On the Expressiveness of Unsynchronized Parallel Communicating Grammar Systems with Regular Components

by

Afshin Bahrampour

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 2024

Abstract

Parallel communicating grammar systems (PCGS) were introduces as a formal language model for parallel and concurrent computation. A PCGS is a set of formal grammars working together on their own strings and communicating by transferring their work strings between each other. A PCGS can be synchronized (when all the components grammars rewrite their strings simultaneously) or unsynchronized (when at any step a component can choose to rewrite its string or wait). Synchronization plays a significant role in the expressiveness of a PCGS. Indeed, synchronized PCGS with context-free components were found to be Turing complete, in contrast to the unsynchronized variant is (not necessarily striclty) weaker than context-sensitive grammars.

We focus in this thesis on PCGS with regular components and more precisely on the weaker, unsynchronized variant. Specifically, We investigate the capability of unsynchronized regular PCGS to generate context-free languages. We find that we can produce many constructs found in context-free languages. However, we also find that certain context-free constructs cannot be generated by using this setup. Consequently, we show that unsynchronized PCGS with regular components lack the full computational ability to generate all context-free languages. In essence, while unsynchronized PCGS with regular components offer promise in modelling the behavior of context-free languages, they ultimately fall short in capturing all of them. This contributes to the effort of understanding the computational capabilities of different grammar systems.

Acknowledgments

I would like to thank my supervisor, Dr. Stefan D. Bruda, for his help in developing the study topics and methodology. His opinion was always beneficial and allowed me to think beyond the box.

In addition, I would like to thank all of the other distinguished teachers in Bishop's University's Department of Computer Science, including Dr. Mohammed Ayoub Aloui Mhamd, from whom I gained valuable knowledge and insights.

I would want to thank my wife, Narges, for her continuous support and presence throughout the journey, which has been a constant source of strength, comprehending, and motivation, regardless of the problems we faced.

Last but not least, I would like to express my gratitude to my parents for their constant support and affection, as well as for believing in me when I didn't believe in myself.

Contents

1 Introduction **1**

2 Preliminaries **3**
- 2.1 Grammars . . . 3
- 2.2 Parallel Communicating Grammar Systems . . . 5
- 2.3 Examples . . . 7

3 Previous Work **13**
- 3.1 Context-Free PCGS are Turing complete . . . 16
- 3.2 Unsynchronized PCGS . . . 16
- 3.3 Recent Investigations on Unsynchronized Context-Free PCGS . . . 18

4 Unsynchronized Regular PCGS and Context-Free Languages **20**
- 4.1 Balanced Symbols . . . 20
- 4.2 Balanced Symbols with Intermediate Content . . . 22
- 4.3 Palindromes . . . 23
- 4.4 Embedded Matching . . . 24
- 4.5 Sequences of Symbols . . . 24
- 4.6 On Generating all the Context-Free Languages . . . 25

5 Conclusions **31**

Bibliography **33**

Chapter 1

Introduction

The study of computing systems that can process information simultaneously has been a primary focus of theoretical computer science for a long time. Among the several frameworks developed to explore the characteristics and capabilities of parallel processing, Parallel Communicating Grammar Systems (PCGS) are distinguished by their unique method of combining parallelism and communication. This thesis delves into the domain of unsynchronized PCGS, particularly those employing regular components, to determine their generative power in comparison to the classical context-free grammars (CFGs).

In a PCGS, communication between grammars can be categorized as either returning or non-returning. In a returning system, after fulfilling a communication request, the queried component replaces its string with the corresponding axiom and proceeds the derivation process from that point. Conversely, in a non-returning system, the component's string remains unchanged following a communication event, allowing the subsequent derivation to continue rewriting the same string. The status of the grammar that sent its current string depends on the type of PCGS being considered. In a non-returning PCGS, the grammar continues working without modifying its string, whereas in a returning PCGS, the grammar erases its current string and resumes the derivation process from the axiom.

PCGS can also be divided into two types: synchronised and unsynchronized. During a rewriting step in a synchronised PCGS, each component must synchronously, apply a rewriting rule. The sole exception is when the related string is entirely composed of terminals, in which case nothing happens with that string during a rewriting step. Otherwise, if some component lacks sufficient rewriting rules, the derivation will block. In unsynchronized PCGS, each grammar has the option of doing a rewriting step or waiting. 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. Synchronized PCGS have received sustained attention through the years. 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.

The synchronized PCGS with context-free components are very powerful. They are known to be Turing complete (that is, much more powerful than the components) [3, 4]. The unsynchronized variant is as expected weaker, being less expressive than context-sensitive grammars [11] but suspected to the equivalent to these grammars being able to model many context-sensitive constructs [1].

As illustrated above people have focused extensively on the more glamorous context-free components, so we thought to characterize the humbler but nonetheless interesting PCGS with regular components for a change. While context-free grammars (CFG) are known for their ability to expressively define a wide variety of language constructions that regular grammars cannot handle (such as nested structures and recursive patterns, which are crucial in programming language design and natural language processing), regular grammars are well-known for their simplicity and efficiency.

We are therefore wondering whether we can put together regular grammars in an unsynchronized PCGS to generate any context-free languages. The answer turns out to be not quite. In fact we find that the relationship between unsynchronized PCGS with regular components and context-free grammars is surprisingly similar with the relationship between unsynchronized PCGS with context-free components and context-sensitive grammars. Indeed, on one hand we find that unsynchronized regular PCGS can generate many typical context-free constructs. On the other hand, we show that these PCGS cannot generate all the context-free languages.

On the other hand, our investigation reveals fundamental limits in unsynchronized regular PCGS. Despite their ability to simulate certain context-free constructs, we show that not all context-free languages can be generated by these systems. We highlight the inherent computational constraints of unsynchronized PCGS when faced with the complexity of certain context-free constructs. This thesis not only demonstrates the ability of unsynchronized PCGS with regular components to model context-free constructs, but it also identifies the limits of their expressiveness.

Considering our results compared with the aforementioned results regarding unsynchronized context-free PCGS we are tempted to speculate that the languages generated by unsynchronized regular PCGS are all context free. On the other hand, we expect that unsynchronized context-free PCGS will be found to be strictly weaker than the context-sensitive grammars. We hope that our approach will be useful for showing the latter result, though it is quite possible that the opposite will turn out to be true (and unsynchronized context-free PCGS will turn out to be equivalent to context-sensitive grammars). Obviously this is mere speculation, subject to future investigations.

Chapter 2

Preliminaries

We start by revisiting some fundamental concepts related to grammars, and then define parallel communicating grammar systems, our main focus. Basically a grammar rewrites a string; by contrast, a parallel grammar system consists of a set of grammars that rewrite in parallel a set of strings in a cooperative fashion (communicating and possibly synchronizing the rewriting processes). The sequencing and criteria of coordination and communication are determined by the cooperative strategy among the component grammars [2].

2.1 Grammars

A grammar is a tuple $G = (N, T, S, P)$, where N is the set of non-terminal symbols, T is the set of terminal symbols, P is a set of rewriting rules (or productions or just rules), and $S \in N$ is the start symbol or axiom. A rule $\alpha \rightarrow \beta \in P$ shows that a substring α can be replaced by β. A rewriting step is the rewriting of the string ω into a string $\omega^{'}$ according to the rules of the grammar, which is written $\omega \Rightarrow \omega^{'}$. Formally, $\omega \Rightarrow \omega^{'}$ iff $\omega = u\alpha v$, there exists a rule $\alpha \rightarrow \beta \in P$, and $\omega' = u\beta v$ for some strings $u, v \in (N \cup T)^*$.

A derivation from ω to $\omega^{'}$ is a chained sequence of rewriting steps which is written $\omega \Rightarrow^* \omega^{'}$. The language generated by a grammar consists of exactly all the strings ω such that $S \Rightarrow^* \omega$ and $|\omega|_N = 0$ that is, all the terminal strings that can be generated by derivations that start from the axiom of the grammar. 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 [6].

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

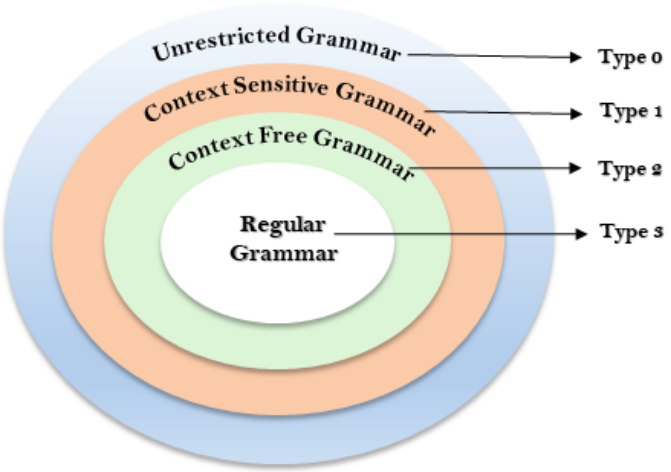

Figure 2.1: The Chomsky hierarchy

1. A type-0 or unrestricted grammar, referred to as G, is characterized by the absence of any restrictions. This kind of grammar has the capability to generate languages that are semi-decidable by Turing machines. The languages produced by a type-0 grammar are known as recursively enumerable languages or RE for short. [10].

2. A grammar G is considered type-1 or context-sensitive grammar, if $|\alpha| \leq |\beta|$ for every rewriting rule $\alpha \rightarrow \beta$ in P. This type of grammar can have a rewriting rule of the form $S \rightarrow \varepsilon$, but only if S is not in the right-hand side of any rewriting rule. Languages generated by a type-1 grammar are called context sensitive, or CS for short [10].

3. A grammar G is classified as type-2 or context-free or CF for short if for every rewriting rule $\alpha \rightarrow \beta$ in P we have $|\alpha| = 1$, indicating that α is a single non-terminal symbol. Within this category, linear grammars represent a specific sub-type. In linear grammars, no rewriting rule is permitted to have more than one non-terminal symbol in its right-hand side. Meanwhile, languages generated by the linear grammar sub-type are referred to as Linear, or LIN. [8, 9].

4. A grammar G is considered type-3 or regular if its rewriting rules are one of these specific forms: $A \rightarrow cB$, $A \rightarrow c$, $A \rightarrow \varepsilon$, or $A \rightarrow B$. In these forms, A, B represent non-terminals, while c is a terminal symbol. Languages generated by a type-3 grammar are called regular, or REG for short.

The types of grammars described above form a hierarchy as shown in Figure 2.1. REG is the smallest and RE the largest class.

2.2 Parallel Communicating Grammar Systems

A parallel communicating grammar system (PCGS for short) [14] provides a theoretical construct that combines the concepts of grammar with parallelism and communication. The idea behind parallel communicating grammar systems (PCGS) is the notion of multiple grammars that work together in parallel, communicate with each other, and generate strings. This concept supports the investigation of the language-theoretic properties of parallel systems.

Definition 2.1. PARALLEL COMMUNICATING GRAMMAR SYSTEM [2]: *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 is a non-terminal and T a termina alphabett, K is the set of query symbol with $K = \{Q_1,\ \ldots,\ Q_n\}$. The sets N, T and K are mutually disjoint. Each G_i, $1 \leq i \leq n$ is the 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 the components of the system. Each query symbol Q_i in K points to the component G_i, $1 \leq i \leq n$.

A PCGS derivations consists 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.

Definition 2.2. DERIVATION IN A PCGS [2]: *Let $\Gamma = (N,\ T,\ K,\ G_1,\ \ldots,\ G_n)$ be a PCGS as above, for two n-tuples $(x_1,\ x_2,\ \ldots,\ x_n)$ and $(y_1,\ y_2,\ \ldots,\ y_n)$, with $x_i,\ y_i \in V_\Gamma^*$, $1 \leq i \leq n$. We write*

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

iff one of the following two cases holds:

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

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

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

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

[and $y_{i_j} = S_{i_j}$, $1 \leq j \leq t$]. 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 $y_{i_j} = S_{i_j}$, $1 \leq j \leq t$]" 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 $x_i \Rightarrow_{G_i} y_i$, $1 \leq i \leq n$ (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. Whenever a query symbol is present, then a communication step has priority and should happen. During this step, each query symbol Q_j in x_i is replaced by x_j, but only if x_j does not contain any query symbols. Simply put, a communication step involves replacing the query symbol Q_j with the string x_j. The outcome of this process is known as Q_j being satisfied by x_j. 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. No rewriting step can take place unless all query symbols are satisfied during a communication step. If some of them remain, the next step will be another communication to satisfy them.

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 [2, 12, 13, 15]:

1. If a component x_i in the current n-tuple $(x_1, ..., x_n)$ includes non-terminals but lacks any non-terminal that can be rewritten in G_i, the derivation cannot proceed.

2. The derivation also comes to a stop in the event of a circular query. This occurs if 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}. recall that communication is prioritized and only strings free of query symbols can be communicated. Consequently in such a cycle neither communication nor componentwise derivation can happen.

Definition 2.3. LANGUAGES GENERATED BY PCGS [2]: *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:*

$$L_f(\Gamma) = \{w \in T^* \mid (S_1, \ldots, S_n) \Rightarrow^* (w, \alpha_1, \ldots, \alpha_n), \alpha_i \in (N \cup T \cup K)^*, 2 \leq i \leq n\}$$

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 first grammar, also called the master grammar.

Definition 2.4. PCGS SEMANTICS [15]: *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.*

In cases where both the returning and non-returning modes of derivation are applicable to 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 or 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). If centralized systems are user we add the symbol C, and for the non-returning mode of derivation we use the symbol N. We thus 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 then the subscript n may also be removed, thus obtaining the classes $PC(X)$, $CPC(X)$, $NPC(X)$, 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).

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

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

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 to 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:

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

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

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

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 $\{A \rightarrow A : A \in N\}$ 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

$$P_1 = \{S_1 \rightarrow aS_1,\ S_1 \rightarrow aQ_2,\ S_2 \rightarrow cQ_2,\ S_2 \rightarrow c\},$$
$$P_2 = \{S_2 \rightarrow bS_2\}.$$

A derivation in Γ_1 *has the following form:*

$$(S_1,\ S_2) \Rightarrow^* (a^k S_1,\ b^k S_2) \Rightarrow (a^{k+1}Q_2,\ b^{k+1}S_2) \Rightarrow$$
$$(a^{k+1}b^{k+1}S_2,\ b^{k+1}S_2) \Rightarrow (a^{k+1}b^{k+1}cQ_2,\ b^{k+2}S_2) \Rightarrow$$
$$(a^{k+1}b^{k+1}cb^{k+2}S_2,\ b^{k+2}S_2) \Rightarrow (a^n b^n c b^{n+1} c \dots c b^{k+j} c,\ b^{k+j}S_2),$$

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

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

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 $L \subseteq L_1 L_2$ and for each $x \in L_1$ ($y \in L_2$) there is an $y \in L_2$ ($x \in L_1$), such that $xy \in L$ [6].

Example 2. *Consider the following PCGS:*

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

with

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

We start with $(S_1,\ S_2,\ S_3)$. *We first use 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:

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

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

$$(S_1,\ S_2,\ S_3) \Rightarrow_r (a^3Q_2,\ bS_2,\ cS_2) \Rightarrow_r (a^3bS_2,\ S_2,\ cS_3) \Rightarrow_r$$
$$(a^3b^3Q_3,\ bS_2,\ c^2S_3) \Rightarrow_r (a^3b^3c^2S_3,\ bS_2,\ S_3) \Rightarrow (a^3b^3c^3,\ b^2S_2,\ cS_3).$$

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

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

Since there is only one query from P_1 to P_2 and only one to P_3, we obtain the same language in both returning and non-returning modes (since the form of the strings being communicated become immaterial after communication).

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

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

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:

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

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,

$$L_r(\Gamma_3) = \{a^n b^n c^n d \mid n \geq 1\}.$$

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

We mainly concentrate in this chapter on synchronized PCGS due to the lack of findings for unsynchronized cases, despite the many outcomes related to the generative capabilities of various PCGS types.

The most powerful PCGS types are context-sensitive (CS) and recursively enumerable (RE). Interestingly, their behaviors are quite alike though not exactly the same. It is immediate that a recursively enumerable grammar is as powerful as any PCGS with recursively enumerable components. That is,

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

for all $Y \in \{PC,\ CPC,\ NPC,\ NCPC\}$ [2].

It turns out that this also applies to PCGS with context-sensitive components in relation to context-sensitive languages:

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

for all $Y \in \{CPC,\ NCPC\}$ [2]. However, it is important to note that this finding pertains to the centralized case and is not applicable to non-centralized PCGS.

Despite these points, it should be noted that PCGS with CS components are not particularly practical due to their reliance on a computationally expensive model, namely the context-sensitive grammar. More useful types employ simpler, computationally less demanding components. Therefore, exploring PCGS with regular or context-free components is more interesting and relevant.

We note that the class of languages produced 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. Essentially, this indicates that a PCGS, in general, exhibits greater power than an individual grammar of the same type, and the power of the system amplifies with the increase in its communication facilities. [15]:

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

The concept is similarly valid for PCGS with context-free components, as follows: [6]:

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

However, in this scenario it is important to note that only increasing communication within the system does not necessarily enhance its power.

In general, the centralized variant is a specific case of a non-centralized PCGS. It is obvious that centralized qualifier limits the communication initiation to the first grammar in the system. Consequently, for any languages created by a centralized PCGS of any type, there is a non-centralized PCGS of the same type that can generate the same language:

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

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 also becomes limited.

These two findings further illustrate the limitations in the generative power of PCGS. Specifically, when limited to only two regular components, the languages generated by centralized or non-centralized PCGS are all context-free.

1. $CPC_2(REG) \subsetneq CF$.
2. $PC_2(REG) \subseteq CF$.

They has been proved in [2].

Enhancing the generative power of a system can also be achieved by increasing the number of components in the system. As previously mentioned, this does not change the generative capacity in the recursively enumerable case, and to an extent, the same holds for the context-sensitive case. However, when examining classes lower in the hierarchy, it becomes evident that an increase in the number of components generally leads to a rise in the system's generative capacity.[2]:

1. There are languages that can be generated by a PCGS with two or more regular components, which cannot be produced by a linear grammar:

   $$Y_n(REG) \setminus LIN \neq \emptyset,$$

   for $n \geq 2,\ Y \in \{PC,\ CPC,\ NPC,\ NCPC\}$.

2. A language exists that can be generated by a PCGS with three or more regular components but cannot be produced 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. A language exists that can be generated by a PCGS with two or more linear components, but cannot be produced by a context-free grammar:

$$Y_n(LIN) \setminus CF \neq \emptyset,$$

for $n \geq 2,\ Y \in \{PC,\ CPC,\ NPC,\ NCPC\}$.

4. A language exists that can be generated by a non-returning PCGS with two or more regular components, which cannot be produced by a context-free grammar:

$$Y_n(REG) \setminus CF \neq \emptyset,$$

for $n \geq 2,\ Y \in \{NPC,\ NCPC\}$.

If the power of the components increases, then the power of the PCGS will generally increase. This applies strictly to the centralized case for regular, linear and context-free components [2]:

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

In the case of non-centralized PCGS, the same relationship is likely to apply, although this requires further investigation.

The following statement implies that the language families $CPC_n(REG)$ and $CPC_n(LIN)$ are not too large:

$$LIN \setminus CPC_*(REG) \cup NCPC_*(REG) \neq \emptyset,$$

Indeed, this means that there are specific languages within the scope of linear grammars that can be effectively generated by these parallel systems. There exists a language that can be generated by a linear grammar but cannot be produced by either non-centralized parallel communicating grammar systems $NCPC_*(REG)$ or centralized parallel communicating grammar systems $CPC_*(REG)$ with any number of regular grammars.

The language family $CPC_2(REG)$, which consists of languages generated by centralized parallel communicating grammar systems using exactly two regular grammars, is strictly included in the set of languages generated by context-free grammars.

$$CPC_2(REG) \subset CF,$$

In other words, every language produced by a centralized parallel communicating grammar system with two regular grammars is also a language generated by a context-free grammar, but there exist languages within the broader set of context-free languages that cannot be generated by such centralized systems with only two regular grammars.

The number of components in a PCGS may also affect the expressive power. In this respect the hierarchies of languages generated by PCGS with either regular or linear grammars (namely, $CPC_n(REG)$ and $CPC_n(LIN)$, $n \geq 1$) where found to be infinite [2].

3.1 Context-Free PCGS are Turing complete

By contrast to the infinite hierarchies mentioned above the hierarchy of synchronized, non-centralized PCGS with context-free components, collapses quite early into the class of recursively enumerable languages. Remarkably, it has been proven that a PCGS with six context-free components, operating in a non-returning manner, can simulate a 2-counter machine [5], thus generating any recursively enumerable language. This Turing completeness extends to returning systems as well, although the exact number of components required might vary. For returning systems it was initially established that 11 context-free components are sufficient to collapse the hierarchy to recursively enumerable languages: $RE = PC_{11}CF = PC_*CF$ [4]. Such a PCGS simulating any 2-counter machine is shown in Figure 3.1. This upper limit was later reduced to just five components [3]. However, these results hinge on the broadcast communication model, where a component holds onto its string until all requests are fulfilled [7], different from the one-step communication model where the component resets immediately after a single communication. Under one-step communication, the hierarchy still collapses, but not this warly. The best result so far requires 95 components to reach the recursively enumerable class [16]. It should be noted however that this result was obtained by modifying the sub-optimal 11-component construction [4], so it is probably improvable.

In terms of size complexity we note that any recursively enumerable language can be generated by returning PCGS with context-free components if the system is designed with a given number of non-terminals [5]. This indicates a nuanced relationship between the number of components, the type of grammars used, and the communication model, all of which influence the generative power and complexity of PCGS.

3.2 Unsynchronized PCGS

As previously noted, unsynchronized PCGS have received relatively little attention. It is quite apparent that the family of languages generated by a PCGS family in unsynchronized mode is included, though not necessarily strictly, within 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 $\{A \rightarrow A : A \in N\}$ 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.

After finding that unsynchronized PCGS are weaker, they have essentially been overlooked in further studies. Results on this matter are scarce and include the following [2]. The centralized case is very weak in some circumstances:

$$L(UCPC_*REG) = REG, L(UCPC_*LIN) = LIN.$$

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

Figure 3.1: A CF-PCGS that simulate a 2-counter Turing machine

On the other hand, $L(UNCPC_2REG)$ contains non-semi linear languages. It is also the case that:

$$L(UPC_2REG) \setminus L(REG) \neq \emptyset$$
$$L(UPC_2REG) \subseteq L(CF)$$
$$L(UPC_2LIN) \setminus L(CF) \neq \emptyset$$
$$L(UCPC_2CF) \setminus L(CF) \neq 0$$

3.3 Recent Investigations on Unsynchronized Context-Free PCGS

It was recently found that language generated by unsynchronized context-free PCGS can be recognized by nondeterministic Turing machine using $O(|w|)$ tape cells for each input instance w, meaning that these languages are all context sensitive [11].

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:

1. $i \neq 1$ and the respective component is not queried, or
2. $i = 1$ 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
3. $i \neq 1$ and x_i is queried by x_j , $j \neq i$, and then x_j become non-direct-significant.

Notably, definition introduces the concept of non-direct-significant components, categorizing components whose structure is irrelevant for derivation, allowing for their removal without affecting lateral effects. 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 3.1. *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.*

Lemma 3.1 establishes that if the length of a component surpasses the length of the generated string w, it becomes non-direct-significant for the generation process. Corollary 3.2 extends this finding to unsynchronized non-returning context-free PCGS.

Corollary 3.2. *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 xi becomes greater than $|w|$, that component becomes non-direct-significant for the generation of w.*

Proposition 3.3. *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

Corollary 3.4. *All languages generated by unsynchronized (returning or non-returning) PCGS with context-free components and no* ε*-rules are context-sensitive.*

Proposition 3.3 asserts a significant computational result, indicating that any language generated by an unsynchronized PCGS, be it returning or non-returning, with context-free components and no ε-rules, is context-sensitive. This theoretical foundation is crucial for understanding the computational complexity of languages generated by such grammar systems. The result was then extended by the elimination of the requirement that ε-rules must be absent:

Proposition 3.5. *Let* Γ *be an unsynchronized (returning or non-returning) PCGS with* n *context-free components* ($n \geq 1$). *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.

Therefore all languages generated by unsynchronized (returning or non-returning) PCGS with context-free components are context-sensitive.

It is important to recognize that all the results other than the one mentioned in this section are quite specific and have not seen significant extension or development in nearly 20 years. Additionally, there are no existing results concerning unsynchronized regular PCGS.

Chapter 4

Unsynchronized Regular PCGS and Context-Free Languages

Regular grammars are valued for their simplicity and effectiveness, while context-free grammars (CFGs) are noted for their broader range of expression. This poses a question: Can we combine the simplicity of regular grammars with the expressive might of CFGs? This is where Parallel Communicating Grammar Systems (PCGS) come in, an interesting concept where multiple grammars operate simultaneously, sharing information to generate language strings. In this chapter, we show how we can express typical context-free language constructs using unsynchronized regular PCGS. We find that typical context-free constructs can be generated in this manner. We then investigate whether all context-free languages can be generated by unsynchronized regular PCGS. However, we find in the end that this is not the case.

We first illustrate the generative power of unsynchronized regular PCGS by generating four very typical context-free languages.

4.1 Balanced Symbols

We first consider a very basic context-free construct namely, counting. It is very common for a context-free language to include a comparison between the number of occurrences of two symbols. In this kind of constructs a pushdown automaton will use its stack as a counter. The simplest such a construction is the language with two symbols having the same number of occurrences:

$$L_{bal} = \{a^n b^n | n \geq 0\}$$

The context-free grammar that defines L_{bal} is $G_{bal} = (N, T, P, S)$ where $N = \{S\}$ is set of non-terminals, $T = \{a, b\}$ is set of terminals and $P = \{S \rightarrow aSb, S \rightarrow \varepsilon\}$ is set of rewriting rules. To simulate this CFG using an unsynchronized PCGS with regular components, we can use two grammars. The first grammar generates the a's and the

second grammar generates the b's. Consider thus $\Gamma_{bal} = (\{S_1, S_2\}, K, \{a, b\}, G_1, G_2)$ where:

$$P_1 = \{S_1 \to aQ_2, B \to S_1\} \tag{4.1}$$
$$P_2 = \{S_2 \to Bb\} \tag{4.2}$$

A derivation in Γ_{bal} can proceed in both returning and non-returning mode, both generating the same strings in the language L_{bal}. The returning mode results in the following kind of derivations:

$$(S_1, S_2) \Rightarrow (aQ_2, Bb) \Rightarrow (aS_2, Bb) \Rightarrow (aBb, S_2) \Rightarrow^{*n}$$
$$(a^n Bb^n, S_2) \Rightarrow_{B \to S_1} \Rightarrow (a^n S_1 b^n, S_2) \Rightarrow (a^n b^n, S_2)$$

The non-returning mode on the other hand would proceed like this:

$$(S_1, S_2) \Rightarrow (aQ_2, Bb) \Rightarrow (aS_2, Bb) \Rightarrow (aBb, Bb) \Rightarrow^{*n}$$
$$(a^n Bb^n, Bb) \Rightarrow_{B \to S_1} \Rightarrow (a^n S_1 b^n, Bb) \Rightarrow (a^n b^n, Bb)$$

To understand the constraints on the derivation within the PCGS, consider what happens if in the first step B is rewritten to ε ($B \to \varepsilon$). By doing this, the system stops working immediately, failing to produce any a's. Such an result is basically incompatible with the principles managing L_{bal}, as it necessitates a balance between the numbers of a's and b's. Therefore, the resulting strings, potentially of the form b_n, fall outside L_{bal}, leading the PCGS design to explicitly disallow this path. Similarly, should the second component S_2 remain unaltered in the initial step, the derivation comes to a stop. This comes from the system's dependency on the concurrent generation of a's and b's for balance, a core requirement of L_{bal}. An absence of this coordinated generation leads to an inconsistency, the string is out of sync with L_{bal}'s balance requirements.. Thus, the PCGS architecture is structured to prevent such inconsistencies by ensuring that every step complies to the generation of strings within L_{bal}. By employing multiple grammars in an unsynchronized method, The PCGS successfully separates the generation of a context-free language into small tasks that regular grammars can handle.

In this example, the CFG's task of generating balanced a's and b's is divided between two grammars in the PCGS. Generating an equal number of symbols is trivial in a synchronized system, but considerably more challenging in the absence of synchronization. Still, we were able to replace the synchronization (which would only require one communication step at the end) with a sequence of communication steps that accomplishes the same outcome. In effect we generate one symbol in each component, put the two symbols in the same component (so that they now match each other), and then repeat as necessary. This simple example's construction demonstrates that an unsynchronized PCGS with regular grammars can simulate the behavior of a CFG.

4.2 Balanced Symbols with Intermediate Content

As an extension of the previous example we expand on the same idea of balanced symbols (so that a pushdown automaton would also use its stack as a counter), but this time with some (not counted) symbols in between. the following language:

$$L_{bal.int} = \{a^n b^m c^n | n, m \geq 0\}$$

This language consists of strings with an equal number of a's and c's, separated by any number of b's. Let $G_{bal.int} = (N, T, P, S)$ where $N = \{S, A, B, C\}$ is set of non-terminals, $T = \{a, b, c\}$ is set of terminals, and the rewriting rules are defined as follows:

$$P = \{S \to aAc, A \to aAc|\varepsilon, C \to cC|\varepsilon, S \to B, B \to bB|\varepsilon\}$$

The equivalent Unsynchronized PCGS with regular components is:

$$\Gamma_{bal.int} = (\{S_1, S_2\}, k, \{a, b, c\}, G_1, G_2)$$

Where:

$$P_1 = \{S_1 \to aQ_2, S_1 \to C, B \to S_1, C \to bC, C \to \varepsilon\}$$
$$P_2 = \{S_2 \to Bc\}$$

A derivation (returning or non-returning) in this system proceeds as follows:

$$(S_1, S_2) \Rightarrow (aQ_2, Bc) \Rightarrow (aS_2, Bc) \Rightarrow (aBc, S_2) \Rightarrow (aS_1c, S_2) \Rightarrow^{*n}$$
$$(a^n S_1 c^n, Bc) \Rightarrow (a^n C c^n, Bc) \Rightarrow (a^n bC c^n, S_2) \Rightarrow^{*m} (a^n b^m C c^n, S_2) \Rightarrow_{C \to \varepsilon}$$
$$\Rightarrow (a^n b^m c^n, S_2)$$

The grammar $\Gamma_{bal.int}$ is designed to generate strings of the form $a^n b^m c^n$. For this to happen, each component of the PCGS plays a specific role. The first component S_1 begins by generating a's, component C generating b's. Simultaneously, the second component S_2 needs to start generating c's to maintain the balance and order of symbols as per the language rules. Delay or inaction in one component can disrupt the harmony and sequence required in the string generation process. The second component S_2 is responsible for generating the c's in the sequence. If it does not start its rewriting process immediately, the sequence of c's will not be initiated, disrupting the intended pattern of the language. If S_2 stays put and does not rewrite, the derivation process can become blocked. This is because the generation of c's (which balance the 'a's) will be delayed or might not occur at all. In the absence of c's, the strings generated would not conform to the language's specification of having equal numbers of a's and c's, separated by any number of b's.

It is worth noting that the inner content (b^m) is generated by a designated non-terminal (C). There is nothing preventing that non-terminal to generate any language that can be generated by an unsynchronized regular PCGS. Similarly, there is nothing preventing the outer part ($a^n \cdots c^n$) to be any other language that can be generated by an unsynchronized regular PCGS. We will elaborate on this later in Section 4.4.

4.3 Palindromes

Another common context-free construct is the presence of a pair of substrings where one if the reversal of the other. In such a case a pushdown automaton will use its stack to match the first substring with the second (in reverse due to the last-in-last-out nature of the stack). The simplest language of this nature is:

$$L_{pal} = \{ww^R | w \in \{a, b\}^*\} \tag{4.3}$$

where w^R denotes the reverse of the string w. Each string in the language L_{pal} is a palindrome because the second half of the string is a mirror image (or reverse) of the first half. The following grammar generates L_{pal}: $G_{bal} = (\{S\}, \{a, b\}, P, S)$ with $P = \{S \to aSa, S \to bSb, S \to \varepsilon\}$.

We can generate the same language using a PCGS where the master grammar starts by generating a symbol and then calls a different component for each kind of symbol thus generated. That component in turn will generate a matching symbol. In all we obtain the following system:

$$\Gamma_{pal} = (\{S_1, S_2, S_3\}, k, \{a, b\}, G_1, G_2, G_3)$$

Where:

$$\begin{aligned} P_1 &= \{S_1 \to aQ_2, A \to bQ_3, B \to \varepsilon\} \\ P_2 &= \{S_2 \to Aa\} \\ P_3 &= \{S_3 \to Bb\} \end{aligned} \tag{4.4}$$

Suppose that we want to generate $abba$. The corresponding derivation (returning or non-returning) goes as follows:

$$\begin{aligned} &(S_1, S_2, S_3) \Rightarrow (aQ_2, Aa, S_3) \Rightarrow (aAa, Aa, S_3) \Rightarrow (aS_1a, S_2, S_3) \Rightarrow \\ &(abQ_3a, S_2, Bb) \Rightarrow (abBba, S_2, S_3) \end{aligned}$$

Now we can delete B using $B \to \varepsilon$ and we end up with $abba$. In fact we can go back and forth between components as many times as we like:

$$(abBba, S_2, S_3) \Rightarrow^{*n} (a^n b^n Bb^b a^n, S_2, S_3) \Rightarrow a^n b^n b^n a^n = (ab)^n (ba)^n$$

Obviously we do not have to alternate between a and b. For example the string $aaabbbbbbaaa$ is obtained in a similar manner:

$$(S_1, S_2, S_3) \Rightarrow (aQ_2, Aa, S_3) \Rightarrow (aS_2, Aa, S_3) \Rightarrow (aAa, S_2, S_3) \Rightarrow$$
$$(abQ_3a, S_2, Bb) \Rightarrow (abS_3a, S_2, S_3) \Rightarrow (abBba, S_2, S_3) \Rightarrow^{*3}$$
$$(aaabbbBbbbaaa, S_2, S_3) \Rightarrow (aaabbbbbbbaaa, S_2, S_3)$$

4.4 Embedded Matching

A simple generalization of the systems that generate balanced symbols and palindromes will generate matched symbols within matched symbols, another typical context-free construct.

We note first that both palindromes and balanced strings are generated (by a context-free grammar or regular PCGS) by matching pairs of symbol. For brevity call both these kind of strings *matchings*.

Consider the PCGS shown in Relation 4.4 that generated palindromes, but instead of erasing B at the end we transform it into a query to a brand new component:

$$P_1 = \{S_1 \rightarrow aQ_2, A \rightarrow bQ_3, B \rightarrow Q_4\}$$
$$P_2 = \{S_2 \rightarrow Aa\}$$
$$P_3 = \{S_3 \rightarrow Bb\}$$

The new component in turn will generate balanced strings, just like the system shown in Relation 4.1:

$$P_4 = \{S_4 \rightarrow aQ_5, C \rightarrow S_4\}$$
$$P_5 = \{S_5 \rightarrow Cb\}$$

The language generated by the resulting PCGS is $\{wa^n b^n w^R | w \in \{a, b\}^*, n \geq 0\}$, which consists of a matching embedded into another matching. Obviously this construction is not restricted to these particular matchings and to only one embedding. We can go with different matchings and we can embed an arbitrary number of such matchings.

4.5 Sequences of Symbols

For completeness we now consider multiple sequences of unrelated symbols. This is not an overly interesting context-free construct (the language is actually regular), but we use this example to show how separate, dedicated components can be made responsible for their own sub-language, while all this sub-languages are put together in the master component. In all we consider the following language:

$$L_{seq} = \{a^i b^j c^k \mid i, j, k \geq 0\}$$

This language includes is generated by the following unsynchronized PCGS with regular components:

$$\Gamma_{seq} = (\{S_1, S_2, S_3, S_4\}, k, \{a, b, c\}, G_1, G_2, G_3, G_4)$$

Where:

$$P_1 = \{S_1 \to Q_2, X \to Q_3, Y \to Q_4\}$$
$$P_2 = \{S_2 \to aS_2, S_2 \to X\}$$
$$P_3 = \{S_3 \to bS_3, S_3 \to Y\}$$
$$P_4 = \{S_4 \to cS_4, S_4 \to \varepsilon\}$$

This PCGS is designed to handle cases where i, j, k can be zero, prevents us from writing $S \to w_i Q_i$ like in the other examples, meaning it can generate strings with no 'a's, 'b's, and 'c's, or any combination thereof. This flexibility is achieved through the independent yet coordinated functioning of each component, guided by the master component P_1. Consider that we want to construct $abbccc$. Derivation (returning or non-returning) is as follows:

$$(S_1, S_2, S_3, S_4) \Rightarrow (Q_2, aS_2, bS_3, cS_4) \Rightarrow (aS_2, aX, bS_3, cS_4) \Rightarrow$$
$$(aX, S_2, bS_3, cS_4) \Rightarrow$$

We have a, now is bb turn:

$$(aQ_3, S_2, S_3, S_4) \Rightarrow (abS_3, aS_2, bS_3, cS_4) \Rightarrow^{*2} (abbS_3, S_2, bY, cS_4) \Rightarrow$$
$$(abbY, S_2, S_3, S_4) \Rightarrow$$

And now we go for the last non-terminal ccc:

$$(abbQ_4, S_2, S_3, S_4) \Rightarrow (abbcS_4, S_2, S_3, S_4) \Rightarrow^{*3} (abbcccS_4, S_2, S_3, cS_4) \Rightarrow_{S_4 \to \varepsilon}$$
$$\Rightarrow (abbccc, S_2, S_3, S_4)$$

The final configuration represents the string $abbccc$, which is in the language L_{seq}. This method effectively simulates the language L_{seq} by allowing independent generation of 'a's, 'b's, and 'c's in any quantity, including none. The unsynchronized nature of the PCGS provides the necessary flexibility to generate all valid strings in the language, illustrating the power of this grammar system in language simulation.

4.6 On Generating all the Context-Free Languages

We have illustrated above the uses of regular components in an unsynchronized PCGS to generate a few common context-free constructs. Now we want to try these techniques in a more general context to simulate an arbitrary context-free grammar.

We hope to generate all context-free languages using unsynchronized (returning or non-returning) PCGS with regular components.

The examples presented earlier illustrate the potential of unsynchronized communication between regular grammars in a PCGS to mimic the behavior of a context-free grammar. Now we want to construct an unsynchronized regular PCGS equivalent to the general form of a context-free rewriting rule namely:

$$A \rightarrow w_1 X_1 w_2 X_2 w_3 X_3 \dots w_n X_n w_{n+1} \tag{4.5}$$

for $n \geq 1$, $X_i \in N$, and $w_i \in T^*$. We have already presented such a simulation for the case $n = 1$. The rule becomes

$$A \rightarrow w_1 B w_2.$$

and the equivalent PCGS is already well known:

$$P_1 = \{S_1 \rightarrow w_1 Q_2, X \rightarrow B_1\}$$
$$P_2 = \{S_2 \rightarrow X w_2\}$$

Indeed, the only possible derivation in this system is:

$$(S_1, S_2) \Rightarrow (w_1 Q_2, X w_2) \Rightarrow (w_1 X w_2, S_2) \Rightarrow (w_1 B_1 w_2, S_2) \Rightarrow w_1 B_1 w_2$$

In this scenario, unsynchronized PCGS with regular components can simulate this structure. This system can effectively show the production of terminal strands w_1 and w_2 and manage a single non-terminal B_1.

This looks promising. Unfortunately the case $n = 2$ (and by extension $n \geq 2$) does not go as smooth. In fact we will show that it is not possible at all to construct an equivalent unsynchronized regular PCGS for this case.

The goal is now to simulate the following rule:

$$A \rightarrow w_1 B_1 w_2 B_2 w_3. \tag{4.6}$$

We run into problems because we now have two non-terminals (B_1 and B_2) that must both find their way into one string. Each non-terminal in turn can potentially expand into more complex structures, and the sequence in which B_1 and B_2 interact with the terminals becomes critical. For $n = 2$, the regular components in the PCGS need to simulate not just the production of terminal strings but also the complex interaction between two non-terminals and their respective expansions. In the unsynchronized PCGS with regular components, each addition of a w_i and B_i pair requires a new set of regular grammars to represent this structure. However, as the complexity of the CFG increases (with more B_i's), the regular components of PCGS struggle to keep up. They cannot effectively represent the recursive or nested structures that the non-terminals B_i in the CFG can generate.

This limitation becomes more apparent as n increases, ultimately highlighting the fundamental gap in computational power between CFGs and unsynchronized PCGS with regular components. Thus, the system fails to accurately represent the CFG production rule for $n = 2$, indicating a fundamental limitation of the PCGS with regular components.

Formally, we show first that it is not possible to generate two non-terminals in the same string using regular components in an unsynchronized PCGS:

Lemma 4.1. *For any derivation* $(S_1, S_2, \ldots, S_n) \Rightarrow^* (w_1, w_2, \ldots, w_n)$ *in an unsynchronized regular PCGS it holds that* $|w_i|_{(N \cup K)} \leq 1$ *for all* $1 \leq i \leq n$. *In other words, in any component at any time during a derivation there is at most one non-terminal or query symbol.*

Proof. Ignoring communication for the time being, in a regular PCGS, each rule is of the form $A \to cB, A \to c, A \to \varepsilon$, or $A \to B$ where $A, B \in N$ (non-terminals) and $a \in T$ (terminals). Therefore, at each step of derivation, a non-terminal can produce at most one terminal followed by at most one non-terminal. We start with an initial configuration $(S_1, S_2, \ldots, S_n)$. Initially, each component S_i is either a single non-terminal or a query symbol, meaning $|S_i|_{(N \cup K)} = 1$. Within any component grammar G_i, a derivation step can involve:

1. Replacing a non-terminal with a terminal symbol, which decreases the count of non-terminals or query symbols in that component ($|w_i|_{(N \cup K)}$ decreases).

2. Replacing a non-terminal with another non-terminal, keeping the count of non-terminals or query symbols constant ($|w_i|_{(N \cup K)}$ remains 1).

Querrying different components does not fare better. We introduce queries using rules of the form $A \to cQ_i$ or $A \to Q_i$ with $A \in N$, $c \in T$, and $Q_i \in K$, and so any query symbol replaces a non-terminal (so overall the number of symbols in $N \cup K$ remains the same). Then a query symbol ends up replaced by a string as above, containing a single non-terminal (that replaces the query symbol), again for no net gain of symbols in $N \cup K$. This can be easily shown by an induction over the length of the chain of queries:

The base case (no queries) is established above. We use the induction hypothesis that at the end of a chain of n queries all components hold strings that contain at most one non-terminal or query symbol. We then go to a chain of $n + 1$ queries by considering what happens when the component G_j queries the string of G_i. The string of G_j does not contain any non-terminal (by the induction hypothesis, since it already contains a query symbol). The string of G_j contains no query (since otherwise it cannot be communicated) and at most one non-terminal (by the induction hypothesis). Therefore after the communication step the string of G_j contain no queries (since the existing query symbol gets erased) and at most one non-terminal (the one coming from G_i). The number of symbols other than terminals cannot increase, as desired.

Therefore neither componentwise derivation steps nor communication steps can increase the number of non-terminals or query symbols in any component string beyond one. □

As an example consider the PCGS developed in Section 4.1. The first step rewrites S_1 into aQ_2 in G_1 (where Q_2 is a query symbol), and S_2 is rewritten as Bb in G_2 (where B is a non-terminal). A communication step involves replacing the sole query symbol in one component with a string from another component. This string will always contain at most one non-terminal, due to the limitation that each component string can only contain one such symbol. In both returning and non-returning derivations the process involves replacing Q_2 with a string from the second component, which always contains one non-terminal (B or S_1). During the derivation process in this PCGS no step can increase the count of non-terminals or query symbols in any component string beyond one. Therefore, $|w_i|_{(N \cup Q)} \leq 1$ for all $1 \leq i \leq n$ throughout the derivation.

Lemma 4.1 alone seems to suggest that rules of Form 4.6 cannot be simulated by an unsynchronized regular PCGS. Indeed, we just proved that we cannot introduce two non-terminals in any current string. However, there is one more way in which such a rule can be simulated: We first generate the w_1B_1 part. Then we proceed with the rewriting of B_1 to a string of terminals, except that we stop one step short, when a non-terminal still exists in the string. If we are clever enough to arrange for that non-terminal to be the last in the string then we can still use the techniques described at the beginning of this section to rewrite that non-terminal into $w_2B_2w_3$. From here on we use again the techniques from the beginning of this section to rewrite this string into a terminal string. This is all possible (even trivial) as long as w_1B_1 can always be rewritten into a string that contains a single non-terminal at the end of the string. That this is not always possible is shown by the following counterexample:

Lemma 4.2. *Let $L_{bal} = \{a^k b^k : k \geq 0\}$ and let Γ_{bal} be an unsynchronized regular PCGS with n components that generates L_{bal}. Let then $(S_1, S_2, \ldots, S_n) \Rightarrow^* (w, w_2, \ldots, w_n)$ with w a non-empty terminal string. Then it must be the case that $(S_1, S_2, \ldots, S_n) \Rightarrow^* (u_1, u_2, \ldots, u_n) \Rightarrow^* (w, w_2, \ldots, w_n)$ with some $u_i = v_1 A v_2$ where A is a non-terminal and neither $v1$ nor $v2$ are empty.*

Proof. We start with the initial configuration of the PCGS, denoted as $(S_1, S_2, \ldots, S_n)$ where each S_i represents the start symbol of each component's grammar. At this stage, none of the components have produced any terminal strings. As the derivation processes, the grammars in each component work together to generate parts of the final string w. To generate a string that belongs to L_{bal}, there must be a mechanism within at least one of the components to ensure the balance of a's and b's. This mechanism is controlled by a non-terminal symbol, denoted as A. A cannot simply be at the beginning or end of the string within any component because in those positions, it could only add a's and b's at one end. However, to balance the a's and b's,

A must influence the production of both symbols within the string. Therefore, there must be an intermediate stage in the derivation process, denoted as $(u_1, u_2, \ldots, u_n)$, where in at least one component u_i, the string takes the form $v_1 A v_2$. Here, A is a non-terminal symbol, and v_1 and v_2 are non-empty strings. This intermediate configuration $v_1 A v_2$ is crucial because it allows A to regulate the production of a balanced sequence of a's followed by b's. A acts as a pivot point, ensuring that the correct number of a's and b's are generated to maintain the balance required by L_{bal}. From this intermediate configuration, the derivation process continues until the final string w is produced. At each step, the components work together, guided by the non-terminal A, to ensure that w is a valid string in L_{bal}, reflecting a balanced number of a's and b's. Thus, we showed that the derivation process in L_{bal} for generating strings necessarily involves a stage where a non-terminal appears in the middle of a string in one of the components. This is essential for ensuring the balanced structure of the final string w. □

Theorem 4.3. *There exists a context-free languages that cannot be generated by any unsynchronized PCGS with regular components.*

Proof. Lemma 4.1 shows that we cannot simulate directly most context-free rules (namely, the ones whose right hand sides contain more than one non-terminal). Therefore the only way to simulate this kind of rules is piecemeal: We simulate a prefix of the right hand side of the rule containing a single non-terminal all the way until the last step that still feature a non-terminal in the current string, and then we use that non-terminal to start the generation of the rest of the right hand side. This only works whenever the non-terminal we continue from is the last in the current string. Lemma 4.2 provides a counterexample showing that this is not always possible. □

To put Theorem 4.3 in a different way, we note that it is fundamentally challenging to directly simulate some specific rules of context-free grammars using unsynchronized regular PCGS, particularly the rules of type $A \to BC$. This difficulty arises primarily from the fact that within any single component of such a system there can only ever be one non-terminal or query symbol at any given stage in the derivation process. The rule $A \to BC$ is crucial in context-free grammars, since it is the workhorse of the Chomsky normal form, where rules can only have the forms $A \to BC$, $A \to a$, and $A \to \varepsilon$. While the last two types are easy to simulate in a regular PCGS, the $A \to BC$ type poses a significant challenge due to its inherent need for merging strings from different non-terminals. To illustrate, consider the context-free rule $A \to w_1 B_1 w_2 B_2 w_3$. In a standard context-free environment, this rule might be broken down into $A \to BC$, $B \to w_1 B_1 w_2$, and $C \to B_2 w_3$, allowing for the gradual construction of the string. However, in an unsynchronized regular PCGS, this method is not feasible because of the limitation on the number of non-terminals or query symbols in each component. There is still a possible way

around $A \rightarrow BC$. That rule implements concatenation, so perhaps we can simulate concatenation of two languages (call them L_1 and L_2) in another way: First create a PCGS that generates L_1. Then modify it so that instead of generating w_1 it generated $w_1 Q_x$, with the component x generating L_2. Lemma 4.2 shows that for a balanced language L_{bal} produced by an unsynchronized regular PCGS, any non-empty terminal string emerging from the system must have gone through a stage where a non-terminal is surrounded by non-empty strings, thus making such a trick impossible.

Chapter 5

Conclusions

Parallel Communicating Grammar Systems (PCGS) represent a fascinating area in theoretical computer science, particularly in the field of formal language theory. PCGS is essentially a collection of several grammars that work in parallel and communicate with each other under certain rules. An interesting aspect of PCGS lies in its capacity to model the complex processes of language production, similar to how different modules in a computer program interact to perform a task. The exploration into the realm of PCGS, specifically focusing on unsynchronized PCGS with regular components, delves into how these systems can be employed to generate more complex languages.

We focus in particular on unsynchronized regular PCGS under the hypothesis that they are powerful enough to generate all the context-free languages. We started by considering several examples to demonstrate how regular grammars within a PCGS can simulate certain context-free constructs. This step-by-step approach with specific examples was meant to build towards a more generalized theory. However, as we delved deeper into the complexities of context-free grammars, particularly when dealing with multiple non-terminals and their interactions with terminals, a significant limitation of unsynchronized PCGS with regular components became apparent.

Our hypothesis appeared to be an ambitious goal, considering the inherent limitations of regular grammars in comparison to the richer structural capabilities of context-free grammars. On the other hand, it seemed reasonable to assume that the communication facilities are powerful enough to compensate for these richer capabilities. We approached the task attempting to break down the rules of a context-free grammar into simpler regular grammars within the PCGS. The purpose of this method was to show that even with the limitations of grammars, the collective power of these grammars in PCGS can reproduce the behavior of a context-free grammar.

However, the challenge quickly became not only to generate the appropriate strings of terminals but also to accurately represent the interactions and expansion

of multiple non-terminals in the context-free grammar. The regular components in the PCGS struggled to emulate these complex structures, especially as the number of non-terminals in the production rules increased. This limitation was a critical roadblock.

Therefore, we pursued a more detailed understanding of the capabilities of unsynchronized PCGS with regular components, leading to Theorem 4.3. This result acknowledges that while PCGS with regular components provides an attractive model for language generation, it inherently lacks the computational power to fully replicate the generative capacity of context-free grammars. This conclusion, while perhaps initially disappointing, is significant as it defines the boundaries and capabilities of unsynchronized PCGS in the context of formal language theory. This illustrates the fact that while PCGS can model a wide array of language structures, there are intrinsic limitations to what can be achieved with regular components, especially when compared to the richer, more flexible structures allowed in context-free grammars.

In all, we showed in this paper that there exist context-free languages that cannot be generated by unsynchronized PCGS using regular components. It is very interesting to go the other way around and try to simulate unsynchronized regular PCGS using context-free grammars. We believe that such a simulation will succeed, thus establishing an actual hierarchy between languages generated by unsynchronized regular PCGS and context-free languages. At the same time we acknowledge the possibility of this not being the case, which will establish that the two classes are not comparable.

Bibliography

[1] S. D. BRUDA, *Pricate communication*, 2023.

[2] 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.

[3] 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.

[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 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.

[6] 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.

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

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

[9] 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.

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

[11] S. LI, *The Generative Power of Unsynchronized Context-Free Parallel Communicating Grammar Systems*, MSc Thesis, Bishop's University, 2022.

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

[13] 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.

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

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

[16] 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.