Technical Report 2014-002
Context-Free Parallel Communicating Grammar Systems Are Not Linear Space

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
20 November 2014

Abstract

Two contradictory results state that the synchronized parallel communicating grammar systems with context-free components (CF-PCGS) are either Turing complete or less expressive than context-sensitive grammars. We show that the latter result is incorrect. Indeed, we discover that the proof of this result relies on coverability trees for CF-PCGS, but that such coverability trees do not contain enough information to support the proof. We also conclude that coverability trees are not really useful in any pursuit other than the one already considered in the paper that introduces them (namely, determining the decidability of certain decision problems over PCGS).

Keywords: formal languages, theory of computation, formal grammar, parallel communicating grammar system, coverability tree, Turing completeness

1 Introduction

Parallel Communicating Grammar Systems (PCGS for short) have been introduced as a language-theoretic treatment of concurrent (or more general, multi-agent) systems [15]. A PCGS extends the concept of a grammar to a structure that consists of several grammars working in parallel and contributing to the generation of strings by communicating with each other.

In a PCGS one grammar component is considered the master of the system and the other component grammars are called helpers or slaves; they all participate in the derivation but may or may not have a direct impact on the generation of the final string produced by the system. The master grammar controls the derivation which is considered complete as soon as it produces a string of terminals regardless of the state of the strings in the other components (hence the name helper or slave component). In order for the helper components to contribute to the derivation, communication steps (sometimes called query steps) are required. In essence a communication step allows the different components in the system to share strings with one another: A grammar proceeds with a communication step by introducing in its string a request for a string from another grammar. Once a communication step has been introduced, all rewriting steps are put on hold until the communication is complete, meaning they are put on hold until the requesting grammar(s) receive the string from the queried component(s).

Our main area of interest is the generative capacity of PCGS. Generally a PCGS with components of a certain type are more powerful than single grammars of the same type. There have also been other attempts to associate the generative power of PCGS with additional representations, including parse trees [2] and coverability trees [14, 16].

We focus in this paper on two contradictory results about PCGS with context-free components (CF-PCGS for short). On one hand, all flavors of synchronized CF-PCGS were found to be computationally

complete [7, 12]. On the other hand, some authors expected that languages produced by a CF-PCGS would be recognizable by $O(n)$ space-bounded Turing machines [3]; if proved to be true this would lead to the conclusion that context-free PCGS generate only context-sensitive languages, and so context-free PCGS are weaker than context-sensitive grammars. This was all subsequently proven [1].

We set to elucidate the contradiction between the results mentioned above. We discover that the reasoning behind one of the contradicting proofs [1] is sound, but the result is correct only if the concept of coverability tree [16] has certain properties. Such a coverability tree imposes a finite structure over an arbitrary derivation in a context-free PCGS. However, this finite structure cannot guarantee certain limits on the number of nonterminals throughout the derivation, which in turn invalidates the aforementioned proof [1]. Essentially coverability trees can represent some properties of CF-PCGS derivations but we demonstrate that essential information required for a successful derivation will be lost in the coverability tree depiction.

2 Preliminaries

The symbol ε denotes the empty string; ω stands for $|\mathbb{N}|$; $|\sigma|_A$ stands for the length of the string σ after all the symbols not in A have been erased. We further use $|\sigma|_a$ instead of $|\sigma|_{\{a\}}$ for singletons $A = \{a\}$.

A grammar [11] is a quadruple $G = (\Sigma, N, S, R)$. Σ is a finite nonempty set; the elements of this set are referred to as terminals. N is a finite nonempty set disjoint from Σ; the elements of this set are referred to as nonterminals. $S \in N$ is a designated nonterminal referred to as the start symbol or axiom. R is a finite set of rewriting rules, of the form $\alpha \to \beta$ where $\alpha \in (\Sigma \cup N)^* N (\Sigma \cup N)^*$ and $\beta \in (\Sigma \cup N)^*$ (α and β are strings of terminals and nonterminals but α has at least one nonterminal). Given a grammar G, the $\Rightarrow_G$ (yields in one step) binary operator on strings from the alphabet $W = (\Sigma \cup N)^*$ is defined as follows: $T_1 A T_2 \Rightarrow_G T_1 u T_2$ if and only if $A \to u \in R$ and $T_1, T_2 \in (\Sigma \cup N)^*$. We often omit the subscript from the yields in one step operator when there is no ambiguity. The language generated by a grammar $G = (\Sigma, N, S, R)$ is $\mathscr{L}(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$, where $\Rightarrow_G^*$ denotes as usual the reflexive and transitive closure of $\Rightarrow_G$. G is a context-free grammar if every rewriting rule $\alpha \to \beta$ in R satisfies $|\alpha| = 1$ (meaning that α is a single nonterminal) [9, 10].

A synchronized[1] PCGS with $n \geq 1$ components [5] is a tuple $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ where N and Σ are the sets of nonterminals and 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 are the components of the system, with G_1 called the master grammar. The elements of K are called query symbols; their indices points to $G_1, \ldots, G_n$ respectively.

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

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

Item 1 defines a step-wise derivation step and Item 2 defines a communication step. A tuple $(x_1, x_2, \ldots, x_n) \in ((N \cup K \cup \Sigma)^*)^n$ as above is called a *configuration* with components x_i, $1 \leq i \leq n$. With $\Rightarrow_\Gamma^*$ denoting the reflexive and transitive closure of $\Rightarrow_\Gamma$, the language generated by a PCGS Γ

[1] All the PCGS are synchronized in what follows so we henceforth omit this qualifier.

is $\mathscr{L}(\Gamma) = \{w \in \Sigma^* : (S_1, S_2, \ldots, S_n) \Rightarrow_\Gamma^* (w, \sigma_2, \ldots, \sigma_n), \sigma_i \in (N \cup K \cup \Sigma)^*, 2 \le i \le n\}$ (we also omit the subscript Γ whenever no ambiguity is thus introduced).

A PCGS is called *returning* if, after communication, a component which has communicated a string resumes the work from its axiom as described by the phrase "[and $y_{i_j} = S_{i_j}$, $1 \le j \le t$]" in Item 2 above. A PCGS is called *non-returning* if this phrase is erased from Item 2.

Let $N = \{A_1, \ldots, A_{n+m}\}$. For a configuration $w = (w_1, \ldots, w_n)$ of Γ let $M_w = ((|w_1|_{X_1}, \ldots, |w_1|_{X_{2n+m}}), \ldots, (|w_n|_{X_1}, \ldots, |w_n|_{X_{2n+m}}))$, where $X_i = A_i$, $1 \le i \le n+m$, and $X_{n+m+j} = Q_j$, $1 \le j \le n$. $M_w(i, j)$ denotes the element $|w_i|_{X_j}$ of M_w. Let $\mathrm{TR}(\Gamma) = \{(r_1, \ldots, r_n) : r_i \in \Gamma_i \vee r_i = \phi\} \cup \{\Lambda\}$, where ϕ is the "phantom" rule that leaves a string unchanged, and Λ denotes a communication step. A transition $t \in \mathrm{TR}(\Gamma)$ is enabled in a configuration if the corresponding rewriting or communication can be applied in that configuration. We write $M_w \overset{t}{\underset{\Gamma}{\twoheadrightarrow}}$ if t is enabled in the configuration w, and $M_w \overset{t}{\underset{\Gamma}{\twoheadrightarrow}} M_{w'}$ whenever w' is result of applying t on w.

A labeled tree $\mathscr{T} = (V, E, l_1, l_2)$ is a coverability tree [16] for $\Gamma = (N, K, \Sigma, G_1, \ldots, G_n)$ if (V, E) is a tree, $l_1 : V \to (\mathbb{N}_\omega^{2n+m})^n$ is the node labeling function, $l_2 : E \to \mathrm{TR}(\Gamma)$ is the edge labeling function, and the following hold (with $d_{\mathscr{T}}(v_1, v_2)$ denoting the set of exactly all the nodes on the path from v_1 to v_2 in the tree $\mathscr{T}$):

1. The root v_0 is labeled by M_{x_0}, where $x_0 = (S_1, \ldots, S_n)$ (initial configuration).
2. The number of outgoing edges $|v^+|$ of $v \in V$ is
   (a) 0 if either no transition is enabled at $l_1(v)$ or there exists $v' \in d_{\mathscr{T}}(v_0, v)$ such that $v \ne v'$ and $l_1(v) = l_1(v')$, and
   (b) the number of transitions enabled at $l_1(v)$ otherwise.
3. For any $v \in V$, $|v^+| > 0$ and any transition t enabled at $l_1(v)$, there exists $v' \in V$ such that $(v, v') \in E$, $l_2(v, v') = t$, and $l_1(v')$ is determined as follows:

   Let $l_1(v) \overset{t}{\underset{\Gamma}{\twoheadrightarrow}} M$. If M contains queries then $l_1(v') = M$. Otherwise for all $1 \le i \le n$ and $1 \le j \le 2n+m$: if there exists $v^* \in d_{\mathscr{T}}(v_0, v)$ such that $l_1(v^*) \le M$ and $l_1(v^*)(i, j) < M(i, j)$, then $l_1(v')(i, j) = \omega$, otherwise $l_1(v')(i, j) = M(i, j)$. The introduction of an ω label is called an ω *breakpoint*.

The coverability tree for any CF-PCGS is always finite and can be effectively constructed [16].

3 Previous Work

Non-returning PCGS with context free components can generate all recursively enumerable languages [12]. Combined with the fact that non-returning systems can be simulated by returning systems [8] based on an earlier result [13], this result establishes that returning PCGS with context-free components are also computationally complete. Direct proofs of Turing completeness for returning CF-PCGS also exist [4, 6, 7].

Another line of investigation contradicts the result described above. A proof that the languages generated by ε-free[2], non-returning CF-PCGS can be accepted in linear space [1] (and so they are context-sensitive [11]) also exists. The construction that establishes the proof is a Turing machine with input σ that uses n work tapes to maintain a configuration $w = (w_1, \ldots, w_n)$ which is repeatedly rewritten according to the PCGS being simulated. Those strings w_i that are shorter than the input σ are kept in

[2] A grammar or PCGS is ε-free whenever rules of the form $A \to \varepsilon$ are not used.

clear. Components w_i longer than σ will not participate directly in the production of σ, but they may still affect the derivation through various side effects. These side effects however depend only on the kind and number of nonterminals in the string, and so these strings are maintained in the following form:

$$w_i = @m_1X_1 \dots m_jX_jm_{j+1}Q_1 \dots m_{j+k}Q_k, \tag{1}$$

with $@ \notin N \cup \Sigma \cup K$. $X_1, \dots, X_j$ and $Q_1, \dots, Q_k$ are all the distinct nonterminals and query symbols in w_i, respectively. The number of occurrences of each nonterminal or query symbol in w_i is given by m_h, $1 \leq h \leq j+k$. All the strings are then rewritten in the usual fashion (with obvious modifications for those strings of form (1)) until σ is produced by the first component or the derivation is blocked [1].

The construction uses $O(|\sigma|)$ storage space as long as an upper bound $m_{\max}$ exists for the values of the counters m_h from the strings of form (1), in the sense that either X_h cannot exceed $m_{\max}$, or once $m_{\max}$ is exceeded then the counter will be always able to maintain itself above $m_{\max}$ (case in which the value of m_h is for all practical purposes equivalent to ω, can be marked as such, and will remain so throughout the derivation). The bound $m_{\max}$ should further be either independent of $|\sigma|$ or at most linear in $|\sigma|$. One such a bound was apparently found [1] starting from the coverability tree of the PCGS being simulated: It is immediate from the construction of this tree and a pumping argument that given a configuration w such that $M_w(i,j) = \omega$ for some component w_i and some nonterminal X_j then the number of occurrences of X_j in the i-th component can be made arbitrarily large. It is tempting to conclude (and it has been so concluded in the original proof) that once $M_w(i,j) = \omega$ then X_j cannot be totally removed by any successive derivation steps from x_i; therefore $m_{\max}$ can be determined by constructing the coverability tree and taking the maximum number $M_w(i,j) \neq \omega$ therein as $m_{\max}$.

4 Why CF-PCGS Are Not Linear Space

The problem with the result described above [1] is the existence of the limit $m_{\max}$ (so that a nonterminal whose number of occurrences surpasses $m_{\max}$ can be considered available in an infinite supply and so its number of occurrences can be replaced with ω). This limit was established using coverability trees. It would appear however that such a limit does not in fact exist.

4.1 An Intuitive Example

While it is true that the number of nonterminals can grow unbounded after an ω breakpoint, this only means that there will always be *some* derivation that makes them so; there is however no guarantee that such a derivation is the successful one. The unbounded growth of some nonterminal is thus not necessarily a feature of a successful derivation. The following example will illustrate this:

Let $E_1 = (\{S_1,S_2\},\{Q_1,Q_2\},\{a,b\},(\{S_1,Q_2\},\{a\},R_1,S_1),(\{S_2\},\{b\},R_2,S_2))$, where

$$\begin{aligned} R_1 &= \{S_1 \to aS_1, S_1 \to aQ_2\} \\ R_2 &= \{S_2 \to S_2S_2, S_2 \to b\} \end{aligned}$$

This system will generate as many S_2 in the second component as a in the first. Then all the S_2 symbols in the second component will be rewritten as b while the first component keep generating a symbols. Any query introduced before all the S_2 disappear will block the derivation. Finally one more step causes the first component to query the second, so that $\mathscr{L}(\Gamma) = \{a^{2n+1}b^{n+1} : n \geq 0\}$.

The coverability tree corresponding to this system is shown in Figure 1. We have omitted the edge labels with the exception of Λ. In order to facilitate the reference to the nodes of the tree we have given them the additional labels v_k, $0 \leq k \leq 7$.

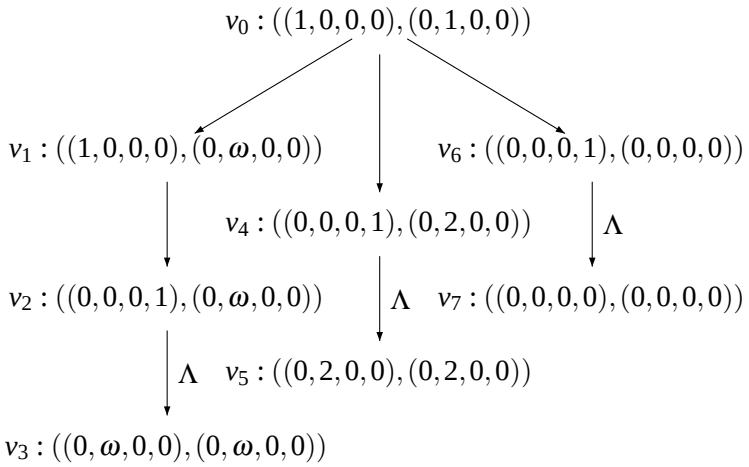

Figure 1: The coverability tree of the sample PCGS E_1.

The paths $v_0 \to v_4 \to v_5$ and $v_0 \to v_6 \to v_7$ are clear. They are caused by the first component introducing Q_2 in the first derivation step. The second component can use either its first or its second rule, resulting in these two paths (the first blocked and the second successful).

The path $v_0 \to v_1 \to v_2 \to v_3$ is a bit more complicated, as it corresponds to all the other derivations in the system. As long as the first component does not query, the second one is free to use its rule $S_2 \to S_2S_2$ as many times as it wishes, hence the occurrence of an "ω breakpoint," meaning that ω appears for S_2 in v_1. This means that there is at least one derivation from v_1 that can increase arbitrarily the number of occurrences of S_2. While this is certainly true, we do note that any successful derivation will nonetheless start at some point to decrease the number of occurrences of S_2 (starting from v_3) in order to reach a terminal string. The outcome of a successful derivation thus depends on the actual number of occurrences of S_2 even if this number is no longer remembered in the coverability tree. Therefore this example shows that the coverability tree does not contain enough information for reconstructing derivations.

The original proof [1] falls precisely into this pitfall. In the particular example of E_1 $m_{\max}$ will be set to 2 based on the coverability tree, whereas the number of occurrences of S_2 can be arbitrarily large and yet significant for some derivation.

The problem is however illustrated by the example above only on an intuitive level. Indeed, in the simulation of E_1 by a Turing machine as outlined in Section 3 no tape content exceeds the length of the input and so no component is ever rewritten in form (1) (the "@ form" henceforth).

4.2 An Actual Counterexample

In order to establish an actual counterexample we consider the following, less intuitive but this time complete example:

$$\begin{aligned}
E_2 &= (\{S,A,O,P,X,Y,Z,\},\{Q_1,Q_2,Q_3\},\{a\},\\
&\quad G_1 = (\{P,S,X,Q_3\},\{a\},R_1,S),\\
&\quad G_2 = (\{O,X,Z\},\{a\},R_2,Z),\\
&\quad G_3 = (\{A,O,P,X,Y\},\{a\},R_3,A))\\
R_1 &= \{S \to S, S \to Q_3, X \to a, P \to a\}\\
R_2 &= \{Z \to OX, X \to XX\}\\
R_3 &= \{A \to A, A \to Q_2, O \to P, X \to YYY\}
\end{aligned}$$

The master G_1 can wait indefinitely before it requests a string from G_3. In the meantime G_2 generates an arbitrary number of X symbols. During this time G_3 can also wait indefinitely, then query the string from G_2, then rewrite all the nonterminals from the string thus communicated (to non-rewritable nonterminals), then block the derivation. At the same time the master rewrites its nonterminals into terminals, but does not have the time (because of G_3) to complete this task. It turns out that no derivation in this system can be successful.

Formally, the system can only proceed as follows up to the first communication step, for some arbitrary $n \geq 0$:

$$(S,Z,A) \Rightarrow^* (S,OX^n,Q_2)$$

If the master queries before G_3 then the derivation will block because the master does not have a rewriting rule for A, hence we need to introduce Q_2 in the second component rather than Q_3 in the first to have any chance of completing the derivation. We then have:

$$(S,OX^n,Q_2) \stackrel{\Lambda}{\Rightarrow} (S,OX^n,OX^n) \Rightarrow (Q_3,w,PX^n)$$

Once G_2 has been queried its contribution to the derivation is complete. It will continue to generate an additional X with every derivation step, but it will never be queried or indeed participate in the derivation in any other way. We will thus replace its content in what follows with a generic w, with the understanding that this w changes throughout the derivation yet its actual value is immaterial.

After the communication step above G_3 can use either of the rules $O \to P$ or $X \to YYY$. If the latter rule is used then the derivation will eventually block. Indeed, the third component must be communicated to the master for the derivation to have a chance to succeed, but the master does not have a rule for rewriting Y and so at the moment of communication Y cannot appear in G_3. The only was this can happen is for G_3 to rewrite O and for the master to introduce Q_3 at the same time, as above. In such a case the derivation will continue as follows:

$$(Q_3,w,PX^n) \stackrel{\Lambda}{\Rightarrow} (PX^n,w,PX^n) \Rightarrow^n (m,w,PY^{n\times 3}) \tag{2}$$

The third component does not have any rewriting rules for the remaining non-terminals, and so the derivation stops here. The master string contained $n+1$ nonterminals to begin with (that is, just after the communication step when it was PX^n) and the n subsequent rewriting steps will rewrite n of those to a, but will leave one nonterminal in the string (either an X or a P). This means that the resulting string m cannot be in $\mathscr{L}(E_2)$ since it contains nonterminals; in other words the derivation blocks without producing a string in $\mathscr{L}(E_2)$.

As argued throughout the description above no other derivation path has even the slightest chance of succeeding. We thus conclude that no matter which derivation path is taken the system eventually blocks, and so $\mathscr{L}(E_2) = \emptyset$.

Consider now the Turing machine simulation of E_2 as presented in Section 3 and working on input a^k for some $k > m_{\max} + 2$. Recall that m_{max} does not depend on the input of the Turing machine (since it is determined before any input is presented to the machine, based on the coverability tree) and so such a k will always exist.

Let the Turing machine simulate a derivation of E_2 as above with $n = k-1$. We already know that such a path is unsuccessful in E_2 (because the third component blocks prematurely). However, in the Turing machine simulation the third component string becomes longer than a^k (this being caused by the application of the rule $X \to YYY$), so it will be rewritten in the @ form, and so as we will see shortly it will fail to block the derivation.

Recall that the string of G_3 evolves in the final stage of the derivation (that is, the $\Rightarrow^*$ phase from Equation (2)) along the following line:

$$PX^n \Rightarrow PY^3X^{n-1} \Rightarrow \cdots \Rightarrow PY^{3i}X^{n-i} \Rightarrow \cdots \Rightarrow PY^{3n}X^0 \tag{3}$$

The occurrences of X and Y can actually be interleaved with each other; however, this interleaving is immaterial from the point of view of the Turing machine simulation, which will rewrite the G_3 component in an @ form as soon as $i = 1$. Indeed, note that $n = k-1$, so $|PY^{3\times i}X^{n-i}| = 1+3i+k-1-i = 2i+k$, and so $|PY^{3\times i}X^{n-i}| > k$ for any $i \geq 1$. The Turing machine will therefore simulate the derivation shown in Equation (3) on its respective (third) work tape as follows:

$$PX^n \Rightarrow @1P3Y(n-1)X \Rightarrow \cdots \Rightarrow @1P(3i)Y(n-i)X \Rightarrow \cdots \Rightarrow @1P(3n)Y0X$$

The simulation continues to work correctly in this form. However, the simulation also uses the rewriting to ω of the X counter. Indeed, $k > m_{\max}+2$, therefore $n > m_{\max}+1$, and so $n-1 > m_{max}$. The simulation above thus becomes[3]:

$$PX^n \Rightarrow @1P3Y\omega X \Rightarrow \cdots \Rightarrow @1P(3i)Y\omega X \Rightarrow \cdots \Rightarrow @1P(3n)Y\omega X$$

The absence of X in the third component no longer happens (for recall that the simulation will never modify an ω counter). Therefore the simulation no longer blocks and so the input string a^k is inadvertently accepted. The counterexample is established.

It may be worth noting that $m_{\max} = 6$ for E_2. This value was computed based on a large coverability tree, which contains more than 50 nodes and so it is not included in this manuscript[4]. It follows that the input string a^9 is accepted by the Turing machine simulation (and so is a^k for any $k \geq 9$). Indeed, Figure 2 shows how the three "component" work tapes of the Turing machine evolve during the acceptance run for a^9.

5 Conclusions

The counterexample E_2 developed in Section 4 shows that the proof that CF-PCGS only generate context-sensitive languages [1] is incorrect: the use of the coverability tree to determine the value of $m_{\max}$ is not adequate. This counterexample also suggests that no such $m_{\max}$ exists.

In the process we also discovered the limitations of coverability trees. Indeed, it is instructive to compare the structure of the coverability tree of E_1 (Figure 1) with the set of possible derivations in E_1. We note that the tree does characterize to some degree all the possible derivations in the system, but at the same time does not characterize them completely. In particular the "downslope" of a successful derivation (when the number of occurrences of S_2 is decreased) does not have any correspondent in the tree. We believe that this shows in an eloquent manner the limitations of these trees.

One could argue that coverability trees offer a simple way of summarizing a complex system; however critical information is necessarily lost in a coverability tree representation given its finite nature. We believe that coverability trees are not really useful in any pursuit other than the one already considered in the paper that introduces them (namely, determining the decidability of certain decision problems over PCGS [16]).

References

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

[3] Note in passing that the counter for Y will also exceed $m_{\max}$ at some point, so the simulation will also replace it with ω. Since the values of this counter is immaterial to the discussion at hand we have not performed such a replacement.

[4] It should be emphasized once more that a finite coverability tree exists by definition and that the actual value of $m_{\max}$ is immaterial for the validity of our counterexample. The absence of the coverability tree from this manuscript therefore affects neither the correctness nor the completeness of our counterexample.

Work tape 1:	Work tape 2:	Work tape 3:	
S	Z	A	$\Rightarrow$
S	OX	A	$\Rightarrow$
S	OXX	A	$\Rightarrow$
S	$OXXX$	A	$\Rightarrow$
S	$OXXXX$	A	$\Rightarrow$
S	$OXXXXX$	A	$\Rightarrow$
S	$OXXXXXX$	A	$\Rightarrow$
S	$OXXXXXXX$	A	$\Rightarrow$
S	$OXXXXXXXX$	Q_2	$\Rightarrow$
S	$OXXXXXXXX$	$OXXXXXXXX$	$\Rightarrow$
Q_3	$@1O\omega X$	$PXXXXXXXX$	$\Rightarrow$
$PXXXXXXXX$	$@1O\omega X$	$PXXXXXXXX$	$\Rightarrow$
$PaXXXXXXX$	$@1O\omega X$	$@1P3Y\omega X$	$\Rightarrow$
$PaaXXXXXX$	$@1O\omega X$	$@1P6Y\omega X$	$\Rightarrow$
$PaaaXXXXX$	$@1O\omega X$	$@1P\omega Y\omega X$	$\Rightarrow$
$PaaaaXXXX$	$@1O\omega X$	$@1P\omega Y\omega X$	$\Rightarrow$
$PaaaaaXXX$	$@1O\omega X$	$@1P\omega Y\omega X$	$\Rightarrow$
$PaaaaaaXX$	$@1O\omega X$	$@1P\omega Y\omega X$	$\Rightarrow$
$PaaaaaaaX$	$@1O\omega X$	$@1P\omega Y\omega X$	$\Rightarrow$
$Paaaaaaaa$	$@1O\omega X$	$@1P\omega Y\omega X$	$\Rightarrow$
$aaaaaaaaa$	$@1O\omega X$	$@1P\omega Y\omega X$	

At this point the first work tape (corresponding to the master component grammar) is identical with the input tape and so the input is accepted.

Figure 2: The run of the Turing machine simulation of the sample PCGS E_2 that inadvertently accepts a^9.

[2] S. D. Bruda and M. S. R. Wilkin, *Parse trees for context-free parallel communicating grammar systems*, in Proceedings of the 13th International Conference on Automation and Information (ICAI 12), Iasi, Romania, June 2012, pp. 144–149.

[3] L. Cai, *The computational complexity of linear PCGS*, Computers and Artificial Intelligence, 15 (1996), pp. 199–210.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

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

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

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

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

[14] V. MIHALACHE, *On the expressiveness of coverability trees for PC grammar systems*, in Grammatical Models of Multi-Agent Systems (Topics in Computer Mathematics), Gordon and Breach Science Publishers, 1999.

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

[16] F. L. TIPLEA, C. ENE, C. M. IONESCU, AND O. PROCOPIUC, *Some decision problems for parallel communicating grammar systems*, Theoretical Computer Science, 134 (1994), pp. 365–385.