Invisible pushdown languages

Eryk Kopczyński

November 3, 2015

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.

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. One of them is to use the *finite automata on trees* (TFAs) [4], which work on trees directly. Another one 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 symbols, or always pops a symbol, or it never pushes or pops symbols – this property allows the tree structure to be easily read. When we safely flatten a language on trees into a language of words – XML is the common and effective way to do that – these two approaches are seen to be equivalent (in some sense), 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 [7, 9]: given two VPDAs (or, equivalently, flattened TFAs) 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?

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 [7] 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 [5].

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 [8], where it is shown that, for a fully recursive DTD, it is decidable whether there is a regular language R such that, for any valid XML document w, it is decidable whether w is valid with respect to D. This is a special case of our result – we take fully recursive DTDs instead of arbitrary finite automata on trees, and we want to separate L from its complement. It is stated as an open problem in [8] whether the problem is still decidable for arbitrary finite automata on trees.

On the other hand, in [3] it is shown that it is decidable whether, for a given visibly push-down language L, there exists a language R such that $R \cap F = L$, where F is the language of all well matched words.

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

2 Preliminaries

We remind the basic notions of automata theory; see [6].

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

A *deterministic finite automaton* (DFA) is a tuple $A = (\Sigma, Q, q_I, F, \delta)$, where Σ is the alphabet of A, Q is the set fo 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 *binary flattened tree grammar* (BFG) over Σ is a context free grammar, whose set of terminal symbols is $\Sigma \cup \{\diamondsuit, \overline{\diamondsuit}\}$, and every production is of form $N \to t$, $N \to \diamondsuit\overline{\diamondsuit}$ or $N \to \diamondsuit N_1 N_2 \overline{\diamondsuit}$, where $N_1, \dots, N_k$ are non-terminals, and t is a terminal. A BFG corresponds to the XML encoding of a regular language of 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 $w \in L(G_1)$, $w \in R$, but for each $w \in L(G_2)$, $w \notin R$.*

Problem 2.2 (CFG-SEPARABILITY)

INPUT *Two context tree 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 [7, 9]. For convenience, we include the idea of the proof here. Encode configurations of a 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. They are separable iff M terminates from the initial configuration w_I.

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

Problem 2.3 (BFG-SEPARABILITY)
INPUT *Two BFGs 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 2.4 *The problem* **BFG-SEPARABILITY** *is undecidable.*

3 Proof

We will reduce **CFG-SEPARABILITY** to **BFG-SEPARABILITY**. To do this, we will take two CFGs G_1 and G_2, and create two BFGs 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. 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 [6].

Given a context free grammar $G = (V, \Sigma, R, S)$ in Chomsky normal form, we will construct a binary flattened tree grammar $G' = (V', \Sigma', R', S')$, 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 R, we have the corresponding production in R':
$$N' \to t \tag{1}$$
- For each production $N \to N_1 N_2$ in R, we have the corresponding bracketed production in R':
$$N' \to \diamondsuit N_1' N_2' \overline{\diamondsuit} \tag{2}$$

- For each $X \in V$, we also have the following productions for X':

$$X' \rightarrow \diamondsuit E' X' \overline{\diamondsuit} \quad (3)$$
$$X' \rightarrow \diamondsuit X' E' \overline{\diamondsuit} \quad (4)$$

- Where the productions for E' are as follows:

$$E' \rightarrow \diamondsuit \overline{\diamondsuit} \quad (5)$$
$$E' \rightarrow \diamondsuit E' E' \overline{\diamondsuit} \quad (6)$$

Consider $\pi : \Sigma' \rightarrow \Sigma$, 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 3.1 *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 3.2 *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 $vwx \in R$ iff $vw'x \in R$. Syntactic equivalence is a congruence with respect to concatenation.

Lemma 3.3 *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 cyclic sequence with period at most $|S|$, $\omega = |S|!$ will work. ■

We say that $T : \Sigma^* \rightarrow \Sigma'^*$ 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 3.4 *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 3.1 – 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 $A' = (\Sigma', Q, q_I, F, \delta)$. 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 3.5 *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 3.2 and thus Theorem 2.4. Indeed, we will show that $T(L(G_1))$ and $T(L(G_2))$ are separated by R – then, after applying Lemma 3.4, we get our claim.

Consider $w \in L(G_i)$; we have to show that $L(A)$ accepts $T(w)$ iff $i = 1$. From Lemma 3.5 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 3.5

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 (3, 4 or 6) and then (5), 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 terminals K' and L', we get $K' \to^* b_1 K' b_2$ and $C' \to^* c_1 L' c_2$, where:

$$b_1 = (\diamondsuit^\nu \heartsuit^\nu)^\omega \diamondsuit^\nu \quad (7)$$

$$b_2 = \overline{\heartsuit}^\nu (\overline{\diamondsuit}^\nu \overline{\heartsuit}^\nu)^\omega \quad (8)$$

$$c_1 = \heartsuit^\omega (\heartsuit^\nu \diamondsuit^\nu)^\omega \quad (9)$$

$$c_2 = (\overline{\heartsuit}^\nu \overline{\diamondsuit}^\nu)^\omega \overline{\diamondsuit}^\nu \quad (10)$$

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

$$N' \to \diamondsuit K' L' \overline{\diamondsuit} \to^* \diamondsuit b_1 K' b_2 c_1 L' c_2 \overline{\diamondsuit} \quad (11)$$

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

$$e_L = \diamondsuit b_1 \quad (12)$$
$$e = b_2 c_1 \quad (13)$$
$$e_R = c_2 \overline{\diamondsuit} \quad (14)$$

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 (11) above, and each production $N \to t$ with $N' \to t$ (1). 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 $l_1, r_n \geq 1$. 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:

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

- $e_R e$ 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 $T(w)$. ■

4 Why binary trees?

Our proof uses *binary* flattened tree grammars – that is, productions of form $N \to \diamondsuit N_1 N_2 \dots N_k \overline{\diamondsuit}$ are allowed only for $k = 0$ or $k = 2$.

If we allowed such rules for any k, a somewhat easier proof would work. The general structure is the same, but it is not required to have G_i's in the Chomsky normal form, and the grammar G' allows inserting empty brackets with productions $E' \to \diamondsuit\overline{\diamondsuit} | \diamondsuit E' \overline{\diamondsuit} | \diamondsuit E' E' \overline{\diamondsuit}$ before and after every symbol: $N \to \diamondsuit E' X_1 E' X_2 \dots X_k E' \overline{\diamondsuit}$, and $e_L = e = e_R = (\diamondsuit^\omega \overline{\diamondsuit} \omega)^\omega$ can be obtained by pumping each pair of $\diamondsuit\overline{\diamondsuit}$ ω times, and then using the rule $E' \to E'E'$ to make sure that $\diamondsuit^\omega \overline{\diamondsuit}^\omega$ appears $k\omega$ times between each two consecutive terminals.

While such a proof was sufficient to solve the problem for visibly pushdown languages, and for XML encoding of trees, it left 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 simpler proof above heavily uses the fact that the XML encoding cannot tell whether $\diamondsuit$ comes from the structurally significant rule $N \to \diamondsuit E' X_1 E' X_2 \dots X_k E' \overline{\diamondsuit}$, or it is a fake inserted by the $X \to \diamondsuit X \overline{\diamondsuit}$ or $E' \to \diamondsuit E' E' \overline{\diamondsuit}$ rules. 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 rules 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" rule $E' \to \diamondsuit\overline{\diamondsuit}$ 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 $\diamondsuit$.

In the binary case, a computer program was used to find the appropriate words e, e_L, e_R which work in Lemma 3.5.

5 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 3.3, and it is used in the proof of Lemma 3.5.)
- 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 3.1 and 3.4.)
- If $T(t_1 \dots t_k) = e_L t_1 e t_2 e \dots e t_k e_R$ for some s, and $L \in \mathcal{C}$, then $T(L) \in \mathcal{C}$. (Used in the proof of 3.4.)

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

References

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

[2] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László 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.

[3] Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer Berlin Heidelberg, 2006.

[4] 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.

[5] Wojciech Czerwinski, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In Adrian Kosowski and Igor 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.

[6] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[7] H. B. Hunt, III. On the decidability of grammar problems. *J. ACM*, 29(2):429–447, April 1982.

[8] Luc Segoufin and Victor Vianu. Validating streaming XML documents. In Lucian Popa, Serge Abiteboul, and Phokion 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.

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