Complexity Results on Balanced Context-Free Languages

Akihiko Tozawa[1] and Yasuhiko Minamide[2]

[1] IBM Research,
Tokyo Research Laboratory, IBM Japan, Ltd.
[2] Department of Computer Science
University of Tsukuba

Abstract. Some decision problems related to balanced context-free languages are important for their application to the static analysis of programs generating XML strings. One such problem is the balancedness problem which decides whether or not the language of a given context-free grammar (CFG) over a paired alphabet is balanced. Another important problem is the validation problem which decides whether or not the language of a CFG is contained by that of a regular hedge grammar (RHG). This paper gives two new results; (1) the balancedness problem is in PTIME; and (2) the CFG-RHG containment problem is 2EXPTIME-complete.

1 Introduction

The study of balanced context-free languages or parenthesis context-free languages dates back to the 1960s and 1970s [McN67, Knu67, Tak75]. Recently balanced context-free languages attract new interests because of their application to XML-related problems [BB02b, BB02a, KM06], e.g., the static analysis of programs generating XML strings. In the previous work [MT06], we give algorithms to two such problems. This paper continues that study and gives answers to problems previously left open.

Let A be a base alphabet. Then, we introduce a paired alphabet consisting of two sets $\acute{A}$ and $\grave{A}$:

$$\acute{A} = \{\, \acute{a} \mid a \in A \,\} \qquad \grave{A} = \{\, \grave{a} \mid a \in A \,\}$$

where $\acute{A}$ and $\grave{A}$ correspond to the set of start tags and the set of end tags, respectively. We consider that $\acute{a}$ and $\grave{a}$ match. We write Σ for $\acute{A} \cup \grave{A}$. Then the fundamental notion on a string over a paired alphabet is whether it is balanced. For example, $\acute{a}\acute{b}\grave{b}\acute{c}\grave{c}\grave{a}$ and $\acute{a}\acute{a}\grave{a}\acute{b}\grave{b}$ are balanced, but $\acute{a}\grave{b}$ and $\acute{a}\acute{b}\grave{b}$ are not. This notion of balanced strings corresponds to well-formed documents in XML. We call the set of all balanced strings $B(\Sigma)$ the Dyck set over Σ [Ber79].

We consider context-free grammars over paired alphabets. The first problem to ask is the balancedness problem. Namely, whether or not the language of a given context-free grammar (CFG) G is balanced, or whether or not $L(G) \subseteq B(\Sigma)$. To

H. Seidl (Ed.): FOSSACS 2007, LNCS 4423, pp. 346–360, 2007.

our knowledge, the best known algorithm for this problem requires exponential time. However this is not optimal. We will prove that this problem in actually in PTIME.

This PTIME algorithm consists of two steps. The first step is to check the balancedness of the language as a grammar of a single kind of parentheses. We here use a fixpoint algorithm based on the algorithms by Knuth [Knu67], and Berstel and Boasson [BB02b]. However, we give a finer analysis of the number of iterations needed to reach fixpoint, which at first glance seems to be exponential to the size of the grammar, but in fact it is linear. The second step is to check each matched letters are of the same kind. Consider a CFG G with a singleton language. Such a CFG is sometimes called a straight line program (SLP). Assume that in this G, we have a production rule $I \to XY$ whose X and Y derive strings $\phi \in \acute{A}^*$ and $\psi \in \grave{A}^*$, respectively, of the same length. Now clearly, the singleton language of G is balanced if ϕ is identical to the reverse of ψ by ignoring $\acute{}$ and $\grave{}$ signs. Plandowski has shown that the problem of deciding the equivalence of two SLPs, and hence the balancedness of this $L(G)$, is in PTIME [Pla94]. We later show how to apply Plandowski's algorithm to the problem for general CFGs.

The second problem is the validation problem. Our previous work discussed the problem whether $L(G) \subseteq L(G')$ holds where G is a CFG and G' is either an (i) XML grammar or (ii) regular hedge grammar (RHG). In particular, the RHG defines an important language class corresponding to regular languages over trees. Indeed, any regular hedge language can be defined only from the following two kinds of productions.

$$X \to \acute{a}Y\grave{a}Z \quad \text{or} \quad X \to \epsilon.$$

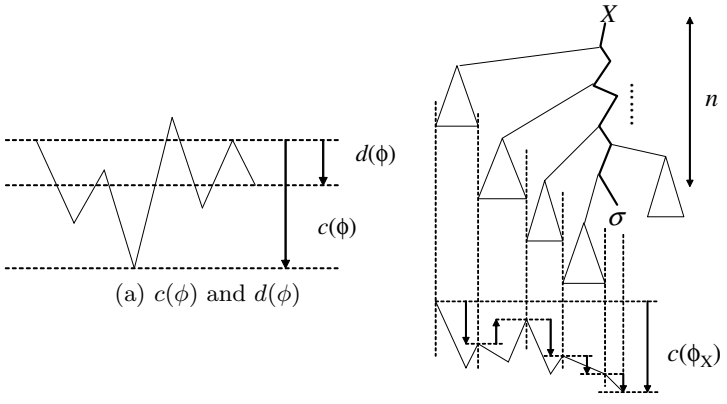
This corresponds to non-deterministic tree automata (NTA) on binary trees. The best known time complexity for the CFG-RHG containment is doubly exponential. This is actually optimal. In this paper, we prove that this problem is 2EXPTIME-hard.

A parenthesis grammar (PG) [McN67, Knu67] is a grammar with a single kind of parentheses, e.g., [and], and with production rules restricted to the following form.

$$X \to [Y_1 \cdots Y_k] \quad \text{or} \quad X \to a$$

Here letters a can be considered as the abbreviation of $\acute{a}\grave{a}$. It is then easy to see that the class of the PG is a subclass of the class of the RHG. We actually prove that the containment $L(G) \subseteq L(G')$ where G' is PG is already 2EXPTIME-hard. The idea of the proof comes from the observation of the gap between derivation trees and parse trees of balanced languages generated from CFGs. Namely, a single node on a parse tree, i.e., matching parentheses, can be split to two $2^{O(n)}$-distant nodes in the corresponding derivation tree.

The rest of the paper is organized as follows. Section 2 explains the algorithm for the balancedness problem. Section 3 proves 2EXPTIME-hardness of the CFG-RHG containment problem. Section 4 summarizes the related work.



(a) $c(\phi)$ and $d(\phi)$

(b) derivation tree of ϕ_X

Fig. 1. Illustration of $c(\phi)$, $d(\phi)$ and ϕ_X

2 PTIME Balancedness Check

We develop an algorithm which decides in polynomial time whether or not the language of a context-free grammar is balanced. The algorithm has two steps. In the first step, we only check *shapes* of strings. The language of a grammar is shape-balanced iff it is balanced by treating it as if using a single pair of parentheses, e.g., $\acute{a}\grave{b}$ is not balanced, but shape-balanced. In this step, we also pick up a string with the deepest valley in the set of strings produced from each nonterminal. The second step is the check of *color-balancedness*, i.e., we never see corresponding $\acute{a}$ and $\grave{b}$ such that $a \neq b$.

In this section, we assume a grammar $G = (\Sigma, V, R, I)$ such that all production rules are in the form $X \to \alpha$ or $X \to \alpha\beta$ where $\alpha, \beta \in \Sigma \uplus V$. We can convert arbitrary CFGs into this form in linear size[1]. We also assume that G is reduced. That is, every nonterminal is accessible from the start symbol and every nonterminal produces at least one terminal string. By a notation $C[]$, we mean a string containing a hole, which is filled with a string ϕ as $C[\phi]$.

2.1 Checking Balancedness of Shapes

The shape of a string is intuitively understood by reading this string from left to right as a line graph. Each letter $\grave{a}$ corresponds to a descending slope of the graph, and each $\acute{a}$ to a climbing slope of the same unit, respectively. Now we obtain the shape of a string by

- Keeping the levels of both ends of this line graph.
- Erase all valleys other than the one deepest in the graph.

[1] More precisely, we obtain G such that $L(G) = L(G') \setminus \{\epsilon\}$ from G'. Removing ϵ does not change the balancedness.

Formally, we say a string ϕ is shape-balanced if repeatedly removing all matching $\acute{a}\grave{b}$ from ϕ gives an empty string. Any string $\phi \in \Sigma^*$ is reduced to the following form with this reduction.

$$\grave{a}_i \cdots \grave{a}_1 \acute{b}_1 \cdots \acute{b}_j$$

We identify the shape of ϕ by $c(\phi)$ and $d(\phi)$ defined as $c(\phi) = i$ and $d(\phi) = i - j$. Here $c(\phi)$ is a nonnegative integer denoting the depth of the deepest valley measured from the left end, and $d(\phi)$ is a possibly-negative integer denoting the level of the right end measured from the left end (cf. Fig. 1(a)). A string ϕ is shape-balanced iff $c(\phi) = d(\phi) = 0$. The language $L(G)$ of a grammar G is shape-balanced if all strings in the language are shape-balanced.

If a grammar has a shape-balanced language, each language corresponding to its nonterminal X has constant $d(\phi)$, i.e., $d(\phi) = d(\psi)$ if $X \stackrel{*}{\Rightarrow} \phi, \psi$. Furthermore, for each nonterminal X, there is a bound m such that $X \stackrel{*}{\Rightarrow} \phi$ implies $c(\phi) \leq m$. To see this, consider the derivation $I \stackrel{*}{\Rightarrow} C[X]$. This $C[]$ must be in the form $\psi_0 \acute{a}_1 \psi_1 \cdots \acute{a}_i \psi_i [] \zeta_j \grave{b}_j \cdots \grave{b}_1 \zeta_0$ with $\psi_0, \ldots, \psi_i, \zeta_0, \ldots, \zeta_j$ shape-balanced. Since $c(C[\phi]) = d(C[\phi]) = 0$ for any $X \stackrel{*}{\Rightarrow} \phi$, we have $c(\phi) \leq i$ and $d(\phi) = i - j$.

The bound of $c(\phi)$ implies the existence of, not necessarily unique, element ϕ_X such that $X \stackrel{*}{\Rightarrow} \phi_X$ with maximum depth $c(\phi_X)$ $(= m)$. Here is the main proposition about this ϕ_X.

Proposition 1. *Assume $G = (\Sigma, V, R, I)$ such that $L(G)$ is shape-balanced. For each $X \in V$, we have ϕ_X with maximum $c(\phi_X)$, such that the height of the derivation tree is bounded by $2n + 1$ where $n = |V|$.*

Note that $L(G)$ is shape-balanced iff $c(\phi_I) = d(\phi_I) = 0$. This proposition bounds the number of iterations to find out ϕ_I, which is linear to the size of the grammar. To prove this proposition, we need analysis on primary paths of derivation trees.

See Fig. 1(b). This figure explains how given a derivation tree for $X \stackrel{*}{\Rightarrow} \phi$, we compute the depth of each valley in ϕ. Assume a path $t_0, t_1, \ldots, t_k (= t)$ in the derivation tree where t_0 is the root, and t is a leaf labeled either as $\acute{a}$ or $\grave{a}$. We would like to compute the depth $c(t)$ of the valley found around this occurrence of $\acute{a}$ or $\grave{a}$. Intuitively, this value is computed from the sum of all $d(\psi)$ where ψ is a substring corresponding to each branch appearing in the left of the path to t.

In this path, each non-leaf node t_i is labeled by a nonterminal $X \in V$ and associated with a rule $X \to \alpha$ or $X \to \alpha\beta \in R$. If t_{i+1} is the first successor of t_i labeled by α, the derivation by $X \to \alpha$ or $X \to \alpha\beta$ does not contribute to deepening the valley around t. On the other hand, if t_{i+1} is the second successor labeled by β, $c(t)$ is increased by $d(\psi)$ where $\alpha \stackrel{*}{\Rightarrow} \psi$ is a sub-derivation for the first successor of t_i. If the leaf node t is labeled by $\acute{a}$, the valley exists on the left of this occurrence of $\acute{a}$. On the other hand, if it is $\grave{a}$, the valley exists on the right, so that we increase $c(t)$ by 1. Now the *primary path* of ϕ is a path to a leaf t in the derivation tree with maximum $c(t)$. Then clearly this $c(t)$ is equal to $c(\phi)$.

Now assume that the grammar has a shaped-balanced language. Then the problem of finding out ϕ_X maximizing $c(\phi_X)$ becomes the problem of finding out a primary path, in a certain derivation tree for X, to the leaf t maximizing $c(t)$.

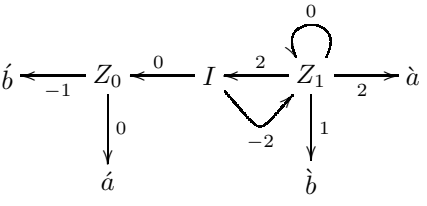
This problem can be considered as a graph problem. We can draw a graph whose vertices correspond to $V \uplus \Sigma$, and whose edge from X to γ corresponds to $X \to \gamma$ or $X \to \alpha\beta \in R$ such that either $\gamma = \alpha$ or β. Now we assign the weight corresponding to the increase in the depth $c(t)$ to each edge of this graph. The grammar has a shaped-balanced language, so that we can determine constant $d(\phi_X)$ for each nonterminal X. Let $d(\phi_{\acute{a}}) = -1$ and $d(\phi_{\grave{a}}) = 1$.

- For each edge from X to α where $\alpha \in V \uplus \acute{A}$,
  - if this edge corresponds to $X \to \alpha$ or $X \to \alpha\beta$, we give the weight 0,
  - if this edge corresponds to $X \to \beta\alpha$, we give the weight $d(\phi_\beta)$.
- For each edge from X to $\grave{a}$,
  - if this edge corresponds to $X \to \grave{a}$ or $X \to \grave{a}\beta$, we give the weight 1,
  - if this edge corresponds to $X \to \beta\grave{a}$, we give the weight $d(\phi_\beta) + 1$.

For example, consider the following grammar with the shape-balanced language.

$$I \to Z_0 Z_1 \qquad Z_0 \to \acute{a}\acute{b} \qquad Z_1 \to Z_1 I \qquad Z_1 \to \grave{b}\grave{a}$$

In this grammar, we always have $d(\phi) = -2$ if $Z_0 \stackrel{*}{\Rightarrow} \phi$, so that the edge from I to Z_1 is given weight -2.



The problem of finding out a primary path with maximum weight first corresponds to the detection of *positive cycles* in the graph. If there is no such cycle, the problem reduces to the longest (maximum-weight) path problem.

Indeed, if the grammar has a shaped-balanced language, the graph constructed as such has no positive cycles. If there is such a cycle, clearly we fail to find any primary path with maximum weight, contradicting the shape-balancedness. On the other hand, if there is no such cycle, then for any path containing the same vertex twice, we can always find another path with at least the same weight, and without such duplicated occurrence of vertices. Now this proves that we can always find a maximum-weight path of length at most $n + 1$.

Fig. 1(b) illustrates how we construct ϕ_X. The length of the primary path in ϕ_X can be made at most $n+1$. We can use arbitrary derivation trees for subtrees not related to the primary path, whose height can be made at most $n + 1$. To sum up, the height of the derivation of ϕ_X can be made at most $2n + 1$, proving Proposition 1.

2.2 Straight Line Programs for ϕ_X

If only concerning the shape-balancedness, it is enough to compute the shape of this ϕ_X, i.e., $c(\phi_X)$ and $d(\phi_X)$. This is rather easy. However for the color-balancedness, we need to compute ϕ_X themselves. Unfortunately, in the worst

case, ϕ_X are not of polynomial length, so that we cannot obtain a PTIME algorithm if they are explicitly represented as strings. We here consider the compressed string representation using sharing graphs.

Definition 1. *(Straight Line Program) A straight line program (SLP) is an acyclic CFG without alternatives in production rules.*

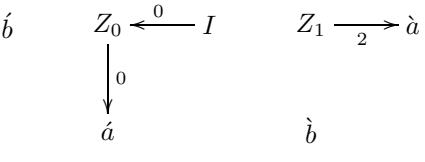
In other words, an SLP is a CFG generating a singleton set. We give an algorithm to find each ϕ_X as an SLP in Fig. 2. This algorithm combines the fixpoint algorithm of shape-balancedness check with Bellman-Ford's algorithm for the longest path problem where the graph does not contain positive cycles [Law76].

This algorithm needs to be repeated $2n+1$ times. After S_{2n+1} is computed, we first check $S_{2n+1}(X) = S_{2n}(X)$ for all $X \in V$. If so, this means that the algorithm reaches the fixpoint, so that it could find ϕ_X with maximum depth for each X. Otherwise, the algorithm could not find ϕ_X with the height of derivation $2n+1$, so that the shape-balancedness has failed from Proposition 1. The first and second component of $S_{2n}(X)$ denote $c(\phi_X)$ and $d(\phi_X)$, respectively, so that we then check that $S_{2n}(I) = (0, 0, \_)$.

After the shape-balancedness check has passed, we construct ϕ_X for each nonterminal X of the grammar. The third component of $S_{2n}(X)$ corresponds to primary paths in the SLP for ϕ_X. We introduce a set of nonterminals $\overline{V} = \{\overline{X} \mid X \in V\}$ and define the set of rules P in the form $\overline{X} \to \theta$ where $\theta = \overline{\alpha}, \overline{\alpha}\beta$ or $\alpha\overline{\beta}$.

$$P = \{\overline{X} \to \theta \mid S_{2n}(X) = (\_, \_, \overline{X} \to \theta),\ X \in V\}$$

This P corresponds to the longest paths trees of Bellman-Ford algorithm computing a collection of longest paths for any source-target pair in the graph. For example, the following trees show the solution of the longest path problem in the previous section, where target vertices correspond to terminals.



This means that we use the same set of rules P to compute any ϕ_X. The following P corresponds to the above solution where $\overline{á} = á$ and $\overline{à} = à$.

$$\overline{I} \to \overline{Z}_0 Z_1 \qquad \overline{Z}_0 \to \overline{á}b́ \qquad \overline{Z}_1 \to b̀\overline{à}$$

Finally, we construct a set of rules U. This is by induction on the depth k of the derivation. Let $U_0 = \{\}$, and at $k+1$-th step, choose and add exactly one rule $Y \to \alpha$ or $Y \to \alpha\beta$ in R to U_{k+1} such that $Y \notin dom(U_k)$, and $\alpha, \beta \in dom(U_k)$. Here $dom(U) = \Sigma \uplus \{Y \in V \mid Y \to \_ \in U\}$. We let $U = U_n$. By construction, $dom(U) = \Sigma \uplus V$, for each $Y \in V$ we have exactly one rule in U, and U induces no cycles.

The following SLP only has the size linear to the original grammar G.

Algorithm 1. *We obtain the SLP for ϕ_X as $(\Sigma, V \uplus \overline{V}, U \uplus P, \overline{X})$.*

Input: CFG (Σ, V, R, I).
Output: A mapping $S_k \in (\Sigma \uplus V) \to (\mathbb{N}\times\mathbb{N}\times(\{\overline{X} \to \theta \mid \theta = \overline{\alpha},\ \overline{\alpha}\beta,\ \alpha\overline{\beta}\}\uplus\{\bullet\})\uplus\{\bot\})$.

1 We first initialize $S_0(X)$ for $X \in V$ and $S_k(\sigma)$ for $\sigma \in \Sigma$ as follows.

$$S_0(X) = \bot,\ S_k(\acute{a}) = (0, -1, \bullet),\ S_k(\grave{a}) = (1, 1, \bullet)$$

2 We iteratively compute $S_{k+1}(X)$ for $X \in V$ as follows.

– For each of (i) $X \to \alpha$ or (ii) $X \to \alpha\beta$, such that $S_k(\alpha)$ and $S_k(\beta)$ (if (ii)) are not $\bot$. Assume that $S_k(\alpha), S_k(\beta)$ and $S_k(X)$ (if not $\bot$) are as follows.

$$S_k(\alpha) = (c_1, d_1, \_),\ S_k(\beta) = (c_2, d_2, \_),\ S_k(X) = (c_0, d_0, \_)$$

In case (ii) with $S_k(X) \neq \bot$, we first confirm that $d_0 = d_1 + d_2$. If this fails, the shape-balancedness has failed. We then compute $S_{k+1}(X)$ as follows. Let $d_3 = d_1 + d_2$, $c_3 = d_1 + c_2$, and $\overline{\overline{\sigma}} = \sigma$.

$$S_{k+1}(X) = \begin{cases} (c_1, d_1, \overline{X} \to \overline{\alpha}) & ((\text{i}),\ c_1 > c_0 \text{ if } S_k(X) \neq \bot) \\ (c_1, d_3, \overline{X} \to \overline{\alpha}\beta) & ((\text{ii}),\ c_1 \geq c_3, \text{ and } c_1 > c_0 \text{ if } S_k(X) \neq \bot) \\ (c_3, d_3, \overline{X} \to \alpha\overline{\beta}) & ((\text{ii}),\ c_1 < c_3, \text{ and } c_3 > c_0 \text{ if } S_k(X) \neq \bot) \\ S_k(X) & (\text{otherwise}) \end{cases}$$

Fig. 2. Combined algorithm for computing primary paths

2.3 Checking Color-Balancedness

The remaining step of the balancedness check is to confirm that each pair of coupled parentheses in strings is of the same color, i.e., of the same base letter in A. If so we call such strings color-balanced, or partially-balanced since they can be defined as substrings of balanced strings. A string is balanced iff it is both shape-balanced and color-balanced.

A color-balanced string ϕ is factorized as $\phi_{-i}\grave{a}_i\phi_{i-1}\cdots\grave{a}_1\phi_0\acute{b}_1\phi_1\cdots\acute{b}_j\phi_j$ such that $\phi_{-i}, \ldots, \phi_j$ are balanced. Even an arbitrary string can be similarly factorized by allowing $\phi_{-i}, \ldots, \phi_j$ to be shape-balanced. According to this factorization, we define

$$\rho(\phi) = \grave{a}_i \cdots \grave{a}_1 \acute{b}_1 \cdots \acute{b}_j$$

Next we define an ordering $\sqsubseteq$ on $\grave{A}^*\acute{A}^*$ as the minimal one satisfying

$$\phi\psi \sqsubseteq \phi\grave{a}\acute{a}\psi$$

for $\phi \in \grave{A}^*$ and $\psi \in \acute{A}^*$. We again extend this to a quasi-ordering $\phi \sqsubseteq \psi \Leftrightarrow \rho(\phi) \sqsubseteq \rho(\psi)$. Note that $\phi \sqsubseteq \psi$ implies $c(\phi) \leq c(\psi)$. Now, the remaining part of the algorithm is fairly simple.

Algorithm 2. *(Balancedness Check) Assume that we already computed ϕ_X with maximum $c(\phi_X)$ for each nonterminal of the given grammar $G = (\Sigma, V, R, I)$. We let $\phi_\sigma = \sigma$. For each $X \in V$, we check the following:*

- ϕ_X *is color-balanced.*
- $\phi_X \sqsupseteq \phi_\alpha$, *if* $X \to \alpha \in R$.
- $\phi_\alpha\phi_\beta$ *is color-balanced and* $\phi_X \sqsupseteq \phi_\alpha\phi_\beta$, *if* $X \to \alpha\beta \in R$.

It is easy to see that a grammar with a balanced language satisfies these conditions. This follows from the fact that under the balancedness, any ϕ_X with maximum $c(\phi_X)$ bounds all strings generated from X also according to $\sqsubseteq$. The other direction is shown by the following proposition; the success of Algorithm 2 implies the color-balancedness of the language of the grammar which, together with the shape-balanced check, proves the balancedness of the language.

Proposition 2. *If the check succeeds,* $X \stackrel{*}{\Rightarrow} \phi$ *implies (i)* ϕ *is color-balanced, and (ii)* $\phi_X \sqsupseteq \phi$.

The proof is by induction on the length of the derivation of ϕ. The base case is about terminal symbols and easy. For inductive step, assume that the proposition holds for strings obtained by derivation whose height is not greater than k. Now consider the derivation $X \stackrel{*}{\Rightarrow} \phi$ with its height $k+1$. If $X \to \alpha\beta$ is used, we have $\phi_1\phi_2 = \phi$ such that $\alpha \stackrel{*}{\Rightarrow} \phi_1$ and $\beta \stackrel{*}{\Rightarrow} \phi_2$. Note that it is safe to replace a substring ψ of a color-balanced string with another color-balanced string ψ' compatible to ψ. Formally, if (a) ψ' and $C[\psi]$ are color-balanced, and (b) $\psi \sqsupseteq \psi'$, we have (c) $C[\psi']$ color-balanced and (d) $C[\psi] \sqsupseteq C[\psi']$. Now we prove the case as follows. By assumption, (a) $\phi_\alpha\phi_\beta$, ϕ_1 and ϕ_2 are color-balanced, we also have (b) $\phi_\alpha \sqsupseteq \phi_1$ and $\phi_\beta \sqsupseteq \phi_2$. Hence (c) ϕ is color-balanced, and (d) $\phi_X \sqsupseteq \phi_\alpha\phi_\beta \sqsupseteq \phi_1\phi_\beta \sqsupseteq \phi_1\phi_2 = \phi$. The case $X \to \alpha$ is easy.

2.4 SLP and CS Equivalence

Finally, we need to confirm that for strings given as SLPs, both the color-balancedness and the check of $\phi \sqsubseteq \psi$ are decidable in PTIME. Fortunately, as noted in the introduction, we can use Plandowski's algorithm deciding the equivalence of two SLPs in PTIME.

First we give some definitions. Let ϕ^o be strings created from ϕ just by taking letters in $\acute{A}$ and removing $\acute{}$ sign. Let ϕ^c be strings created from ϕ similarly for $\grave{A}$ but in addition, by reversing the obtained string. For example, $(\grave{a}\grave{b}\acute{c})^o = c$ and $(\grave{a}\grave{b}\acute{c})^c = ba$. We use $\phi[j,k]$ to denote a substring of ϕ starting from its j-th letter and ending before the k-th letter.

Proposition 3. *Let* $c_j = |\rho(\phi_j)^c|$, $o_j = |\rho(\phi_j)^o|$, *and* $m = min(o_1, c_2)$.

(i) We have $\phi_1\phi_2$ *color-balanced iff* ϕ_1 *and* ϕ_2 *are color-balanced, and*

$$\rho(\phi_1)^o[o_1 - m, o_1] = \rho(\phi_2)^c[c_2 - m, c_2]$$

(ii) We have $\phi_1 \sqsubseteq \phi_2$ *iff* $s = c_2 - c_1 = o_2 - o_1 \geq 0$, *and*

$$\begin{aligned}\rho(\phi_1)^c &= \rho(\phi_2)^c[s, c_2]\\ \rho(\phi_1)^o &= \rho(\phi_2)^o[s, o_2]\\ \rho(\phi_2)^o[0, s] &= \rho(\phi_2)^c[0, s]\end{aligned}$$

Input: SLP (Σ, V, R, I)
Output: CS $(A, V^c \uplus V^o, R', I^c \text{ or } I^o)$

1 For a rule $X \to \alpha\beta \in R$, we let $o_1 = c(\phi_\alpha) - d(\phi_\alpha)$, $c_2 = c(\phi_\beta)$, and $m = min(c_2, o_1)$.

$$X^o \to \alpha^o[0, o_1 - m]\beta^o$$
$$X^c \to \beta^c[0, c_2 - m]\alpha^c$$

2 For a rule $X \to \alpha$, we just define

$$X^c \to \alpha^c, X^o \to \alpha^o$$

3 Finally for letters we define

$$\acute{a}^c = \grave{a}^o = \epsilon,\ \acute{a}^o = \grave{a}^c = a,$$

Fig. 3. Translation of SLP for ϕ_X into CS for $\rho(\phi_X)$

(iii) A composition system (CS) is an SLP which allows occurrences of nonterminals in the form $X[j,k]$ in the rhs of productions. Given an SLP generating ϕ, the translation in Fig. 3 gives CS such that $I^c \stackrel{*}{\Rightarrow} \rho(\phi)^c$ and $I^o \stackrel{*}{\Rightarrow} \rho(\phi)^o$.

It is known that the equivalence problem of two CSs also has a PTIME algorithm, since a CS can always be converted back into a polynomial-size SLP [Hag00, Sch06]. We use this algorithm with the property (ii) to determine $\phi_1 \sqsubseteq \phi_2$. We also use it with the property (i) to create a proof tree showing that each ϕ_X is (or is not) color-balanced by checking, for each production $Y \to \alpha\beta$ needed in constructing ϕ_X, that two CS, i.e., those using start symbols $\alpha^o[o_1 - m, o_1]$ and $\beta^c[c_2 - m, c_2]$, are equivalent.

Now the following theorem is immediate from Proposition 2 and 3.

Theorem 1. *The balancedness problem is in PTIME.*

3 2EXPTIME-Completeness of CFG-RHG Containment

We show that the CFG-RHG containment problem is 2EXPTIME-complete. In our previous work, we developed a decision algorithm for the problem which has doubly exponential time complexity [MT06]. Here we prove that this algorithm is actually optimal by showing that the problem is 2EXPTIME-hard.

As noted in the introduction, we actually show the 2EXPTIME-hardness of the CFG-PG containment problem. The result for the CFG-RHG containment is immediately obtained by regarding PGs as RHGs. For this, we distinguish a single pair of parenthesis [and]. We use an abbreviation $a = \acute{a}\grave{a}$ and use A to denote the set of strings in this form.

3.1 The Key Observation

Seidl [Sei90] showed that the containment between the languages of two nondeterministic tree automata (NTA) is EXPTIME-complete. In fact, NTA defines

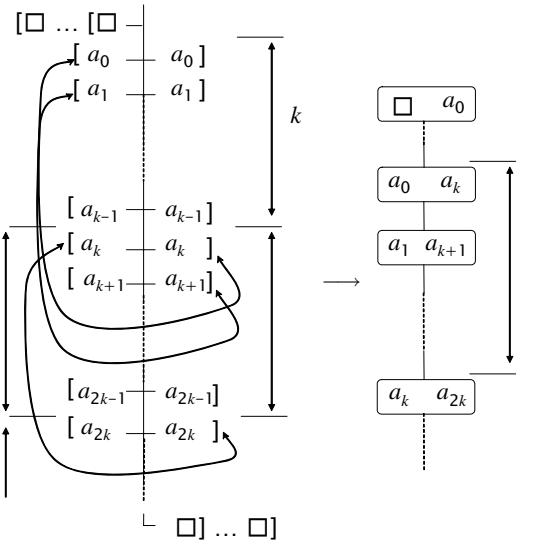


Fig. 4. A derivation tree and parse tree

the class of tree languages corresponding to languages of parse trees obtained from string languages defined by PG. If the problem is PG-PG or RHG-RHG containment, or even if the lhs of the containment is a balanced grammar such that $c(\phi_X) = d(\phi_X) = 0$ for all $X \in V$, the complexity relaxes to EXPTIME. In the case of NTA-containment, the size of the lhs is merely a polynomial factor. On the other hand, in the CFG-PG containment problem, the primary factor of the complexity is CFG on lhs. We here explain the high 2EXPTIME complexity from the gap between a derivation tree and parse tree of each string in the language of this CFG. Each node in a derivation tree corresponds to a nonterminal, while each node in a parse tree corresponds to matched parentheses. In the case of balanced grammars, two trees are not so different in the sense that each matching parentheses also exist closely, i.e., as siblings, in a derivation tree. This is generally not the case for a grammar just with a balanced language.

First note that for arbitrary $k \in \mathbb{N}$ and $\phi \in \Sigma^*$, we can construct a CFG (SLP) I^k_ϕ of size $O(\log k)$ that accepts ϕ^k. For this, we define $X_0 \to \phi$, $X_{i+1} \to X_i X_i$, and $I^k_\phi \to X_{i_1} \dots X_{i_n}$ where $i_1, .., i_n$-th bits are set in the binary encoding of k. Now let $\square \in A$ and define a grammar G as

$$I \to I^k_{[\square} X,\ X \to I^k_{\square]} \text{ and } X \to [aXa] \text{ for } a \in A$$

generating the following strings.

$$\overbrace{[\square \cdots [\square}^{k}[a_0[a_1 \cdots \overbrace{\square] \cdots \square]}^{k} \cdots a_1]a_0]$$

See also a derivation tree for this grammar described as the left tree of Fig. 4. In the figure, double-ended arrows between parentheses indicate matched

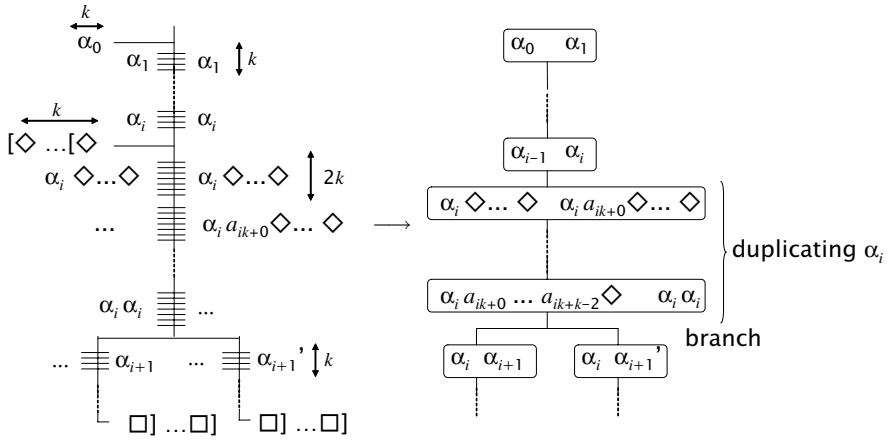


Fig. 5. Simulation of a branch in ATM

parentheses. We can see that each matched parentheses exist in k-distant positions in the derivation tree. Hence, a parse tree of the language of this grammar looks like the right tree of Fig. 4.

Now the procedure that decides $L(G) \subseteq L(G_{\mathrm{pg}})$ can be considered as a procedure which checks whether all parse trees above are contained in $L(G_{\mathrm{pg}})$ as trees. This PG compares the sequence $a_0 \cdots a_{k-1}$ with $a_k \cdots a_{2k-1}$, $a_k \cdots a_{2k-1}$ with $a_{2k} \cdots a_{3k-1}$, and so on. The first idea for the hardness proof is to use this fact to simulate a Turing machine (TM) using k-space. Let $\alpha_i = a_{ik} \cdots a_{(i+1)k-1}$ be a configuration of TM. It is possible to construct a PG to check if each transition from α_i to α_{i+1} is a valid single computation step of the given TM.

Although this idea indeed works as we will later formally discuss, we need one more trick to prove the 2EXPTIME-hardness. Note here that the size of G is $O(\log k)$. This means that the above TM can only solve EXPSPACE problems. To obtain the 2EXPTIME-hardness, we consider alternating Turing machines (ATM). An ATM is a Turing machine with conjunctive transitions in the form $q \vdash q_1 \wedge q_2$[2]. A computation of an ATM is thus a tree with branching degrees at most 2, and each 2-degree branch corresponds to this $\wedge$-transition. It is known that the class of 2EXPTIME is identical to the problems solvable by ATM using exponential space [CKS81].

The idea to simulate ATM is to add the following production rules to the previous grammar.

$$X \to I^k_{[\diamond} Y,\ Y \to XX \text{ and } Y \to [aYa] \text{ for } a \in A$$

A derivation tree of this grammar may look like the left tree of Fig. 5. We simplify the figure by omitting most of [and], and we also group each length-k

[2] In the literature, conjunctive transitions $q \vdash q_1 \wedge q_2$ are often expressed by using $\forall$-states, which correspond to branches in the computation tree, of possibly more than degree 2. It is easy to see that such an ATM using many-degreed branches can be easily converted into an equivalent ATM using at most 2-degree branches.

sequence of leaves $a_{ik}, \cdots, a_{(i+1)k-1}$ along the main branch as α_i. Now, we at some timing start to use Y instead of X in the main branch of the derivation, and this branch by Y eventually ends with two branches of derivations by X. Now the right tree of Fig. 5 explains how such derivation is parsed. In this figure, each node corresponding to the derivation by Y is illustrated as twice as wide as nodes for X. This reflects $d(\phi_Y) = 2k$ and $d(\phi_X) = k$. This also means that the PG can now simulate a machine with $2k$-space here. We would like to use this machine to duplicate the information of α_i before the branch. Such a duplication can be done by $O(k)$-computation steps, whose each step can be checked by constant size PG. After successfully creating $\alpha_i\alpha_i$, we can simulate alternating transitions α_i to α_{i+1}, and α_i to α'_{i+1}, by checking two child branches independently.

Using this idea, we will prove the following theorem in the next section.

Theorem 2. *The CFG-PG containment problem is 2EXPTIME-hard.*

3.2 Proof of 2EXPTIME-Hardness

Let P be a 2EXPTIME problem and M be an ATM solving this problem using exponential space. This ATM M is a tuple $(Q, \Gamma, \vdash, q_0, \Box, H)$ where

- Q is a set of states,
- Γ is a set of symbols,
- $\vdash \subseteq (\Gamma \times Q \times \{-1, 0, 1\} \times \Gamma \times Q) \uplus (Q \times \{q \wedge q' \mid q, q' \in Q\})$ is a transition relation,
- q_0 is an initial state,
- $\Box \in \Gamma$ is a blank symbol, and
- H is a set of accepting states.

A configuration of M is $\alpha = s_0 \cdots s_{i-1} s_i^q s_{i+1} \cdots \in \Gamma^*(\Gamma \times Q)\Gamma^\omega$ where s_i^q indicates that the current state is q and the head is at i-th position. The computation of M starts from $\alpha_0 = x_0^{q_0} x_1 \cdots x_{n-1} \Box \cdots$ where $x = x_0 \cdots x_{n-1}$ is an input. If $|x| = n$, the computation of M uses no more than $k = 2^{p(n)}$ space for some polynomial p. A computation history of M is a finite tree $T = (T, \alpha)$ associated with a function $\alpha \in T \to \Gamma^*(\Gamma \times Q)\Gamma^\omega$. This T is of branching degrees at most 2, so that the set of nodes T is given as a finite downward-closed subset of $\{1, 2\}^*$ under lexicographic ordering on $\{1, 2\}^*$. For each node $t \in T$, $\alpha(t)$ represents the configuration at t. Each degree 2 branch in T corresponds to alternating transitions $q \vdash q_1 \wedge q_2$. If $x \in P$, we can find T whose all leaves t satisfy $s_i^q \in \alpha(t)$ for some $q \in H$. Otherwise, any T has some leaves with non-accepting states.

Given an input $x = x_0 \cdots x_{n-1}$, we can construct the following CFG G in deterministic polynomial time and with its size polynomial to the input:

$$
\begin{aligned}
&I \to [\#[x_0^{q_0} \cdots [x_{n-1} I_{[\Box}^{k-n} X \\
&X \to [aXa] \quad \text{for } a \in A' \\
&X \to [\#I_{[\diamond}^{k} Y \\
&X \to \#]I_{\Box]}^{k} \\
&Y \to [aYa] \quad \text{for } a \in A \\
&Y \to XX
\end{aligned}
$$

where we let $k = 2^{p(n)}$, $Q^\bullet = \{\bullet\} \uplus Q$, $A' = \{\#\} \uplus \Gamma \times Q^\bullet = \{\#\} \uplus \Gamma \uplus \Gamma \times Q$, $A = \{\#\} \uplus (\Gamma \times Q^\bullet \uplus \{\diamond\}) \times \{!\}^\bullet$. We add the symbol $\#$ to recognize boundaries of each configuration.

We would like to create a PG G_{pg} or equivalently NTAs such that any parse tree in $L(G) \setminus L(G_{\mathrm{pg}})$ corresponds to an accepting computation history in the sense as explained in the last section. We model the parse tree as in Fig. 4 by a tree $U = (U, \lambda)$ associated with a labeling function $\lambda \in U \to A \times A$. Any $\phi \in L(G)$ can be parsed as such U. This U also has branching degrees at most 2. Any NTA running on U can be efficiently converted into an equivalent PG on ϕ.

First, we rule out ill-formed trees such that either $\#$ is not inserted appropriately, or 2-degree branches occur at inappropriate positions. For this, we use NTA N_1 which accepts any tree U with $u \in U$ such that either (i) $\lambda(u) = (\#, s), (s, \#)$ where $s \neq \#$, or (ii) $\lambda(u) \neq (\#, \#)$ and u is of branching degree 2.

Before simulating an alternation step of the ATM, we need to copy a configuration. By the production $X \to [\#I^k_{[\diamond} Y$, we prepare the copy and obtain the configuration below.

$$\# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond \; \diamond \quad \cdots \quad \diamond \; \#$$

At the first step, we place two markers denoted by ! at the two positions after $\#$, then repeatedly move two markers to the right and copy the contents.

$$\begin{array}{l}
\# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond \; \diamond \quad \cdots \quad \diamond \; \# \\
\# \; s_0^! \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond^! \; \diamond \quad \cdots \quad \diamond \; \# \\
\\
\# \; s_0^! \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond^! \; \diamond \quad \cdots \quad \diamond \; \# \\
\# \; s_0 \; s_1^! \cdots s_i^q \cdots s_{k-1} \; \# \; s_0 \; \diamond^! \quad \cdots \quad \diamond \; \# \\
\qquad\qquad\qquad\qquad \vdots \\
\# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1}^! \; \# \; s_0 \; s_1 \quad \cdots \quad \diamond^! \; \# \\
\# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; s_0 \; s_1 \quad \cdots \quad s_{k-1} \; \#
\end{array}$$

This process is checked by an NTA N_2, which checks each sequence of nodes $u_0, \ldots, u_n, \ldots, u_m \in U$ such that (i) each u_{i+1} is the successor of u_i, (ii) $\lambda(u_i) = (\#, \#)$ iff $i = 0, n, m$, (iii) $\lambda(u_i) = (\diamond, \_)$ or $(\diamond^!, \_)$ for some $n < i < m$. Note that any tree in $L(G) \setminus L(N_1)$ contains such a sequence only when $n = k+1$ and $m = 2k+2$. N_2 accepts a tree which contains an incorrect copying sequence, e.g., $\lambda(u_i) = (s^!, s)$, $\lambda(u_j) = (\diamond^!, s')$ and $s \neq s'$ for some i, j. This N_2 also accepts a tree if one of (i) $\lambda(u_{m-1}) = (\diamond^!, \_)$ and (ii) the node u_m is of branching degree 2, exclusively holds.

Finally, we construct an NTA N_3 for checking the transitions of M. This is similar to N_2 except that we check a sequence $u_0, \ldots u_n$ such that $\lambda(u_0) = \lambda(u_n) = (\#, \#)$ without occurrence of ! and $\#$ on nodes $u_1 \ldots u_{n-1}$. This N_3 accepts U whenever it finds a sequence $u_0, \ldots, u_n$ such that u_0 is of branching degree 1, but the sequence is either (i) not obeying $\vdash$ when u_n is also of branching-degree 1, or (ii) not containing accepting state, i.e., no u_i such that $\lambda(u_i) = (s^q, \square)$ for some $q \in H$, when u_n is a leaf. This N_3 also accepts U whenever it

finds $u_0 \in U$ of branching degree 2, followed by two sequences $u_0, u_1, \ldots, u_n$ and $u_0, u'_1, \ldots, u'_m$, which does not have i and j such that (i) $\lambda(u_{i'})$, $\lambda(u'_{j'})$ are in the form (s, s) if $i' \neq i$ and $j' \neq j$, and (ii) $\lambda(u_i) = (s^q, s^{q_1})$ and $\lambda(u'_j) = (s^q, s^{q_2})$ for some alternating transition $q \vdash q_1 \wedge q_2$.

We obtain the parenthesis grammar G_{pg} from an NTA accepting the union of $L(N_1)$, $L(N_2)$ and $L(N_3)$. The size of this G_{pg} is independent of x. Now for any x, we construct G in deterministic polynomial time from x so that we have $x \in P$ iff $\neg L(G) \subseteq L(G_{\mathrm{pg}})$. Hence the CFG-PG containment is 2EXPTIME-hard.

4 Related Work

In the same paper as the SLP equivalence [Pla94], Plandowski has also shown the existence of a polynomial-size *test set* for given CFG G. A test set $T \subseteq L(G)$ is such that given two morphisms $h, h' \in \Sigma \to M$ for a free group M, if $h(\phi) = h'(\phi)$ for all $\phi \in T$ then this holds for all $\phi \in L(G)$. He computed this T as a set of SLPs. Hence if we can efficiently decide whether or not $h(\phi) = \epsilon$ for all $\phi \in T$, by letting $h'(\sigma) = \epsilon$ (= unit of M) for all $\sigma \in \Sigma$, we obtain the PTIME algorithm to a problem similar to the balancedness problem which decides whether or not $h(L(G)) = \{\epsilon\}$. One difference here is that $aa^{-1} = a^{-1}a = \epsilon$ holds in a free group, while $\grave{a}\acute{a}$ is irreducible in the balancedness problem. We are not sure if we have another polynomial time algorithm for the balancedness problem in this direction.

Meyer and Stockmeyer showed that the equivalence problem for regular expressions extended with squaring is EXPSPACE-hard and further investigated the complexity of the problems for various variants of regular expressions [MS72, SM73]. For tree languages, Seidl showed that the equivalence of nondeterministic tree automata is EXPTIME-complete using an alternating Turing Machine as our discussion [Sei90]. The proofs of these completeness results are based on the hardness of the corresponding universality problems for languages of the rhs of the containment. On the other hand, in our proof of 2EXPTIME-completeness of CFG-RHG containment, a CFG in the lhs of the containment is essential.

References

[BB02a] Jean Berstel and Luc Boasson. Balanced grammars and their languages. In *Formal and Natural Computing: Essays Dedicated to Grzegorz*, volume 2300 of *LNCS*, pages 3–25, 2002.

[BB02b] Jean Berstel and Luc Boasson. Formal properties of XML grammars and languages. *Acta Informatica*, 38(9):649–671, 2002.

[Ber79] Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbucher, 1979.

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[Hag00] Christian Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, Universität Stuttgart, Fakultät Informatik, 2000.

[KM06] Christian Kirkegaard and Anders Møller. Static analysis for Java Servlets and JSP. In *Proc. 13th International Static Analysis Symposium, SAS '06*, volume 4134 of *LNCS*, August 2006.

[Knu67] Donald E. Knuth. A characterization of parenthesis languages. *Information and Control*, 11(3):269–289, 1967.

[Law76] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*, chapter 3. 1976.

[McN67] Robert McNaughton. Parenthesis grammars. *Journal of the Association for Computing Machinery*, 14(3):490–500, 1967.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Conf. Rec. 13th IEEE Symp. on Switching and Automata Theory*, pages 125–129, 1972.

[MT06] Yasuhiko Minamide and Akihiko Tozawa. XML validation for context-free grammars. In *Proc. of The Fourth ASIAN Symposium on Programming Languages and Systems*, volume 4279 of *LNCS*, pages 357–373, 2006.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *Algorithms – ESA '94 (Utrecht)*, volume 855 of *LNCS*, pages 460–470. Springer, 1994.

[Sch06] Saul Schleimer. Polynomial-time word problems, 2006. `http://front.math.ucdavis.edu/math.GR/0608563`.

[Sei90] Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

[Tak75] Masako Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 21(1):1–36, 1975.