PARSE TREES, INTERFERENCE, AND MIXED NODES FOR CONTEXT-FREE PARALLEL COMMUNICATING GRAMMAR SYSTEMS

by

MEHRDAD NASERDOUST

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 2020

Abstract

Parallel Communicating Grammar Systems (PCGS) were introduced for a better understanding of concurrent systems on a language-theoretic level. Some research has been done regarding relationship between PCGS and practical computing systems, though this has not received sustained attention. In another more practical attempt, parse trees for PCGS have been investigated. It is relatively straightforward to construct a parse tree for every PCGS derivation, but going the other way around (reconstructing a derivation for every PCGS parse tree) is not always possible. The possibility of doing this has been found to depend on the property of interference in a PCGS. We refine this result, showing that there is one extra property that needs to be taken into account. We show that even in the absence of interference there exist parse trees that cannot correspond to any derivation when strings containing nonterminals are communicated between PCGS components. This reduces the utility of PCGS parse trees even further than previously thought.

Acknowledgments

I would like to thank the Computer Science department at Bishop's University for giving me the opportunity to pursue a Master's degree. I would like to underline the support, patience and guidance received from Dr. Stefan D. Bruda, without him none of this would have been possible. The door to Dr. Bruda's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

I also want to express my gratitude to my family and friends for their continued support and encouragement, supporting me emotionally throughout my studies.

Contents

List of Figures

Chapter 1

Introduction

Parallel Communicating Grammar Systems (PCGS) have been introduced as a language-theoretic treatment of concurrent (or more generaly, multi-agent) systems [7, 20]. A PCGS contains different component grammars that work in parallel on separate strings. Components may or may not synchronize with each other, but they also cooperate by communicating with each other. More precisely, components may request strings generated by others. Synchronization and communication facilities of PCGS whose components are of a certain type are generally more powerful than a single grammar of the same type [5, 7, 10, 20, 21].

Specifically, PCGS whose components are context-free grammars (or just CF-PCGS for short) appear to be capable of offering convenient models for the control flow of the concurrent execution of several threads running complex, possibly recursive code. Recursion in particular generates an infinite state space when classical verification techniques such as model checking or model-based testing are used. Therefore, the analysis of such complex, concurrent and recursive processes need to be considered on a more abstract, non-regular level. A straight move to the context-free level however will not solve the problem completely. Indeed, context-free process algebrae such as the basic process algebra or BPA [2] can specify such context-free properties, but their ability to model concurrency is limited by the basic fact that context-free languages are not closed under intersection [17]. This limitation has generated an investigation into visibly pushdown languages [1], culminating with multi-stack visibly pushdown languages [3, 16] and eventually a process algebra [18].

This approach however is on the automata side and therefore they are inherently inconvenient for specifying systems. Because of all these reasons PCGS with context-free components appear to be an excellent basis for developping a model for the specification of recursive, concurrent systems.

The motivation of PCGS is claimed to be the study of concurrent systems. Relatively recently some links with practice have been attempted [15, 4]. Parse trees in particular are one of the most important tool for context-free languages; they are used for important proofs such as the pumping lemma, but most importantly they form the basis of programming language compilers. Parse trees have the potential to have similar importance for PCGS with context-free components. This is supported by the fact that they continue to exist in the context of context-free PGCS derivations; indeed, any context-free PCGS derivation has an associated parse tree [4].

Going the other way around and offering a full characterization of context-free PCGS derivations using parse trees turns out to depend on the two particular notions of "interference" [4] and "mixed meta-nodes" (Chapter 4). The notion of interference has already been investigated, but the notion of mixed meta-nodes was overlooked. We therefore focus now on the mixed meta-nodes and we find that they impose further limitations for characterizations of derivations by parse trees. Existence of mixed meta-node in a PCGS can cause a parse tree to grow in such a way that we cannot find the corresponding derivation for it.

In all, it turns out that a complete characterization of derivations in CF-PCGS is even more limited than previously thought. Indeed, such a full characterization is only possible if the notion of interference and mixed meta-nodes are both absent. Such a property thus only holds for one very restricted context-free PCGS variant (Chapter 5). One consequence of the existence of such a characterization is in the realm of generative power, namely that this variant of context-free PCGS is the weakest of them all.

Parse trees are a powerful tool for having a better understanding of context-free PCGS. Our results regarding the relationship between these two notions is useful in a more abstract way in a possible quest toward grammatical approaches to formal verification of concurrent and recursive systems. Some of these results [4] suggest that synchronized context-free PCGS are unnecessarily

powerful for our purpose. It would appear that the unsynchronized variant has all the ingredients needed for modeling such systems and at the same time it is less complex. In light of our results combined with on earlier attempt [4], further restrictions on the PCGS being used as a model for a possible process algebra might be contemplated.

The rest of this thesis is organized as follows. We start by summarizing the necessary preliminaries in Chapter 2. The parse trees for context-free PCGS were introduced and analyzed earlier [4], there the notions of meta-tree and meta-node were also introduced; we summarize these results in Chapter 3. We also introduce here the notion of interference and its consequences on the relation between parse trees and derivations in synchronized as well as non-returning CF-PCGS. Our contribution starts in Chapter 4, where we show that the original definition of meta-nodes is too restricted and so we relax it. We also introduce the notion of mixed meta-nodes here. In Chapter 5 we show how the existence of mixed meta-nodes cause the existence of parse trees that do not correspond to any derivation. We are thus able to identify the class of CF-PCGS for which a complete characterization of derivations using parse trees is possible. We conclude in Chapter 6.

Chapter 2

Preliminaries

In this section we briefly recall some basic notions regarding parallel communicating grammar systems which are necessary to follow the paper. We assume that the reader is familiar with formal language theory, so we only specify some notations. For further details consult [17].

For some alphabet (i.e., finite set) V, some word $x \in V^*$, and a set $U \subseteq V$, we denote the number of occurrences of elements from U in x by $|x|_U$. By abuse of notation we write $|x|_a$ instead of $|x|_{\{a\}}$ for singleton sets $U = \{a\}$. The symbol ε will be used to denote the empty string, and only the empty string.

2.1 Context-Free Grammars

A *grammar* [17] is a tuple $G = (N, \Sigma, R, S)$ where N describes a finite set of nonterminals. Σ describes a finite set of terminals. $S \in N$ is the axiom or start symbol, and $R \subseteq ((N \cup \Sigma)^* N (N \cup \Sigma)^*) \times (N \cup \Sigma)^*$ describes a finite set of productions or (rewriting) rules where each rule maps a nonterminal to a string $S \in (N \cup \Sigma)^*$. A rewriting rule (σ, σ') is customarily written $\sigma \to \sigma'$. A rewriting step replaces a string $w = u\sigma v$ with $w' = u\sigma' v$ whenever $\sigma \to \sigma' \in R$ (written $w \Rightarrow_G w'$, though we often omit the subscript G whenever the grammar is understood from the context). A grammar G generates the language $\mathscr{L}(G)$ of exactly all the strings that can be derived from the axiom S, i.e., $\mathscr{L}(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$, where $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$ and is called a derivation. A *context-free grammar*, in particular, is a grammar with $R \subseteq N \times (N \cup \Sigma)^*$.

Definition 1 ([17]). *A* parse tree *for a context-free grammar $G = (N, \Sigma, R, S)$ is a tree whose nodes are labelled with symbols from the set $N \cup \Sigma$. It is defined inductively as follows (based on Figure 3.1(a–d) on page 12): For every $a \in N \cup \Sigma$ the tree depicted in Figure 3.1(a) is a parse tree with* yield *a; for every $A \to \varepsilon \in R$ the tree from Figure 3.1(b) is a parse tree with yield ε. Suppose that the n trees from Figure 3.1(c) are parse trees with yields $y_1, y_2, \ldots, y_n$ and that $A \to A_1 A_2 \ldots A_n \in R$; then the tree shown in Figure 3.1(d) is a parse tree with yield $y_1 y_2 \ldots y_n$.*

Note that the yield of a parse tree is the sequence of leaf labels as obtained by an inorder traversal of the tree. For every parse tree with root S and yield w there exists a derivation $S \Rightarrow^* w$ (and the other way around) in a natural way [17].

2.2 Parallel Communicating Grammar Systems

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 allows us to investigate properties of parallel systems.

Definition 2 ([7]). *A PCGS is an $(n+3)$ tuple $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ for some $n \geq 1$, where N is nonterminal alphabet, K is the set of query symbols, $K = \{Q_1, Q_2, \ldots, Q_n\}$, and Σ is a terminal alphabet. The sets N, Σ, and K are mutually disjoint. $G_i = (N \cup K, \Sigma, R_i, S_i)$, $1 \leq i \leq n$ are grammars; they represent the components of the system. The indices $1, \ldots, n$ of the symbols in K point to $G_1, \ldots, G_n$, respectively.*

A derivation in PCGS consists of applying a series of rewriting rules in each component and reacting to communication requests between components. Communication happens as a result of introducing query symbols during the rewriting process, and has always priority over rewriting. In other words no rewriting can happen in a configuration featuring query symbols (meaning, communication requests that are not yet satisfied).

Definition 3 ([7]). *Let $\Gamma = (N, K, T, G_1, \cdots, G_n)$ be a PCGS as above, and $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_n)$ be two n-tuples with $x_i, y_i \in (N \cup T \cup K)^*$, for every $1 \leq i \leq n$. We write $(x_1, \ldots, x_n) \Rightarrow$*

$(y_1,\ldots,y_n)$ *iff one of the following two cases holds:*

1. $|x_i|_K = 0$ *for every* $1 \leq i \leq n$, *and for each* $1 \leq i \leq n$, *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* $1 \leq i \leq n$, *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* $z_j \in (N \cup T \cup K)^*$, $|z_j|_K = 0$, $1 \leq j \leq t+1$. *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* $y_i = x_i$.

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

1. There is no query symbol in $x_1,\ldots,x_n$, then we have a component-wise derivation ($x_i \Rightarrow_{G_i} y_i$,for every $1 \leq i \leq n$, which means that one rule is used per component G_i), unless x_i is all terminals ($x_i \in T^*$) in which case it remains unchanged ($y_i = x_i$).

2. We have query symbols and so a communication step is required. When this occurs 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, a communication step involves the query symbol Q_j being replaced by the string x_j; the result of this replacement is referred to as Q_j being *satisfied* (by x_j). Once the communication step is complete the grammar G_j continues processing from its axiom, unless the system is non-returning. Communication steps always have priority over rewriting steps; if not all query symbols are satisfied during a communication step, they will be satisfied during the next communication step (as long as the replacement strings do not contain query symbols).

For both component-wise and communication steps we use $\Rightarrow$, but we may use $\overset{\Lambda}{\Rightarrow}$ for communication steps whenever we want to emphasize that a communication takes place. A sequence of interleaved rewriting and communication steps will be denoted by $\Rightarrow^*$, the reflexive and transitive closure of $\Rightarrow$.

Definition 4 ([7])**.** *The language generated by a PCGS* Γ *is*

$$\mathscr{L}(\Gamma) = \{w \in T^* : (S_1, S_2, ..., S_n) \Rightarrow^* (w, \sigma_2, ..., \sigma_n), \sigma_i \in (N \cup T \cup K)^*, 2 \leq i \leq n\}.$$

We start from the tuple of axioms $(S_1, S_2, ..., S_n)$ in each component. We proceed by rewriting and/or communication steps until G_1 produces a terminal string. We do not restrict the form of, or indeed care about the rest of the components of the final configuration. In other words they might contain nonterminals or query symbols.

As with any model certain behaviors have been defined in semantic terms to simplify their description. The following terms will be used frequently in what follows.

Definition 5 ([22])**.** *We call a PCGS Γ centralized if query symbols are only introduced in the first component grammar G_1 which means only G_1 can control the communications. If component grammar G_i, $i \neq 1$ can coordinate communications steps, meaning any component grammar can introduce communication symbols, then the system is non-centralized.*

We call a PCGS a returning system if the components return to their axiom after they are queried by other component. If those component continue to process the current string and are not reset to their axiom, then the system will be considered non-returning.

A PCGS system is considered synchronized whenever a component grammar uses exactly one rewriting rule per derivation step (unless the component grammar is holding a terminal string, case in which it is allowed to stay put). If the components are allowed to wait instead of proceeding with a rewriting rule then that system is non-synchronized.

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 $PC_n(X)$ [7]. 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:

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

Chapter 3

Previous Work

Our result is in the area of generative power of PCGS so we start by summarizing here the existing results in this area. It should be noted that not all structural variations have been studied in this respect. Most of the existing results are about centralized systems, and even then not all of the centralized variants have been studied thoroughly. We then proceed with our main focus namely, the introduction of meta-trees (Definition 8) and the associated results which mainly refer to context-free non-returning systems (synchronized and unsynchronized).

In the following we refer in the usual way to the various classes in the Chomsky hierarchy as follows: RE, CS, CF, LIN, and REG are the classes of recursively enumerable, context-sensitive, context-free, linear, and regular languages, respectively.

As already mentioned, PCGS are generally more powerful than grammars of the same type. This is not however always the case, especially at the top of the Chomsky hierarchy. Specifically, CS and RE are the two most powerful PCGS and grammar types; surprisingly their behavior is quite similar. We start with the immediate finding that a RE grammar is just as powerful as a PCGS with RE components. Due to this the PCGS of this type with $n > 1$ components are not very interesting since they are just as powerful as a PCGS with one component. In other words a PCGS with unrestricted components are Turing equivalent and are just as powerful as RE grammars: $\mathrm{RE} = Y_n(\mathrm{RE}) = Y_*(\mathrm{RE})$, $n \geq 1$, for all $Y \in \{PC, CPC, NPC, NCPC\}$ [8].

A bit more interesting is the fact that the same holds to some degree for PCGS with context-sensitive components versus context-sensitive languages: $CS = Y_n(CS) = Y_*(CS)$, $n \geq 1$, for $Y \in \{CPC, NCPC\}$ [8]. Note that this result describes the centralized case; we would expect that the non-centralized case to be more powerful.

Context-sensitive grammars are computationally expensive and so limited in usefulness. As is the case with normal grammars, the most useful PCGS classes are therefore the simple ones. This is in fact more the case with PCGS; anticipating a bit, the fact that CF PCGS will turn out to be Turing complete make RE and CS PCGS not just of limited utility, but not useful at all.

Moving to the more interesting PCGS classes, we first note that the class of languages generated by a centralized returning PCGS with regular components is a subset of the class of languages generated by a non-centralized, returning PCGS with regular components. This indicates that the generative power of a PCGS is greater than of a single grammar component, and that the more communication facilities we have the more powerful the resulting system is: $CPC_n(\text{REG}) \subsetneq PC_n(\text{REG})$, $n > 1$ [22]. A similar result was found for PCGS with context free components; however in this case the inclusion is not necessarily strict, so increased communication may not make the system more powerful: $CPC_*(\text{CF}) \subseteq PC_*(\text{CF})$ [12].

We note in general that the centralized variant is a particular case of a non-centralized PCGS. Indeed, that centralized qualifier restricts the initiation of the communication to the first grammar in the system. As a consequence the class of languages generated by a centralized PCGS of any type can be generated by a non-centralized PCGS of the same type: $CPC_n(X) \subseteq PC_n(X)$ for any $n \geq 1$. As a more general comment on this matter it should be noted that the fact that the generative power of a PCGS is greater that of a single grammar component is largely due to the introduction of communication facilities. Once these facilities are restricted, the generative power is also restricted.

The following two results demonstrate that there are limitations to the generative power of PCGS. When we have only two regular components the languages generated by centralized PCGS are all context free. Even the non-centralized variant is limited to generating context-free languages.

- $CPC_2(\text{REG}) \subsetneq \text{CF}$, [8].

- $PC_2(\text{REG}) \subseteq \text{CF}$ [8].

Another way to increase the generative power of a system is to increase the number of components in the system. We have shown that this does not change the generative capacity in the RE and (to some degree) CS case. However if we examine classes that are lower in the hierarchy we notice that an increase in the number of components generally increases the generative capacity of the system [8].

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

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

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

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

Obviously an increase in the power of the components will generally increase the power of a PCGS. This holds strictly in the centralized case for REG versus LIN versus CF components: $CPC_n(\text{REG}) \subsetneq CPC_n(\text{LIN}) \subsetneq CPC_n(\text{CF})$, $n \geq 1$, [8]. Presumably the same relationship would hold for the non-centralized case, but as far as we know this has not been investigated.

We already mentioned the number of components as an important factor in the generative power of PCGS. It therefore makes sense to consider the hierarchies generated by this factor. Some of these hierarchies are in fact infinite, namely $CPC_n(\text{REG})$ and $CPC_n(\text{LIN})$, $n \geq 1$ [8].

Some hierarchies however collapse. We have already mentioned that $CPC_n(CS)$ and $NCPC_n(CS)$, $n \geq 1$, do not give infinite hierarchies, for all of these classes coincide with CS. Lower

classes also produce collapsing hierarchies; for instance non-centralized CF-PCGS with 11 components generate the whole class of RE languages [10]. A later paper found that a CF-PCGS with only 5 components can generate the entire class of RE languages by creating a PCGS that has two components that track the number of non-terminals and use the fact that for each RE language L there exists and Extended Post Correspondence problem P [14] such that $L(P) = L$. In other words, RE $= PC_5$CF $= PC_*$CF [9].

There have also been other papers that have examined the size complexity of returning and non returning CF systems even further. It has been shown that every recursively enumerable language can be generated by a context fee returning PCGS, where the number of nonterminals in the system is less than or equal to a natural number k [6]. It has also been shown that non-returning CF-PCGS can generate the set of recursively enumerable languages with 6 context free components by simulating a 2-counter Turning machine [11].

The above results [6, 9, 10] use a particular communication model called broadcast communication, which is not necessarily implied by the canonical PCGS definition. In the one-step communication model which is arguably implied by the PCGS definition the hierarchy PC_*CF also collapse, though not necessarily at $n = 11$ or $n = 5$ [23].

Turing completeness was also shown for non-returning systems [11, 19]. In particular, if $k \geq 2$ and $L \subseteq \{a_1, \ldots, a_k\}^+$ is a recursively enumerable language, then there exists a non-returning CF-PCGS without ε-rules (meaning without rules of the form $A \to \varepsilon$) that generates L [11]. If we consider that non-returning systems can be simulated by returning systems via the help of assistance grammars holding intermediate strings [13], these results [11, 19] also apply to returning systems (though the number of components necessary for this to happen does not remain the same).

3.1 PCGS Parse Trees

Our research is a continuation of the investigation into parse trees for context-free PCGS [4]. We therefore review these results in more depth according to the paper already mentioned [4].

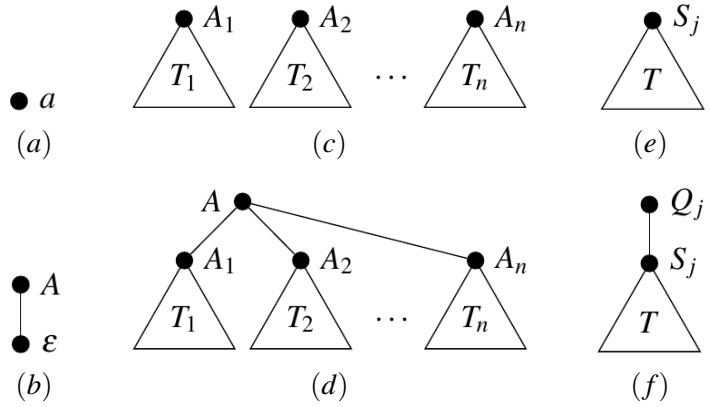

Figure 3.1: Parse trees for context-free grammars (a,b,c,d); supplementary parse trees for CF-PCGS (e,f).

Definition 6 (PCGS Parse Trees)**.** *Let $\Gamma = (N,K,\Sigma,G_1,\ldots,G_n)$ be a PCGS with context-free components.*

A parse tree for some component $G_i = (N \cup K,\Sigma,R_i,S_i)$, $1 \leq i \leq n$, of Γ is defined inductively as follows: For every $a \in N \cup K \cup \Sigma$ the tree depicted in Figure 3.1(a) is a parse tree with yield *a, and for every $A \to \varepsilon \in R_i$ the tree depicted in Figure 3.1(b) is a parse tree with yield ε. Suppose that the n trees from Figure 3.1(c) are parse trees with yields y_1, y_2, …, y_n and that $A \to A_1A_2\ldots A_n \in R_i$; then the tree shown in Figure 3.1(d) is a parse tree with yield $y_1y_2\ldots y_n$. If the tree depicted in Figure 3.1(e) is a parse tree of G_j, then the tree from Figure 3.1(f) is a parse tree for G_i, $1 \leq i,j \leq n$, $i \neq j$.*

The yield of a parse tree continues to be the sequence of leaf labels as obtained by an inorder traversal of that parse tree.

Definition 7 (PCGS Parse Forest)**.** *A parse forest for Γ is an n-tuple $\mathscr{T} = (T_1,T_2,\ldots,T_n)$, where T_i is a parse tree for G_i as in Definition 6, $1 \leq i \leq n$. The first component T_1 of a parse forest $\mathscr{T}$ is called the master parse tree (of $\mathscr{T}$).*

Definition 8 (Meta-tree)**.** *Let $\Gamma = (N,K,\Sigma,G_1,\ldots,G_n)$ be a context-free PCGS and let T_k, $1 \leq k \leq n$ be a tree in the parse forest of some derivation $(S_1,S_2,\ldots,S_n) \Rightarrow_\Gamma^* (x_1,x_2,\ldots,x_n)$.*

A meta-node *(of T_k) is a maximal region of T_k which (a) has all its edges produced by applications of rules from a single component G_i of Γ, and (b) is rooted at S_i, the axiom of G_i.*

The meta-tree *$\mu(T_k)$ of T_k is the tree of meta-nodes constructed using function μ defined recursively (and naturally) such that for a given parse tree T, $\mu(T)$ produces the following meta-tree:*

1. *The root r of $\mu(T)$ is the meta-node rooted at the root of T. It has one child for every leaf labelled with a query symbol Q_j in r (zero children if no such a label exists in r).*

2. *For each edge (Q_j, S_j) in T that originates from a leaf of r labelled Q_j the respective child of r is $\mu(T')$, where T' is the tree rooted at S_j.*

The yield of $\mu(T)$ is defined as being the same as the yield of the underlying parse tree T.

Proposition 1. *Every derivation resulting in a configuration $(x_1, x_2, \ldots, x_n)$ in a PCGS with context-free components has an equivalent parse forest; the yields of the parse trees in that forest are x_1, x_2, $\ldots$, x_n, respectively*

Definition 9 (Checkpoint and Interference)**.** *Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a CF-PCGS. We use $\gamma_0 \Rightarrow_\Gamma \gamma_1 \Rightarrow_\Gamma \cdots \Rightarrow_\Gamma \gamma_p$ to refer to any complete derivation in Γ, meaning that there is no γ_{p+1} such that $\gamma_p \Rightarrow_\Gamma \gamma_{p+1}$. We further put $\gamma_k = (\gamma_{k1}, \gamma_{k2}, \ldots, \gamma_{kn})$, meaning that we use γ_{ki} to refer to the i-th component of the configuration γ_k, for $0 \leq k \leq p$ and $1 \leq i \leq n$.*

A checkpoint *of (component) G_i by G_j, $1 \leq i, j \leq n$ during some derivation in Γ is either p (the end of the derivation) or some $1 \leq k < p$ such that $|\gamma_{kj}|_{Q_i} \neq 0$ (the event of G_j querying G_i). Note that during a particular derivation there may be multiple checkpoints of G_i by G_j.*

A component G_j interferes *with another component G_i in some derivation whenever there exists a checkpoint $\mathscr{C}$ of G_i by G_j and a string w_i (the* interference string*) such that*

1. *$S_i \Rightarrow^*_{G_i} w_i$ (w_i can be produced by grammar G_i if it acts alone outside Γ), and*

2. *$\gamma_{\mathscr{C}i} \neq w_i$ (w_i cannot be produced by G_i at step $\mathscr{C}$ in the respective derivation of Γ).*

Definition 10 (Unique-query PCGS)**.** *A unique-query PCGS is a PCGS in which no rewriting rule contains two or more occurrences of the same query symbol.*

Proposition 2. *There exists a master parse tree of some synchronized PCGS that does not correspond to any derivation in that PCGS. There exists a parse forest of some synchronized PCGS that does not correspond to any derivation in that PCGS. This all holds for both returning and non-returning PCGS.*

Proposition 3. *There exists a master parse tree of some non-returning, unsynchronized PCGS that does not correspond to any derivation in that PCGS. There exists a parse forest of such a PCGS that does not correspond to any derivation in that PCGS.*

Proposition 4. *There exists a master parse tree of some returning, unsynchronized, non-unique-query PCGS that does not correspond to any derivation in that PCGS. There exists a parse forest of such a PCGS that does not correspond to any derivation in that PCGS.*

Chapter 4

Meta-Trees and Mixed Meta-Nodes

The meta-trees as introduced in Definition 8 are too restrictive. Indeed, while it is true that any derivation has an equivalent parse tree, it is possible that the parse tree cannot be covered by meta nodes. In other words, it is not the case that all derivations in a context-free PCGS have an equivalent meta-tree. It is possible to have derivations and meta-trees that do not correspond to each other, even when all the constraints are in place.

The cause of this issue is the first constraint in the definition of meta-trees namely, "*a meta-node (of T_k) is then a maximal region of T_k which (a) has all the edges produced by applications of rules from a single component G_i of* Γ". While it is true that a meta-node is only expanded using rules from a given component, the construction of that meta-node might have been started in a different component, and so overall the rules that are being used do not come from the same component. If this is the case, then such a parse tree cannot be covered by meta-nodes as defined originally.

We will therefore start by providing a relaxed definition for meta-nodes (and so meta-trees).

Definition 11 (Meta-tree)**.** *Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a context-free PCGS and let T_k, $1 \leq k \leq n$ be a tree in the parse forest of some derivation $(S_1, S_2, \ldots, S_n) \Rightarrow_\Gamma^* (x_1, x_2, \ldots, x_n)$. A* meta-node *(of T_k) is then a maximal region of T_k which is rooted at S_i, the axiom of G_i and does not include any edge (Q_k, X), $1 \leq k \leq n$, $X \in N \cup T$. In other words, a meta-node is a maximal sub-tree of T_K rooted at some axiom and not containing any internal node labeled with a query symbol.*

The meta-tree *$\mu(T_k)$ of T_k is then a the tree of meta-nodes constructed using a function μ defined*

recursively (and naturally) such that for some parse tree T $\mu(T)$ produces the following meta-tree:

1. *The root r of $\mu(T)$ is the meta-node rooted at the root of T. It has one child for every leaf labeled with a query symbol Q_j in r (zero children if no such a label exists in r).*

2. *For each edge (Q_j, S_j) in T that originates from a leaf of r labelled Q_j the respective child of r is $\mu(T')$, where T' is the tree rooted at S_j.*

The yield of $\mu(T)$ is defined as being the same as the yield of the underlying parse tree T.

For a better understanding of the new meta-tree notion we proceed with the following example. Figure $4.1(a)$ depicts the meta-nodes $\mathbf{M_1}$, $\mathbf{M_2}$ and $\mathbf{M_3}$ in the parse tree $\mu(\mathbf{T})$ shown in Figure $4.1(b)$. Note that this construction is not valid according to the original definition if for example the rewriting rule $A \Rightarrow a$ belongs to the first component (according to the original definition that rule has to come from the second component since it is used in the construction of a meta-node rooted at S_2). Figure $4.1(c)$ is the meta-tree $\mu(\mathbf{T})$ rooted at $\mathbf{M_1}$, which in turn has $\mathbf{M_2}$ and $\mathbf{M_3}$ as its children. As we can see meta-tree $\mu(T_k)$ of T_k start with its axiom which is the master parse tree and contains 2 edges which correspond to all the queries performed during the derivation.

Definition 12 (Mixed meta-nodes)**.** *Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a context-free PCGS and let T_k, $1 \leq k \leq n$ be a tree in the parse forest of some derivation $(S_1, S_2, \ldots, S_n) \Rightarrow^*_\Gamma (x_1, x_2, \ldots, x_n)$. A* mixed meta-node *of T_k is then a meta-node of T_k which at the time it is communicated from one component to another contained at least one leaf labeled with a nonterminal.*

Lemma 5. *There exists a meta-tree $\mu(T)$ for every parse tree T from any PCGS parse forest. Furthermore $\mu(T)$ covers all the nodes from T.*

Proof. That $\mu(T)$ is a tree is immediate by the definition of a parse tree. During the derivation components are building their respective meta-trees by expanding meta-nodes until a query symbol is thus introduced. This is followed immediately by the connection of that query symbol with the meta-tree constructed by the component that is being queried. This in turn introduces a new meta-node that becomes the child of the initial meta-node, and so on.

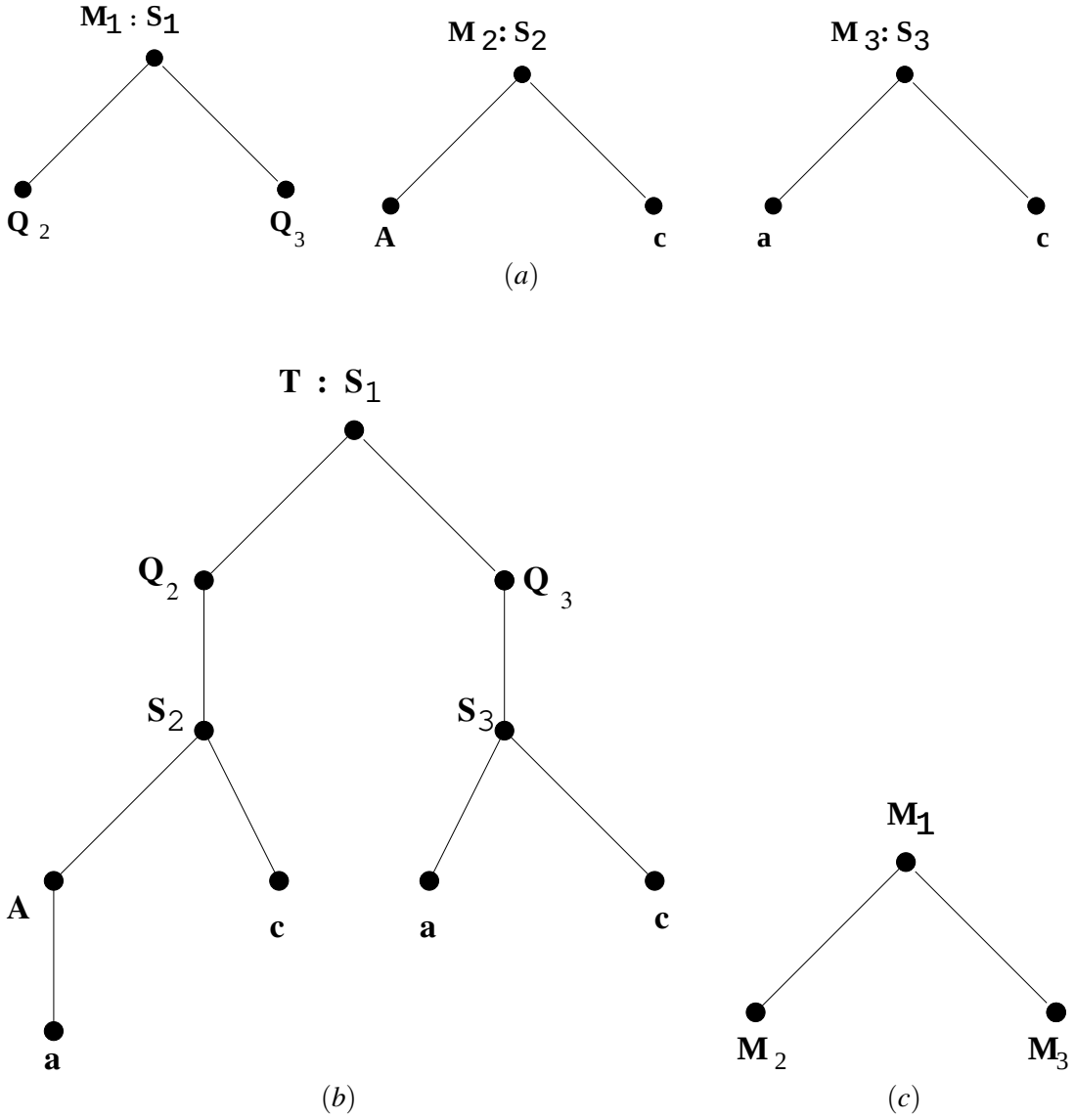

Figure 4.1: Meta-nodes M_1,M_2 and M_3 (a), a parse tree T (b), and the associated meta-tree (c)

By induction over the depth of the nodes in T we can show that $\mu(T)$ covers all the nodes of T. At level 0 the tree starts with S_k of T which is the root of the root meta-node. The children of this meta-node are all introduced by edges (Q_j, S_i) in the parse tree. Note that these are the only kind of edges whose start nodes are labeled with query symbols.

At a level $n > 0$ we have a meta-node that starts at some axiom S_i by the inductive hypothesis. We keep including in this meta-node all the nodes of the parse trees until we reach either leaf nodes or nodes labeled with query symbols. In the latter case we note again that the only edge starting from a query symbol Q_j has the form (Q_j, S_i) and so starts a new meta-tree (rooted at S_j) on the next level. □

Chapter 5

Complete Characterization of PCGS Derivations by Parse Trees

Parse trees are useful for context-free grammars because all derivations have equivalent parse trees and the other way around. According to Theorem 1, we already know that there exists a parse tree for every derivation in a context-free PCGS. However, whether this also works the other way around (any parse tree corresponds to an equivalent derivation) depends on other characteristics of the PCGS in discussion. Whenever there exists a derivation for every parse tree of a PCGS we say that the respective PCGS is completely characterized by parse trees. This property turns out to depend on the presence of interference and mixed meta-nodes.

It is already known that interference causes PCGS not to be completely characterized by parse trees. As a consequence, synchronized PCGS, unsynchronized, non-returning PCGS, as well as unsynchronized, returning PCGS that are not unique-query cannot be completely characterized by parse trees (see Propositions 2, 3, and 4).

By contrast, unsynchronized, returning, unique-query PCGS were believed to be completely characterized by parse trees. We consider this class of PCGS further and we discover that the second property (mixed meta-nodes) also plays a role.

Theorem 6. *Every (master) parse tree of a PCGS* Γ *with root* S_1 *and yield* w *corresponds to a (not necessarily unique) derivation* $(S_1, S_2, \ldots, S_n) \Rightarrow_\Gamma^* (w, x_2, \ldots, x_n)$ *for some* $x_i \in N \cup \Sigma$, $2 \leq i \leq n$ *iff there is no interference and the construction of the parse tree did not use mixed meta-node.*

Proof. Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ and let T be some master parse tree with $\mu(T)$ its corresponding meta-tree.

We start by considering the case in which there is no interference and no mixed meta-nodes where used, and proceed by structural induction over $\mu(T)$.

Consider a meta-node rooted as S_i, $0 \leq i \leq n$ that has no queries anywhere inside it. All the leaves were labeled with terminals at communication time, so the rules that were used to construct the meta-node are all coming from the same component (the one that produced the node originally) through a component-wise derivation. Given that there is no interference in Γ, the respective components has the time to carry on any derivation, including the one corresponding to the meta-node in discussion. Therefore we can just choose a suitable component-wise derivation in Γ that produces our meta-node, as desired; the base case is established.

Now let us consider the scenario where query symbols Q_{i_1}, Q_{i_2}, ..., Q_{i_k} are presented. We start constructing meta-nodes rooted as S_{i_1}, S_{i_2}, S_{i_k}. By the previous argument all these meta-nodes correspond to component-wise derivations. By the induction hypothesis and absence of interference, each meta-node will have the time to reach any combination starting from their axiom. By the absence of the nonterminals as leaves at the moment of querying we are sure that all the rewriting rules used in the meta-node M_i are coming from G_i. The parent node does not interfere with its children since it does not perform any query. In the meta-node M_i we will have a configuration that corresponds to a string which includes query symbols $Q_{i_1}, Q_{i_2}, \ldots, Q_{i_k}$. By the definition of a meta-tree, the query symbols should become the root of corresponding components. The actual children do correspond to such derivations (by induction hypothesis). Therefore the whole tree corresponds to a derivation, as desired. The induction is complete.

Consider now the checkpoint $\mathscr{C}$ of G_i by G_j such that G_j interferes with G_i at $\mathscr{C}$ in an otherwise successful derivation of Γ with parse tree T. Let w_i be the interference string. The checkpoint $\mathscr{C}$ corresponds in T to a node labelled Q_i having S_i as sole child (which in turn is the root of some subtree). Replace then the aforementioned tree rooted at S_i with the tree corresponding to the derivation $S_i \Rightarrow^*_{G_i} w_i$. We still have a parse tree, yet such a tree cannot correspond to any derivation

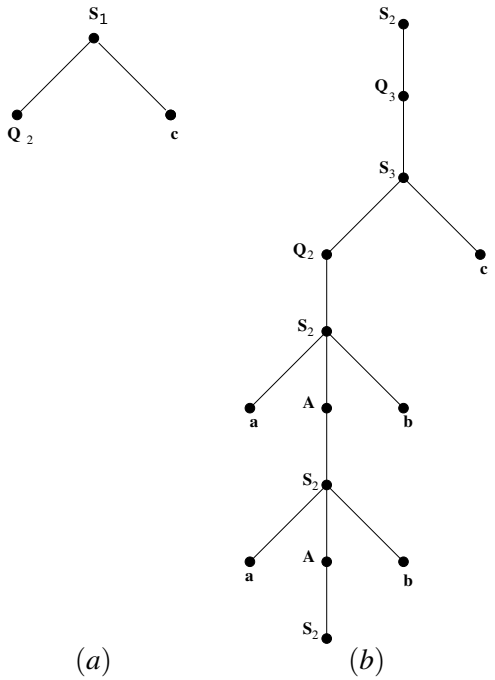

Figure 5.1: Possible parse trees for the first and second component of a PCGS.

in Γ since this would imply that w_i has communicated to G_j at checkpoint $\mathscr{C}$ (an impossibility since the interference string w_i is not available at that point).

Finally, let at least one nonterminal A be present in the yield of a parse tree at the time if is queried by the other component. Existence of such a nonterminal will allow the tree to grow in such a way that the respective derivation cannot be obtained in the system. We establish this case using the following counterexample:

Let $\Gamma = (\{S_1, S_2, S_3, A\}, K, \{a, b, c\}, G_1, G_2, G_3)$ be an unsynchronized, returning, unique-query PCGS such that

$$\begin{aligned} R_1 &= \{S_1 \to Q_2 c, S_2 \to ab\} \\ R_2 &= \{S_2 \to aAb, S_2 \to Q_3\} \\ R_3 &= \{S_3 \to Q_2 c, A \to S_2\} \end{aligned}$$

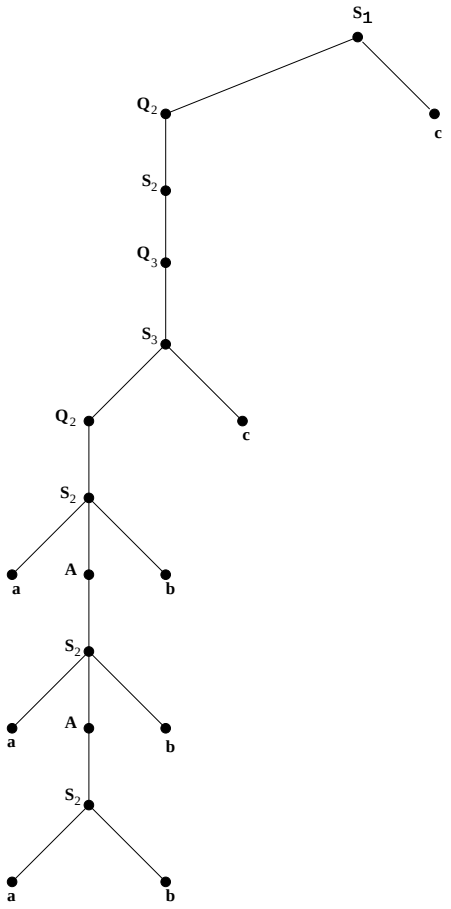

Figure 5.2: A parse tree that cannot correspond to any derivation in the respective PCGS.

This system generated the language $\mathscr{L}(\Gamma) = \{a^n b^n c^m | n > 1, m \geq n\}$. Indeed, a cycle between R_2 and R_3 would result in suitable numbers of a, b and c as follows: $(S_1, S_2, S_3) \Rightarrow^* (Q_2 c, a^n S_2 b^n c^m, S_3) \Rightarrow (a^n S_2 b^n c^{m+1}, S_2, S_3)$ for some $n > 1$, which will be finally rewritten to $(a^{n+1} b^{n+1} c^{m+1}, S_2, S_3)$. There is no other possible derivation in the system.

On the side of constructing the parse trees, we start in the first component. As we can see in Figure 5.1, the first component should wait for the construction of the other meta-tree, so we wait until we reach a desired tree in the second component. A particular parse tree constructed by the second component is shown in Figure 5.1. This tree starts being constructed in the second component starting its level 4 where the rule $S_2 \Rightarrow aAb$ is used. Then this tree is queried by the third component (rule $S_3 \Rightarrow Q_2 c$). The nonterminal A in the yield of this tree (which is a mixed meta-node) is then rewritten using the rule $A \Rightarrow S_2$ (level 5). This is where the contradiction starts to happen. Indeed, the tree thus obtained gets further communicated to the second component (level 0), where it continues to expand but this time using rules from the second component. Accordingly, we use at level 6 $S_2 \Rightarrow aAb$ (since we are in the tree being operated on by the second component). On the other hand we are also in the meta-node corresponding to the third component and so at level 7 we use $A \Rightarrow S_2$.

Finally, we communicate the tree just derived in the second component to the first. We then rewrite S_2 in the first component to ab, thus reaching the master parse tree shown in Figure 5.2, with yield $a^3 b^3 c^2$. Clearly, this does not correspond to any derivation in Γ; as desired. □

We already know that derivations in PCGS that are either synchronized or non-returning cannot be completely characterized by parse trees. We further know that this is also the case for unsynchronized, unique-query PCGS that are not unique-query. These results were established earlier based on interference. The existence (or absence) of mixed meta-nodes allows us to further refine the remaining class.

First however we need to see what are the features of a PCGS that ensure the absence of mixed-meta-nodes in their parse trees. We note that when a string w is communicated from a component i to another component j we connect to the parse tree of component i a tree rooted at s_j with

yield w. A mixed meta-node in particular corresponds with a yield w such that $|w|_N \neq 0$ (that is a yield that contains no nonterminal). The only way to avoid mixed meta-nodes is to disallow the communication of strings containing nonterminals.

Definition 13. *A* PCGS with communication by terminal strings *the communication of strings that contain nonterminals is not allowed.*

Corollary 7. *Every parse tree with root S_1 and yield w of an unsynchronized, returning, unique-query PCGS Γ corresponds to a (not necessarily unique) derivation $(S_1, S_2, \ldots, S_n) \Rightarrow^* (w, x_2, \ldots, x_n)$ in Γ for some $x_i \in N \cup \Sigma$, $2 \leq i \leq n$ iff Γ is a PCGS with communication by terminal strings.*

Proof. If (by contrapositive): If we have any interference or any mixed meta-node in Γ there is at least one parse tree that does not correspond to any derivation in Γ by Theorem 6. The lack of both interference and mixed meta-nodes only happen in unsynchronized, returning, unique-query PCGS with communication by terminal strings.

Only if: There is no interference in unsynchronized, returning, unique-query PCGS. Indeed, in such a system any checkpoint reduces the queried component to its axiom, which thereafter has any time it needs to reach any possible form. Since the system is with communication by terminal strings no mixed meta-nodes are ever communicated. The result then follows directly from Theorem 6. □

Chapter 6

Conclusions

Like for context-free grammars, parse trees would be a very useful tool for CF-PCGS both theoretically and practically if they turn out to completely characterize derivations. The existence of an equivalent parse tree for every derivation is already established (Proposition 1). However, the other way around (meaning that there is a derivation for every parse tree) for different types of PCGS derivations depends on notions of interference and as it turns out also mixed meta nodes (Definition 12).

One case of derivations believed to have complete characterization by parse trees was the unsynchronized, non-returning, unique query CG-PCGS. Our incursion into this area revealed that existence of mixed meta-nodes in PCGS causes parse trees that do not correspond to any derivation. In this respect, we noticed that the the definition of meta-trees needs to be modified since it is not valid for all the parse trees. As a result we gave a more general definition for meta-trees (Definition 11). With the new definition we were able to define the notion of mixed meta-nodes (Definition 12). Existence of mixed meta-nodes in a PCGS will result in some parse trees that cannot correspond to any derivation (Theorem 6). Having said that, we have complete characterizations in an unsynchronized, non-returning, unique-query PCGS if and only if we only have communication by terminal strings.

A relatively straightforward argument similar to one made earlier [4] will show that a CF-PCGS in which we have complete characterization of derivations by parse trees is not more powerful than

a single context-free grammar. It follows that the vast majority of CF-PCGS variants do have more expressive power than their components (all of them *except* unsynchronized, non-returning, unique-query PCGS with communication by terminal strings).

The ultimate goal of showing a complete characterization of PCGS by parse trees is for using PCGS as a basis for formal specification. In fact, we are aiming at modeling only the interactions of (albeit complex) computing systems with their environment. In this respect, even though the unsynchronized PCGS are less powerful, they still seem suitable. Similar characteristics of these systems, as they are arguably closer to the way an actual concurrent system works, make them a great candidate for particular tasks of system specification. Based on the findings from this thesis imposing more restrictions such as unique queries and communication by terminal strings become very appealing, though it remains to be seen how meaningful such restrictions would be from the point of view of formal specification and verification.

Bibliography

[1] R. ALUR AND P. MADHUSUDAN, *Visibly pushdown languages*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 04), ACM Press, 2004, pp. 202–211.

[2] J. A. BERGSTRA AND J. W. KLOP, *Process theory based on bisimulation semantics*, in Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, vol. 354 of Lecture Notes in Computer Science, Springer, 1988, pp. 50–122.

[3] S. D. BRUDA AND M. T. BIN WAEZ, *Unrestricted and disjoint operations over multi-stack visibly pushdown languages*, in Proceedings of the 6th International Conference on Software and Data Technologies (ICSOFT 2011), vol. 2, Seville, Spain, July 2011, pp. 156–161.

[4] S. D. BRUDA AND M. S. R. WILKIN, *Parse trees and unique queries in context-free parallel communicating grammar systems*, Tech. Rep. 2013-001, Department of Computer Science, Bishop's University, apr 2013.

[5] L. CAI, *The computational complexity of linear PCGS*, Computer and AI, 15 (1989), pp. 199–210.

[6] 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.

[7] E. CSUHAJ-VARJÚ, J. DASSOW, J. KELEMEN, AND G. PAUN, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[8] 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.

[9] E. CSUHAJ-VARJÚ, P. GHEORGHE, AND G. VASZIL, *PC grammar systems with five context-free components generate all recursively enumerable languages*, Theoretical Computer Science, 299 (2003), pp. 785–794.

[10] 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.

[11] 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.

[12] 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.

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

[14] V. GEFFERT, *Context-free-like forms for the phrase-structure grammars*, in Mathematical Foundations of Computer Science, vol. 324 of Lecture Notes in Computer Science, Springer, 1988, pp. 309–317.

[15] M. A. GRANDO AND V. MITRANA, *A possible connection between two theories: grammar systems and concurrent programming*, Fundamenta Informaticae, 76 (2007), pp. 325–336.

[16] S. LA TORRE, P. MADHUSUDAN, AND G. PARLATO, *A robust class of context-sensitive languages*, in Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science (LICS 07), Washington, DC, 2007, IEEE Computer Society, pp. 161–170.

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

[18] D. MADUDU, S. D. BRUDA, AND M. T. BIN WAEZ, *Cmvp, a process algebra based on multi-stack visibly pushdown languages and their disjoint operations*, Tech. Rep. 2019-002, Department of Computer Science, Bishop's University, Jan. 2019.

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

[20] G. PAUN AND L. SANTEAN, *Parallel communicating grammar systems: the regular case*, Analele Universitatii din Bucuresti, Seria Matematica-Informatica, 38 (1989), pp. 55–63.

[21] G. PAUN AND L. SANTEAN, *Further remarks on parallel communicating grammar systems*, International Journal of Computer Math., 34 (1990), pp. 187–203.

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

[23] 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 (2016), pp. 113–170.