PATTERN AMBIGUITIES FOR PURE CONTEXT–FREE GRAMMARS

Erkki Mäkinen and Ferucio Laurenţiu Ţiplea

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1996-6

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1996-6, AUGUST 1996

PATTERN AMBIGUITIES FOR PURE CONTEXT–FREE GRAMMARS

Erkki Mäkinen and Ferucio Laurenţiu Ţiplea

University of Tampere
Department of Computer Science
P.O.Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-4009-9
ISSN 0783-6910

Pattern Ambiguities for Pure Context–Free Grammars

Erkki MÄKINEN[a] **and Ferucio Laurenţiu ŢIPLEA**[b]

[a]Department of Computer Science
University of Tampere, P.O. Box 607
FIN-33101 Tampere, Finland
e-mail: em@cs.uta.fi

[b]Faculty of Informatics
"Al. I. Cuza" University of Iaşi
6600 Iaşi, Romania
e-mail: fltiplea@infoiasi.ro

Abstract

We consider here some special forms of ambiguity for pure context-free grammars, namely *pattern avoiding ambiguity*, *pattern preserving ambiguity*, *pattern ambiguity*, and *grammar avoiding ambiguity*. The first two properties are undecidable for arbitrary pure context-free grammars, and in a particular but general enough case we can effectively decide whether or not a pure context-free grammar is pattern preserving ambiguous.

1 Introduction and Preliminaries

The notion of ambiguity is one of the key concepts of formal language theory. For results concerning ambiguity in standard Chomsky type context-free grammars, the reader is referred e.g. to [2]. In this paper we study some special forms of ambiguity for pure context-free grammars. This study is motivated by an encryption method proposed in [5]. We start with some terminology and notation (for details concerning formal language theory the reader is referred to [2], [9]).

A pure context-free grammar (PCF grammar, for short) [6] is a 3-tuple $G = (V, V_0, P)$, where V is a finite alphabet, the set of *axioms* V_0 is a finite subset of V^+, and P is a finite set of productions of the form $a \longrightarrow v$, where $a \in V$ and $v \in V^*$. If $v = \lambda$ then the production $a \longrightarrow v$ is called a λ-*production*. The relation $\Longrightarrow$ (yields directly) and its reflexive and transitive closure $\stackrel{*}{\Longrightarrow}$ is defined as usual. G is said to be *production-complete* [5] if for each symbol $a \in V$ there is at least one production with a in its left hand side.

Let $G = (V, V_0, P)$ be a pure context-free grammar. We consider a finite set Lab of *labels* and a bijective function ρ from P into Lab. The couple $\gamma = (G, \rho)$ is called a *labelled PCF grammar* ($lPCF$ grammar, for short). If a production $u \longrightarrow v$ is associated with the label r by ρ, we will write $r : u \longrightarrow v$. The leftmost derivation of y from x by using the sequence of productions $\alpha = r_1 \dots r_n \in Lab^+$ is denoted by $x \overset{\alpha}{\Longrightarrow}_{\gamma,l} y$.

With a labelled grammar γ we associate a morphism $g : Lab^* \longrightarrow \{0,1\}^*$ with the property $| g(r) | \leq 1$ for all $r \in Lab$. A couple (γ, g) as above will be called an *arbitrarily labelled grammar*, ($alPCF$ grammar, for short). The terminology is justified by the fact that $g \circ \rho$ associates to each production $p \in P$ an element of the set $\{0, 1, \lambda\}$, and thus $g \circ \rho$ can be thought as an arbitrary (not necessarily one-to-one) labelling of G, permiting λ as a label. We have to remark that the labelling ρ helps us to easily handle the productions, whereas the labelling g is in fact the true labelling permiting different "actions" with the same "name" or without name (labelled by λ).

An example of an $alPCF$ grammar (γ, g) is presented in the next table. The last column lists the productions of the grammar, the second one describes the labeling ρ, whereas the first one gives the morphism g:

g	ρ	P
λ	r_1	$S \longrightarrow ab$
0	r_2	$a \longrightarrow ab$
0	r_3	$a \longrightarrow cb$
0	r_4	$c \longrightarrow cb$
0	r_5	$c \longrightarrow ab$
1	r_6	$b \longrightarrow aa$

A compact representation for such grammars is obtained by writing both labels in the front of each production. For example, the first two rules of the above grammar can be written as $\lambda : r_1 : S \longrightarrow ab$, $0 : r_2 : a \longrightarrow ab$.

In the next we shall consider only $lPCF$ grammars with an axiom denoted by S. For such a grammar γ define the *left Szilard language* of γ [7],[8], denoted $Szl(\gamma)$, by $Szl(\gamma) = \{\alpha \in Lab^+ | S \overset{\alpha}{\Longrightarrow}_{\gamma,l} y\}$; the strings $\alpha \in Szl(\gamma)$ are called *control words of* γ.

2 On An Encryption Method

In [1] an encryption method based on control words associated to leftmost derivations in context-free grammars has been proposed. We recall this method in terms of PCF grammars [5].

An $alPCF$ grammar (γ, g) defines an encryption method as follows. Assume the plaintext space being the phrases in English. Such a phrase u is encoded, by a one-to-one coding, into a string $x \in \{0,1\}^+$. From this string we compute a new string $\alpha \in Lab^+$ such that $g(\alpha) = x$ and α is a control word of γ (suppose such strings exist[1] and we arbitrarily choose one of them). A possible encryption of u is then the sentential form obtained in γ rewriting the axiom with the control word α.

[1] an $alPCF$ grammar (γ, g) satisfying the property $(\forall x \in \{0,1\}^+)(\exists \alpha \in Szl(\gamma))(g(\alpha) = x)$ is called $(0,1)$-*total* [1], [5], [4]

Concerning this encryption method many problems arise [1], [5], [4]. Let us suppose we are legal users of a cryptosystem like that described above:

$$\underbrace{u}_{plaintext} \stackrel{encode}{\longrightarrow} x \in \{0,1\}^+ \stackrel{g^{-1}}{\longrightarrow} \alpha \in Szl(\gamma) \stackrel{\gamma}{\longrightarrow} S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \underbrace{y}_{cryptotext}$$

(the strings $x \in \{0,1\}^+$ will be called *patterns*, and if $g(\alpha) = x$ we say that α *extends* x with respect to g).

Starting from x the sender has to look for a control word α extending x, and to rewrite the axiom S of γ with respect to α; the result will be the cryptotext. In general, there could exist more than one control word extending x, and therefore the sender has to choose one of them. Then we have to take into consideration the next two cases:

(S1) (easy encryption)
for any two distinct control words α and β extending the same pattern ($g(\alpha) = g(\beta)$) we have $S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S$[2]. That is, the cryptotext associated to x does not depend on the control word extending the pattern $g(\alpha) = g(\beta)$. Clearly, the sender may choose in the easiest way a control word and to encrypt with respect to it; the problem is that the cryptotext could be less safe;

(S2) (supplementary encryption key)
for any two distinct control words α and β extending the same pattern ($g(\alpha) = g(\beta)$) we have $\neg(S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S)$. That is, the cryptotext associated to x heavily depends on the control word extending the pattern $g(\alpha) = g(\beta)$. Such a property permits to the sender the use of an encryption key in order to choose a control word; this could be time consuming but in this way the cryptosystem is strenghtened.

¿From the receiver's point of view we have to take into considerations another two cases:

(R1) (easy decryption)
for any two distinct parsings α and β of the same cryptotext we have $g(\alpha) = g(\beta)$. Therefore, the receiver may parse the cryptotext in the easiest way, but the cryptosystem could be less safe;

(R2) (supplementary decryption key)
for any two distinct parsings α and β of the same cryptotext we have $g(\alpha) \neq g(\beta)$. Therefore, the receiver needs supplementary information in order to decrypt (a supplementary encryption/decryption key). In fact, this case is similar to the case (S2).

As we will see in the next section, these considerations lead to different forms of ambiguity for $alPCF$'s.

[2] "$S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S$" denotes "$\exists y :\ S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} y \ \wedge\ S \stackrel{\beta}{\Longrightarrow}_{\gamma,l} y$"; sometimes it is necessary to fix y, and then we write $S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} y \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S$

3 Pattern Ambiguities

An *lPCF* γ is *ambiguous* if $(\exists\alpha,\beta\in Szl(\gamma))(\alpha\neq\beta\ \wedge\ S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S)$. This notion of ambiguity may be translated to *alPCF* (γ,l) by means of γ. But, in this case, the ambiguity can be devided in two classes with respect to g:

1. $\exists\alpha,\beta\in Szl(\gamma)$ such that $\alpha\neq\beta$ and $g(\alpha)=g(\beta)$ and $S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S$. We will call this one *pattern preserving ambiguity* (pp-ambiguity or ppa, for short), because the computations α and β leading to ambiguity have in common the pattern $g(\alpha)=g(\beta)$;

2. $\exists\alpha,\beta\in Szl(\gamma)$ such that $\alpha\neq\beta$ and $g(\alpha)\neq g(\beta)$ and $S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S$. We will call this one *pattern avoiding ambiguity* (pa-ambiguity or paa, for short), because the computations α and β leading to ambiguity have no pattern in common.

It is clear that the negations of these two kinds of ambiguities correspond to the cases (S2) (or (R2)) and (R1), respectively.

The above two ambiguities are referring to the grammar but we may consider ambiguity with respect to the pattern as well. Therefore, we say that (γ,g) is *pattern ambiguous* if there exist $x\in\{0,1\}^+$ and $\alpha,\beta\in Szl(\gamma)$ such that $\alpha\neq\beta$ and $g(\alpha)=g(\beta)$. That is, there is an x which can be extended to two different control words. In other words, (γ,g) is pattern ambiguous if $(\exists\alpha,\beta\in Szl(\gamma))(\alpha\neq\beta\ \wedge\ g(\alpha)=g(\beta))$. This ambiguity can be divided in two kinds of ambiguities with respect two the grammar γ:

1. $\exists\alpha,\beta\in Szl(\gamma)$ such that $\alpha\neq\beta$ and $g(\alpha)=g(\beta)$ and $S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S$. In fact this one is the pattern preserving ambiguity;

2. $\exists\alpha,\beta\in Szl(\gamma)$ such that $\alpha\neq\beta$ and $g(\alpha)=g(\beta)$ and $\neg(S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S)$. This pattern ambiguity avoids grammar ambiguity and we will call it *grammar avoiding ambiguity* (ga-ambiguity or gaa, for short).

It is clear that the negation of ga-ambiguity corresponds to the case (S1). Let us conclude our discussion:

- **ga-ambiguity — easy encryption**

$$\begin{array}{ll}
\text{(gaa)} & (\exists\alpha,\beta\in Szl(\gamma))(\alpha\neq\beta\ \wedge\ g(\alpha)=g(\beta)\ \wedge\ \neg(S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S))\\
\text{(non-gaa)} & (\forall\alpha,\beta\in Szl(\gamma))(\alpha=\beta\ \vee\ g(\alpha)\neq g(\beta)\ \vee\ S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S),\ \text{or}\\
 & (\forall\alpha,\beta\in Szl(\gamma))((\alpha\neq\beta\ \wedge\ g(\alpha)=g(\beta))\ \Rightarrow\ S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S);
\end{array}$$

- **pa-ambiguity — easy decryption**

$$\begin{array}{ll}
\text{(paa)} & (\exists\alpha,\beta\in Szl(\gamma))(\alpha\neq\beta\ \wedge\ g(\alpha)\neq g(\beta)\ \wedge\ S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S)\\
\text{(non-paa)} & (\forall\alpha,\beta\in Szl(\gamma))(\alpha=\beta\ \vee\ g(\alpha)=g(\beta)\ \vee\ \neg(S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S)),\ \text{or}\\
 & (\forall\alpha,\beta\in Szl(\gamma))((\alpha\neq\beta\ \wedge\ S\overset{\alpha}{\Longrightarrow}_{\gamma,l}\cdot\overset{\beta}{\Longleftarrow}_{\gamma,l}S)\ \Rightarrow\ g(\alpha)=g(\beta));
\end{array}$$

- **pp-ambiguity — supplementary encryption/decryption key**

$$
\begin{array}{ll}
\text{(ppa)} & (\exists\alpha,\beta \in Szl(\gamma))(\alpha \neq \beta \ \wedge \ g(\alpha) = g(\beta) \ \wedge \ S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S) \\
\text{(non-ppa)} & (\forall\alpha,\beta \in Szl(\gamma))(\alpha = \beta \ \vee \ g(\alpha) \neq g(\beta) \ \vee \ \neg(S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S)), \text{ or} \\
 & (\forall\alpha,\beta \in Szl(\gamma))((\alpha \neq \beta \ \wedge \ g(\alpha) = g(\beta)) \ \Rightarrow \ \neg(S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S)), \text{ or} \\
 & (\forall\alpha,\beta \in Szl(\gamma))((\alpha \neq \beta \ \wedge \ S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S) \ \Rightarrow \ g(\alpha) \neq g(\beta)).
\end{array}
$$

Example 3.1 *(1) Let (γ, g_1) be the alPCF given by:*

$$0 : r_1 : S \longrightarrow ac,\ 0 : r_2 : S \longrightarrow bc,\ \lambda : r_3 : a \longrightarrow aa,$$
$$\lambda : r_4 : b \longrightarrow aa,\ \lambda : r_5 : c \longrightarrow cc.$$

It is easy to see that this grammar is both pp-ambiguous (take $\alpha = r_1r_3$ and $\beta = r_2r_4$) and ga-ambiguous (take $\alpha = r_1$ and $\beta = r_2$) but not pa-ambiguous.

(2) Let (γ, g_2) be the alPCF given by:

$$0 : r_1 : S \longrightarrow ac,\ 1 : r_2 : S \longrightarrow bc,\ \lambda : r_3 : a \longrightarrow aa,$$
$$\lambda : r_4 : b \longrightarrow aa,\ \lambda : r_5 : c \longrightarrow cc.$$

It is easy to see that this grammar is pa-ambiguous (take α and β as in (1)) but neither pp-ambiguous nor ga-ambiguous.

(3) Let (γ', g_3) be the alPCF given by:

$$0 : r_1 : S \longrightarrow ac,\ 0 : r_2 : S \longrightarrow bc.$$

It is easy to see that this grammar is ga-ambiguous (take $\alpha = r_1$ and $\beta = r_2$) but not pp-ambiguous. Consider the grammar (γ'', g_4) given by

$$0 : r_1 : S \longrightarrow ac,\ \lambda : r_2 : S \longrightarrow bc,\ 0 : r_3 : b \longrightarrow a.$$

This grammar is pp-ambiguous (take $\alpha = r_1$ and $\beta = r_2r_3$) but not ga-ambiguous.

Remark 3.1 *(1) It follows directly from the definitions that pp- or pa-ambiguity of (γ, g) implies the ambiguity of γ. The converse does not hold true as Example 3.1 shows us: the grammar γ is ambiguous but (γ, g_1) is not pa-ambiguous, and (γ, g_2) is not pp-ambiguous.*

(2) Similarly, pp- or ga-ambiguity implies pattern ambiguity, but the converse does not hold true: (γ, g_2) in Example 3.1 is pattern ambiguous (take $x = 0$, $\alpha = r_1$, and $\beta = r_1r_3$) but neither pp- nor ga-ambiguous.

(3) From Example 3.1 we can see that the ga-, pp- and pa-ambiguities are not implying each others.

We will show that the ambiguity, pp- and pa-ambiguity are in general undecidable.

Lemma 3.1 *It is undecidable whether an $lPCF$ grammar is ambiguous.*

Proof The proof is based on the recursive unsolvability of the Post Correspondence Problem and it follows the same line as for context-free grammars (see for example [3]). □

Lemma 3.2 *pp-ambiguity is undecidable for $alPCF$ grammars.*

Proof We will reduce the ambiguity problem for $lPCF$ grammars to the pp-ambiguity problem for $alPCF$ grammars.

Let γ be an $lPCF$ grammar. Consider (γ', g), where γ' is obtained from γ by adding a new symbol S' and a new production $0 : p : S' \longrightarrow S$; for the other productions r of γ', we will set $g(r) = \lambda$. We show that γ is ambiguous iff (γ', g) is pp-ambiguous.

If α and β are two distinct control words of γ such that $S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} y \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S$ then $\alpha' = p\alpha$ and $\beta' = p\beta$ are two distinct control words of (γ', g) with $g(\alpha') = 0 = g(\beta')$ and $S' \stackrel{\alpha'}{\Longrightarrow}_{\gamma',l} y \stackrel{\beta'}{\Longleftarrow}_{\gamma',l} S'$. Therefore, the ambiguity of γ leads to the pp-ambiguity of (γ', g).

The converse can be easily obtained observing that any non-empty control words of (γ', g) is of the form $p\alpha$, where α is a control word of γ. □

Before showing that the pa-ambiguity is undecidable we consider the concept of *2-branched grammar.*

Definition 3.1 *An $lPCF$ grammar γ is said to be* 2-branched *if, for any symbol a in γ, there exist at most two productions with a in the left hand side.*

Lemma 3.3 *pa-ambiguity is undecidable for $alPCF$ grammars.*

Proof We will reduce the ambiguity problem for $lPCF$ grammars to the pa-ambiguity problem for $alPCF$ grammars.

Let γ be an $lPCF$ grammar. Transform first γ into a new 2-branched $lPCF$ grammar γ' as follows: assume $a \longrightarrow u_1, \ldots, a \longrightarrow u_k$ are all the productions with a in left hand side and $k > 2$. Consider the new symbols $a_1, \ldots, a_{k-2}$ and replace the above productions by the new productions $a \longrightarrow u_1, a \longrightarrow a_1, a_1 \longrightarrow u_2, \ldots, a_{k-3} \longrightarrow u_{k-2}, a_{k-3} \longrightarrow a_{k-2}, a_{k-2} \longrightarrow u_{k-1}, a_{k-2} \longrightarrow u_k$. By iterating this procedure we finally get a 2-branched $lPCF$ γ'.

Claim γ is ambiguous iff γ' is ambiguous.

Proof of Claim Assume first γ is ambiguous. That is, there exist two distinct control words α and β such that $S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} y$ and $S \stackrel{\beta}{\Longrightarrow}_{\gamma,l} y$ for some y. If these control words use rules of the form $a \longrightarrow u_i$ with $i > 1$ (as above) then they will be replaced by the sequence of rules $a \longrightarrow a_1, a_1 \longrightarrow a_2, \ldots, a_{i-1} \longrightarrow u_i$, for $i < k$, or by the sequence $a \longrightarrow a_1, a_1 \longrightarrow a_2, \ldots, a_{i-2} \longrightarrow u_k$, for $i = k$. Finally, we will get two distinct control words of γ', α' and β', such that $S \overset{\alpha'}{\Longrightarrow}_{\gamma',l} y$ and $S \overset{\beta'}{\Longrightarrow}_{\gamma',l} y$. Hence, γ' is ambiguous.

Conversely, let us assume that γ' is ambiguous, and let α and β be two distinct control words producing the same word y from the axiom S. First, we may assume that both α and β do not end by rules $x \longrightarrow z$, where z is a new symbol. Indeed, if we suppose $\alpha = \alpha' r_1$, $\beta = \beta' r_2$, $r_1 : x_1 \longrightarrow z_1$, $r_2 : x_2 \longrightarrow z_2$, and z_1 and z_2 are new symbols, then we may write

$$S \overset{\alpha'}{\Longrightarrow}_{\gamma',l} y_1 = y_1^1 x_1 y_1^2 \Longrightarrow_{\gamma',l} y_1^1 z_1 y_1^2 = y$$
$$S \overset{\beta'}{\Longrightarrow}_{\gamma',l} y_2 = y_2^1 x_2 y_2^2 \Longrightarrow_{\gamma',l} y_2^1 z_2 y_2^2 = y.$$

Both x_1 and x_2 are the leftmost symbols that are rewritten and therefore $y_1^1 = y_2^1$ and $x_1 = x_2$. Since both derivations produce the same string, y, it follows that $z_1 = z_2$, $y_2^1 = y_2^2$, and so $r_1 = r_2$ and $\alpha' \neq \beta'$. Since z_1 is a new symbol, there must exist a rule having z_1 in its left hand side. As a result, we can add a new rule with $z_1 = z_2$ in the left hand side, to the end of α and β. Continuing this process as long as it is necessary we finally get two new control words with the desired property.

If a is the leftmost symbol in a string $w = w_1 a w_2$ produced by γ', and the rule $a \longrightarrow a'$ is used, where a' is a new symbol, then the leftmost symbol to be rewritten in $w_1 a' w_2$ is a'. Consequently, if α and β contain rules with new symbols in their right hand sides then these rules come in sequences of the form:

$$a \longrightarrow a_1, a_1 \longrightarrow a_2, \ldots, a_i \longrightarrow u_j,$$

where $i = j - 1$ or $i = j - 2$, $a_1, \ldots, a_i$ are new symbols, and $a \longrightarrow u_j$ is a rule in γ. This fact permits us to "pack" such sequences of rules into one rule, namely $a \longrightarrow u_j$. Finaly we will get two control words of γ, α' and β', such that $S \overset{\alpha'}{\Longrightarrow}_{\gamma,l} y$ and $S \overset{\beta'}{\Longrightarrow}_{\gamma,l} y$.

The proof of the Claim is complete if we can show that $\alpha' \neq \beta'$. We know that $\alpha = \beta$. So, we can write $\alpha = \theta r_1 \alpha_1$ and $\beta = \theta r_2 \beta_1$, where $r_1 \neq r_2$. As r_1 and r_2 rewrite the leftmost symbol (θ being a common prefix of α and β) it follows that r_1 and r_2 have the same left hand side, say x. The above remarks lead to the fact that α and β contain two subsequences $w_1 = \theta_2 r_1 \alpha_1^1$ and $w_2 = \theta_4 r_2 \beta_1^1$ such that:

- $\theta = \theta_1\theta_2 = \theta_3\theta_4$ and $\alpha_1 = \alpha_1^1\alpha_1^2$ and $\beta_1 = \beta_1^1\beta_1^2$, for some $\theta_1, \theta_3, \alpha_1^2, \beta_1^2$;
- these two sequences have the form:

  $$a \longrightarrow a_1, \ldots, r_1, \ldots, a_i \longrightarrow u_j,$$

  and respectively

  $$a \longrightarrow a_1, \ldots, r_2, \ldots, a_s \longrightarrow u_t,$$

  where $a \longrightarrow u_j$ and $a \longrightarrow u_t$ are rules in γ, and $u_j \neq u_t$ (r_1 or r_2 may be the first rules of these sequences).

The "packing operation" (described above) applied to α and β will replace these two subsequences by the rules $a \longrightarrow u_j$ and $a \longrightarrow u_t$, respectively. Consequently, α' and β' will be distinct control words.

The Claim is proved.

Consider now the *alPCF* (γ', g), where γ' is obtained from γ as described above and g is given by the following setting: for any a, if there is at most one production with a in the left hand side then label it by 0; otherwise (there are at most two productions with a in the left hand side) label one of them by 0 and the other one by 1. We show that γ' is ambiguous iff (γ', g) is pa-ambiguous.

If α and β are two distinct control words of γ' such that $S \overset{\alpha}{\Longrightarrow}_{\gamma',l} y \overset{\beta}{\Longleftarrow}_{\gamma',l} S$ then $g(\alpha) \neq g(\beta)$ because there must exist a branching symbol for these control words (note that the derivation is leftmost). Therefore, the ambiguity of γ' leads to the pa-ambiguity of (γ', g).

The converse follows from definitions. $\Box$

4 A Sufficient Criterion for the Decidability of pp-ambiguity

As mentioned in the previous sections the non-ppa property would be useful both for sender and receiver in order to strenghten the cryptosystem. Unfortunately, we have seen that this property is in general undecidable. In what follows we shall derive a sufficient criterion for the decidability of pp-ambiguity. The PCF grammars that we will consider have no λ-productions or renamings (that is, productions of the form $a \longrightarrow b$ with $a, b \in V$), and they are production-complete.

Definition 4.1 *Let $\gamma = (G, \rho)$ be an lPCF grammar, $au \in V^+$ ($a \in V$), $b \in V$ and $\alpha \in Lab^+$. We say that α* minimally covers *au with respect to b if one of the next two properties is satisfied:*

1. *if $u = \lambda$ then $\alpha \in Lab$ and $b \overset{\alpha}{\Longrightarrow}_{\gamma,l} cv$, for some $c \in V$ and $v \in V^*$;*
2. *if $u \neq \lambda$ then $\alpha = r_1 \cdots r_n$ ($n \geq 1$), $b \overset{r_1 \cdots r_{n-1}}{\Longrightarrow}_{\gamma,l} cv_1 \overset{r_n}{\Longrightarrow}_{\gamma,l} dv_2$, u is a suffix of v_2 but not of v_1 (for $n = 1$ we have $cv_1 = b\lambda$).*

Hence, we may informally say that α minimally covers au with respect to b if α is "a minimal sequence" such that the string yielded from b by α "covers" the string au, excepting the first letter.

Definition 4.2 *Let γ be an lPCF grammar and $\alpha, \beta \in Szl(\gamma)$. We say that α and β are* compatible control words *if α minimally covers bv with respect to S or β minimally covers au with respect to S, where $S \overset{\alpha}{\Longrightarrow}_{\gamma,l} au$ and $S \overset{\beta}{\Longrightarrow}_{\gamma,l} bv$.*

For an alphabet V denote by $\leq_{pref}$ the *prefix (partial) order* on V, that is $u \leq_{pref} v$ iff u is a prefix of v. Let u and v words over V. We say that u and v are *comparable* (with respect to the prefix order) if $u \leq_{pref} v$ or $v \leq_{pref} u$. If u and v are comparable we say that the pair (u', v') is obtained from (u, v) by *left maximal cancellation* if u' and v' are respectively obtained from u and v by deleting the common maximal prefix of them.

Let $T \subseteq V$. By $\#(T, \alpha)$ we denote the number of occurrences of letters $a \in T$ in α. Then the *balance* of α and β over T is defined by:

$$B_T(\alpha, \beta) = |\#(T, \alpha) - \#(T, \beta)|.$$

Definition 4.3 *An $alPCF$ grammar (γ, g) is* balance k-bounded, *$k \geq 1$, if the next property is satisfied: for all $\alpha, \beta \in Szl(\gamma)$ such that $g(\alpha) = g(\beta)$ and $S \stackrel{\alpha}{\Longrightarrow}_{\gamma,l} \cdot \stackrel{\beta}{\Longleftarrow}_{\gamma,l} S$, and for all $\alpha' \leq_{pref} \alpha$ and $\beta' \leq_{pref} \beta$ such that α' and β' are compatible, it follows that $B_{\{0,1\}}(\alpha', \beta') \leq k$.*

Theorem 4.1 *It is effectively decidable whether or not a given balance k-bounded $alPCF$ grammar (γ, g) is pattern preserving ambiguous.*

Proof Let (γ, g) be a balance k-bounded $alPCF$ grammar. We shall generate a graph $\mathcal{G} = (X, E)$ whose nodes are of the form

$$(au, b, x_1, x_2) \text{ or } (a, bu, x_1, x_2)$$

where

- $a, b \in V$;
- $u \in V^*$ such that $|u| \leq max\{|w| | \exists a \longrightarrow w \in P\}$;
- $x_1, x_2 \in \{0, 1\}^*$.

The arcs of $\mathcal{G}$ will be labelled by couples (α, β) where $\alpha, \beta \in Lab^*$ and either $\alpha = \lambda$ or $\beta = \lambda$. The next procedure generates this graph:

1. generate the node (S, S, λ, λ). This node is called the *initial node* of $\mathcal{G}$ and it is marked as an *open node*;
2. while there are open nodes left choose one of them, say δ, close it and perform one of the next three steps (if both the steps 2.1 and 2.2 can be performed with respect to δ then choose 2.1):

   2.1 if $\delta = (au, b, x_1, x_2)$ then

   - for any minimal covering sequence $\beta \in Lab^+$ of au with respect to b,

   $$b \stackrel{\beta}{\Longrightarrow}_{\gamma,l} cvu,$$

   generate (provided that it does not yet exist) the node (a, cv, y_1, y_2), where y_1, y_2 are obtained from $x_1, x_2 g(\beta)$ by left maximal cancellation. If $y_1 \neq \lambda$ and $y_2 \neq \lambda$, or $B_{\{0,1\}}(y_1, y_2) > k$ then mark the node (a, cv, y_1, y_2) as a *final* node; otherwise, mark it as an *opened* node;

- the arc $(\delta, (a, cv, y_1, y_2))$ will be labelled by (λ, β);

2.2 if $\delta = (a, bv, x_1, x_2)$ then

- for any minimal covering sequence $\alpha \in Lab^+$ of bv with respect to a,

$$b \overset{\alpha}{\Longrightarrow}_{\gamma,l} cuv,$$

generate (provided that it does not yet exist) the node (cu, b, y_1, y_2), where y_1, y_2 are obtained from $x_1 g(\alpha), x_2$ by left maximal cancellation. If $y_1 \neq \lambda$ and $y_2 \neq \lambda$, or $B_{\{0,1\}}(y_1, y_2) > k$ then mark the node (cu, b, y_1, y_2) as a *final* node; otherwise, mark it as an *opened* node;

- the arc $(\delta, (cu, b, y_1, y_2))$ will be labelled by (α, λ);

2.3 if no node can be generated from δ as described in 2.1 or 2.2 then mark δ as a *final* node;

The graph $\mathcal{G}$ is finite and can be effectively constructed. Indeed, as the grammar is production-complete and it has no λ-productions or renamings, it follows that there are finitely many minimal covering sequences of a string au with respect to b, and these sequences can be effectively computed. Moreover, the use of such sequences lead to the fact that for any node (u, v, x_1, x_2) the next two properties hold:

(1) $|u|, |v| \leq max\{|w| | \exists a \longrightarrow w \in P\}$;

(2) $|x_1|, |x_2| \leq k + max\{|w| | \exists a \longrightarrow w \in P\}$
(each rule of the grammar add at least one new symbol, and therefore every minimal covering sequence of a string au' with respect to b has the length at most $|u'|$. Now, the statement follows from (1)).

Let $\delta_1, \ldots, \delta_n = (u, v, x_1, x_2)$ $(n \geq 2)$ be a path in $\mathcal{G}$ such that δ_1 is the initial node. Consider the sequence of arc labels determined by this path,

$$(\alpha_1, \beta_1) \cdots (\alpha_{n-1}, \beta_{n-1}),$$

where $\beta_i = \lambda$ for i odd and $\alpha_j = \lambda$ for j even, and let $\alpha = \alpha_1 \cdots \alpha_{n-1}, \beta = \beta_1 \cdots \beta_{n-1}$. The next properties hold true:

(3) α and β are compatible control words;

(4) $B_{\{0,1\}}(\alpha, \beta) = B_{\{0,1\}}(x_1, x_2)$;

(5) if δ_n is a *final* node then there is no $\alpha', \beta' \in Szl(\gamma)$ such that $\alpha \leq_{pref} \alpha'$, $\beta \leq_{pref} \beta'$, $g(\alpha') = g(\beta')$, $S \overset{\alpha'}{\Longrightarrow}_{\gamma,l} \cdot \overset{\beta'}{\Longleftarrow}_{\gamma,l} S$. Indeed, if this is not the case then:

a) if $x_1 \neq \lambda$ and $x_2 \neq \lambda$ then $g(\alpha') \neq g(\beta')$; a contradiction;

b) if $B_{\{0,1\}}(x_1, x_2) > k$ then (γ, g) would not have the balance k-bounded; a contradiction;

(6) if $u = v \in V$ and $x_1 = x_2 = \lambda$ and $\alpha \neq \beta$ then (γ, g) is pattern preserving ambiguous; the converse holds also true (that is, if (γ, g) is pp-ambiguous then there is a node δ satisfying (6)).

Finally, to decide whether (γ, g) is pattern preserving ambiguous or not we have to generate the graph $\mathcal{G}$ and to verify whether (4) holds true or not. The theorem is proved. $\square$

One can easily notice that the graph in the proof of the above theorem is a version of a domino graph introduced in [10], [11]. By a rough approximation we can say the complexity of our algorithm is exponential because the graph has an exponential number of nodes (it is enough to count the sequences of 0 and 1's in the last two components of the nodes).

References

[1] M. Andraşiu, A. Atanasiu, Gh. Păun, A. Salomaa. A New Cryptosystem Based on Formal Language Theory, *Bull. Math. Soc. Sci. Math. Roumanie*, Tome 36(84), nr.1, 1992, pp. 1-15.

[2] M.A. Harrison. *Introduction to Formal Language Theory*, Addison-Wesley 1978.

[3] J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[4] E. Mäkinen. $(0,1)$-Totality is Undecidable for Arbitrary Context-Free Grammars,*Fundamenta Informaticae* (to appear).

[5] C. Matei, F.L. Ţiplea. $(0,1)$- Total Pure Context-Free Grammars, in *Proc. of 2nd Conference on Development in Language Theory*, Magdeburg, 1995.

[6] H.A. Maurer, A. Salomaa, D. Wood. Pure Grammars, *Inform. and Control* 44 (1), 1980, pp. 47-72.

[7] E. Moriya. Associate Languages and Derivational Complexity of Formal Grammars and Languages, *Inform. and Control* 22, 1973, pp. 139-162.

[8] M. Penttonen. On Derivation Languages Corresponding to Context-free Grammars,*Acta Inform.* 3, 1973, pp. 285-291.

[9] A. Salomaa. *Formal Languages*, Academic Press, New York, London, 1973.

[10] A. Weber, T. Head. The Finest Homophonic Partition and Related Code Concepts, in *Proc. of the 19th Symposium on Mathematical Foundations of Computer Science*, Kosice, Slovakia, 1994.

[11] A. Weber. Computing Deciphering and Synchronization Delays of a Code by means of Dominoes, J.W. Goethe-Universität, Frankfurt am Main, 1995.