Technical Report 2013-001: Parse Trees and Unique Queries in Context-Free Parallel Communicating Grammar Systems*

Stefan D. Bruda and Mary Sarah Ruth Wilkin
Department of Computer Science
Bishop's University
2600 College St
Sherbrooke, Quebec J1M 1Z7, Canada
stefan@bruda.ca, swilkin@cs.ubishops.ca

16 April 2013

Abstract

Parallel communicating grammar systems (PCGS) were introduced awhile ago purportedly to analyze concurrent systems on a language-theoretic level. To our knowledge however no actual relationship between PCGS and practical computing systems was ever investigated. We believe that PGCS with context-free components (CF-PCGS) have high practical potential, especially in the area of formal methods, so we start to bring CF-PCGS to a more practical level by studying a construct that has proven useful elsewhere: the parse tree. We can attach a parse forest (and then tree) to any CF-PCGS derivation in a natural way. However, the other way around (finding a derivation for each parse forest) holds only for one, very restrictive variant of CF-PCGS. Overall beside providing a convenient tool to be used in conjunction with CF-PCGS, this work strongly suggests the aforementioned PCGS variant as the most promising model for practical applications in general and for grammatical approaches to formal verification of concurrent, recursive systems in particular.

Keywords: parallel communicating grammar systems, context-free grammars, parse trees, concurrency

1 Introduction

Parallel Communicating Grammar Systems (PCGS) have been introduced as a language-theoretic treatment of concurrent (or more general, multi-agent) systems [5, 13]. A PCGS consists of several component grammars that work in parallel on separate strings. The components also communicate with each other: One component grammar may request strings generated by others, and several components may make queries at the same time. Because of the synchronization and communication facilities, PCGS whose components are of a certain type are generally more powerful than a single grammar of the same type [4–6, 13, 14].

Our overall research interest is the formal specification and verification of recursive, concurrent systems. We thus focus in particular on PCGS whose components are context-free grammars (context-free PCGS or just CF-PCGS for short), for indeed context-free grammars (or equivalently pushdown automata) model naturally the control flow of sequential computation in typical programming languages with nested, and potentially recursive invocations of program modules such as procedures and methods. Many non-regular properties are therefore required for software verification. We need to specify and verify properties such as "if p

*This research was supported by the Natural Sciences and Engineering Research Council of Canada. Part of this research was also supported by Bishop's University.

holds when a module is invoked, the module must return, and q must hold upon return" [1]. Non-regular properties however generate an infinite state space, which cannot be handled by finite-state process algebrae or by standard verification techniques such as model checking. Context-free process algebrae such as basic process algebra or BPA [2] can specify such context-free properties. Still, most of the software use many parallel components (such as multiple threads). In addition, many conformance-testing techniques (such as may/must testing [7]) use test cases that run in parallel with the process under test. Concurrency is therefore required for software verification, but cannot be provided by simple context-free process algebrae since context-free languages are not closed under intersection [12].

On the automata side the class of multi-stack visibly pushdown languages (MVPL) has been introduced to address such a need [3,11]; no equivalent mechanism is known on the grammatical (and thus process-algebraic) side. In addition, permitting concurrency in a compositional manner is done in MVPL using constructs that do not seem to be naturally portable to the grammatical (or process-algebraic) side.

For all these reasons PCGS with context-free components appear particularly useful as the basis of process algebrae suitable for specifying recursive, concurrent systems such as complex application software. Indeed, the context-free components are an excellent model of recursive subsystems, while the communication between these components (which is done in a "remote procedure call"-like fashion) seems particularly suitable for putting these subsystems together.

The motivation of PCGS is claimed to be the study of concurrent systems. Relatively recently some (strenuous) links with practice have beet attempted [10], but overall virtually no work has been performed on actually linking PCGS with any practical field. PCGS with context-free components in particular (which we believe to have the most practical utility) were ignored almost completely. Instead, PCGS have been studied relatively extensively (though not completely) with respect to theoretical properties such as generative power and then their study has largely stalled.

One of the most important constructs in the realm of context-free languages are the parse trees. They offer straightforward proofs of important results such as the pumping theorem, and they are also extremely useful in practice; indeed, they form the basis of compiler functionality, for compilers spend a huge amount of time constructing and then traversing such trees. We believe that parse trees can have similar importance for PCGS with context-free components. We therefore introduce the concept of parse trees for context-free PCGS (Section 3) as a natural extension of context-free parse trees, showing that any context-free PCGS derivation has an associated parse tree. We actually start with the introduction of a parse forest rather than a single tree, but we find out that the whole forest is not more descriptive than a single tree so we are able to revert to the original concept of parse tree without loss of information.

Going the other way around and offering a full characterization of context-free PCGS derivations using parse trees turns out to depend on a particular notion of "interference" (Section 4): full characterization is only possible for systems in which such an interference is not present. Such a property thus only holds for one very restricted context-free PCGS variant (Section 5) which will turn out to be called unsynchronized, returning, unique-query context-free PCGS. Only for such a variant parse trees offer a complete characterization of derivations. 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 (Section 6).

Beside offering a powerful tool for the study of context-free PCGS, our result is also useful in a more abstract way in our quest toward grammatical approaches to formal verification of concurrent and recursive systems. Indeed, our result suggests that synchronized context-free PCGS are unnecessarily powerful for our purpose. It would appear that the unsynchronized

variant has all the ingredients needed for modelling such systems and at the same time it is less complex. A less complex formalism implies that the associated algorithms have a lower complexity and all the other constructs (such as automata and temporal logic) are simpler and so easier to analyze and use. All of these are obvious advantages in the realm of formal methods. In addition, some sub-family of unsynchronized context-free PCGS can even make full use of tools (such as the parse trees) that are of reduced utility for the other variants.

2 Preliminaries

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

2.1 Context-Free Grammars

A (formal) *grammar* [12] is a tuple $G = (N, \Sigma, R, S)$, with N and Σ the set of nonterminals and terminals, respectively, $S \in N$ the axiom (or start symbol), and $R \subseteq ((N \cup \Sigma)^* N (N \cup \Sigma)^*) \times (N \cup \Sigma)^*$ the set of rewriting rules (often just rules for short). A rewriting rule (σ, σ') is customarily written $\sigma \to \sigma'$. A rewriting step replaces the string $w = u\sigma v$ with $w' = u\sigma' v$ whenever $\sigma \to \sigma' \in R$ (written $w \Rightarrow_G w'$). A grammar G generates the language $\mathcal{L}(G)$ of exactly all the strings that can be derived from the axiom S, i.e., $\mathcal{L}(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$, where $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$ (we often omit the subscript whenever the grammar is understood from the context) and is called a derivation. A *context-free grammar* in particular is a grammar with $R \subseteq N \times (N \cup \Sigma)^*$.

Definition 1. *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 1(a–d)): For every $a \in N \cup \Sigma$ the tree depicted in Figure 1(a) is a parse tree with* yield *a; for every $A \to \varepsilon \in R$ the tree from Figure 1(b) is a parse tree with yield ε. Suppose that the n trees from Figure 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 1(d) is a parse tree with yield $y_1 y_2 \ldots y_n$ [12].*

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 A and yield y there exists a derivation $A \Rightarrow^* y$ (and the other way around) in a natural way [12].

2.2 Parallel Communicating Grammar Systems

A PCGS with n components, $n \geq 1$ [5] is a $(n+3)$-tuple $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ where N is the set of nonterminals, Σ is the set of terminals, $K = \{Q_1, Q_2, \ldots, Q_n\}$ (the sets N, K and Σ are mutually disjoint) and $G_i = (N \cup K, \Sigma, R_i, S_i)$, $1 \leq i \leq n$, are grammars.

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

The derivation in a PCGS is defined as follows: Given a PCGS $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$, $(x_1, x_2, \ldots, x_n) \Rightarrow_\Gamma (y_1, y_2, \ldots, y_n)$, $x_i, y_i \in (N \cup K \cup \Sigma)^*, 1 \leq i \leq n$, iff one of the following cases holds:

1. $|x_i|_K = 0$, $1 \leq i \leq n$, and for all $1 \leq i \leq n$, we have $x_i \Rightarrow_{G_i} y_i$, or [$x_i \in \Sigma^*$ and] $x_i = y_i$.

2. $|x_i|_K > 0$ for some $1 \leq i \leq n$; for each such i let $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \dots z_t Q_{i_t} z_{t+1}$, with $t \geq 1$, $z_j \in (N \cup K \cup \Sigma)^*$, and $|z_j|_K = 0$. Then $y_i = z_1 x_{i_1} z_2 x_{i_2} \dots z_t x_{i_t} z_{t+1}$ [and $y_{i_j} = S_{i_j}$, $1 \leq j \leq t$] whenever $\left|x_{i_j}\right|_K = 0$, $1 \leq j \leq t$. If on the other hand $\left|x_{i_j}\right|_K \neq 0$ for some $1 \leq j \leq t$, then $y_i = x_i$. For all i, $1 \leq i \leq n$, for which y_i was not specified above we have $y_i = x_i$.

The first case is called a component-wise derivation step and the second a communication step. Note that communication has priority over component-wise derivation. The query symbol to which a string has been communicated is called satisfied. A tuple $(x_1, x_2, \dots, x_n) \in ((N \cup K \cup \Sigma)^*)^n$ as above is called a *configuration* of the system. We call x_i a component of the configuration or only a component if the reference to the configuration is understood from the context. Rules $Q_j \to \alpha$ are never used so we can assume that no such rules are present.

The derivation in a PCGS is blocked if no component-wise derivation can be applied to a nonterminal symbol in some component, or circular queries appear. The latter happens when G_{i_1} introduces Q_{i_2}, G_{i_2} introduces Q_{i_3}, …, $G_{i_{k-1}}$ introduces Q_{i_k} and G_{i_k} introduces Q_{i_1}; in such a case no rewriting step is possible (as communication has priority), but no communication steps are possible either.

With $\Rightarrow_\Gamma^*$ denoting as usual the reflexive and transitive closure of $\Rightarrow_\Gamma$, the language generated by a PCGS Γ is $\mathcal{L}(\Gamma) = \{w \in \Sigma^* : (S_1, S_2, \dots, S_n) \Rightarrow_\Gamma^* (w, \sigma_2, \dots, \sigma_n), \sigma_i \in (N \cup K \cup \Sigma)^*, 2 \leq i \leq n\}$ (again as usual we omit the subscript Γ whenever no ambiguity is introduced by such an omission). The derivation starts from the tuple of axioms $(S_1, S_2, \dots, S_n)$. A number of rewriting and/or communication steps are performed 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).

Several variants of PCGS can be defined. If a component-wise derivation requires that any component containing non-terminals be rewritten (i.e., the bracketed phrase "[$x_i \in \Sigma^*$ and]" is present in the first case above) then the system is *synchronized* (note however that any component that is already a string of terminals will not be rewritten under any circumstance). Otherwise we have an *unsynchronized* system, that allows components to either perform derivations or stay put. A PCGS is called *returning* (to the axiom) if, after communication, a component which has communicated a string resumes the work from its axiom as described by the phrase "[and $y_{i_j} = S_{i_j}$, $1 \leq j \leq t$]" in the second case above. A PCGS is called *non-returning* if components continue working using the current string after a query (i.e., the bracketed sentence above is erased from the definition).

3 PCGS Parse Trees (and Meta-Trees)

The notion of parse trees can be naturally extended to PCGS with context-free components (just PCGS henceforth, with the understanding that all grammars from now on are context-free unless otherwise stated). In fact we use almost the same construction as in Definition 1, though we also need to account for communication between components. We also need to keep track of which portion of which tree has been generated by which component gramar.

Definition 2. PCGS PARSE TREES: *Let $\Gamma = (N, K, \Sigma, G_1, \dots, G_n)$ 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 1(a) is a parse tree with yield *a; for every $A \to \varepsilon \in R_i$ the tree depicted in Figure 1(b) is a parse tree with yield ε. Suppose that the n trees from Figure 1(c) are parse trees with yields y_1, y_2, …, y_n and that $A \to A_1 A_2 \dots A_n \in R_i$; then the tree*

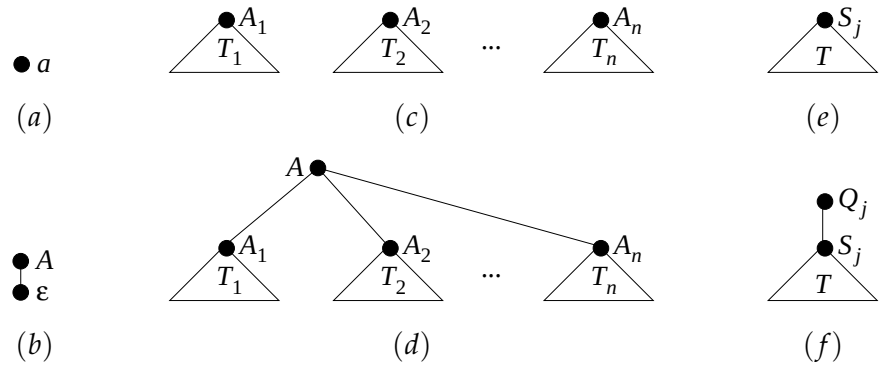

Figure 1: Parse trees for context-free grammars as well as PCGS (a, b, c, d); supplementary parse trees for context-free PCGS (e, f).

shown in Figure 1(d) is a right parse tree with yield $y_1 y_2 \dots y_n$. If the tree depicted in Figure 1(e) is a parse tree of G_j, then the tree from Figure 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.

Note that parse trees are identified as belonging to a specific component G_i in the PCGS. They are therefore constructed using only rewriting rules from that component, until a query node Q_j is generated. When this happens, the sub-tree rooted at the subsequent S_j is only allowed to use rewriting rules from G_j, and so on.

Similar with the context-free case, we can create parse trees that correspond to derivations in a PCGS. However now we have n components so logically we should construct simultaneously n parse trees; the n parse trees are collectively referred to as a *parse forest*.

Definition 3. PCGS PARSE FOREST: *A parse forest for Γ is an n-tuple $\mathcal{T} = (T_1, T_2, \dots, T_n)$, with T_i a parse tree for G_i as in Definition 2, $\leq i \leq n$. The first component T_1 of a parse forest $\mathcal{T}$ is called the master parse tree (of $\mathcal{T}$).*

As it turns out Definition 3 will only have a temporary role, as we will eventually show that the master parse tree alone is enough to characterize a PCGS derivation. For the time being the following establishes the usefulness of parse forests.

Lemma 1. *Every derivation resulting in a configuration $(x_1, x_2, \dots, x_n)$ 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, …, x_n, respectively*

Proof. Let $\Gamma = (N, K, \Sigma, G_1, \dots, G_n)$ be a context-free PCGS with $G_i = (N \cup K, \Sigma, R_i, S_i)$, $1 \leq i \leq n$. We construct the trees in the parse forest simultaneously in a natural way as follows:

1. All the n trees are initialized as single nodes labelled S_i, $1 \leq i \leq n$, respectively.

2. If no leaf node labelled with a query symbol exists in any of the n parse trees, then each component i chooses a leaf labelled with a nonterminal A such that $A \to w_1 w_2 \dots w_k \in R_i$, $w_j \in N \cup K \cup \Sigma$. The respective (former) leaf gains k children labelled w_1, w_2, …, w_k (or just one child labelled ε whenever $k = 0$).

   If Γ is synchronized then every component must perform such an expansion as long as a nonterminal leaf is present in its parse tree; otherwise, a component can either perform

the expansion or leave its parse tree unchanged. In all cases, if there is no leaf labelled by a nonterminal in the parse tree for some component, then that component does not alter its parse tree.

3. If at least one component has a leaf node labelled with a query symbol, then the following process takes place for as long as such leaf nodes are present: A leaf labelled with a query symbol (say, Q_j) in some component (say, i) is chosen, such that the parse tree for component j has no leafs labelled with query symbols. The leaf labelled with Q_j gains one child labelled S_j which is the root of a copy of the current parse tree of component j.

   If Γ is returning, then the parse tree for component j is reset to a single node labelled S_j; otherwise the parse tree for component j remains unchanged.

Showing that given a derivation such a construction exists proceeds as follows: We consider the component strings x_i, $1 \leq i \leq n$, as they are rewritten, together with the corresponding parse trees T_i, $1 \leq i \leq n$, in the forest as they are constructed during the same derivation. We proceed by induction over the number m of derivation steps performed.

For $m = 0$ the component strings are all initialized with the respective axioms, while the parse trees have all one (root) node labelled with the axiom; their yields are obviously identical with the component strings.

The component strings at step m are then rewritten to obtain the component strings at step $m + 1$ as follows:

Suppose that no component string contains query symbols, so we have a component-wise derivation step. By inductive assumption it follows that there is no leaf labelled with a query symbol in any of the corresponding parse trees. The rewriting that takes place in x_i picks a nonterminal A (such that $x_i = x'_i A x''_i$) to be rewritten using a rule $A \to \alpha_1\alpha_2 \ldots \alpha_k \in R_i$, with $\alpha_i \in N \cup \Sigma \cup K$, $1 \leq i \leq k$. Then A is replaced in the string by $\alpha_1\alpha_2 \ldots \alpha_k$ (such that x_i becomes $x'_i\alpha_1\alpha_2 \ldots \alpha_k x''_i$). By definition the corresponding node A in T_i gains k children labelled α_1, α_2, $\ldots$, α_k. The yield of T_i was $x'_i A x''_i$ (by inductive assumption) and is changed as follows: it contains x'_i (since nothing changes to the left of the node labelled A), followed by $\alpha_1\alpha_2 \ldots \alpha_k$ (A disappears from the yield since the node is now internal; its place is taken by the labels of its children according to the inorder traversal), followed by x''_i (since nothing changes to the right of the node labelled A), as desired.

Whenever x_i contains only terminals (case in which x_i does not change), the yield of T_i contains only terminals by inductive assumption, which means that all the leafs of T_i are labelled with terminals. In such a case no expansion can take place, so the tree remains unchanged, again as desired.

We note that whether some components can remain unchanged in the current step even if they can be rewritten depends on whether the system is synchronized or not; the same dependency is specified in the definition for the respective forest.

Suppose now that query symbols are present in some components, so a communication step takes place. Let component x_i contain Q_j (so that $x_i = x'_i Q_j x''_i$); then after the communication step x_i becomes $x'_i x_j x''_i$. The yield of T_i was by inductive assumption $x'_i Q_j x''_i$. Now the leaf labelled Q_j becomes internal (so it disappears from the yield). It's place is taken by the yield of the subtree now rooted at Q_j (by the definition of inorder traversal); this yield is however x_j (indeed, the tree rooted at Q_j is a copy of T_j, which has yield x_j by inductive assumption) and so the yield of T_i becomes $x'_i x_j x''_i$ (again by the definition of inorder traversal), as desired. T_j is reset to the initial form iff x_j is reset to the axiom (namely, whenever the system is returning) by the respective definitions. That circular queries block equally the derivation and the construction

of the parse trees is also immediate from definitions. □

In order for parse forests to be useful we must also be able to get a parse forest and reconstruct some derivation that gave birth to it. It turns out that this is possible. We can actually do even better and reconstruct the derivation starting solely from the master parse tree of the forest. In order to do this we will find the following notion of meta-tree useful.

Definition 4. META-TREE: *Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a context-free PCGS and let T_l, $\leq l \leq n$ 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 *is then a maximal region of T_l which (a) has all the edges produced by applications of rules from one component G_i of Γ, and (b) is rooted at S_i, the axiom of G_i.*

The meta-tree *$\mu(T_l)$ of T_l is a the tree of meta-nodes constructed naturally as follows: The root is the meta-node rooted at the root of T_l. There exists an edge in $\mu(T_l)$ for exactly all the edges (Q_j, S_j) in T_l; this edge connects the meta-node that contains Q_j (the parent) with the meta-node rooted at S_j (the child).*

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

Lemma 2. *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. The yield of $\mu(T)$ is the same as the yield of T.*

Proof. That $\mu(T)$ is a tree is immediate by the definition of a parse tree. Indeed, component-wise derivations build a meta-node; the introduction of a query symbol is followed immediately by the connection of that symbol with the respective axiom, which then starts a new meta-node that becomes the child of the initial meta-node, and so on.

A simple inductive argument over the depth of the nodes in T further shows that $\mu(T)$ covers all the nodes of T. Indeed, the root S_l of T (at depth 1) is evidently the root of a meta-node (and so inside some meta-node). Then if a node A (at depth d) with children A_1, A_2, …, A_k (at depth $d+1$) is already in a meta-node rooted at S_i, then the rewriting rule that generates its children must come from G_i by the definition of a PCGS parse tree and so all A_1, A_2, …, A_k are in the same meta-node. If a query Q_j (depth d) is in some meta-node, then its only child must be S_j (again by the definition of a PCGS parse tree), which means that S_j (depth $d+1$) is the beginning of a new meta-node (and so belong to some, although different meta-node). □

Overall we established a natural bijection μ between every PCGS parse tree and its meta-tree. In passing, the inductive argument used in the proof of Lemma 2 effectively establish an algorithm for computing μ.

Now we can show that a valid derivation can be constructed from every master parse tree.

Lemma 3. *Given a master parse tree T with yield w for some PCGS Γ, one can reconstruct a derivation in Γ that produces w, provided that T was constructed based on a valid derivation in Γ.*

Proof. Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ and let $\mu(T)$ be the meta-tree of T. In what follows the length of a derivation $A \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_m$ is m. We proceed with our proof using a structural induction over $\mu(T)$.

There are no query symbols in any of the leaves of $\mu(T)$, so any derivation consistent with the respective leaf will do. Such a valid derivation can be obtained out of the meta-node using the standard technique used for context-free grammars and their parse trees [12]. At least one such a derivation must exist since the whole tree T comes from a valid derivation in Γ, so the base case is established. It is worth noting additionally that the length of any derivation corresponding to a leaf is equal to the number of internal nodes in that leaf (since each of these

internal nodes correspond to the application of one rewriting rule and so with one step in the derivation).

Consider now the inductive step that is, a meta-node N with κ internal nodes containing exactly all the query symbols $Q_{N_1}, Q_{N_2}, \ldots, Q_{N_p}$ (which are all necessarily leafs). Call the subtrees rooted at the roots of the children of N $N_1, N_2, \ldots, N_p$, respectively, and let their number of internal nodes that are not labeled by query symbols be $\kappa_1, \kappa_2, \ldots, \kappa_p$, again respectively.

The proof of this step proceeds more conveniently if split it into one case for Γ being synchronized and another for Γ being unsynchronized.

Synchronized: During the time N is expanded its children are also expanded by the same number of steps (Γ being synchronized). It follows that the query symbol Q_{N_i} must be generated after precisely κ_i steps (indeed, this is how many steps are in the derivation that produced its sub-parse tree). Out of all the possible derivations that generate N we thus choose one derivation that introduce every Q_{N_i} after κ_i steps. Such a derivation must exist since the whole tree T was generated by a valid derivation to begin with.

The derivation we thus choose happens in the component given by the root of N. The other components' derivations are given by the inductive assumption. Putting all of these together and then satisfying the p query symbols in the usual way (communication and if applicable reduction to axiom for the communicated component) we obtain a whole derivation for N as a parse tree, as desired.

Unsynchronized: We proceed as in the synchronized case, except that now we no longer need to introduce the query symbols after any precise number of steps. We therefore just choose a derivation that introduces all the query symbols present in N. We then proceed in the same manner to construct the complete derivation for the parse tree N. □

Note that the algorithm implied by the construction of Lemma 3 has a high time complexity (since multiple derivations must be tried for most meta-nodes) and can likely be improved. At this stage however just having an algorithm suffices.

Now we can finally state the main result that establishes the usefulness of parse trees for PCGS with context-free components. Indeed, we showed that every derivation in a PCGS has a corresponding parse forest (Lemma 1) and so a corresponding parse tree (the master parse tree of that forest). Once a master parse tree is given, one derivation that generates it can be determined (Lemma 3); this essentially makes the rest of the forest unnecessary. Putting these two points together we have:

Theorem 4. *Every derivation in a PCGS Γ with context-free components that produces a string w (in the sense of the language generated by a PCGS that is, by the first component grammar) has an equivalent parse tree with yield w. Conversely, given a master parse tree T with yield w that has been constructed according to a valid derivation in Γ, one can reconstruct a derivation in Γ that produces w.* □

We essentially showed that a given derivation in a PCGS is characterized by a single parse tree. Whether this goes the other way around (that is, whether any parse tree corresponds to a derivation) has a more complex answer that will be studied in the next couple of sections.

4 Interference and Characterization by Parse Trees

The usefulness of parse trees for context-free grammars is that they characterize exactly all the derivations in a grammar, meaning that every derivation has an equivalent parse tree but also

every parse tree corresponds to at least one derivation in the grammar. The first property is already established for PCGS and their parse trees in Theorem 4 above. PCGS parse trees are a relatively straightforward extension of the concept of (context-free) parse trees, so they have the potential of having the second property as well.

However tempting (and useful) this might be however we show that this is most of the time not the case. More precisely, the existence of the second property will turn out to be dependent on the concept of interference:

Definition 5. CHECKPOINT AND INTERFERENCE: *Let $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a PCGS. A* checkpoint *of component G_i by G_j, $\leq i, j \leq n$ during some derivation in Γ is either the end of the derivation or the event of G_j querying G_i.*

A component G_j interferes *with another component G_i whenever there exists a checkpoint $\mathcal{C}$ of G_i by G_j and a string w_i (the* interference string*) such that (a) $S_i \Rightarrow^*_{G_i} w_i$, and (b) w_i cannot be produced by G_i by $\mathcal{C}$ in the respective derivation of Γ.*

Clearly the components of a PCGS will somehow interfere with each other (else the resulting system will not serve any purpose). Our notion of interference is more restricted and is meant to only identify those cases in which one component controls another in between queries. As it turns out, our notion of interference makes the difference for whether parse trees or forests characterize completely PCGS derivations.

Theorem 5. *Every master parse tree with root S_1 and yield w of a PCGS Γ 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 in Γ.*

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. (Recall that the notion and properties of a meta-tree were introduced in Definition 4 and Lemma 2.)

We consider first the case in which no interference is present and we proceed by structural induction over the (structure of) $\mu(T)$.

A leaf meta-node rooted at S_i, $0 \leq i \leq n$ has no queries anywhere inside it; moreover the rewriting rules that create the leaf come all from a single component. It is thus equivalent to some component-wise derivation starting from the axiom S_i in the respective component grammar G_i. In the absence of interference multiple components can create any possible combination of such leafs concurrently, including the ones corresponding to the actual leafs of the given meta-tree. Indeed we note first that all the components start from their axioms. The fact that there is no interference in Γ means that the components can reach any combination of individual outcomes, including the ones corresponding to the leafs being considered. We can then simply choose a component-wise derivation in Γ that produces a configuration which includes the yields of all the leafs, as desired; the base case is established.

Consider now some meta-node rooted at S_i, $0 \leq i \leq n$ together with all its children rooted at S_{i_1}, S_{i_2}, S_{i_k} (and so introduced by the query symbols Q_{i_1}, Q_{i_2}, Q_{i_k}, respectively). By the same argument as above this meta-node corresponds to a component-wise derivation in the respective component. In the absence of interference the components corresponding to its children have the time to reach any combination of configurations starting from their respective axioms (by induction hypothesis), including the configurations that correspond to the actual children of the node in discussion. The parent node is immaterial in the process as it does not perform any query and so does not interfere with its children. Our meta-node corresponds to a derivation in the respective component resulting in some string that includes the query symbols Q_{i_1},

$Q_{i_2}, \ldots, Q_{i_k}$ (by the definition of a parse tree). These query symbols must then become roots to trees corresponding to derivations in their respective components (again by the definition of a parse tree). The actual children do corresponds to such derivations (by induction hypothesis). Therefore the whole tree corresponds to a derivation, as desired. The induction is complete.

Consider now the checkpoint $\mathcal{C}$ of G_j by G_i such that G_i interferes with G_j at $\mathcal{C}$ in an otherwise successful derivation of Γ with parse tree T. Let w_i be the interference string. The checkpoint $\mathcal{C}$ 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 in Γ since this would imply that w_i is communicated to G_j at checkpoint $\mathcal{C}$ (an impossibility since the interference string w_i is not available at that point). □

5 Complete Characterization of PCGS Derivations with Parse Trees

Theorem 5 together with Definition 5 allows us to easily see which (if any) PCGS variant feature complete characterization of their derivation based on parse trees.

Consider first synchronized PCGS. Such a system can perform "inter-component counting," in the sense that the number of steps being the same in two (or more) components allows for the generation of comparable numbers of symbols in both (all three, etc.). In such a case the number of nodes in the component parse trees is kept synchronized, so taking an arbitrary parse tree from one component and plugging in into the master parse tree will not do (since this synchronization is no longer taken into account). More to the point such an "inter-component counting" is a clear interference by Definition 5 so we cannot have a complete characterization by parse forests:

Corollary 6. *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.*

Proof. Once the existence of a master parse tree not corresponding to any derivation is established, the existence of a parse forest with the same property is immediate (just take the master parse tree thus found and add some arbitrary but valid $n-1$ parse trees for the remaining components).

Interference between different components of a synchronized PCGS is clearly present (since the components are synchronized and therefore restrict each other in the number of derivation steps available). Theorem 5 establishes the desired result.

A constructive proof of this result is also easy to establish by finding an interference string. In order to better illustrate the concept of interference string we also present this proof.

Let $\Gamma = (\{S_1, S_2, S_2'\}, \{Q_1, Q_2\}, \{a, b\}, G_1, G_2)$ be a PCGS with $G_1 = (\{S_1, S_2'\}, \{a, b\}, R_1, S_1)$ and $G_2 = (\{S_2, S_2'\}, \{a, b\}, R_2, S_2)$ such that

$$\begin{aligned} R_1 &= \{S_1 \to aS_1, S_1 \to Q_2, S_2' \to \varepsilon\} \\ R_2 &= \{S_2 \to S_2', S_2' \to bS_2'\} \end{aligned}$$

A derivation in Γ can only proceed as follows: Once the first component queries the second the derivation is almost finished. Indeed, the only possible rewriting will erase S_2' (communicated from the second component), thus reaching a string of terminals. The only successful derivations thus involve some $p+1 > 0$ component-wise derivations followed by a communication step, followed by one final component-wise derivation.

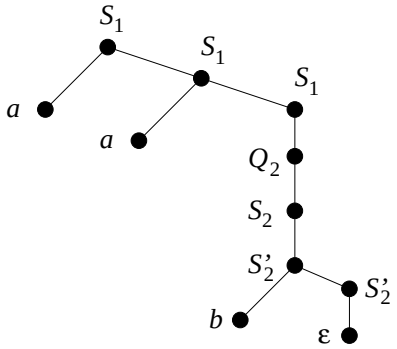

Figure 2: Master parse tree that cannot correspond to any derivation in the respective PCGS

Whenever Q_2 is introduced by the first component-wise derivation ($p = 0$) we have:

$$(S_1, S_2) \Rightarrow (Q_2, S_2') \Rightarrow (S_2', \sigma) \Rightarrow (\varepsilon, \sigma')$$

with σ being either S_2 or S_2' and so σ' being S_2' or bS_2' depending on whether the system is returning or not. In either case the result of the derivation is ε.

If Q_2 is introduced later ($p > 0$) we start with $(S_1, S_2) \Rightarrow (aS_1, S_2')$. Each subsequent $p - 1$ derivations introduce one a in the first component and one b in the second, so after p steps we obtain:

$$(S_1, S_2) \Rightarrow^* (aa^{p-1}S_1, b^{p-1}S_2') = (a^pS_1, b^{p-1}S_2')$$

A final component-wise derivation introduces Q_2 in the first component while the second gains another b, so we have:

$$(S_1, S_2) \Rightarrow^* (a^pQ_2, bb^{p-1}S_2') = (a^pQ_2, b^pS_2')$$

The second component is then communicated to the first, followed by the erasure of S_2' in the first component. Therefore:

$$(S_1, S_2) \Rightarrow^* (a^pb^pS_2', \sigma) \Rightarrow (a^pb^p, \sigma')$$

where σ is either S_2 or b^pS_2' (depending on whether Γ is returning or not) and so σ' is either S_2' or $b^{p+1}S_2'$, respectively. Clearly whether Γ is returning or not is immaterial as far as the string produced by the derivation is concerned.

As argued earlier there is no other possible derivation. We thus conclude that $\mathcal{L}(\Gamma) = \{a^pb^p : p \geq 0\}$.

Consider now the tree depicted in Figure 2. It is a master parse tree for Γ according to Definition 2, but its yield is $aab \notin \mathcal{L}(\Gamma)$ and so cannot have any equivalent derivation. Incidentally, this tree was constructed specifically as a counterexample using the pertinent portion of the proof of Theorem 5 (b being an interference string of G_2 by G_1). □

The unsynchronized, non-returning PCGS do not offer such strong synchronization, so it is reasonable to believe that parse trees might after all describe exactly all their derivations. Unfortunately, we still have interference and so this is yet again not the case:

Corollary 7. *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 some synchronized PCGS that does not correspond to any derivation in that PCGS.*

Proof. Again once the existence of a master parse tree not corresponding to any derivation is established, the existence of a parse forest with the same property is immediate (just take the master parse tree thus found and add some arbitrary but valid $n - 1$ parse trees for the remaining components).

We will establish interference using a system with two components having the following sets of rewriting rules, respectively:

$$\begin{aligned} R_1 &= \{S_1 \to aQ_2, S_2 \to bQ_2, S_2 \to c\} \\ R_2 &= \{S_2 \to xS_2\}. \end{aligned}$$

We have $(S_1, S_2) \Rightarrow^* (aQ_2, x^n S_2) \Rightarrow (ax^n S_2, x^n S_2)$. This first checkpoint does not feature any interference. If the derivation continues however using $S_2 \to bQ_2$ rather than $S_2 \to c$, we get $(ax^n S_2, x^n S_2) \Rightarrow^* (ax^n Q_2, x^{n+k} S_2)$ for some $k \geq 0$. Clearly the second component is forbidden to produce any string $x^l S_2$ with $l \leq n$ even if it is perfectly capable of doing so if left to its own devices. The interference is established and so the result is an immediate consequence of Theorem 5. □

The unsynchronized, returning case is slightly more interesting, though the result is still mostly negative:

Corollary 8. *Let a* unique-query PCGS *be a PCGS in which no rewriting rule contains two or more occurrences of the same query symbol. Then:*

Every parse tree with root S_1 and yield w of an unsynchronized, returning 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 unique-query PCGS.

Proof. If (by contrapositive): If Γ is not unique query then interference happens as follows: Let $A \to \sigma_1 Q_i \sigma_2 Q_i \sigma_3$ be one of the rules of component G_j than makes Γ non-unique-query. G_j then interferes with G_i since G_j imposes a fixed number of steps (zero!) on G_i between the satisfaction of the first and the second occurrence of Q_i. By Theorem 5 at least one parse tree that does not correspond to any derivation in Γ exists.

Only if: When Γ is unique query then no interference is possible. Indeed, any checkpoint reduces the queried component to its axiom, and then the respective component has as much time as needed to derive by itself any possible string. The whole positive part of the proof of Theorem 5 applies literally to this case. □

6 Unique-Query Context-Free PCGS

Between other things, the complete characterization of derivations in unique-query context-free PCGS by parse trees implies that this variant is by far the least powerful PCGS with context-free components possible. Indeed, in such a case we do not gain anything from using a PCGS; a simple context-free grammar will do just as well.

Corollary 9. *Exactly all the languages generated by unique-query, unsynchronized, returning context-free PCGS are context free.*

Proof. That every context-free language can be generated by a unique-query, unsynchronized, returning context-free PCGS is immediate since a context-free grammar is a special case of PCGS (with one component and no use for query symbols).

Let now $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ be a unique-query, unsynchronized, returning context-free PCGS with $G_i = (N \cup K, \Sigma, R_i, S_i)$, $1 \leq i \leq n$. Let σ_i be a renaming such that $\sigma_i(x)$ is the string x in which all the occurrences $\eta \in N$ are replaced by (η, i). We extend naturally σ_i to rewriting rules as $\sigma_i(A \to \alpha) = \sigma_i(A) \to \sigma_i(\alpha)$, and to sets of rewriting rules as $\sigma_i(\rho) = \{\sigma_i(r) : r \in \rho\}$.

Consider then the context-free grammar $G = ((N \times \{1, 2, \ldots, n\}) \cup K, \Sigma, R, (S_1, 1))$, where $R = \bigcup_{0 \leq i \leq n} R'_i$ with $R'_0 = \{Q_j \to (S_j, j) : 1 \leq j \leq n\}$ and $R'_i = \sigma_i(R_i)$, $\leq i \leq n$. We find that there is a natural bijection β between the set of parse trees of Γ and the set of parse trees of G such that the yield of T is identical with the yield of $\beta(T)$ and thus we complete the proof. Indeed, this establishes that $w \in \mathcal{L}(\Gamma) \implies w \in \mathcal{L}(G)$ by Theorem 4, that $w \in \mathcal{L}(G) \implies w \in \mathcal{L}(\Gamma)$ by Theorem 8, and so that $\mathcal{L}(\Gamma) = \mathcal{L}(G)$, as desired.

Intuitively, a derivation in G simulates the derivation in Γ component by component while the respective parse tree is constructed (and so generates an isomorphic parse tree). The nonterminals are the old nonterminals in Γ with an added associated index to keep track which rewriting (i.e., node expansion) happens in which component. The occurrence of a query symbol will change this index in G, meaning that in Γ whatever happens afterward is the result of a derivation in a different component. This all follows faithfully the definition of PCGS parse trees (Definition 2).

Formally, given some parse tree T of Γ we construct $\beta(T)$ in a natural way by considering every meta-node N in the meta-tree $\mu(T)$ and relabeling all the nodes in N with root S_i from l to $\sigma_i(l)$. (Recall that the notion and properties of a meta-tree were introduced in Definition 4 and Lemma 2.)

That β is one-to-one is immediate given that σ_i is one-to-one and that only the node labels are changed by β while the structure of the tree remains the same.

Let now T' be a parse tree of G. T' is rooted at (S_i, i) with $i = 1$, the axiom of G and so $\beta^{-1}(T')$ is rooted at S_i for $i = 1$ (the axiom of G_1). Then every node labeled (A, i) is expanded using some rule from $\sigma_i(R_i)$ (only these are usable given the second component of the label), meaning that the corresponding node in $\beta^{-1}(T')$ is expanded using some rule from R_i. When a node labeled Q_j is encountered, it can only be expanded using the rule $Q_j \to (S_j, j)$, so the label of its sole child is (S_j, j), and so from then on only rules from $\sigma_j(R_j)$ will be applicable (same reason as above). Therefore in $\beta^{-1}(T')$ a node labeled Q_j can only have one child labeled S_j, and from then on only rules from R_j will be applicable. As far as $\beta^{-1}(T')$ is concerned the above description matches exactly Definition 2 and so $\beta^{-1}(T')$ is a parse tree for Γ. The function β is thus onto.

The fact that β is a bijective relabeling of nodes in a tree which does not change terminal symbols established immediately that the yields of T and $\beta(T)$ are the same for any T. □

It is also worth noting that for synchronized context-free PCGS non-returning systems are (not necessarily strictly) weaker than their returning counterparts [8]. This turns out to be reversed in the unique-query, unsynchronized case (returning, unique-query PCGS being the weakest of them all). Indeed, a non-returning unique-query context-free PCGS T can simulate a returning unique-query context-free PCGS S in a trivial manner: we construct the context-free grammar corresponding to S as in Corollary 9 above and we just call that T (that is, a non-returning unique-query context-free PCGS). T is the lowest common denominator of all the context-free PCGS (having one component and no use for queries) so it can be any kind of context-free PCGS. We therefore showed the following:

Corollary 10. *Any returning unique-query context-free PCGS can be simulated by a non-returning unique-query context-free PCGS.* □

We are not aware of any results in the intermediate case (namely, unsynchronized PCGS that are not necessarily unique query).

7 Conclusions

Parse trees have established themselves in the realm of context-free grammars as an authoritative theoretical tool as well as a significant practical tool. We believe that such trees can be equally useful in the realm of PCGS. We therefore introduced the concept of parse trees for context-free PCGS (Definition 2) such that every derivation in a PCGS has an equivalent parse tree (Theorem 4).

A parse tree is however most useful if the correspondence the other way around also holds (meaning that there is a derivation for every parse tree), such that a parse tree characterizes completely a PCGS derivation. We identified the notion of interference (Definition 5), whose presence is exactly all that is needed for the lack of complete characterization by parse trees (Theorem 5). Our incursion into the area thus proved to be substantially less successful than anticipated, as parse trees still cannot fully characterize derivations most of the time (Corollaries 6, 7, and 8).

The bright spot in this whole incursion are the unsynchronized, returning, unique query context-free PCGS which have in this respect very nice properties, namely that parse trees do characterize derivations completely (Corollary 8). As a consequence, this PCGS flavour turns out to have the same generative power as context-free grammars (Corollary 9) that is, to be the weakest of them all (indeed the weakest possible variant of context-free PCGS!).

7.1 Incidental Results

Our results were obtained using the intermediate step of a parse forest (Definition 3), which in the end got reduced to a single parse tree. While parse forests do not convey any more information than their master parse trees, they may be needed as an intermediate step in any process of actually constructing parse trees. That is, the only immediate way arising from our results of actually constructing a parse tree for a context-free PCGS is to construct the parse forest first and then discard all but the first tree of that forest. Parse forests do not have much theoretical utility but may prove to be needed in practice.

The concept of splitting a PCGS parse tree into regions thus obtaining a meta-tree (Definition 4 and Lemma 2) is natural, very simple, and worth remembering. It has proven extremely useful in our effort and might have further utility elsewhere.

A final nod goes to an interesting reversion in the power of returning versus non-returning context-free PCGS. In the synchronized case it has been long known that the non-returning variant can be simulated by the returning flavour, making the returning PCGS more powerful [8]. When it comes to unique-query, unsynchronized PCGS the simulation goes the other way around (Corollary 10). It would be interesting to know how is the wind blowing in the intermediate case (unsynchronized PCGS that are not necessarily unique query); given the role reversal in the extremes, the simulation can essentially go either way, or it may even be that the two variants are not comparable.

7.2 PCGS in Formal Methods?

So what about the bigger picture? Our eventual goal is exploiting PCGS in formal specification and verification. The attention received by PCGS so far was from the formal languages com-

munity and so people have focused on the "more powerful is better" facet of these systems. It did not take long to identify Turing complete variants [6], the apex of such a pursuit. Power (and Turing-completeness in particular) are essential properties for identifying those real-life phenomena (computational or otherwise) that can be modelled using the respective formalism.

Our goal is in this respect less ambitious. Instead of trying to model general computational phenomena, we are aiming at modelling only the interactions of (albeit complex) computing systems with their environment, so that such interactions can be specified and then verified (the bread and butter of formal methods). In this context the facet of interest of any underlying formalism becomes "less powerful is better," as long as the properties of interest can still be modelled. In this respect we note that overall unsynchronized PCGS appear to be more amenable to practical applications, they being less powerful but still expressive enough to model complex, potentially recursive systems. They seem even better suited for the particular task of system specification, as they are arguably closer to the way an actual concurrent system works.

Unfortunately unsynchronized PCGS have received little attention: They have been found to be weaker in terms of generative power compared to their synchronized counterparts, and then they have been effectively ignored. In practice however (and especially in specification and verification) the increased generative power per se does not matter; on the contrary, the simpler the formalism the better it is as long as the features of interest can be modelled by it. It thus appears that unsynchronized PCGS fit this profile very well, and so focusing on this variant is a promising line of investigation.

Other variants have also been proposed, including centralized PCGS [5] (where only the master grammar is allowed to query) or PCGS with terminal transmission [9] (in which only terminal strings can be communicated). They further simplify the formalism which is in general a good thing, but when it comes to formal verification we believe that these further simplifications reduce too much the ability of the formalism to model real-life phenomena. We thus believe that unsynchronized context-free PCGS are in the sweet spot when it comes to formal methods.

This all being said, we note that the unique query variant is the only one that can make full use of parse trees (though whether the use of parse trees is indeed required for our purposes remains to be seen). More interestingly, the computational power of the unique-query variant does not vary with the number of components, which is appealing from the point of view of modelling parallel systems with arbitrary (and possibly dynamic) number of concurrent threads of execution. Whether the unique query restriction is a reasonable restriction remains however to be seen.

In general, before starting to use them in formal methods (or indeed any practical domain), unsynchronized PCGS need to be analyzed thoroughly, especially with respect to generative capacity and closure properties. Such an analysis is as we already mentioned missing almost completely. Doing this is included in our short-term, immediate interests. At the same time the possible ways of modelling the behaviour of complex application software using PCGS need to be investigated, and this is yet another of our immediate interests. Indeed, we moved tentatively our interest from MVPL to PCGS simply because of the necessarily awkward form of an MVPL-based specification; should a PCGS-based specification be equally awkward, our pursuit becomes substantially less interesting.

References

[1] R. ALUR, K. ETESSAMI, AND P. MADHUSUDAN, *A temporal logic of nested calls and returns*, in Pro-

ceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 04), vol. 2988 of Lecture Notes in Computer Science, Springer, 2004, pp. 467–481.

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

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

[6] 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.

[7] R. DE NICOLA AND M. C. B. HENNESSY, *Testing equivalences for processes*, Theoretical Computer Science, 34 (1984), pp. 83–133.

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

[9] H. FERNAU, *Parallel communicating grammar systems with terminal transmission*, Acta Informatica, 37 (2001), pp. 511–540.

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

[11] 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.

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

[13] G. PAUN AND L. SANTEAN, *PCGS: The regular case*, Ann. Univ. Buc., Matem. Inform. Series, 38 (1989), pp. 55–63.

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