COMPUTATIONAL COMPLEXITY OF UNSYNCHRONIZED PARALLEL COMMUNICATING GRAMMAR SYSTEMS WITH CONTEXT-FREE COMPONENTS

by

JAVAD OMIDVAR

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
May 2021

Abstract

Parallel Communicating Grammar Systems (PCGS) were introduced as language-theoretic treatments of concurrent systems. The communication between grammars makes the whole system more powerful in solving a common task than its grammar components. PCGS can be synchronized (with the component grammars working in lock-step) or unsynchronized. Synchronized PCGS have received significant attention, but the unsynchronized variant has largely been ignored. In this thesis we investigate the generative capacity of unsynchronized PCGS with contex-free components (unsynchronized CF-PCGS for short). Synchronized CF-PCGS have been shown to be Turing complete. The hypothesis that unsynchronized CF-PCGS are weaker has been circulated, and we show that this is true and also false. For one thing, we show that returning unsynchronized CF-PCGS are Turing complete. To do so, we will adapt a simulation of a 2-counter Turing machine to the unsynchronized case. On the other hand, we also show that non-returning unsynchronized CF-PCGS are linear space (and so they only generate context-sensitive languages). This is established however under the assumption that no ε-rules are used by the component grammars.

Acknowledgments

I would first like to thank my supervisor, Professor Stefan Bruda, whose expertise was invaluable in formulating the research questions and methodology. His feedback was always helpful and made my mind able to think out of box.

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

I would like to thank my wife, Aisan, who was always supportive, always by my side, and understood my situation no matter how hard was the way we have been through.

Last but not least, I would especially like to thank my mother, who passed away during my Master's and I was not there to be with her. Too unfortunate she cannot see me graduate. Your memory will be eternal.

Contents

Chapter 1

Introduction

Parallel Communicating Grammar Systems (PCGS), have been introduced [18] as a language-theoretic treatment of concurrent systems. They consist of several grammars which work separately in a parallel environment. But the thing which makes them powerful is the fact that they may cooperate by communicating with each other toward a common task. Communication consists of a request from one component grammar resulting for the string currently derived in another component grammar. Communication has priority over derivations, so once a query appears it must be satisfied before any rewriting takes place. A derivation in a PCGS therefore consists of component-wise derivation and communication steps interleaved with each other.

The first grammar in a PCGS is designated as the master grammar while the others are helpers or slaves. The master is responsible for deriving a string with the possible help of the slaves. The difference is stressed in centralized PCGS, where only the master is allowed to initiate communication. By contrast, any component can initiate communication in a non-centralized system [4, 21] (while the master continues to be responsible for the overall output). Both centralized and non-centralized PCGS can communicate in a returning or non-returning manner. If the components of a PCGS return to their axiom after communication then the PCGS is returning. On the other hand, if the components continue to process the current string and do not reset to their axiom, then the system will be considered non-returning [4].

PCGS can also be differentiated based on their synchronization behaviour. In a synchronized PCGS all the components must perform a rewriting step at the same time in the absence of any queries (unless the component grammar is holding a terminal string, case in which it is allowed to stay put). By contrast, whenever components are allowed to either perform a rewriting step or wait then we call the respective system non-synchronized.

PCGS have been investigated with respect to their generative power, but some variants received substatially more attention than others. In particular, there are many efforts related to synchronized PCGS. In particular, it has been shown that non-returning PCGS with context free components are Turing complete, which means they can generate all recursively enumerable languages [12]. On the other hand, we know that non-returning systems can be simulated by returning systems [8]. Consequently, returning CF-PCGS are also Turing complete [13]. This result has also been developed independently by a direct simulation of a 2-counter Turing machine [5].

By contrast, unsynchronized PCGS have received very little attention. In particular, unsynchronized PCGS with context-free components (unsynchronized CF-PCGS for short) have not been investigated at all in respect to their generative power. This thesis sets out to investigate this area.

Our findings outline a wide variety in the generative power of unsynchronized CF-PCGS. First, we start from a paper which proved that returning CF-PCGS are Turing complete based on a direct simulation of a 2-counter Turing machine [5]. We show that this simulation also works in an unsynchronized manner. That is, the same simulation takes place successfully even when the original system becomes unsynchronized. We therefore conclude that unsynchronized CF-PCGS can generate all the recursively enumerable languages, so they are Turing complete.

Secondly, we investigate computational complexity of unsynchronized non-returning CF-PCGS. We find that under certain constraints (namely, whenever none of the components of the PCGS feature ε-rules) these systems can be simulated by $O(n)$ space-bounded Turing machines. Therefore the languages generated by this family of CF-PCGS are all context sensitive.

Chapter 2

Preliminaries

We briefly review some basic notions regarding grammars and then parallel communicating grammar systems which are essential to follow the paper. Considering some elements of the non-terminal and terminal alphabets. The basic intuition of a grammar model of distributed systems can be the following: component grammars forming the distributed system can execute rewritings on a shared set of words. The order of participation and the start and end conditions of participation are defined by the strategy of cooperation between the components [4].

A grammar is a tuple $G = (N, \Sigma, S, R)$, where N is a set of nonterminal symbols, Σ is a set of terminals, R is a set of rewriting rules (or productions), and $S \in N$ is the start symbol or axiom. A rule $\alpha \to \beta \in R$ specifies that a substring α can be replaced by β. A rewriting step is the application of a rewriting rule on a string w obtaining the string w' (written $w \Rightarrow w'$). A derivation from w to w' is a chained sequence of rewriting steps (written $w \Rightarrow^* w'$). The language generated by a grammar consists of exactly all the strings w such that $S \Rightarrow^* w$ and $|w|_N = 0$ (all the terminal strings that can be generated by derivations that start from the axiom of the grammar).

According to the Chomsky hierarchy, we have four classes of grammars, and they can be classified by the form of their rewriting rules. Let $G = (N, \Sigma, S, R)$ be a grammar; then:

- The grammar is a type-0 or unrestricted if G does not have any restriction. These grammars create languages which can be semi-decided by Turing machines. Any languages generated by this type of grammars are called recursively enumerable, or RE for short [11].

- If $|\alpha| \leq |\beta|$ for each rewriting rule $\alpha \to \beta$ in R then G is a type-1 or context sensitive grammar. 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. These languages are referred to as context sensitive, or CS for short [11].

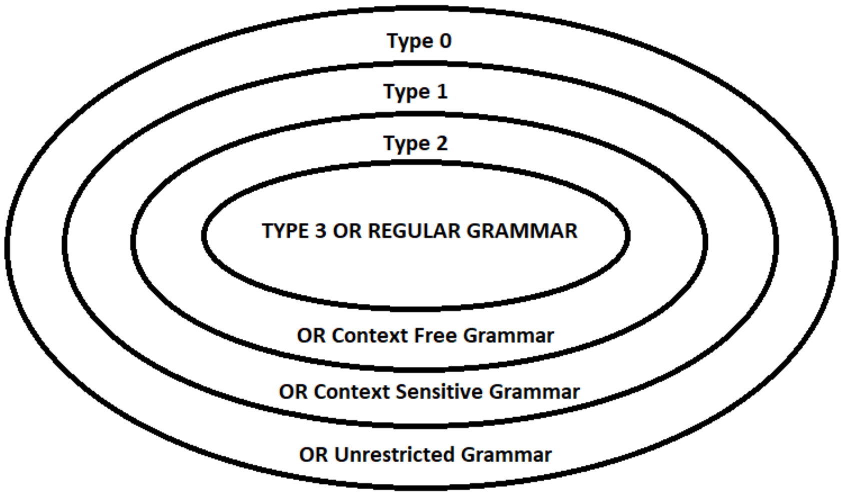

Figure 2.1: The Chomsky hierarchy

- If every rewriting rule $\alpha \rightarrow \beta$ in R satisfies $|\alpha| = 1$ (meaning that α is a single nonterminal), G is a context-free grammar, or type-2. Context-free grammars include a further specific type where no rewriting rule is allowed to have more than one non-terminal symbol on its right hand side (called linear grammars). The languages generated by type-2 grammars are referred to as context free or CF for short, and the languages generated by the linear grammar sub-type are referred to as Linear or LIN [9, 10].

- Whenever the rewriting rules of G are restricted to the following forms, G is called a type-3 or regular grammar: $A \rightarrow cB$, $A \rightarrow c$, $A \rightarrow \varepsilon$, or $A \rightarrow B$, where A, B are nonterminals and c is a terminal. A type-3 grammar can generate languages which are called regular, or REG for short. A language is called semi-linear if and only if it is letter equivalent to a regular language. When two languages are indistinguishable from each other when we only look at the relative number of occurrences of symbols in their words without taking their order into consideration, they are called letter equivalent [11].

As shown in Figure 2.1, four language classes are showed in a hierarchy, REG is the smallest and RE the largest class.

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 language-theoretic properties of parallel systems.

Definition 2.1. PARALLEL COMMUNICATING GRAMMAR SYSTEM [4]: *Let $n \geq 1$ be a natural number. A PCGS of degree n is an $(n + 3)$ tuple $\Gamma = (N, K, T, G_1, \ldots, Gn)$*

where N is a nonterminal alphabet, T is a terminal alphabet, and K is the set of query symbols, $K = \{Q_1, Q_2, \ldots, Q_n\}$. The sets N, T, K are mutually disjoint; let $V_\Gamma = N \cup K \cup T$. $G_i = (N \cup K, T, R_i, S_i), 1 \leq i \leq n$ are Chomsky grammars. The grammars $G_i, 1 \leq i \leq n$, represent the components of the system. The indices $1, \ldots, n$ of the symbols in K point to $G_1, \ldots, G_n$, respectively.

PCGS derivations consist of sequences of rewriting steps and communication steps. Communication has priority over rewriting, so a rewriting step is impossible if a query symbol (that requests communication) is present.

Definition 2.2. DERIVATION IN A PCGS [4]: *Let $\Gamma = (N, K, T, G_1, \cdots, Gn)$ be a PCGS as above, and $(x_i, x_2, \ldots, x_n)$ and $(y_i, y_2, \ldots, y_n)$ be two n-tuples with $x_i, y_i \in V_\Gamma^*$, $1 \leq i \leq n$. We write $(x_i, \ldots, x_n) \Rightarrow (y_i, \ldots, y_n)$ iff one of the following two cases holds:*

1. *$|x_i|_K = 0, 1 \leq i \leq n$, and for each $i, 1 \leq i \leq n$, we have $x_i \Rightarrow_{G_i} y_i$ (in the grammar G_i), or $x_i \in T^*$ and $x_i = y_i$.*
2. *There exists i, $1 \leq i \leq n$, such that $|x_i|_K > 0$. Then, for each such i, we write $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}$, $t \geq 1$, for $z_j \in V_\Gamma^*$, $|z_j|_K = 0$, $1 \leq j \leq t+1$. If $|x_{i_j}|_K = 0$, $1 \leq j \leq t$, then $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ [and $y_{i_j} = S_{i_j}, 1 \leq j \leq t$]. When, for some $j, 1 \leq j \leq t$, $|x_{i_j}|_K \neq 0$, then $y_i = x_i$. For all $i, 1 \leq i \leq n$, for which y_i is not specified above, we have $y_i = x_i$.*

We consider the PCGS as returning *if [and $y_{i_j} = S_{i_j}, 1 \leq j \leq t$] exists in the definition. Otherwise the system is* non-returning.

Like for grammars, we use $\Rightarrow$ to denote a derivation step (component-wise rewriting or communication). $\Rightarrow^*$ denotes a series of rewriting and communication steps (the reflexive and transitive closure of $\Rightarrow$). We also use (sparingly) the notation $\overset{\Lambda}{\Rightarrow}$ to explicitly identify a communication step.

In other words, an n-tuple $(x_1, \ldots, x_n)$ yields $(y_1, \ldots, y_n)$ if:

1. If there is no query symbol in $x_1,\ldots,x_n$, then the derivation is component-wise ($x_i \Rightarrow_{G_i} y_i, 1 \leq i \leq n$). In this case, one rule is used per component G_i, unless x_i is all terminals ($x_i \in T^*$). In this case, the string does not change ($y_i = x_i$).
2. If a query symbols appears, then a communication step has priority and should happen. For this, each query symbol Q_j in x_i is replaced by x_j, if and only if x_j does not contain query symbols. In other words, in a communication step the query symbol Q_j is replaced by the string x_j; the result of this replacement is referred to as Q_j being *satisfied* (by x_j).

   The grammar G_j continues processing from its axiom or from x_j depending on whether the system is returning or non-returning, respectively.

   No rewriting step can take place unless all query symbols are satisfied during a communication step. If some of them remains, the next step will perform another communication to satisfy them. The replacement string must not contain query symbols.

We have two ways in which the derivation can be blocked [4, 14, 15, 21]:

1. if component x_i of the current n-tuple $(x_1, \ldots, x_n)$ does not contain a nonterminal that can be rewritten in G_i, or

2. if a circular query appears; expressly, if G_{i_1} queries Q_{i_2}, G_{i_2} queries Q_{i_3}, and so on until $G_{i_{k-1}}$ queries Q_{i_k} and G_{i_k} queries Q_{i_1}, as the communication step has higher priority, and no communication is possible because only strings without query symbols can be communicated, then a derivation will be impossible.

Definition 2.3. LANGUAGES GENERATED BY PCGS [4]: *The language generated by a PCGS Γ is $\mathcal{L}(\Gamma) = \{w \in T^* : (S_1, S_2, ..., S_n) \Rightarrow^* (w, \sigma_2, ..., \sigma_n), \sigma_i \in V_\Gamma^*, 2 \leq i \leq n\}$.*

The derivation begins from the tuple of axioms $(S_1, S_2, ..., S_n)$. G_1 and creates a terminal string after one or more rewriting and/or communication steps are performed. The end result is a terminal string produced by the master grammar. Note that the form of the strings in the slave grammars is not restricted.

Definition 2.4. PCGS SEMANTICS [21]: *In a centralized PCGS only the first component grammar G_1 can introduce query symbols. If on the other hand any component can initiate communication we call the system non-centralized.*

A PCGS is called a returning system when the component grammars return to their axiom after a communication step. Whenever the grammars continue the derivation from the current string they are non-returning.

A system is synchronized when each component grammar uses exactly one rewriting rule in each component-wise derivation step (with the exception of a component grammar holding a terminal string, which is not modified). In a non-synchronized system each component may chose to either rewrite or wait in any step that is not a communication step.

The family of languages generated by a non-centralized, returning PCGS with n components of type X (where X is an element of the Chomsky hierarchy) will be denoted by $PC_n(X)$. The language families generated by centralized PCGS will be represented by $CPC_n(X)$. The fact that the PCGS is non-returning will be indicated by the addition of an N, thus obtaining the classes $NPC_n(X)$ and $NCPC_n(X)$. Let M be a class of PCGS, $M \in (PC, CPC, NPC, NCPC)$; then we define:

$$M(X) = M_*(X) = \bigcup_{n \geq 1} M_n(X)$$

The language generated by a PCGS Γ is denoted by $L(\Gamma)$ when Γ works as a synchronized system and by $L_u(\Gamma)$ when Γ works in an unsynchronized manner.

Space-bounded-Turing machines will be used to define the computational complexity of certain classes of CF-PCGS.

Definition 2.5. SPACE-BOUNDED TURING MACHINE [20]: *Given a Turing machine M and an input string $x \in T^*$, the working space of M on x is the sum of the lengths of all the work tapes of M to accept x. More generally, let S be any function from $\mathbb{N}$ to $\mathbb{N}$ and let $L \subseteq T^*$. We say that M semi-decides L in space S provided that M semi-decides L and uses at most $S(n)$ tape cells on any input of length n in T^*. If M is a nondeterministic Turing machine then we write $L \in NSPACE(S(n))$. We say also that M is an $S(n)$ space-bounded Turing machine.*

2.1 Examples of PCGS communication and coordination models

PCGSs can be classified based on their grammar structure, behaviour after satisfying a query, and timing manner.

Recall that a PCGS in which only one grammar can produce query symbols is called centralized. If two or more grammars can request for a string by query, we have a non-centralized PCGS.

As an example, the following PCGS is a centralized system, since only P_1 can initiate the queries:

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

By contrast, the following is a non-centralized PCGS, for indeed both P_1 and P_2 can initiate queries:

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

A PCGS can be either returning or non returning, which will affect the behaviour of the component strings after satisfying a query. After providing the string for the grammar which requested it by issuing the respective query symbol, the grammar which satisfied the query will continue the derivation from its axiom in a returning system. On the other hand, in a non-returning system, the grammar which provides the string satisfying the the query will continue the rewriting from the current form of the respective component.

Consider the following PCGS as an example:

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

Now consider the following derivation:

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

When Q_2 or Q_3 are satisfied, the components P_1 and P_2 return to their respective axiom. Thus, the system is returning. On the other hand, if the following derivation happens then the PCGS will be non-returning.

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

A PCGS is called synchronized if all the grammars perform exactly one rewriting during a global rewriting step (with the excetption of those grammars whose component strings do not feature any nonterminal). Conversely, if each grammar can either perform a rewriting or wait then the system is called unsynchronized. Note that in both cases a query symbol generates an immediate communication step in the same way for both the synchronized and unsynchronized systems.

Consider the following PCGS as an example:

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

The following derivation assumes that the system is synchronized:

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

By contrast,the following is a possible derivation when the system is considered unsynchronized.

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

Note incidentally that all the derivations in a synchronized system are also valid derivations in the unsynchronized version. The lack of synchronization can introduce additional derivations but cannot remove derivations.

Chapter 3

Previous Work

In this chapter, we review some of the previous results related to generative capacity of PCGS. Most research considers synchronized systems.

Context-sensative and recursively enumerable are the most powerful types, and they work similar to each other as well. First of all, it is immediate that a RE grammar and a PCGS with RE components are equally powerful so we have RE grammar: $\text{RE} = Y_n(\text{RE}) = Y_*(\text{RE})$, $n \geq 1$, for all $Y \in \{PC, CPC, NPC, NCPC\}$ [4].

To some degree, a similar situation holds for PCGS with context-sensitive components compared with context-sensitive languages: $CS = Y_n(CS) = Y_*(CS)$, $n \geq 1$, for $Y \in \{CPC, NCPC\}$ [4]. The non-centralized PCGS with context-free components are presumably more powerful.

This all being said, note that PCGS with CS components are not exactly useful since they rely on an computationally expensive model (the context-sensitive gramar). The most useful types use simpler, computationaly cheaper components. Consequently, the investigation or PCGS with regular or context-free components is more interesting.

The following result points that the class of languages which a centralized PCGS with regular components generates is a subset of the class of languages which are created by a non-centralized PCGS with regular components. We can conclude that a PCGS is generally more powerful than a single grammar component. Additionally, having more communications makes the system even more powerful: $CPC_n(\text{REG}) \subsetneq PC_n(\text{REG})$, $n > 1$ [21]. The same idea holds for PCGS with context-free components: $CPC_*(\text{CF}) \subseteq PC_*(\text{CF})$ [7]. Note however that the inclusion is not strict, so a non-centralized system is not necessarily strictly more powerful.

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 which can generated the same : $CPC_n(X) \subseteq PC_n(X)$ for any $n \geq 1$.

Certain subclass of centralized PCGS with regular components can generate at most the class of CF languages: If Γ is a regular, centralized or non-centralized, returning PCGS such that $\mathrm{com}(\Gamma) = 1$, then $\mathcal{L}(\Gamma)$ is context free [21]. These regular PCGS are more powerful in generative capacity than a regular grammar. However, their power is limited to the context-free languages.

The limitations of the generative power of PCGS can be investigated by the result below. When there are only two regular components, the languages generated by centralized, or even non-centralized PCGS are all context free.

- $CPC_2(\mathrm{REG}) \subsetneq \mathrm{CF}$ [4].
- $PC_2(\mathrm{REG}) \subseteq \mathrm{CF}$ [4].

The generative power of a system can be increased by increasing the number of components in the system as well. It has been shown that this does not change the generative capacity in the RE and to some degree in the CS case, but if we investigate classes that are lower in the hierarchy, it can be seen that an increase in the number of components usually increases the generative capacity of the system [4].

Another series of papers has examined the size complexity of returning and non returning CF systems. It was shown that returning CF-PCGS can generate any recursively enumerable language if the number of nonterminals in the system is less than or equal to a natural number k [3]. By simulating a 2-counter Turning machine, it has also been shown that non-returning CF-PCGS can generate the set of recursively enumerable languages with 6 context free components [6]. Further information of the Turing completeness of non-returning CF-PCGS is also available [6, 12]. Specifically, if $k \geq 2$ and $L \subseteq \{a_1, \ldots, a_k\}^+$ is a recursively enumerable language, then there is a non-returning CF-PCGS without ε-rules that generates L [6].

By considering the fact that the non-returning systems can be simulated by returning systems by assistance grammars holding intermediate strings [8], the results mentioned above [6, 12] also apply to returning systems. However, the number of components for this to happen will not remain the same, meaning that as far as we know a Turing-complete returning system is substantially more complex than a returning Turing complete CF-PCGS.

Unsynchronized PCGS have received substantially less attention. Available results on these systems include the following. It has been proven that $L_u(CPC_*LIN) \subseteq L(LIN)$ [4]. However, this is not true for the non-returning case and also for the non-centralized case. In particular, $L_u(NCPC_2CF)$ contains one-letter non-regular languages [4].

Several papers have shown that synchronization is useful, meaning that unsynchronized PCGS are weaker than the synchronized ones [16, 17]:

1. $L_u(CPC_*REG) = L(REG), L_u(CPC_*\ LIN) = L(LlN)$
2. $L_u(NCPC_2, REG) - L(CF) \neq \emptyset, L_u(CPC_2CF) - L(CF) = \emptyset$
3. $(L(CPC_*REG) \cap L(NCPC_*REG)) - (L_u(PC_*CF) \cup L_u(NPC * CF)) \neq \emptyset$

Unsynchronized PCGS have been also investigated elsewhere [19]. Some of the findings are:

1. $L_u(PC_2REG) \subseteq L(CF)$.
2. $L_u(PC_2LIN) - L(CF) \neq \emptyset$
3. $L_u(NCPC_2REG)$ contains non-semi-linear languages.
4. $L_u(NCPC_2CF)$ contains one-letter non-regular languages.

Chapter 4

Returning Unsynchronized CF-PCGS are Turing Complete

In this chapter we show that unsynchronized PCGS with context-free components are Turing complete. It has been proved that CF-PCGS are Turing complete by simulating an arbitrary 2-counter Turing machine [5]. To prove that unsynchronized CF-PCGS are Turing complete as well, we attempt to use the same construction but this time in an unsynchronized manner. We will show that the 2-counter machine must follow the same steps, as all the other variations introduced by the unsynchronized nature of the simulating PCGS will lead the system to a blocked state.

Overall we have the following:

Theorem 4.1. *Any recursively enumerable language can be generated by an unsynchronized returning PCGS with context-free components. Therefore this variant of PCGS is Turing complete.*

Proof. One proof for the fact that synchronized returning CF-PCGS are Turing complete [5] is based on constructing a PCGS that simulated an arbitrary 2-counter machine. This PCGS is shown in Figure 4. We consider the same PCGS, but this time working in an unsynchronized manner. We follow the original simulation, showing in the process that this simulation can only proceed in the original manner, all the alternative derivations introduced by the now unsynchronized nature of the system resulting in blocked derivations (that do not affect the language generated by the PCGS).

In what follows we refer to the PCGS in discussion (see Figure 4) as Γ. We further refer to its components by the names given to the respective set of rules in the figure.

We start from the initial configuration:

$$\Gamma = (S, S_1, S_2, S_3, S_4, S_1, S_2, S_3, S_4, S, S)$$

$$
\begin{aligned}
P_{GM} &= \{S \to [I], [I] \to C, C \to Q_{a_1}\} \cup \\
&\quad \{< 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 \\
&\quad \{< 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 \\
&\quad \{< 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, \\
&\quad x \in \Sigma, c'_1, c'_2 \in \{Z, B\}, e'_1, e'_2 \in \{-1, 0, +1\}\} \cup \\
&\quad \{< 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 | \\
&\quad (x, q, c_1, c_2, q', e_1, e_2, +1) \in R, c'_1, c'_2 \in \{Z, B\}, \\
&\quad 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 \\
&\quad \{[x, q, c_1, c_2, e_1, e_2] \to [e_1]', [+1]' \to AAC, [0]' \to AC, [-1]' \to C | \\
&\quad x \in \Sigma, q \in E, c_1, c_2 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \cup \\
&\quad \{[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 \\
&\quad \{[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, \\
&\quad 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 \\
&\quad \{[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] \\
&\quad [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 \\
&\quad \{[x, q, c_1, c_2, e_1, e_2] \to [e_2]', [+1]' \to AAC, [0] \to AC, [-1] \to C | \\
&\quad x \in \Sigma, q \in E, c_1, c_2 \in \{Z, B\}, e_1, e_2 \in \{-1, 0, +1\}\} \cup \\
&\quad \{[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 \\
&\quad \{[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], \\
&\quad [I] \to [I] | x \in \Sigma, q \in E, \\
&\quad 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 \\
&\quad \{[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] \\
&\quad [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 >, \\
&\quad < 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, \\
&\quad 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)}, \\
&\quad S^{(4)} \to Q_2^{c_1} Q_3^{c_1} Q_2^{c_2} Q_3^{c_2} S^{(1)}\}.
\end{aligned}
$$

Figure 4.1: A CF-PCGS with broadcast communication that simulates a 2-counter Turing machine [5].

After the first rewriting step, Γ can only enter the following configuration:

$$\Gamma = ([I], u_1, u_2, u_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, Q_m, S^{(3)})$$

Indeed, if P_{GM} waits we end up in the configuration

$$(S, u_1, u_2, u_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, Q_m, S^{(3)})$$

In $P_2^{C_1}$ u_1 eventually will become Q_m or $Q_4^{C_1}$. If it is Q_m then Q_m can be replaced by S, $[I]$, or C. Only C works, and then $P_2^{C_1}$ will return to Q_m. If $Q_4^{C_1}$ is introduced, then it can be replaced by S_4, $S_4^{(1)}$, $S_4^{(2)}$, or $Q_4^{C_1}$. In all cases the the system will block.

It should be mentioned on a more general note that the only two situations in which an unsynchronized system can block is because of circular queries, or because there are no rewriting rules for any of the components in the system that contain one or more nonterminals. The second situation is not technically a blocked state since the system can still be considered to be in a waiting state. However, since no possible continuation exists we will still call this kind of a situation blocking. We used the term "block" above in this sense, as will be the case throughout the remainder of this paper.

If on the other hand $P_1^{C_1}$ waits, then the resulting configuration is

$$([I], S_1, u_2, u_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, Q_m, S^{(3)})$$

In this situation, $P_4^{C_1}$ can only use a (possible chain of) the following rules:

$$S_4 \to S_4^{(1)}, S_4^{(1)} \to S_4^{(2)}, S_4^{(2)} \to Q_1^{C_1}$$

No matter how long (or short) this chain is, $P_1^{C_1}$ cannot perform any derivation and so the system will block.

If $P_2^{C_1}$ waits then we reach

$$([I], u_1, S_2, u_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, Q_m, S^{(3)})$$

In this situation $P_4^{C_1}$ can again use only a chain of the following rules

$$S_4 \to S_4^{(1)}, S_4^{(1)} \to S_4^{(2)}, S_4^{(2)} \to Q_1^{C_1}$$

and again no rewriting is possible in $P_1^{C_1}$.

Suppose now that $P_3^{C_1}$ waits. We then enter the following configuration:

$$([I], u_1, u_2, S_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, Q_m, S^{(3)})$$

which result in a blocked derivation just like the one caused by $P_2^{C_1}$ waiting.

If the waiting component is $P_4^{C_1}$ then the current configuration becomes

$$([I], u_1, u_2, u_3, S_4, u_1', u_2', u_3', S_4^{(1)}, Q_m, S^{(3)})$$

In this case, $A \to a$ can never happen, and so A can never be eliminated resulting in a failed (blocked) derivation.

Whenever $P_1^{C_2}$, $P_2^{C_2}$, $P_3^{C_2}$, or $P_1^{C_2}$ wait we end up in the same blocked configuration as the one resulting from $P_1^{C_1}$, $P_2^{C_1}$, $P_3^{C_1}$, or $P_1^{C_1}$ waiting, respectively.

If P_{a1} waits then we get:

$$([I], u_1, u_2, u_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, S, S^{(3)})$$

In this case the only way for P_{GM} to continue is by using $S \to [I]$ and $[I] \to C$. Then the derivation blocks, since $< I >$ never happens.

Finally, P_{a2} waiting results in the following configuration.

$$([I], u_1, u_2, u_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, Q_m, S)$$

The only effect is on γ in the end, which does not produce any new case.

According to the aforementioned cases, the first rewriting for all the grammars is unavoidably synchronized and so we reach the following configuration, just like in the original proof:

$$([I], u_1, u_2, u_3, S_4^{(1)}, u_1', u_2', u_3', S_4^{(1)}, Q_m, S^{(3)})$$

where u_1, u_2, u_3 are either $Q_m or Q_4^{c_1}$ and u_1', u_2', u_3' are either $Q_m or Q_4^{c_2}$. If any of these symbols is not Q_m, the system is blocked after the communication, so the next step is:

$$([I], Q_m, Q_m, Q_m, S_4^{(1)}, Q_m, Q_m, Q_m, S_4^{(1)}, Q_m, S^{(3)}) \overset{\Lambda}{\Rightarrow}$$
$$(S, [I], [I], [I], S_4^{(1)}, [I], [I], [I], S_4^{(1)}, [I], S^{(3)})$$

This in turn is followed by the following rewriting:

$$(S, [I], [I], [I], S_4^{(1)}, [I], [I], [I], S_4^{(1)}, [I], S^{(3)}) \Rightarrow$$
$$([I], [I]', [I], [I], S_4^{(2)}, [I]', [I], [I], S_4^{(2)}, < I >, S^{(4)})$$

In this rewriting step, if P_{GM} waits then we obtain:

$$(S, [I]', [I], [I], S_4^{(2)}, [I]', [I], [I], S_4^{(2)}, < I >, S^{(4)})$$

After three rewriting steps, P_1, P_2, and P_3 turn to aQ_m, and they do not have S in their grammars, so the system blocks.

If $P_1^{C_1}$ waits then we obtain:

$$([I],[I],[I],[I],S_4^{(2)},[I]',[I],[I],S_4^{(2)},<I>,S^{(4)})$$

Eventually, Q_1 will appear in $P_4^{C_1}$. This grammar does not have any definition for [I], so the system blocks.

Since $P_2^{C_1}, P_3^{C_1}$ do not change in this step, they do not cause any new case if they wait.

If $P_4^{C_1}$ waits then we get:

$$([I],[I]',[I],[I],S_4^{(1)},[I]',[I],[I],S_4^{(2)},<I>,S^{(4)})$$

In this case, $A \to a$ never happens, so A can never be eliminated from the configuration and so the derivaton cannot result in a string in the language.

If any of $P_1^{C_2}$, $P_2^{C_2}$, $P_3^{C_2}$, or $P_4^{C_2}$ wait, then we end up in the same situation as $P_1^{C_1}$, $P_2^{C_1}$, $P_3^{C_1}$, or $P_4^{C_1}$ waiting, respectively.

If P_{a1} waits then we obtain:

$$([I],[I]',[I],[I],S_4^{(2)},[I]',[I],[I],S_4^{(2)},[I],S^{(4)})$$

However, P_{GM} only has the rewriting rules $S \to [I]$ and $[I] \to C$ available and so blocks, since $<I>$ never happens.

If P_{a2} waits then we get:

$$([I],[I]',[I],[I],S_4^{(2)},[I]',[I],[I],S_4^{(2)},<I>,S^{(3)})$$

The only effect is on γ in the end, which does not produce any new case.

According to the discusion above, we have again only one way to proceed, as follows:

$$\begin{aligned}&([I],[I]',[I],[I],S_4^{(2)},[I]',[I],[I],S_4^{(2)},<I>,S^{(4)}) \Rightarrow\\&(C,AC,[I],[I],Q_1^{C_1},AC,[I],[I],Q_1^{C_2},<I>,Q_2^{C_1}Q_3^{C_1}Q_2^{C_2}Q_3^{C_2}S^{(1)})\end{aligned}$$

Next comes a communication step, which is the same no matter whether the system is synchronized or not (since communication always has priority over componentwise derivation):

$$\begin{aligned}&(C,AC,[I],[I],Q_1^{C_1},AC,[I],[I],Q_1^{C_2},<I>,Q_2^{C_1}Q_3^{C_1}Q_2^{C_2}Q_3^{C_2}S^{(1)}) \overset{\wedge}{\Rightarrow}\\&(C,S_1,S_2,S_3,AC,S_1,S_2,S_3,AC,<I>,[I][I][I][I]S^{(1)})\end{aligned}$$

The next rewriting step proceeds as follow:

$$\begin{aligned}&(C,S_1,S_2,S_3,AC,S_1,S_2,S_3,AC,<I>,[I][I][I][I]S^{(1)}) \Rightarrow\\&(Q_{a1},u_1,u_2,u_3,aC,u_1',u_2',u_3',aC,<I>,[I][I][I][I]S^{(2)})\end{aligned}$$

In this rewriting step, if P_{GM} waits then we get:

$$(C, u_1, u_2, u_3, aC, u'_1, u'_2, u'_3, aC, < I >, [I][I][I][I]S^{(2)})$$

if $u = Q_m$ then the system will block at the next step. However, if $u = Q_4$ then the system will continue and the subsequent communication step will introduce either $S_4^{(1)}$ C, $< I >$, or Q_m. In all these cases the system will block.

If $P_1^{C_1}$ waits then we obtain the following configuration:

$$(Q_{a1}, S_1, u_2, u_3, aC, u'_1, u'_2, u'_3, aC, < I >, [I][I][I][I]S^{(2)})$$

After three steps, $Q_1^{C_1}$ appears in $P_4^{C_1}$ and the system blocks.

If $P_2^{C_1}$ waits we get:

$$(Q_{a1}, u_1, S_2, u_3, aC, u'_1, u'_2, u'_3, aC, < I >, [I][I][I][I]S^{(2)})$$

Eventually, $P_2^{C_1}$ will be assigned to P_{a_2}, so γ' will contain S_2 and then the system blocks.

If $P_3^{C_1}$ waits, the situation will be the same as $P_2^{C_1}$ waiting.

If $P_4^{C_1}$ waits then we obtain:

$$(Q_{a1}, u_1, u_2, u_3, aC, u'_1, u'_2, u'_3, AC, < I >, [I][I][I][I]S^{(2)})$$

In the next step $P_2^{C_1}$ will turn into $Q_4^{C_1}$ and then AC will replace $P_2^{C_1}$. Since this grammar does not have any rewriting rule for that string, the system blocks.

If any of $P_1^{C_2}$, $P_2^{C_2}$, $P_3^{C_2}$, or $P_4^{C_2}$ wait, the situation would be the same as $P_1^{C_1}$, $P_2^{C_1}$, $P_3^{C_1}$, or $P_4^{C_1}$ waiting, respectively.

P_{a1} does not change in this step, so waiting will not affect on the system.

If P_{a2} waits then we get:

$$(Q_{a1}, u_1, u_2, u_3, aC, u'_1, u'_2, u'_3, AC, < I >, [I][I][I][I]S^{(1)})$$

This does not produce any new case.

Once more, the only way to continue is through a synchronized rewriting step. Note in addition that u_1, u_2, and u_3 are either Q_m or Q_4^{C1}, and u'_1, u'_2, and u'_3 are either Q_m or Q_4^{C2}. If any of these strings is Q_m then the system is blocked after the communication. In all we have

$$(Q_{a1}, Q_4^{C1}, Q_4^{C1}, Q_4^{C1}, aC, Q_4^{C2}, Q_4^{C2}, Q_4^{C2}, AC, < I >, [I][I][I][I]S^{(2)}) \overset{\wedge}{\Rightarrow}$$
$$(< I >, aC, aC, aC, S_4, aC, aC, aC, S_4, S, [I][I][I][I]S^{(2)})$$

The next rewriting step is as follow:

$$(< I >, aC, aC, aC, S_4, aC, aC, aC, S_4, S, [I][I][I][I]S^{(2)}) \Rightarrow$$
$$(u[x', q, Z, Z, e_1, e_2], aQ_m, aQ_m, aQ_m, S_4^{(1)}, aQ_m, aQ_m, aQ_m, S_4^{(1)}, Q_m, [I][I][I][I]S^{(3)})$$

In this rewriting step, if P_{GM} waits then we end up in the following configuration:

$$(< I >, aQ_m, aQ_m, aQ_m, S_4^{(1)}, aQ_m, aQ_m, aQ_m, S_4^{(1)}, Q_m, [I][I][I][I]S^{(3)})$$

None of $P_1^{C_1}$, $P_2^{C_1}$, $P_3^{C_1}$, $P_1^{C_2}$, $P_2^{C_2}$, and $P_3^{C_2}$ have rules for $< I >$, so system blocks.

If $P_1^{C_1}$ waits then we get:

$$(u[x', q, Z, Z, e_1, e_2], aC, aQ_m, aQ_m, S_4^{(1)}, aQ_m, aQ_m, aQ_m, S_4^{(1)}, Q_m, [I][I][I][I]S^{(3)})$$

Eventually, $aC \Rightarrow aQ_m$ and Q_m can only be satisfied by S, $[I]$, or C. if P_m becomes aS then the system blocks. if P_m becomes aC then we are back where we started, which results in a loop. if P_m becomes $a[I]$ then in the next steps, $a[I]$ will replace $Q_1^{c_1} at P_4^{c_1}$, and that grammar does not have a rule for $a[I]$ so the system blocks.

If $P_2^{C_1}$ or $P_3^{C_1}$ waits then we obtain:

$$(u[x', q, Z, Z, e_1, e_2], aQ_m, aC, aC, S_4^{(1)}, aQ_m, aQ_m, aQ_m, S_4^{(1)}, Q_m, [I][I][I][I]S^{(3)})$$

Eventually, $aC \Rightarrow aQ_m$ and Q_m can be satisfied by either S, $[I]$, or C. As above the system blocks for S, and enters a loop for C. P_m becoming $a[I]$ only affects γ' in the end.

If $P_4^{C_1}$ waits then we get:

$$(u[x', q, Z, Z, e_1, e_2], aQ_m, aQ_m, aQ_m, S_4, aQ_m, aQ_m, aQ_m, S_4^{(1)}, Q_m, [I][I][I][I]S^{(3)})$$

This will not add or remove any string.

If any of $P_1^{C_2}$, $P_2^{C_2}$, $P_3^{C_2}$, or $P_4^{C_2}$ waits, then the situation would be the same as $P_1^{C_1}$, $P_2^{C_1}$, $P_3^{C_1}$, $P_4^{C_1}$ waiting, respectively.

If P_{a1} waits then we obtain:

$$(u[x', q, Z, Z, e_1, e_2], aQ_m, aQ_m, aQ_m, S_4^{(1)}, aQ_m, aQ_m, aQ_m, S_4^{(1)}, S, [I][I][I][I]S^{(3)})$$

Eventually, S will rewrite to Q_M which in turn can be replaced by S, $[I]$, or C. We obtain a loop for S. The system blocks for C. If P_m becomes $[I]$ then $[I] \Rightarrow < I > \Rightarrow u[x', q, Z, Z, e_1, e_2]$ and this will be passed to $P_1^{C_2}$, $P_2^{C_2}$, $P_3^{C_2}$, $P_4^{C_2}$, $P_1^{C_1}$, $P_2^{C_1}$, $P_3^{C_1}$, and $P_4^{C_1}$. Neither of these components have any rewriting rule for that, so the system blocks.

If P_{a1} waits then the resulting configuration is:

$$(u[x', q, Z, Z, e_1, e_2], aQ_m, aQ_m, aQ_m, S_4^{(1)}, aQ_m, aQ_m, aQ_m, S_4^{(1)}, Q_m, [I][I][I][I]S^{(2)})$$

This configuration does not produce any new case.

Given the observations above it is obvious that the unsynchronized PCGS will follow exactly the same steps as the synchronized PCGS and so will proceed in exactly the same way toward correctly simulating the 2-counter machine. Thus unsynchronized returning CF-PCGS can generate all recursively enumerable languages. □

Chapter 5

Non-returning Unsynchronized CF-PCGS are Linear Space

We now study the computational complexity of unsynchronized non-returning CF-PCGS based on space-bounded Turing machines. We will show that in the absence of ε-rules languages generated by unsynchronized non-returning CF-PCGSs can be recognized by nondeterministic Turing machines using $O(|w|)$ tape cells for each input instance w. We adapt for this purpose an earlier proof for synchronized systems [1]. That proof was based on the existence of a threshold on the number of occurrences of a nonterminal in a long enough string, above which the number of occurrences of that nonterminal is no longer relevant to the overall derivation in the respective system. It was later shown that this threshold does not actually exist, and so the original proof was invalidated [2]. However, we are able to show that the respective threshold in unsynchronized systems is effectively 1, meaning that a nonterminal appearing in a (long enough) string is important, but that nonterminal subsequently disappearing is not. Therefore a variant of the original proof holds in the unsynchronized case.

Definition 5.1. *During a derivation process in a PCGS, a component of the current configuration x_i is called non-direct-significant for the recognizing of the string w if*

1. *either $i \neq 1$ and x_i is not queried anymore 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 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.

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). This definition introduces the class of components for which the structure is irrelevant for the derivation.

Lemma 5.1. *Let $\Gamma = (N, K, T, G_1, ..., G_n)$ be an unsynchronized non-returning PCGS and w a string. Let also $(x1, ..., x_n)$ 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 recognizing of w.*

Proof. We consider two situations:

Let $i = 1$. If $|x_1|_K = 0$, then x_1 will be rewrited using the rules of G_1. But these are context-free rules and there are not ε-productions, so the length of x_1 does not decrease. If $|x_1|_K \neq 0$, a communication step will be performed. But the communication step does not reduce the length of the component because there are not null components to be queried (there are not ε-productions). So, the length of x_1 does not decrease anymore and this leads to the rejection of w. Note that querying a component does not result in the length of the respective component to be reduced since the system is non-returning. Therefore x_1 is non-direct-significant according to the definition 5.1.

For $i \geq 2$, either the component x_i is never queried, therefore it is non-direct-significant, or it is queried by the first component the length of x_1 becomes greater than $|w|$, therefore x_1 becomes non-direct-significant (according to the point 1). So x_i is non-direct-significant. or it 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. □

Theorem 5.2. *Let Γ be an unsynchronized 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 $\Gamma = (N, K, T, G_1, ..., G_n)$ 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 (with no ε-rules). We will construct the 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. The alphabet of the tape of M is $N \cup K \cup T \cup @, @ \neq N \cup K \cup T$. Given an input string w, M will simulate step by step the derivation of w by Γ.

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 and will work on it as follows:

1. 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
   (a) select a rule from the rule set P_i and rewrites x_i according to this rule, or
   (b) remains unchanged.

If there are some i for which $|x_i|_N = 0$ or such rule does not exist, that specific i has to choose the second option. If $|x_i| > |w|$ then, according with Lemma 5.1, x_i become non-direct-significant. Therefore its structure is irrelevant and it will be replaced by the string

$$@T_1...T_jQ_1...Q_k \tag{5.1}$$

where @ is a special symbol $(@ \neq N \cup T \cup K)$, $T1, ..., T_j$ are the distinct nonterminals in x_i and $Q_1, ..., Q_k$ are the distinct query symbols in x_i, each appearing $q_1, q_2, \ldots, q_k$ times, respectively. We have to explain now how the rewriting works on strings of the form 5.1. Let the rewriting rule be

$$A \rightarrow A_1A_2...A_m \tag{5.2}$$

where $A \in N, A_1, ..., A_m \in N \cup K$. Then, there is $T_r = A$, $1 \leq r \leq j$, (if not, the rule is not applicable). If A_j does not already exists in x_i, then it is added to x_i. Since it is an unsynchronized system, each grammar can wait optionally for an arbitrary amount of time. In fact, the possibility of waiting is the only addition to unsynchronized systems over their synchronized counterpart. Accordingly, we can assume that each non-terminal appears infinite times without modifying the overall language generated by the PCGS.

2. 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 5.2 or, after replacement, x_i becomes longer than $|w|$, then x_i becomes non-direct-significant, so it will be replaced by a string of the form 5.2.

   This communication step is repeatedly performed until there are no query symbols in the current configuration.

   In non-direct-significant components we argue that the number of occurrences of a query symbol does not matter since all the occurrences will be satisfied in the same way and the respective component will remain non-direct-significant. In other words, the only thing a query symbol can do in a non-direct-significant component is to block the derivation (or not). This effect happens no matter how many times the respective symbol appears in the component and so we do not need to maintain a count on the number of occurrences (needing to satisfy a query symbol is important even in non-direct-significant components, but how many times the query needs to be satisfied is not).

M repeats steps of type 1 and 2 until the w and x_i are identical, when M accepts the input and halts.

Note that counterintuitively no nonterminal symbol is ever erased from a non-direct-significant component. This might appear to cause a problem in circumstances such as a nonterminal A being rewritten into another nonterminal B that will influence the derivation differently from A (for example by blocking further rewriting in contrast with A that would allow the derivation to continue). In a normal component the occurrence of A is replaces by B (so A disappears), whereas in our representation of non-direct-significant components once A is replaces by B we keep *both* A and B in the string. We argue that the continuing presence of A does not affect the possible derivations as follows: If A were not present, the derivation in the respective component would block. However, this does not have any influence in the overall derivation of the system, 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 (for the respective component) does not result in an overall blocked derivation. Therefore the fact that A is still present (and so an inadvertent escape from blocking) is irrelevant. It is also the case that A is not required to rewrite itself into B at any fixed time; not applying the respective rewriting rule for an arbitrary number of rewriting steps is also an option. Overall, A and B can both appear in the string at any moment of the global derivation. That is, in non-direct-significant components A and B appearing sequentially one after the other (like in the real derivation) has the same effect as the two nonterminals appearing simultaneously (as in our representation).

Let us count the amount of work space used by M during the derivation. In the worst case, it become of the form 5.1. Because we have a fixed finite number t of nonterminals for a given PCGS and exactly n query symbols, the length of such component on the tape is independent of $|w|$ and is less than $1 + (t + n)$, where $t = \mathbb{N}$.

A communication step may use temporarily an amount of tape space double than the space used by a single component temporarily (e.g. a string of length $|w|$ is queried by another string of length $|w|$; before the reduction to form 5.1 we have to use $2|w|$ tape cells). Therefore, the number of cells used by a component is less than $2\max(|w|, 1 + (t + n))$. We have n components and we need some extra space on the tape to keep the rules of the system and $|w|$ which we denote it by P_l. So, the space used by M is upper-bounded by $2n\max(|w|, 1 + (t + n)) + P_l$. However, neither of t, n, or P_l are $|w|$-dependent, so the overall space used is linear in $|w|$, as desired. □

Corollary 5.3. *All languages generated by unsynchronized non-returning PCGS with context-free components and no ε-rules are context sensitive.*

Proof. This result follows directly from Theorem 5.2 given that all context-sensitive languages can be accepted in linear space [20]. □

Chapter 6

Conclusion

PCGS introduce an inherently concurrent model in formal languages. A possible longer term interest to exploit the model in general (and CF-PCGS specifically) in formal methods is interesting precisely because of this inherent parallelism. However, some formal language questions have to be investigated before starting to use it, one of which being the generative power of CF-PCGS.

According to the previous results related to the expressiveness of synchronized CF-PCGS, these PCGS are Turing complete. However, neither of them investigated the unsynchronized CF-PCGS. We attempted here to remedy this situation.

First, we investigated one system designed earlier to show Turing completeness in the synchronized case [5]. We proceeded to follow the system as it works in an unsynchronized manner. We considered all other variants that an unsynchronized system can fall into. In order to do this we checked what happens if one or multiple grammars wait and the rest of the system continues the process. We concluded that these extra possible derivations fall into one of the following three scenarios:

1. The system will block. Indeed, most of the cases will result in blocking the system. For example, another grammar will query for the string from the waiting grammar, but that grammar did not thus reach the intended string. The new string causes the component that issued the query to be unable to continue.

2. A loop happens and a grammar follows a number of rewriting steps to go back to the string it started from. In this case, the process will continue as in the synchronized case.

3. The new variant does not create any new output or does not affect the final result.

Using these observations we were able to construct a CF-PCGS capable of simulating an arbitrary 2-counter Turing machine, and so show that CF-PCGS are indeed Turing complete.

In addition, we showed that unsynchronized non-returning CF-PCGS can be simulated in linear space. For this purpose we identified components that exceed the length of the desired string (the non-direct-significant ones) and we provided a compact representation for them, thus reducing the space requirements to $O(n)$.

It would seem that synchronization has the most effect on non-returning systems. On one hand, unsynchronized returning CF-PCGS are as powerful as their synchronized version. We can then conclude that the returning option makes the system so powerful that the lack of synchronization cannot weaken it. On the other hand, when we eliminate synchronization in a non-returning system we end up with a CF-PCGS that is considerably weaker. It should be noted however that we impose one extra restriction in the non-returning case namely, the absence of ε-rules. It is easy to see that this restriction is central to the proof. Whether the difference between Turing completeness and linear space is given by the returning nature of the system or by the absence of ε-rules remains to be seen.

Bibliography

[1] S. D. BRUDA, *On the computational complexity of context-free parallel communicating grammar systems*, in New Trends in Formal Languages, G. Paun and A. Salomaa, eds., vol. 1218 of Lecture Notes in Computer Science, Springer, 1997, pp. 256–266.

[2] 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.

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

[4] 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.

[5] 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.

[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, *Nonreturning 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 contraints*, 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 parallel communicating grammar systems with context-free components*, in Mathematical Linguistics and Related Topics, The Publishing House of the Romanian Academy of Science, 1994, pp. 258–270.

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

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

[16] G. PAUN, *On the power of synchronization in parallel communicating grammar systems*, Stud. Cerc. Matem, 41 (1989).

[17] G. PAUN, *On the synchronization in parallel communicating grammar systems*, Acta Informatica, 30 (1993), pp. 351–367.

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

[19] G. PAUN, S. VICOLOV, AND A. SALOMAA, *On the generative capacity of parallel communicating grammar systems*, International journal of computer mathematics, 45 (1992), pp. 49–59.

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

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