Invisible Pushdown Languages

Eryk Kopczyński
University of Warsaw
erykk@mimuw.edu.pl

Abstract

Context-free languages allow one to express data with hierarchical structure, at the cost of losing some of the useful properties of languages recognized by finite automata on words. However, it is possible to restore some of these properties by making the structure of the tree visible, such as is done by visibly pushdown languages, or finite automata on trees. In this paper, we show that the structure given by such approaches remains invisible when it is read by a finite automaton (on word). In particular, we show that separability with a regular language is undecidable for visibly pushdown languages, just as it is undecidable for general context-free languages.

Categories and Subject Descriptors F.1.1 [*Automata*]; F.4.3 [*Classes defined by grammars or automata*]

Keywords regular languages of trees, visibly pushdown automata, separability, XML

1. Introduction

Finite automata are a well known formalism for describing the simplest formal languages. Regular languages – ones which are recognized by finite automata – have very nice closure properties, such as decidability of most problems such as universality or disjointness, equivalence of deterministic finite automata (DFA) and non-deterministic finite automata (NFA), and closure under complement.

However, most programming and natural languages have to describe a hierarchical (tree) structure, and finite automata on words are no longer appropriate. To capture such a hierarchical structure, Noam Chomsky proposed the classic notion of *context-free languages*. Context-free languages are recognized by context-free grammars (CFGs), or equivalently by pushdown automata (PDA).

However, context-free languages do not have as good properties as regular ones – for example, universality and disjointness are no longer decidable, deterministic PDA are less powerful than non-deterministic ones, and they are not closed under complement. These properties fail since, although words from a context-free language have an underlying tree structure, it is hard to tell what this structure is just by looking at the word – two completely different derivation trees can yield a very similar output, consider for example the English sentences *Time flies like an arrow* and *fruit flies like a banana*, or *The complex houses married and single soldiers and their families* – after reading the four first words of the latter sentence, one could think that *the complex houses* is the subject and *married* is the verb, while in fact, *the complex* is the subject and *houses* is the verb. This is also a big problem in practical computer science, since such a possibility of incorrect parsing leads to many errors – one famous example is the SQL injection attack, which is based on fabricating SQL queries which will be parsed incorrectly, allowing unauthorized access to a database.

There are two popular approaches to solve this. The classic approach is to use the *finite automata on trees* (TFAs) [5], which work on trees directly. There are several ways of flattening a tree to a string in such a way that the tree structure can be unambiguously read from the output. One common way is the *XML encoding*; an XML encoding of a regular language of trees can be viewed as a context-free grammar where every terminal $a \in \Sigma$ has a matching closing tag $\bar{a}$, and every production is of form $N \to aX_1 \dots X_k\bar{a}$. Nowadays, XML [4] is widely used in web services and databases for the representation of data structures. XML documents are validated with DTDs (document type definitions) [9], which can be viewed as a special kind of finite automata working on the tree of the document. Another common way is the *functional encoding*, which can be viewed as a CFG where every production is of form $N \to a(X_1 \dots X_k)$, where (and) are special bracket symbols.

Another approach is to use *visibly pushdown automata* (VPDAs), also known as languages of *nested words* [2], where every symbol in our alphabet has a fixed type with respect to the stack – it either always pushes a new symbol, or always pops a symbol, or it never pushes or pops symbols – this property allows the tree structure to be easily read. For example, XML and functional grammars given above can be recognized by VPDAs – opening tags and brackets always push a symbol on the stack, while closing tags and brackets always pop.

Since both of these approaches boil down to working on trees directly in a regular way, they can be seen as equivalent, and most properties of regular languages of words are retained – non-deterministic and deterministic VPDAs and finite automata on trees are equivalent, and universality and intersection problems are decidable. Hence, representing our data as trees, instead of forcing a linear word structure, definitely solves many problems – both theoretical and practical – efficiently.

In this paper, we show that not all problems are solved by these approaches. In particular, we show that, informally, although (flattened) TFAs and VPDAs are successful at making the structure visible to powerful computation models such as Turing machines, the structure still remains invisible to the simple ones, such as finite automata on words. We use our technique to show that the following problem is undecidable, just as in the usual "invisible" context-free case [8, 12]: given two VPDAs (or TFAs flattened in some way) accepting languages L_1 and L_2 such that L_1 and L_2 are disjoint, is there a regular language R such that R accepts all words from L_1, but no words from L_2?

Our proof is a reduction from the context-free case; essentially, it is shown that the extra structural information can be made in-

LICS '16, July 05 - 08, 2016, New York, NY, USA

ACM 978-1-4503-4391-6/16/07?$15.00
DOI: http://dx.doi.org/10.1145/2933575.2933579

visible to all finite automata. We show that this can be done even when, intuitively, the finite automaton has access to any structural information that it could reasonably see while traversing the tree – therefore, our method works not only with VPDAs, but also with any "reasonable" string encoding of trees, such as XML or functional encoding.

A similar property is also obtained for separating by other classes of languages, as long as the corresponding problem for CFGs is undecidable, and the separating class has basic closure properties and a pumping property – the precise conditions are listed in the sequel. In [8] it is shown that the separability problem of context-free languages is undecidable for any class which includes all *definite* languages. On the other hand, it has been shown recently that the problem of separability of CFLs by *piecewise testable languages* is decidable [6].

Our method solves the following open problem, which has appeared on Rajeev Alur's website in early 2013 [1]:

> A Challenging Open Problem
>
> Consider the following decision problem: given two regular languages L_1 and L_2 of nested words, does there exist a regular language R of words over the tagged alphabet such that Intersection(R,L_1) equals L_2? [...]

We say that L_2 is a *regular restriction* of L_1 iff the above holds. Since disjoint languages L_1 and L_2 are separable iff L_2 is a regular restriction of $L_1 \cup L_2$, and separability is undecidable, restriction-regularity is undecidable too.

Rajeev Alur's question is inspired by [11], where it is shown that, for a *fully recursive* DTD d, it is decidable whether the language accepted by d is *(regularly) recognizable*, that is, there is a regular language R such that, for any properly structured word w (in this case, a proper XML document), w is valid with respect to d iff it is accepted by R. This can be viewed as a special case of separability – we take DTDs instead of arbitrary finite automata on trees, and we want to separate L from its complement. For a fully recursive DTD d, a *local automaton* (also called *standard automaton*) A_d is constructed, and it is shown that d is streamable iff it is validated by A_d. More progress is made in [10], where an attempt is made to generalize the result to DTDs which are not fully recursive, by introducing the notion of a *separating semigroup* H for the DTD d, and allowing the local automaton to acknowledge H. The existence of such a separating semigroup can be reduced to the word problem for finite groups; however, this problem is known to be undecidable.

On the other hand, in [3] it is shown that regular recognizability is decidable for visibly push-down languages (an input has to recognize a given language knowing that the input is a well matched word, that is, symbols that push and pop symbols on the stack match correctly). The method used in [3] works for VPDAs and the functional encoding of trees, but it does not extend to the XML encoding, where every opening tag a has to be matched by a matching closing tag $\bar{a}$, potentially providing extra information; and thus, the problem of recognizability of XML in full generality remains open, as far as we know (and, although the problem of regularity of a context-free language is also undecidable, the reduction presented in this paper does not immediately work in this case either). In the conclusion of [3], the authors mentioned that they were working on the regular restriction problem for VPDAs, but as far as we know, they were not successful.

2. Preliminaries

We remind the basic notions of automata theory; see [7].

For an alphabet Σ, Σ^* denotes the set of words over Σ, and ϵ denotes the empty word.

A *deterministic finite automaton* (DFA) over Σ is a tuple $A = (\Sigma, Q, q_I, F, \delta)$, where Σ is the alphabet of A, Q is the set of states, $q_I \in Q$ is the initial state, $F \subseteq Q$ is the final state, and $\delta : Q \times \Sigma \to Q$ is the transition function. We extend δ to $\delta : Q \times \Sigma^* \to Q$ in the following way: $\delta(q, \epsilon) = q$, $\delta(q, wx) = \delta(\delta(q, w), x)$.

A *context-free grammar* (CFG) is a tuple $G = (V, \Sigma, R, S)$, where V is the set of non-terminal symbols, Σ is the set of terminal symbols, R is a set of *productions* of form $N \to X_1 \dots X_k$ where N is a non-terminal symbol and and each X_i is either a terminal or non-terminal symbol, and $S \in V$ is the start symbol. The language accepted by G, $L(G) \subseteq \Sigma^*$, is the set of words which can be obtained from the start symbol S by replacing non-terminal symbols with words, according to the productions.

A *flattened tree grammar* (FTG) over Σ is a context-free grammar, whose set of terminal symbols is $\Sigma' := \Sigma \cup \{\diamondsuit, \overline{\diamondsuit}\}$, and every production is of form $N \to t$ or $N \to \diamondsuit N_1 N_2 \dots N_k \overline{\diamondsuit}$, where $N_1, N_2, \dots, N_k$ are non-terminals, and t is a terminal. A *binary flattened tree grammar* (BFG) is a flattened tree grammar which uses only $k \in \{0, 2\}$. A BFG corresponds to the XML encoding of a regular language of binary trees ($\diamondsuit$ and $\overline{\diamondsuit}$ correspond to the opening `<a>` and closing `</a>` tag, respectively), and languages recognized by flattened tree grammars are visibly pushdown languages. These two facts are routine to check – we omit this to avoid having to state the definitions of VPDAs and TFAs; we have decided to use flattened tree grammars in this paper since they are easier to define than both of these formalisms.

DEFINITION 2.1. *We say that two languages L_1 and L_2 are* separable *if there is a regular language R such that for each $w \in L_1$, $w \in R$, but for each $w \in L_2$, $w \notin R$.*

PROBLEM 2.2 (CFG-SEPARABILITY).
INPUT *Two context-free grammars G_1 and G_2 such that $L(G_1)$ and $L(G_2)$ are disjoint*
OUTPUT *Are $L(G_1)$ and $L(G_2)$ separable?*

The CFL separation problem is known to be undecidable [8, 12]. For convenience, we include the idea of the proof here. Encode configurations of a deterministic Turing machine M as words, and say that $w_1 \to w_2$ iff a machine in configuration w_1 reaches the configuration w_2 in next step. It can be easily shown that (for simple encodings) the languages $\{w_1 \# w_2^R : w_1 \to w_2\}$ and $\{w_1^R \# w_2 : w_1 \to w_2\}$ are context-free, and thus the languages L_1 and L_2 below are also context-free.

$$\begin{aligned} L_1 &= \{w_1 \# w_2 \# \dots w_{2k} \# a^{2k} : w_1 = w_I, w_{2i-1} \to w_{2i}^R\} \\ L_2 &= \{w_1 \# w_2 \# \dots w_{2k} \# a^{k} : w_1 = w_I, w_{2i}^R \to w_{2i+1}\} \end{aligned}$$

The languages L_1 and L_2 are separable iff M terminates from the initial configuration w_I. Indeed, if M terminates after n steps, then we can recognize whether $w \in L_1$ or $w \in L_2$ by reading the configurations w_i given by w, until we find one which does not match the run of M – if i is even, then we know that $w \notin L_1$, and if i is odd, then we know that $w \notin L_2$. If the sequence ends after $l \leq n$ configurations, count the number of a's, and say $w \notin L_1$ iff there are not exactly $2k$ of them, and $w \notin L_2$ if there are not exactly k of them. Since the run is fixed, this can be done with a finite automaton (which accepts the word if it proves that $w \notin L_2$, and rejects if it proves that $w \notin L_1$).

On the other hand, if M does not terminate, consider two words w, w' from L_1 and L_2 respectively where the number of configurations given is $2k$, and they correctly describe the run of M, up to $2k$ steps. The finite automaton A now has to tell whether the number of a's is k or $2k$. Since for every finite automaton A there is a number $\omega \in \mathbb{N}$ such that A cannot tell a^ω from $a^{2\omega}$ in any context (Lemma 4.3 below), A cannot separate the words w, w' for $k = \omega$.

3. Overview of the main result

PROBLEM 3.1 (BFG-SEPARABILITY, FTG-SEPARABILITY).

INPUT *Two BFGs (FTGs)* G_1 *and* G_2 *such that* $L(G_1)$ *and* $L(G_2)$ *are disjoint*
OUTPUT *Are* $L(G_1)$ *and* $L(G_2)$ *separable?*

Our main result is the following:

THEOREM 3.2. *The problems* **BFG-SEPARABILITY** *and* **FTG-SEPARABILITY** *are undecidable.*

We will prove Theorem 3.2 in both cases by reducing **CFG-SEPARABILITY**. We will transform a CFG G into a FTG (BFG) G' by adding extra productions and structural symbols $\diamondsuit$, $\overline{\diamondsuit}$. This transformation won't change the structure of the word: if we take the language $L(G')$ and remove all the structural symbols, we still have the language $L(G)$.

Therefore, if $L(G_1)$ and $L(G_2)$ are separable, then so are $L(G'_1)$ and $L(G'_2)$ – a regular language separating $L(G_1)$ and $L(G_2)$ can also separate $L(G'_1)$ and $L(G'_2)$ simply by ignoring the structural symbols (Lemma 4.1). We still have to show that if $L(G'_1)$ and $L(G'_2)$ are separable by some R, then $L(G_1)$ and $L(G_2)$ are also separable; in other words, a regular language cannot use the extra structural information given by the structural symbols.

This is done the following way: take $w' \in L(G'_i)$, and the corresponding $w \in L(G_i)$. Using the fact (Lemma 4.3) that for every regular language R there is an ω such that R cannot tell v^ω from $v^{2\omega}$ in any context (they are *syntactically equivalent*), we construct the word w'' which is also in $L(G'_i)$, and is syntactically equivalent to $T(w)$, where $T(w)$ is w padded with specific sequences of structural symbols. This way, we make all the structural information contained in w' invisible for R. If R separates $L(G'_1)$ and $L(G'_2)$, we could also separate $L(G_1)$ from $L(G_2)$ in the following way: take $w \in L(G_i)$, obtain $T(w)$ by padding; $w \in L(G_1)$ iff $T(w) \in R$.

Since BFGs are FTGs, undecidability of separability of FTGs follows from the undecidability of separability of BFGs. However, in the next section, we show the undecidability of **FTG-SEPARABILITY** separately. This is because the proof is considerably easier in this case. The two proofs differ in how the CFG G is transformed to G', and how to construct the padding T, and how to transform w' into w'' syntactically equivalent to $T(w)$. Note that, since the languages recognized by FTGs are visibly pushdown, undecidability of separability of visibly pushdown languages already follows from undecidability of **FTG-SEPARABILITY**.

4. Separability of FTGs

In this section, we show that separability is undecidable for FTGs.

We will reduce the problem **CFG-SEPARABILITY** to **FTG-SEPARABILITY**. To do this, we will take two CFGs G_1 and G_2, and create two FTGs G'_1 and G'_2 such that G'_1 and G'_2 are separable iff G_1 and G_2 are. Without loss of generality, we can assume that grammars G_i do not accept the empty word, or any word of length 1.

Given a context-free grammar $G = (V, \Sigma, P, S)$, we will construct a flattened tree grammar $G' = (V', \Sigma', P', S')$, in the following way:

- For each $X \in V$, we have a non-terminal X'. We also have one special non-terminal E'. The starting symbol of G' is S'.
- For each production $N \to t$ in P, we have the corresponding production in P':
$$N' \to t \tag{1}$$
- For each production $N \to N_1 \dots N_k$ in P, we have the corresponding bracketed production in P':
$$N' \to \diamondsuit E' N'_1 E' N'_2 E' \dots N_k E' \overline{\diamondsuit} \tag{2}$$
- Where the productions for E' are as follows:
$$E' \to \diamondsuit \overline{\diamondsuit} \tag{3}$$
$$E' \to \diamondsuit E' \overline{\diamondsuit} \tag{4}$$
$$E' \to \diamondsuit E' E' \overline{\diamondsuit} \tag{5}$$
- For each non-terminal N, we also have the following production in P':
$$N' \to \diamondsuit N' \overline{\diamondsuit} \tag{6}$$

Consider $\pi : \Sigma' \to \Sigma$, the homomorphism which simply removes the structural symbols $\diamondsuit$ and $\overline{\diamondsuit}$. By applying π to all the production rules for G', we obtain a grammar $\pi(G')$ which accepts exactly $\pi(L(G'))$. It is straightforward to check that $\pi(G')$ is in fact equivalent to G – the only difference is that E' is inserted in some places, but all words generated by E' reduce to the empty word after applying π. Therefore, $\pi(L(G'_i))$ equals $L(G_i)$, which makes the following straightforward:

LEMMA 4.1. *If* $L(G_1)$ *and* $L(G_2)$ *are separable, then so are* $L(G'_1)$ *and* $L(G'_2)$.

Proof $\pi^{-1}(R)$ is a regular language which separates $L(G'_1)$ and $L(G'_2)$ – in other words, the automaton separating these two languages works exactly as the one separating $L(G_1)$ and $L(G_2)$ (it just ignores all the closing and structural symbols). ■

The rest of this section will prove the other direction:

THEOREM 4.2. *If* $L(G'_1)$ *and* $L(G'_2)$ *are separable, then so are* $L(G_1)$ *and* $L(G_2)$.

Assume that $L(G'_1)$ and $L(G'_2)$ are separable. Therefore, there is a finite automaton A such that $R = L(A)$ accepts all words from $L(G'_1)$, but no words from $L(G'_2)$. We say that two words $w_1, w_2 \in \Sigma'^*$ are *syntactically equivalent* with respect to R iff for any words $v, x \in \Sigma'^*$, we have $vw_1x \in R$ iff $vw_2x \in R$. Syntactic equivalence is a congruence with respect to concatenation.

LEMMA 4.3. *There is a number* $\omega \in \mathbb{N}$ *such that for any* $w \in \Sigma'^*$, w^ω *is syntactically equivalent to* $w^{2\omega}$ *with respect to* R.

Proof The set S of all the equivalence classes is a semigroup with concatenation as the operation. This semigroup is called the *syntactic semigroup* of A, and it is finite – if for two words w_1 and w_2 we have $\delta(q, w_1) = \delta(q, w_2)$ for each $q \in Q$, then they are syntactically equivalent. For any finite semigroup $(S, \cdot)$, there is a number $\omega \in \mathbb{N}$ such that for any $s \in S$, we have $s^{2\omega} = s^\omega$ – since $k \mapsto s^k$ yields an ultimately periodic sequence with period at most $|S|$, $\omega = |S|!$ will work. ■

We say that $T : \Sigma^* \to \Sigma'^*$ is a *padding* iff there exist $e_L, e, e_R \in (\Sigma' - \Sigma)^*$ such that, for any $w = t_1 \dots t_n \in \Sigma^*$, $T(w)$ is the word $e_L t_1 e t_2 e \dots e t_n e_R$.

LEMMA 4.4. *For a padding* T, *the languages* $L(G_1)$ *and* $L(G_2)$ *are separable, iff* $T(L(G_1))$ *and* $T(L(G_2))$ *are.*

Proof The forward direction is straightforward, and proven just as Lemma 4.1 – the automaton simply ignores all the symbols from $\Sigma' - \Sigma$ (in fact, the stronger version of this lemma where $e_L, e, e_R \in \Sigma'^*$ is also true).

For the backward direction, we take the DFA A' which separates $T(L(G_1))$ and $T(L(G_2))$: $A' = (\Sigma', Q, q_I, F, \delta)$. We can assume that there are no transitions to the initial state q_I in A' – otherwise, we create a copy of q_I and make it the new initial state.

We construct a new DFA $A'' = (\Sigma, Q, q_I, F', \delta')$ in the following way: take $\delta'(q_I, t) = \delta(q_I, e_L t)$, and $\delta'(q, t) = \delta(q, et)$ for $q \neq q_I$. For F' we take the set of states q such that $\delta(q, e_R) \in F$. The automaton A'' working on $w \in \Sigma^*$ simulates the automaton A' working on $T(w)$, hence it accepts w iff A' accepts $T(w)$. ■

LEMMA 4.5. *There is a padding T with the following property: for each $w \in L(G_i)$, there is a word $w' \in L(G'_i)$ which is equivalent to $T(w)$ with respect to R.*

This proves Theorem 4.2 and thus Theorem 3.2. Indeed, we will show that $T(L(G_1))$ and $T(L(G_2))$ are separated by R – then, after applying Lemma 4.4, we get our claim.

Consider $w \in L(G_i)$; we have to show that A accepts $T(w)$ iff $i = 1$. From Lemma 4.5 we know that there is some $w' \in L(G'_i)$ which is equivalent to $T(w)$ with respect to R. Therefore, we know that $T(w) \in R$ iff $w' \in R$, and since $w' \in L(G'_i)$, $T(w) \in R$ iff $i = 1$. ■

Proof of Lemma 4.5 in the FTG case

We will show that the padding T given by $e_L = e = e_R = (\diamondsuit^\omega \overline{\diamondsuit}^\omega)^\omega$ satisfies our claim.

Let $w \in L(G_i)$. We know that $w = \pi(w')$ for some $w' \in L(G'_i)$. We have $w' = e_0 w_1 e_1 w_2 \dots e_{k-1} w_k e_k$, where each $e_i \in \{\diamondsuit, \overline{\diamondsuit}\}^*$, for $i = 0, \dots, k$.

We can assume the following facts about e_i:

- Each e_i starts with $\diamondsuit$ and ends with $\overline{\diamondsuit}$. This follows from our productions P' (the production 2 puts E' before and after each non-terminal, and E' must start with $\diamondsuit$ and end with $\overline{\diamondsuit}$, according to productions 3, 4, 5).
- Let $\chi(e_i)$ be the number of times $\diamondsuit\overline{\diamondsuit}$ appears in e_i. Without loss of generality, we can assume that $\chi(e_i)$ is divisible by ω for each i. Indeed, suppose that $e_i = u\diamondsuit\overline{\diamondsuit}v$. Since the only production in P' which could produce $\diamondsuit\overline{\diamondsuit}$ is the production $E' \to \diamondsuit\overline{\diamondsuit}$ 3, we could replace the application of that production with $E' \to \diamondsuit E' E' \overline{\diamondsuit} \to \diamondsuit\diamondsuit\overline{\diamondsuit} E' \overline{\diamondsuit} \to \diamondsuit\diamondsuit\overline{\diamondsuit}\diamondsuit\overline{\diamondsuit}\overline{\diamondsuit}$ (productions 4, 5), increasing $\chi(e_i)$ by 1. Repeat until $\chi(e_i)$ is divisible by ω.
- For each production $N' \to \diamondsuit N'_1 \dots N'_k \overline{\diamondsuit}$ in P' (where N' or any N'_i could be E'), we also have a production $N' \to \diamondsuit N' \overline{\diamondsuit}$ (productions 4, 6). Therefore, every time we introduce the structural symbols $\diamondsuit$ and $\overline{\diamondsuit}$, we can introduce as many of them as we want (as long as they match). Without loss of generality, we can assume that each structural symbol is multiplied ω times. Thus, $e_i \in (\diamondsuit^\omega + \overline{\diamondsuit}^\omega)^*$.

From these three properties, e_i has to be syntactically equivalent with respect to R to $e = e_L = e_R$. Indeed, since syntactic equivalence is a congruence, and w^ω is equivalent to $w^{2\omega}$ for any w, $e_i \in (\diamondsuit^\omega + \overline{\diamondsuit}^\omega)^*$ has to be syntactically equivalent to $(\diamondsuit^\omega \overline{\diamondsuit}^\omega)^{\chi(e_i)}$, and since $\chi(e_i)$ is divisible by ω, it has to be syntactically equivalent to $(\diamondsuit^\omega \overline{\diamondsuit}^\omega)^\omega$.

Hence, the word w' is equivalent to $T(w)$. ■

5. Separability of BFGs

While the proof above is sufficient to solve the problem for visibly pushdown languages, and for XML encoding of trees, it leaves us with a craving for more, for the following reasons.

First, if we consider languages of terms, it is natural to consider different symbols for nodes with different arities (numbers of children) – while the proof above heavily uses the fact that the XML encoding cannot tell whether $\diamondsuit$ comes from the structurally significant production $N \to \diamondsuit E' X_1 E' X_2 \dots X_k E' \overline{\diamondsuit}$, or it is a fake inserted by the $X \to \diamondsuit X \overline{\diamondsuit}$ or $E' \to \diamondsuit E' E' \overline{\diamondsuit}$ productions. Since the arities here are respectively $2k+1$, 1 and 2, this fails if the automaton sees them.

Moreover, if we consider the process of flattening a tree as the result of an automaton which traverses the tree recursively and react to what it is seeing on its path, it is natural to assume that such an automaton sees the arity, and moreover, between returning from the i-th child of v and progressing to the $(i+1)$-th child, the automaton should see that i of n children are progressed. This corresponds to flattening productions of form $N \to \diamondsuit_0^k X_1 \diamondsuit_1^k X_2 \diamondsuit_2^k X_3 \dots X_k \diamondsuit_k^k$.

Restricting ourselves to the binary case allows us to solve the problem in full generality – while the encoding in the definition of BFG does not explicitely say whether $\diamondsuit$ comes from the "empty leaf" production $E' \to \diamondsuit\overline{\diamondsuit}$ or from one of the binary branching rules, a finite automaton can easily tell which one is the case by looking at the neighborhood. Also, while we do not write the infix structural symbols $\diamondsuit_1^2$ explicitely, the automaton can tell that it is at such a branching point iff the last symbol was $\overline{\diamondsuit}$, and the next one is $\diamondsuit$.

The rest of this section shows that separability is undecidable even when restricted to BFGs.

Proof of Theorem 3.2 in the BFG case

The general structure of the proof is the same as of the proof of the FTG case, given in Section 4.

We have to adjust the productions P' so that we obtain a BFG. As in the proof of the FTG case, we can assume that grammars G_i do not accept the empty word, or any word of length 1. We can also assume that these grammars are in the *Chomsky normal form*, that is, each production is of form $N \to N_1 N_2$ or $N \to t$, where N, N_1 and N_2 are non-terminals, and t is a terminal. It is well known that any context-free grammar is effectively equivalent to a grammar in Chomsky normal form [7].

Given a context-free grammar $G = (V, \Sigma, P, S)$ in Chomsky normal form, we will construct a binary flattened tree grammar $G' = (V', \Sigma', P', S')$, in the following way:

- For each $X \in V$, we have a non-terminal X'. We also have one special non-terminal E'. The starting symbol of G' is S'.
- For each production $N \to t$ in P, we have the corresponding production in P':
$$N' \to t \tag{7}$$
- For each production $N \to N_1 N_2$ in P, we have the corresponding bracketed production in P':
$$N' \to \diamondsuit N'_1 N'_2 \overline{\diamondsuit} \tag{8}$$
- For each $X \in V$, we also have the following productions for X':
$$X' \to \diamondsuit E' X' \overline{\diamondsuit} \tag{9}$$
$$X' \to \diamondsuit X' E' \overline{\diamondsuit} \tag{10}$$
- Where the productions for E' are as follows:
$$E' \to \diamondsuit\overline{\diamondsuit} \tag{11}$$
$$E' \to \diamondsuit E' E' \overline{\diamondsuit} \tag{12}$$

Lemmas 4.1, 4.3, and 4.4 are proven exactly in the same way as in the FTG case. It is sufficient to prove the counterpart of Lemma 4.5 to prove Theorem 4.2, and thus Theorem 3.2.

Proof of Lemma 4.5 in the BFG case

Let $\heartsuit = \diamondsuit\diamondsuit\overline{\diamondsuit}$, $\overline{\heartsuit} = \diamondsuit\overline{\diamondsuit}\overline{\diamondsuit}$. Thus, for any non-terminal $N' \in V'$, by applying one of productions (9, 10 or 12) and then (11), we have $N' \to^* \heartsuit N'\overline{\diamondsuit}$ and $N' \to^* \diamondsuit N'\overline{\heartsuit}$. Also, let $\nu = \omega - 1$.

By applying the above many times to non-terminals K' and L', we get $K' \to^* b_1 K' b_2$ and $L' \to^* c_1 L' c_2$, where:

$$b_1 = (\diamondsuit^\nu \heartsuit^\nu)^\omega \diamondsuit^\nu \quad (13)$$
$$b_2 = \overline{\heartsuit}^\nu (\overline{\diamondsuit}^\nu \overline{\heartsuit}^\nu)^\omega \quad (14)$$
$$c_1 = \heartsuit^\nu (\heartsuit^\nu \diamondsuit^\nu)^\omega \quad (15)$$
$$c_2 = (\overline{\heartsuit}^\nu \overline{\diamondsuit}^\nu)^\omega \overline{\diamondsuit}^\nu \quad (16)$$

Now, whenever we have a production $N \to KL$ in P, we can do the following in P':

$$N' \to \diamondsuit K' L' \overline{\diamondsuit} \to^* \diamondsuit b_1 K' b_2 c_1 L' c_2 \overline{\diamondsuit} \quad (17)$$

This can be written as $N' \to e_L K' e L' e_R$, where:

$$e_L = \diamondsuit b_1 \quad (18)$$
$$e = b_2 c_1 \quad (19)$$
$$e_R = c_2 \overline{\diamondsuit} \quad (20)$$

We claim that the padding T given by the words e_L, e, e_R defined above satisfies our claim.

Indeed, take $w \in L(G_i)$. Consider the derivation tree of w in G_i; repeat this derivation in G_i', replacing each production $N \to KL$ with $N' \to e_L K' e L' e_R$ according to the chain of productions (17) above, and each production $N \to t$ with $N' \to t$ (7). In the end, for $w = t_1 \dots t_n$, we obtain the word $w' \in L(G_i')$, which contains the symbols $t_1, \dots, t_n$ separated with e, possibly accompanied by e_L's on the right side and e_R's on the left side, and with at least one e_L before t_1 and at least one e_R after t_N. In other words,

$$w' = e_L^{l_1} t_1 e_R^{r_1} e e_L^{l_2} t_2 e_R^{r_2} e e_L^{l_3} \dots t_n e_R^{r_N}$$

where all l_i, r_i are integers, and $l_1, r_n \geq 1$. Let $\equiv$ be the relation of syntactic equivalence with respect to R; note that $w^\omega \equiv w^{2\omega}$ for any $w \in \Sigma'^*$, and that $\equiv$ is a congruence with respect to concatenation. Remembering that $\diamondsuit$ is a left factor of $\heartsuit$ and thus $\diamondsuit^{\omega+\nu}\heartsuit \equiv \diamondsuit^\nu \heartsuit$, and similarly $\overline{\heartsuit}\overline{\diamondsuit}^{\omega+\nu} \equiv \overline{\heartsuit}\overline{\diamondsuit}^\nu$, it can be checked that the following equivalences (*) hold:

- $e_L e_L$ is equivalent to e_L:

$$\begin{aligned} & e_L e_L = \diamondsuit b_1 \diamondsuit b_1 = \\ = \; & \diamondsuit(\diamondsuit^\nu\heartsuit^\nu)^\omega\diamondsuit^\nu\diamondsuit(\diamondsuit^\nu\heartsuit^\nu)^\omega\diamondsuit^\nu \equiv \\ \equiv \; & \diamondsuit(\diamondsuit^\nu\heartsuit^\nu)^\omega\diamondsuit^\omega(\diamondsuit^\nu\heartsuit^\nu)^\omega\diamondsuit^\nu \equiv \\ \equiv \; & \diamondsuit(\diamondsuit^\nu\heartsuit^\nu)^\omega(\diamondsuit^\nu\heartsuit^\nu)^\omega\diamondsuit^\nu \equiv \diamondsuit(\diamondsuit^\nu\heartsuit^\nu)^\omega\diamondsuit^\nu = \diamondsuit b_1 = e_L \end{aligned}$$

- $e_R e_R$ is equivalent to e_R:

$$\begin{aligned} & e_R e_R = c_2\overline{\diamondsuit}c_2\overline{\diamondsuit} = \\ = \; & (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\diamondsuit}^\nu\overline{\diamondsuit}(\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\diamondsuit}^\nu\overline{\diamondsuit} \equiv \\ \equiv \; & (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\diamondsuit}^\omega(\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\diamondsuit}^\nu\overline{\diamondsuit} \equiv \\ \equiv \; & (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega(\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\diamondsuit}^\nu\overline{\diamondsuit} \equiv (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\diamondsuit}^\nu\overline{\diamondsuit} = c_2\overline{\diamondsuit} = e_R \end{aligned}$$

- $e e_L$ is equivalent to e:

$$\begin{aligned} & e e_L = b_2 c_1 \diamondsuit b_1 = \\ = \; & b_2\heartsuit^\nu(\heartsuit^\nu\diamondsuit^\nu)^\omega\diamondsuit(\diamondsuit^\nu\heartsuit^\nu)^\omega\diamondsuit^\nu \equiv \\ \equiv \; & b_2\heartsuit^\nu(\heartsuit^\nu\diamondsuit^\nu)^\omega\diamondsuit\diamondsuit^\nu(\heartsuit^\nu\diamondsuit^\nu)^\omega \equiv \\ \equiv \; & b_2\heartsuit^\nu(\heartsuit^\nu\diamondsuit^\nu)^\omega(\heartsuit^\nu\diamondsuit^\nu)^\omega \equiv b_2\heartsuit^\nu(\heartsuit^\nu\diamondsuit^\nu)^\omega = b_2 c_1 = e \end{aligned}$$

- $e_R e$ is equivalent to e:

$$\begin{aligned} & e_R e = c_2\overline{\diamondsuit}b_2 c_1 = \\ = \; & (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\diamondsuit}^\nu\overline{\diamondsuit}\overline{\heartsuit}^\nu(\overline{\diamondsuit}^\nu\overline{\heartsuit}^\nu)^\omega c_1 \equiv \\ \equiv \; & (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\heartsuit}^\nu(\overline{\diamondsuit}^\nu\overline{\heartsuit}^\nu)^\omega c_1 \equiv \\ \equiv \; & (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega(\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\heartsuit}^\nu c_1 \equiv (\overline{\heartsuit}^\nu\overline{\diamondsuit}^\nu)^\omega\overline{\heartsuit}^\nu c_1 = b_2 c_1 = e \end{aligned}$$

These rules allow us to reduce all l_i and r_i to zeros (except l_1 and r_n, which can be reduced to 1). Hence, the word w' is equivalent to $T(w)$. ■

Remark. The appropriate words e, e_L, e_R which work in Lemma 4.5 in the BFG case have been found with a computer program. The approach which yielded the answer was to assume a fixed value of ω, consider $uv^\omega w$ to be equivalent to $uv^{2\omega}w$, and exhaustively search for the words e_L, e, e_R such that $e_L K' e L' e_R$ can be derived from N', and the required equivalences (*) hold. By looking at the words obtained for $\omega = 2, 3$, one could guess the general formula. The previous approach was to use a computer program to search for a solution where the words e_L, e, e_R are equal, just as it works in the non-binary case; however, no such solution was found in the binary case (even when considering a fixed finite syntactic semigroup instead of just a fixed ω, which ensured that the program had a finite amount of cases to check, and thus find an answer if one existed); however, this previous approach has shown that the weaker, but still sufficient, equivalences (*) should be possible to obtain.

6. Conclusion

We can also consider the separation problem for other classes of languages – that is, is it decidable whether languages accepted by two BFGs are separable by a language of class $\mathcal{C}$? From the proof above, this problem is undecidable for class $\mathcal{C}$, as long as the following conditions are satisfied:

- The respective problem for CFGs is undecidable.
- The class $\mathcal{C}$ has the following pumping property: for any $L \in \mathcal{C}$, there exists some ω such that for any words v, w, x, $vw^\omega x \in L$ iff $vw^{2\omega}x \in L$. (This is Lemma 4.3, and it is used in the proof of Lemma 4.5.)
- If π is a homomorphism which ignores a subset of symbols, and $L \in \mathcal{C}$, then $\pi^{-1}(L) \in \mathcal{C}$. (Used in the proofs of Lemma 4.1 and 4.4.)
- For a padding $T(t_1 \dots t_k) = e_L t_1 e t_2 e \dots e t_k e_R$, $T(L_1)$ and $T(L_2)$ are separable iff L_1 and L_2 are. (This is Lemma 4.4.)

Hence, the separability problem is also undecidable for other classes of languages, such as the class of languages recognizable by first-order logic.

Acknowledgments

Thanks to Charles Paperman for introducing me to the separability problem for VPDAs, and for helping me with the references.

This work is supported by Poland's National Science Centre grant 2013/11/D/ST6/03075.

References

[1] R. Alur. homepage. `https://web.archive.org/web/20130103185520/http://www.cis.upenn.edu/~alur/nw.html`. A snapshot from 2013-01-03.

[2] R. Alur and P. Madhusudan. Visibly pushdown languages. In L. Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211.

ACM, 2004. ISBN 1-58113-852-0. doi: 10.1145/1007352.1007390. URL http://doi.acm.org/10.1145/1007352.1007390.

[3] V. Bárány, C. Löding, and O. Serre. Regularity problems for visibly pushdown languages. In B. Durand and W. Thomas, editors, *STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-32301-3. doi: 10.1007/11672142_34. URL http://dx.doi.org/10.1007/11672142_34.

[4] T. Bray, M. Sperberg-McQueen, J. Paoli, F. Yergeau, and E. Maler. Extensible markup language (XML) 1.0 (third edition). W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-xml-20040204.

[5] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.

[6] W. Czerwinski, W. Martens, L. van Rooijen, and M. Zeitoun. A note on decidable separability by piecewise testable languages. In A. Kosowski and I. Walukiewicz, editors, *Fundamentals of Computation Theory - 20th International Symposium, FCT 2015, Gdańsk, Poland, August 17-19, 2015, Proceedings*, volume 9210 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2015. ISBN 978-3-319-22176-2. doi: 10.1007/978-3-319-22177-9_14. URL http://dx.doi.org/10.1007/978-3-319-22177-9_14.

[7] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321455363.

[8] H. B. Hunt, III. On the decidability of grammar problems. *J. ACM*, 29(2):429–447, Apr. 1982. ISSN 0004-5411. doi: 10.1145/322307.322317. URL http://doi.acm.org/10.1145/322307.322317.

[9] E. Maler, T. Bray, F. Yergeau, J. Cowan, M. Sperberg-McQueen, and J. Paoli. Extensible markup language (XML) 1.1. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-xml11-20040204/.

[10] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against dtds. In T. Schwentick and D. Suciu, editors, *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings*, volume 4353 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2007. ISBN 3-540-69269-X. doi: 10.1007/11965893_21. URL http://dx.doi.org/10.1007/11965893_21.

[11] L. Segoufin and V. Vianu. Validating streaming XML documents. In L. Popa, S. Abiteboul, and P. G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 53–64. ACM, 2002. ISBN 1-58113-507-6. doi: 10.1145/543613.543622. URL http://doi.acm.org/10.1145/543613.543622.

[12] T. G. Szymanski and J. H. Williams. Non-canonical extensions of bottom-up parsing techniques. Technical report, Ithaca, NY, USA, 1975.