Fundamenta Informaticae 154 (2017) 37–46
DOI 10.3233/FI-2017-1551
IOS Press

It is Undecidable if Two Regular Tree Languages can be Separated by a Deterministic Tree-walking Automaton

Mikołaj Bojańczyk[*†]
The University of Warsaw
Institute of Informatics
Banacha 2, 02-097 Warsaw, Poland
bojan@mimuw.edu.pl

Abstract. The following problem is shown undecidable: given regular languages L, K of finite trees, decide if there exists a deterministic tree-walking automaton which accepts all trees in L and rejects all trees in K. The proof uses a technique of Kopczyński from [1].

1. Introduction

Regular languages have a sort of anti-Rice theorem: for every natural property X, one can decide which regular languages have property X. Examples of such properties include: "empty", "infinite", "universal", "commutative", "upward closed in the Higman ordering", "definable in first-order logic", etc. A nonexample is "contains an accepting computation of the universal Turing machine", see also [2] for an example where the undecidability is less aparent. There are properties for which no algorithm is known, but it is believed that with sufficient work an algorithm will be found, e.g. the property "definable in level Σ_5 of the first-order quantifier hierarchy (see [3] for a discussion on how algorithms were provided for the first 4 levels)". Trees – at least finite ones – look similar, with algorithms for properties like emptiness, finiteness, or upward closure being quite straightforward. Of

[*]Address for correspondenece: The University of Warsaw, Institute of Informatics, Banacha 2, 02-097 Warszawa, Poland
[†]The work was supported by the grant Lipa, which has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 683080).

course trees can be more challenging, so some questions remain open, e.g. it is not known if one can decide which regular tree languages are definable in first-order logic [4]. Nevertheless, the prevailing opinion seems to be that the final answer to this and similar questions will be "decidable".

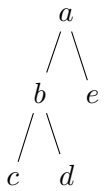
This paper gives an example of an undecidable property of regular tree languages, namely this:

Theorem 1.1. The following problem is undecidable:

- **Input.** Two regular tree languages, given as bottom-up automata;
- **Question.** Can they be separated by a deterministic tree walking automaton, i.e. is there a deterministic tree walking automaton which accepts all trees in the first language, and rejects all trees in the second language?

The undecidable question in the above theorem is a property not of one, but of two regular tree languages. Questions about separation, like the one above, are currently an important theme in the theory of regular languages, see e.g. the references in Section 5 of the survey [3].

This paper is based on a result by Kopczyński [1], which showed that it is undecidable if two visibly pushdown word languages can be separated by a regular word language. Since a visibly pushdown word language can be viewed as a tree language, Kopczyński's result can be rephrased as follows: it is undecidable if two given regular tree languages can be separated by a regular property of the words which are their XML encodings, see Figure 1. Because of the similarity of visibly pushdown languages to pushdown languages, the revolutionary character of Kopczyński's result was less apparent – after all, so many questions about pushdown automata are undecidable (like universality, or more close to this topic, separation by regular word languages).



<a><c></c><d></d><e></e>

a tree its XML encoding

Figure 1. XML encoding

This underlying technical development of the paper differs only slightly from [1]. Our problem has the same instances (pairs of regular tree languages, also known as visibly pushdown languages), it asks a very similar separation question, and we use the same reduction to prove undecidability. Since our separating mechanism is stronger than the one used by Kopczyński (deterministic tree-walking automata, as opposed to regular properties of the XML encoding), we need a stronger lemma to prove correctness of the reduction, but this stronger lemma is simply taken from the literature; thus making the proof shorter than [1] but not self-contained.

I would like to thank Sylvain Schmitz and the anonymous referees for helpful comments.

2. Trees and their automata

This section defines basic tree terminology, and introduces the two models of tree automata that will be considered: the stronger model of deterministic bottom-up tree automata, and the weaker model of deterministic tree-walking automata.

Trees and terms

In this paper, a *ranked alphabet* is a finite set where each element has an associated arity (a natural number, with zero being used for letters that are used to label leaves). For a finite ranked aphabet Σ, define a tree over Σ to be a finite, sibling-ordered tree, where every node has a label from Σ and the number of children is the arity of the label. For $n \geq 0$, define an n-ary term over Σ to be a tree over the alphabet $\Sigma \cup \{*\}$, where * is a letter of arity zero that appears exactly n times. Every occurrence of * is called a *port*, the idea is that trees or terms can be substituted into a port. We write $\mathsf{trees}_n\Sigma$ for the set of n-ary terms. In the case $n = 0$ of trees we omit the subscript 0. If t is an n-ary term, and $t_1, \ldots, t_n$ are terms, then we write $t(t_1, \ldots, t_n)$ for the term (whose arity is the sum of arities of the terms $t_1, \ldots, t_n$)) obtained from t by substituting t_i for the i-th port in t. Note that our notion of term uses each argument once, as opposed to the more typical notion which allows each argument to be used several times.

We consider two automaton models for trees, as described below.

Deterministic bottom-up tree automata

A *deterministic bottom-up tree automaton* consists of: an input ranked alphabet Σ, a state space Q, a set $F \subseteq Q$ of accepting states, and for each letter $a \in \Sigma$ of arity n a transition function:

$$\delta_a : Q^n \to Q.$$

The automaton is evaluated on a tree in a bottom up way. The state in a tree is obtained by reading the root label, and applying its transition function to the states in the child subtrees. The language recognised by such an automaton is the set of all trees which are evaluated to an accepting state. A tree language is called *regular* if it is recognised by such an automaton.

Deterministic tree-walking automata

A computation of a deterministic bottom-up tree automaton, as described above, can be viewed as a branching computation, since the state in a node depends on the states in all of its children. In contrast, a tree-walking automaton, as described below, is a sequential device, where a computation has a linear structure. The syntax of a deterministic tree-walking automaton consists of a ranked input alphabet Σ, a set of states Q, an initial state $q_0 \in Q$, and for each letter $a \in \Sigma$ of arity n a transition function

$$\delta_a : \underbrace{Q \times \{\text{root}, 1, \ldots, \text{maxarity}\}}_{\text{what the automaton sees}} \to \underbrace{\{\text{accept,reject}\} \cup \big(Q \times \{\text{parent}, 1, \ldots, n\}\big)}_{\text{what the automaton does}},$$

where maxarity stands for the maximal arity of letters in the input alphabet. In a given input tree, a *configuration* of the automaton is a pair of the form (state of the automaton, node of the tree). The automaton begins in the configuration which consists of the initial state and the root of the input tree. When in a configuration (q, v), the automaton applies the transition function corresponding to v's label, with the argument to the function being the state q and the child number of v (i.e. the number i such that v is the i-th child of its parent, or "root" if v has no parent). Based on the result of the transition function, the automaton chooses to accept/reject the tree, or change its state and make a move to some neighbouring node (or no move at all). In principle, there can be runs that do not accept because the automaton enters a loop, or runs where the automaton walks out of the tree by e.g. moving to the parent in the root node. As shown in [5], every deterministic tree-walking automaton can be converted in polynomial time into one which always ends up by using an accept or reject command.

3. Undecidability of separation

We say that two sets are *separated* by a set M if M contains the first set and is disjoint with the second. The contribution of this paper is the following theorem.

The proof of the above theorem uses a technique from [1], which shows undecidability for separation of visibly pushdown languages by regular word languages. As in [1], we reduce from the following undecidability result, which was shown even under the assumption that the input grammars are deterministic, see Theorem 4.6 in [6].

Theorem 3.1. The following problem is undecidable:

- **Input.** Two context-free word languages, given by grammars.
- **Question.** Can they be separated by some regular word language?

The reduction we use is actually the same transformation from context-free grammars to tree languages as used by Kopczyński in [1], only the correctness proof is different, since we reduce to a slightly different problem (the problem used by Kopczyński had the same instances, but a weaker class of separating languages, and therefore fewer "yes" instances).

The main result about deterministic tree-walking automata that is needed for the correctness proof is the following lemma on deterministic tree-walking automata, which is taken from [7]. For a tree language $L \subseteq \mathsf{trees}\Sigma$ define two terms $t, t' \in \mathsf{trees}_n\Sigma$ to be L-equivalent if

$$s(t(s_1, \ldots, s_n)) \in L \quad \text{iff} \quad s(t'(s_1, \ldots, s_n)) \in L$$

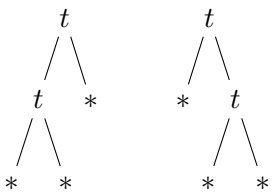
holds for every $s \in \mathsf{trees}_1\Sigma$ and $s_1, \ldots, s_n \in \mathsf{trees}\Sigma$. The following Rotation Lemma was proved[1] in [7].

[1]The careful reader will note that [7] proves a weaker result, namely Lemma 18, which uses a very slightly coarser notion of L-equivalence, call it *weak L-equivalence*, see page 4 in [7]. In weak L-equivalence, we require that

$$s(t(s_1, \ldots, s_n)) \in L \quad \text{iff} \quad s(t'(s_1, \ldots, s_n)) \in L$$

holds for every $s \in \mathsf{trees}_1\Sigma$ and $s_1, \ldots, s_n \in \mathsf{trees}\Sigma$ which satisfy the additional condition that each port is a left child in

Lemma 3.2. (Rotation Lemma)
Let Σ be a ranked alphabet, which contains a letter a of rank 2 and a letter c of rank 0. Let L be a tree language over Σ which is recognised by a deterministic tree-walking automaton. There exists some $t \in \mathsf{trees}_2\{a, c\}$ such that following two terms are L-equivalent:



Kopczyński obfuscation

We now present the reduction from separation of context-free word languages by a regular word language (the problem in Theorem 3.1) to separation of regular tree languages by a deterministic tree-walking automaton. Consider a context-free grammar G in Chomsky normal form, with terminals Γ and nonterminals N. Since we use Chomsky normal form, nonterminals get transformed into pairs of nonterminals, and therefore we can view Γ as ranked letters of arity zero, and N as ranked letters of arity 2, and we can view derivations of the grammar as trees in $\mathsf{trees}(\Gamma \cup N)$.

Choose some fresh letters a, c, of arities 2 and 0 respectively. The *Kopczyński obfuscation* of G, denoted by $\mathsf{kop}(G)$, is the set of all trees that can be obtained from some derivation of the grammar, and replacing each nonterminal by a binary term over the alphabet $\{a, c\}$, possibly using different terms for different occurrences of nonterminals. A more formal definition is that

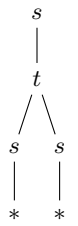
$$\mathsf{kop}(G) = \bigcup_{t \text{ a derivation of } G} \mathsf{kop}(t),$$

while $\mathsf{kop}(t)$ is the set of trees over alphabet $\Gamma \cup \{a, c\}$ defined by

$$\begin{aligned} \mathsf{kop}(\sigma) &= \{\sigma\} \\ \mathsf{kop}(\sigma(t_1, t_2)) &= \{s(s_1, s_2) : s \in \mathsf{trees}_2\{a, c\}, s_1 \in \mathsf{kop}(t_1), s_2 \in \mathsf{kop}(t_2)\} \end{aligned}$$

---

s and each s_i has at least two nodes. In the proof of Lemma 18, the term t has the property that it is weakly L-equivalent to



for some s where the only leaf port is a left child. For such terms, weak L-equivalence coincides with L-equivalence as used in the Rotation Lemma.

where the first line is used for trees with just one node, and the second line for other trees. We use the name Kopczyński because mapping a grammar to its Kopczyński obfuscation was the reduction used in [1], as it is also in this paper. It is not difficult to see that the obfuscation is a regular tree language and that a tree automaton for the obfuscation can be computed based on the grammar. The following lemma shows that taking the Kopczyński obfuscation reduces the undecidable problem in Theorem 3.1 to the problem in Theorem 1.1, thus proving undecidability of the latter.

Lemma 3.3. Let G, H be context free grammars, with terminals Γ. The following conditions are equivalent:

1. The tree languages

$$\mathsf{kop}(G), \mathsf{kop}(H) \subseteq \mathsf{trees}(\Gamma \cup \{a, c\})$$

   can be separated by a deterministic tree-walking automaton.

2. The word languages

$$L(G), L(H) \subseteq \Gamma^*$$

   generated by these grammars can be separated by a regular word language.

The implication from 2 to 1 in the above lemma is straightforward. This is because for every regular word language L, in particular the separator, there is a deterministic tree-walking automaton that accepts an input tree if and only if L contains the sequence of leaves read from left to right. The idea is to use depth-first search, see e.g. Example 1 in [8].

It remains to prove the converse implication from 1 to 2. Here our task is more difficult than in [1], because deterministic tree-walking automata are relatively powerful, and can be quite challenging to prove that they cannot do something. We use the following corollary of the Rotation Lemma. For $t \in \mathsf{trees}_2\Sigma$, define t^* to be the smallest set of terms that contains $*$ (a unary term with the port in the root) and which is closed under composition with t in the following sense:

$$t_1, t_2 \in t^* \qquad \text{implies} \qquad t(t_1, t_2) \in t^*.$$

Lemma 3.4. Let $L \subseteq \mathsf{trees}\Sigma$ and t be as in the Rotation Lemma and let Γ be the rank 0 symbols in Σ. There is a regular word language $K \subseteq \Gamma^*$ such that

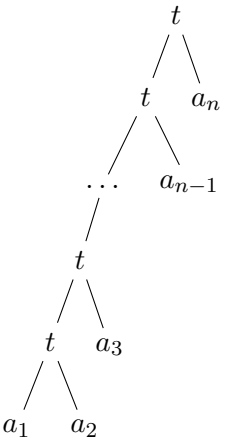
$$a_1 \cdots a_n \in K \qquad \text{iff} \qquad s(a_1, \ldots, a_n) \in L$$

holds for every $n \geq 2$, $a_1, \ldots, a_n \in \Gamma$ and n-ary $s \in t^*$.

Before proving the above lemma, note that it implies that as long as s is taken from t^*, then membership of $s(a_1, \ldots, a_n)$ in L does not depend on the branching structure of s, but only on the sequence $a_1, \ldots, a_n$. More precisely, for every $a_1, \ldots, a_n \in \Gamma$ and every $s, s' \in t^*$ with exactly n ports we have

$$s(a_1, \ldots, a_n) \in L \qquad \text{iff} \qquad s'(a_1, \ldots, a_n) \in L.$$

Proof:
For $a_1, \ldots, a_n \in \Gamma$, define $\mathsf{comb}(a_1, \ldots, a_n)$ to be the following tree:



Every two binary trees with the same number of leaves can be transformed into each other via a sequence of rotations. Therefore, repeated application of the Rotation Lemma shows that every n-ary $s \in t^*$ satisfies

$$s(a_1, \ldots, a_n) \in L \qquad \text{iff} \qquad \mathsf{comb}(a_1, \ldots, a_n) \in L.$$

To complete the proof, it suffices to show that

$$K = \{a_1 \cdots a_n \in \Gamma^* : \mathsf{comb}(a_1, \ldots, a_n) \in L\}$$

is a regular word language. Since deterministic tree-walking automata can only recognise regular tree languages, see e.g. Fact 1 in [8], there is a bottom-up tree automaton $\mathcal{A}$ that recognises L. We define a deterministic word automaton recognising K as follows. The states are the same as in $\mathcal{A}$ plus a special initial state. When the automaton is in the initial state and reads a letter $\sigma \in \Gamma$, it moves to the state of $\mathcal{A}$ after reading a one node tree σ. When the automaton is in a state of $\mathcal{A}$, then the transition function is defined by

$$\delta(q, \sigma) = t(q, \sigma) \qquad \text{for } \sigma \in \Gamma$$

where $t(q, \sigma)$ is the state of $\mathcal{A}$ after reading a tree obtained from $t(*, \sigma)$ by putting some tree evaluated to q into the port. By definition, this word automaton maps a word $a_1 \cdots a_n \in \Gamma^*$ to the state of the tree automaton $\mathcal{A}$ after reading the tree $\mathsf{comb}(a_1, \ldots, a_n)$, and therefore the language K is regular.
□

Using the above lemma, we complete the implication from 1 to 2 in Lemma 3.3. Suppose that $\mathsf{kop}(G)$ can be separated from $\mathsf{kop}(H)$ by some deterministic tree-walking automaton recognising a language $L \subseteq \mathsf{trees}(\Gamma \cup \{a, c\})$. Apply the Rotation Lemma to L, yielding t, and apply Lemma 3.4 yielding a regular word language $K \subseteq \Gamma^*$. We claim that K separates the context-free word languages

generated by G and H. Indeed, suppose that $a_1 \cdots a_n$ is generated by G. By taking the corresponding derivation and replacing each nonterminal by t, we see that there is some n-ary term $s \in t^*$ such that

$$s(a_1, \ldots, a_n) \in \mathsf{kop}(G).$$

Since $\mathsf{kop}(G)$ is contained in L, it follows that $a_1 \cdots a_n \in K$. Conversely, if $a_1 \cdots a_n$ is generated by H, then there is some n-ary term $s \in t^*$ such that

$$s(a_1, \ldots, a_n) \in \mathsf{kop}(H).$$

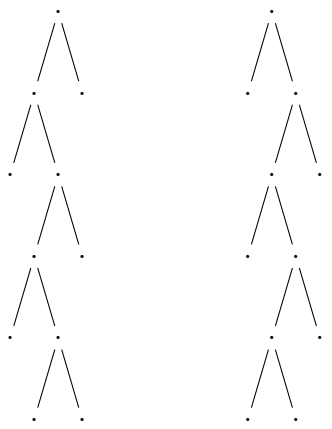
Since $\mathsf{kop}(H)$ is disjoint with L, it follows that $a_1 \cdots a_n \notin K$. This completes the proof of Lemma 3.3, and therefore also of Theorem 1.1.

4. What is the scope of the technique?

The proof of Theorem 1.1 works not just for deterministic tree-walking automata, but also for any class of regular languages $\mathcal{L}$ that satisfies the Rotation Lemma and is strong enough to express properties like: "the sequence of leaves, when read from left to right, belongs to a regular language K". However, this makes the technique sound more powerful than it is: the Rotation Lemma is a very strong lemma, and seems to hold only for deterministic tree-walking automata and their special cases. For example, the Rotation Lemma fails for nondeterministic tree-walking automata, and all fragments of first-order logic beyond Boolean combinations of Σ_1 sentences, for which separation is decidable [9].

It seems therefore that the technique of Kopczyński obfuscation is exhausted by deterministic tree-walking automata. As an example, we claim that one can find:

- a grammar G generating the palindromes; and



derivations in G derivations in H

Figure 2. In a derivation from G, the right child of the root is a leaf, while in a derivation from H, the left child of the root is a leaf.

- a grammar H generating the non-palindromes;

such that the Kopczyński obfuscations $\mathsf{kop}(G)$ and $\mathsf{kop}(H)$ can be separated by a nondeterministic tree-walking automaton, thus showing that the reduction in Lemma 3.3 fails for nondeterministic tree-walking automata. The trick is to choose the grammars so that their derivations have shapes as in Figure 2; then the technique from Lemma 2 in [7] can be used to separate $\mathsf{kop}(G)$ from $\mathsf{kop}(H)$. This counterexample also works for other separators, e.g. for first-order logic. The counterexample only means that the same reduction cannot be used, but the problem might still be undecidable.

Conclusion.

The conclusion is that some questions about regular tree languages can indeed be undecidable. The particular undecidability proof in this paper strongly depends on the Rotation Lemma – which is true essentially only for deterministic tree-walking automata – and on separation. To highlight the role of separation, consider the class $\mathcal{L}$ of regular tree languages L such that $t \in L$ depends only on the sequence of leaves in t, read from left to right. Then membership of a regular tree language in $\mathcal{L}$ is decidable (see Theorem 1 in [10] for a stronger result) but separation of two regular tree languages by $\mathcal{L}$ is undecidable, using the same proof as here or in [1].

References

[1] Kopczyński E. Invisible Pushdown Languages. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. 2016 pp. 867–872. doi:10.1145/2933575.2933579. URL http://doi.acm.org/10.1145/2933575.2933579.

[2] Hinz F, Dassow J. A undecidability result for regular anguages and its applications to regulated rewriting. Bulletin of the EATCS, 1989;38:168–173. ISSN-0252-9742.

[3] Place T, Zeitoun M. The Tale of the Quantifier Alternation Hierarchy of First-Order Logic over Words. SIGLOG Newsletter, 2015;2(3):4–17. doi:10.1145/2815493.2815495.

[4] Thomas W. Logical Aspects in the Study of Tree Languages. In: CAAP'84, 9th Colloquium on Trees in Algebra and Programming, Bordeaux, France, March 5-7, 1984, Proceedings. 1984 pp. 31–50. ISBN:0-521-26750-1.

[5] Muscholl A, Samuelides M, Segoufin L. Complementing deterministic tree-walking automata. Inf. Process. Lett., 2006;99(1):33–39. doi:10.1016/j.ipl.2005.09.017. URL http://dx.doi.org/10.1016/j.ipl.2005.09.017.

[6] Szymanski TG, Williams JH. Noncanonical Extensions of Bottom-Up Parsing Techniques. SIAM J. Comput., 1976;5(2):231–250. doi:10.1137/0205019. URL http://dx.doi.org/10.1137/0205019.

[7] Bojańczyk M, Colcombet T. Tree-walking automata cannot be determinized. Theor. Comput. Sci., 2006;350(2-3):164–173. doi:10.1016/j.tcs.2005.10.031. URL http://dx.doi.org/10.1016/j.tcs.2005.10.031.

[8] Bojańczyk M. Tree-Walking Automata. In: Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers. 2008 pp. 1–2. doi:10.1007/978-3-540-88282-4_1. URL http://dx.doi.org/10.1007/978-3-540-88282-4_1.

[9] Goubault-Larrecq J, Schmitz S. Deciding Piecewise Testable Separability for Regular Tree Languages. In: 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy. 2016 pp. 97:1–97:15. doi:10.4230/LIPIcs.ICALP.2016.97. URL http://dx.doi.org/10.4230/LIPIcs.ICALP.2016.97.

[10] Wilke T. An Algebraic Characterization of Frontier Testable Tree Languages. Theor. Comput. Sci., 1996;154(1):85–106. doi:10.1016/0304-3975(95)00131-X. URL http://dx.doi.org/10.1016/0304-3975(95)00131-X.