldots \alpha_m A_m \alpha_{m+1}$$

where $A \in N$, A_1, $\ldots$, $A_m \in N \cup K$ and α_1, $\ldots$, $\alpha_{m+1} \in T^*$. Then, there is $N_r = A$, $1 \leq r \leq j$, (if not, the rule is not applicable). For all $1 \leq k \leq j$, if A_k does not already exists in x_i, then it is added to x_i.

We note again that no nonterminal is ever removed from the representation of form (4.1). Ignoring (for the moment) ε-rules the same argument as of why this does not affect the overall result of the derivation applies.

Now it is time to consider ε-rules. We still have not restricted their use, but interestingly enough it is easy to see that these rules do not have any effect on a non-direct-significant component. Indeed, no nonterminal is ever removed from a string represented in form (4.1), which is actually the only thing an ε-rule ever does. Note that keeping all the nonterminals in a non-direct-significant component does not affect the overall derivation as argued on page 22 in the proof of Theorem 4.3. Thus it continues to be the case that the only way for a non-direct-significant component to become direct-significant is by its reduction to axiom as a consequence of a query (see below).

We are able to argue however that (effectively) not using ε-rules in non-direct-significant components does not affect the overall derivation as follows: Using or not using these rules can only be meaningful when their use transforms a non-direct-significant component back into a direct-significant one. Indeed, the other use of these rules can be explained away by the argument that we refereed to in the paragraph above (that not eliminating a nonterminal in a non-direct-significant component does not affect the overall derivation, see page 22). According to Lemma 4.5 however, it is never necessary to go from a non-direct-significant back to a direct-significant component. Indeed, any derivation that originally goes this route can according to the lemma be replaced by an equivalent derivation that never encounters any non-direct-significant (that is, superlinear) form of the respective component. We can therefore let this derivation continue in the superlinear territory and never come back (and presumably fail), since the nondeterministic nature of M ensure that the alternate derivation guaranteed by Lemma 4.5 will also be explored (and possibly succeed). That is, we ignore ε-rules in components of form (4.1) without loss of generality, since alternate derivations are always guaranteed by Lemma 4.5.

(ii) If there are query symbols in the current configuration, then M simulates communication steps. If there are circular queries, M rejects the input and halts. Otherwise, M nondeterministicaly selects a component x_i for which Q_j, $1 \leq j \leq q$, are all the query symbols and $|x_j|_K = 0$, $1 \leq j \leq q$. M sequentially replaces Q_j by x_j. If either the current x_j is of the form (4.1) or x_i becomes longer than $|w|$ after replacement, then x_i becomes non-direct-significant, so it will be replaced by a string of the form (4.1). This communication step is repeatedly performed until there are no query symbols in the current configuration. The queried components are reduced to their respective axiom as applicable (that is, if Γ is returning).

M repeats steps (i) and (ii) until w and x_1 are identical, case in which M accepts the input and halts.

It is immediate that our machine works in exactly the same manner as the machine used in the proof of Theorem 4.3. It follows that the space requirements are identical and so upper bounded by $2 \times n \max\left(|w|,\ 1 + (t + m)\right) + \rho$, where neither of t, n, or ρ are $|w|$-dependent. That is, the overall space used is again linear in $|w|$, as desired. □

Corollary 4.7. *All languages generated by unsynchronized (returning or non-returning) PCGS with context-free components in the present of ε-rules are context sensitive.*

Proof. Again this result follows directly from Theorem 4.6 given that all context-sensitive languages can be accepted in linear space [16]. □

Chapter 5

Conclusion

PCGS introduces an inherently concurrent model in the discipline of formal languages. Due to this inherent parallelism, to exploit this model in general (and context-free PCGS in particular) in formal methods is one of our longer-term interest. It is necessary to address several formal language questions before applying it into our studies, the generative power of context-free PCGS being one of them. Indeed, this is a particularly important question that affects many aspects of any PCGS-based specification formalism including expressiveness and also algorithm complexity.

From the previous results on PCGS we know the expressiveness of synchronized context-free PCGS, which all turn out to be Turing complete [4, 18]. However, there are no discussions about the unsynchronized context-free PCGS. We attempted here to remedy this oversight.

First, we gave a proof that the language generated by unsynchronized context-free PCGS in the absence of ε-rules are context sensitive, being recognizable by nondeterministic Turing machines using $O(|w|)$ tape cells for each input instance w (Theorem 4.3 and Corollary 4.4). We adapted for this purpose an earlier construction [1]. The earlier use of this construction failed because the non-direct-significant components can affect the derivation through various side effects including nonterminals that cannot be overwritten and thus block the derivation, as well as nonterminals that introduce circular queries. Neither of these side effects can happen in an unsynchronized system, simply because each grammar has the liberty of choosing at any step that is not a communication step to either rewrite or wait. In other words, if a component contains nonterminals that cannot be rewritten, it can choose to wait to let other grammars rewrite, and so the derivation will no longer be blocked. Therefore, the construction that failed in the synchronized case succeeds in the unsynchronized case. We are thus able to base our proof on representing long (non-direct-significant) components in a compact manner that reduces the overall space requires to simulate the system to linear without affecting the overall result.

We were then able to handle ε-rules in our simulation with virtually no change in the representation and algorithm (Theorem 4.6 and Corollary 4.7). We continue to use the same concept of non-direct-significant components together with the same representation of these components. The only potential problem is that some component can become non-direct-significant and then direct-significant again through rewriting. This is a problem because the conversion from direct-significant to our compact representation of non-direct-significant components necessarily looses information, so that the conversion the other way around is no longer possible. However, we show that such a transition (from non-direct-significant to direct-significant) is actually unnecessary by showing that any such a derivation has an equivalent derivation that never touches the non-direct-significant territory (Lemma 4.5, the crux of handling ε-rules).

In all, we showed in this paper that the languages generated by all the families of unsynchronized PCGS with context-free components can be accepted in linear space and so they are all context sensitive:

Theorem 5.1. *Any language generated by an unsynchronized PCGS with context-free components can be accepted by a linear space-bounded Truing machine. Therefore all languages generated by unsynchronized PCGS with context-free components are context-sensitive.*

Proof. Immediate from Theorem 4.3, Corollary 4.4, Theorem 4.6, and Corollary 4.7. □

Bibliography

[1] S. D. BRUDA AND M. S. R. WILKIN, *Limitations of coverability trees for context-free parallel communicating grammar systems and why these grammar systems are not linear space*, Parallel Processing Letters, 26 (2016), p. 1650012.

[2] E. CSUHAJ-VARJÚ, *On size complexity of context-free returning parallel communicating grammar systems*, in Where Mathematics, Computer Science, Linguistics and Biology Meet, C. Martin-Vide and V. Mitrana, eds., Springer, 2001, pp. 37–49.

[3] E. CSUHAJ-VARJÚ, J. DASSOW, J. KELEMEN, AND G. PAUN, *Grammar systems: a Grammatical Approach to Distribution and Cooperation*, Gordon and Breach Science Publishers S.A., 1994.

[4] E. CSUHAJ-VARJÚ AND G. VASZIL, *On the computational completeness of context-free parallel communicating grammar systems*, Theoretical Computer Science, 215 (1999), pp. 349–358.

[5] E. CSUHAJ-VARJÚ, G. PAUN, AND G. Vaszil, *PC Grammar Systems with five Context-Free Components Generate all Recursively enumerable Languages*, Theoretical Computer Science, 299 (2003), pp. 785–794.

[6] E. CSUHAJ-VARJÚ AND G. VASZIL, *On the size complexity of non-returning context-free PC grammar systems*, in 11th International Workshop on Descriptional Complexity of Formal Systems (DCFS 2009), 2009, pp. 91–100.

[7] J. DASSOW, G. PAUN, AND G. ROZENBERG, *Grammar systems*, in Handbook of Formal Languages – Volume 2: Linear Modeling: Background and Applications, Springer, 1997, pp. 155–213.

[8] S. DUMITRESCU, *Non-returning PC grammar systems can be simulated by returning systems*, Theoretical Computer Science, 165 (1996), pp. 463–474.

[9] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability A Guide to the Theory of NP-Completeness*, Macmillan Higher Education, 1979.

[10] G. KATSIRELOS, S. MANETH, N. NARODYTSKA, AND T. WALSH, *Restricted global grammar constraints*, in Principles and Practice of Constraint Programming (CP 2009), vol. 5732 of Lecture Notes in Computer Science, 2009, pp. 501–508.

[11] H. R. LEWIS AND C. H. PAPADIMITRIOU, *Elements of the Theory of Computation*, Prentice Hall, 2nd ed., 1998.

[12] N. MANDACHE, *On the computational power of context-free PCGS*, Theoretical Computer Science, 237 (2000), pp. 135–148.

[13] V. MIHALACHE, *On the generative capacity of parallel communicating grammar systems with regular components*, tech. rep., Turku Centre for Computer Science, Turku, Finland, 1996.

[14] D. PARDUBSKA AND M. PLATEK, *Parallel communicating grammar systems and analysis by reduction by restarting automata*, tech. rep., Department of Computer Science, Comenius University, Bratislava, Slovakia, 2008.

[15] G. PAUN AND L. SANTEAN, *Parallel communicating grammar systems: the regular case*, Analele Universitatii din Bucuresti, Seria Matematica-Informatica, 2 (1989), pp. 55–63.

[16] J. ROTHE, *Complexity theory and cryptology, Texts in Theoretical Computer Science*, An EATCS Series, Springer, 2005.

[17] L. SANTEAN, *Parallel communicating grammar systems*, Bulletin of the EATCS (Formal Language Theory Column), 1 (1990).

[18] M. S. R. WILKIN AND S. D. BRUDA, *Parallel communicating grammar systems with context-free components are Turing complete for any communication model*, Acta Universitatis Sapientiae, Informatica, 8:2 (2016), pp. 113–170.