

Effective Characterizations of Tree Logics

Mikołaj Bojańczyk *
Warsaw University, Poland
bojan@mimuw.edu.pl

ABSTRACT

A survey of effective characterizations of tree logics. If \mathcal{L} is a logic, then an effective characterization for \mathcal{L} is an algorithm, which inputs a tree automaton and replies if the recognized language can be defined by a formula in \mathcal{L} . The logics \mathcal{L} considered include path testable languages, frontier testable languages, fragments of Core XPath, and fragments of monadic second-order logic.

Categories and Subject Descriptors

F.4.1 [Mathematical logic and formal languages]: Mathematical logic; H.2.3 [Database management]: Languages—Query languages

General Terms

Languages, Theory

1. INTRODUCTION

We say a logic (such as first-order logic, or CoreXPath) has an effective characterization if the following decision problem is decidable: “given as input a finite automaton, decide if the recognized language can be defined using a formula of the logic”. Representing the input language by a finite automaton is a reasonable choice, since many known logics (over words or trees) are captured by finite automata.

This type of problem has been successfully studied for word languages. Arguably best known is the result of McNaughton, Papert and Schützenberger [25, 19], which says that the following three conditions on a regular word language L are equivalent: a) L can be defined by a star-free regular expression; b) L can be defined in first-order logic with order and label tests; c) the syntactic monoid of L does not contain a non-trivial group. Since condition b) can be effectively tested, the above theorem gives an effective characterization of star-free expressions, and first-order logic. This

*Author supported by Polish government grant no. N206 008 32/0810.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS’08, June 9–12, 2008, Vancouver, BC, Canada.

Copyright 2008 ACM 978-1-60558-108-8/08/06 ...\$5.00.

result demonstrates the importance of this type of work: an effective characterization not only gives a better understanding of the logic in question, but it often reveals unexpected connections with algebraic concepts. During several decades of research on word languages, effective characterizations have been found for fragments of first-order logic with restricted quantification and a large group of temporal logics, see [20] and [33] for references.

Probably the most important question for trees is: does first-order logic on trees have an effective characterization? There are a number of variants of this question, depending on the type of tree considered (binary, ranked, or unranked) and the exact flavor of first-order logic (what predicates are available: child, descendant, document order). No effective characterization is known for any of the variants. The only exception, see [2], is first-order logic where the child relation is used instead of the descendant relation.

First-order logic on trees and the Potthof example.

To illustrate the idiosyncratic nature of first-order logic on trees, we present an example due to Potthof [22]. In the example, the trees considered are finite and binary, i.e. each node has exactly zero or two children. The first-order formulas will use predicates $left(x, y)$ and $right(x, y)$ for the left and right children, and $x < y$ for descendants. The variables in the formula quantify over tree nodes, e.g. the formula

$$\forall x(\exists y left(x, y) \Rightarrow \exists z right(x, z))$$

says that each node x with a left child y also has a right child z ; this formula is a tautology in binary trees.

The Potthof example refers to parity. As is well known, word languages that refer to parity, such as “words of even length”, or “words with an even number of a ’s” cannot be defined in first-order logic. For (binary) trees, however, the situation becomes more complicated. First off, the property “trees with an even number of nodes” is first-order definable, for the simple reason that every binary tree has an odd number of nodes. On the other hand, and in analogy to words, the property “trees with an even number of a ’s” is not definable in first-order logic. However, and this is the surprising example of Potthof, the property “trees where some leaf has even depth (i.e. an even number of proper ancestors)” can actually be defined in first-order logic. The reason is that this property is equivalent to:

(*) Either the zigzag of the root has even length, or there are two siblings with zigzags of even and odd lengths, respectively.

In the above, the zigzag of a node x contains the left child of x , the right child of the left child of x , the left child of the right child of the left child of x , and so on until a leaf is reached. It is not hard to write a first-order formula $\varphi(x, y)$ which says that y is in the zigzag of x . The formula is a conjunction of several conditions, one of which is

$$\neg \exists y_1, y_2, y_3. x \leq y_1 \wedge \text{left}(y_1, y_2) \wedge \text{left}(y_2, y_3) \wedge y_3 \leq y.$$

Note that the zigzag has even length if and only if the only leaf it contains is a left child; therefore the property (*) on zigzags can be defined in first-order logic.

What conclusions can we draw example above?

A first, and more obvious, conclusion is that trees are more complicated than words, and require new insights. There are other examples of unusual tree behavior, some of which can be found in this survey.

A second, possibly more debatable, conclusion is that some of the quirky phenomena illustrated above may come from using binary trees, which have the arbitrary restriction that each node has either two or zero successors. Maybe the difference between binary trees and unranked trees—a cosmetic difference in most other settings—begins to play a role tackling the difficult problem of effective characterizations. Accordingly, some of the recent research in tree logics has moved from binary trees to unranked trees. This is also the direction chosen by this survey, where most of the results are about unranked trees. Unranked trees also have the advantage of being closer to the XML data model.

Scope and organization.

This paper is not a survey of logics for trees, or even of logics for unranked trees, for which excellent sources are readily available. The reader interested in an introduction to the links between automata and logic is referred to [30]. A survey of logics that focuses on unranked trees can be found in [17]. In general terms, this survey is about the differences in expressive power between various tree logics. Specifically, though, the focus is on finding effective characterizations of tree logics. Most attention is devoted to those logics that have known effective characterizations, sometimes at the cost of more interesting logics without known effective characterizations.

The paper is organized as follows.

In Section 2, we present the basic definitions. The most important concept is that of identity, which is illustrated on three toy logics.

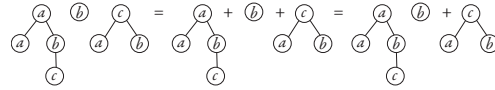
Sections 3–5 contain the core of the paper, a survey of existing effective characterizations of tree languages. These are grouped in to three categories. Section 3 is about something that can be called “pattern testing”, these are languages like “trees that contain a path in $(ab)^*$ ”, or “trees that contain a in a leaf”. Section 4 is about CoreXPath and its fragments. Section 5 is about first-order and monadic second-order logic.

Finally, Section 6 presents an undecidability result: it is impossible to give a “general” algorithm characterizing classes of tree languages.

2. PRELIMINARIES

The trees in this paper are finite and labeled. The siblings are ordered, although some logics will not be able to refer to this order. Some of the trees will be *binary* (each node has

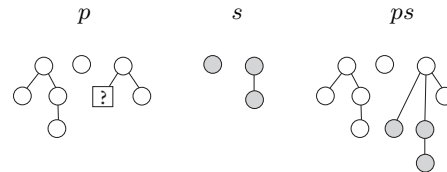
two or zero children), but most of the time trees will be *unranked*, where there is no restriction (apart from finiteness) on the number of children. A *forest* is an ordered sequence of unranked trees. The empty forest is denoted by 0. We concatenate forests using the + symbol.



Forests and trees alike will be denoted by the letters s, t, u, \dots . Labels of nodes will be denoted by a, b, c . We use letters A, B, C to denote the alphabets, i.e. finite sets of labels. A set L of forests over given alphabet A is called a *forest language over A*.

The notions of node, descendant and ancestor relations between nodes are defined in the usual way. We use x, y, z to denote nodes. We write $x < y$ to say that x is a proper ancestor of y or, equivalently, that y is a proper descendant of x . A *path* is a set of nodes $\{x_1, \dots, x_n\}$ where x_{i+1} is a child of x_i , for all i . A maximal path connects the root with a leaf.

If we take a forest and replace one of the leaves by a special symbol \square , we obtain a *context*. Contexts will be denoted using letters p, q, r . The empty context, where the only node is the hole, is denoted by \square . A forest s can be substituted in place of the hole of a context p , the resulting forest is denoted by ps , as illustrated below:



There is a natural composition operation on contexts: the context qp is formed by replacing the hole of q with p . This context satisfies $(pq)s = p(qs)$ for all forests s .

Finally, we also allow concatenating a forest s to a context p . The result, denoted by $s + p$ is the unique context that satisfies $(s + p)t = s + pt$ for all forests t . The concatenation $p + s$ is defined symmetrically, with $(p + s)t = pt + s$.

Regular languages.

There are many equivalent definitions of regular tree languages. We will present two here: languages recognized by automata and languages with a finite index Myhill-Nerode congruence. A third definition—in terms of monadic second-order logic—will appear later on.

The first definition is in terms of automata. A *tree automaton works* as follows. Fix an input alphabet A . The automaton has a finite state space Q and a finite set of *transitions* of the form $L, a \rightarrow q$, where $L \subseteq Q^*$ is a regular word language, $a \in A$ is a letter of the input alphabet and $q \in Q$ is a state. A *run* of the automaton assigns (nondeterministically) a state to each node of the input tree. These states have to be consistent with that transition function in the following sense. Let x be a node with label a and children x_1, \dots, x_n , listed from left to right. If the node x is assigned q by the run, and the nodes x_1, \dots, x_n are assigned states q_1, \dots, q_n , then there must be a transition $L, a \rightarrow q$ such that the word $q_1 \dots q_n$ belongs to the language L . A

run is called *accepting* if the state assigned to the root belongs to a designated set of accepting states. Formally, the automaton is given by a tuple

$$\langle Q, A, \delta, F \rangle ,$$

where Q is the state space, A is the input alphabet, δ is the set of transitions and $F \subseteq Q$ is the set of accepting states. The set of trees accepted by an automaton is called the language *recognized* by the automaton. A tree language is called *regular* if it is recognized by some tree automaton.

An equivalent definition uses the Myhill-Nerode *syntactic congruence* (actually, there will be two congruences, one for forests and one for contexts). We will use this definition more often, since it is used in “identities”, the key technical tool in this paper. Let L be a tree language. Two forests s, s' are called *equivalent under L* , written $s \sim_L s'$, if

$$ps \in L \quad \Leftrightarrow \quad ps' \in L$$

holds for every context p such that both ps and ps' are trees. Two contexts q, q' are called *equivalent under L* , also written $q' \sim_L q$, if for any forest t , the two forests qt and $q't$ are equivalent under L . When the language L in question is clear from the context, we omit the subscript in \sim_L and simply write \sim . Note the asymmetry in the definition: although the equivalence relation is defined on forests (or contexts with possibly many roots), the language L itself only talks about trees. This is the reason for the clause that restricts the contexts p to ones where both ps and ps' are trees. Thanks to this clause, the language “all trees” has only one equivalence class for contexts, and one for contexts. Without the clause the tree language “all trees” would not have the same equivalence relation as its complement, the empty language. Using a standard technique, one can show that a tree language is regular if and only if both its syntactic equivalences have finite index.

The syntactic equivalences are a congruence with respect to the operations

$$s + t \quad ps \quad pq \quad s + p \quad p + s$$

on forests s, t and contexts p, q , as defined in the previous section. Note that this is a congruence in a two-sorted algebra (forests and contexts), which is called a *forest algebra* [8].

We will be using algorithms to reason about the syntactic congruence. How do we represent a syntactic congruence on the input of such an algorithm? We have “multiplication tables” for each of the five operations outlined above, as well as a mapping, which to every letter a in the alphabet assigns the equivalence class of the context $a\Box$ with label a in the root and a hole below. Note that the equivalence class of the empty forest (resp. empty context) is uniquely defined by the multiplication tables, since this is the neutral element for forest concatenation (resp. context composition). Since every context and forest can be built from the empty context, empty forest, and contexts $a\Box$, the mapping α and the multiplication tables uniquely specify the syntactic congruence. The representation of the congruence can be computed based on an automaton for the language, although the congruence may have exponentially more classes than the automaton has states (e.g. the tree language “label a at depth n ”).

Identities.

The focus of this paper is to provide identities that describe the expressive of tree logics. An identity can be seen as a pumping lemma: it says that one tree can be replaced by another, and the logic will not notice.

We begin our discussion with a simple identity, called *commutativity*, which will be used many times in the paper:

$$s + t \sim t + s .$$

This identity should be read as follows: in any situation, the forest $s + t$ can be replaced by the forest $t + s$. In other words, the language is closed under reordering children.

In the identities, the letters used for the variables will implicitly identify the type of the variable. Unless otherwise stated, variables s, t and u stand for forests, while p, q and r stand for contexts. Note that in the particular case of commutativity, restricting quantification to only trees would have the same effect.

Below we present three toy logics, with their simple characterizations expressed in terms of identities. The idea is to successively demonstrate the concepts that will appear in our effective characterizations. The first example is the class of languages that only depend on the set of labels in the tree; this example is characterized by two simple identities. In the second example, we introduce the ω exponent into the identities, a way of talking about loops in an identity. In the third example, we use a language class that is not closed under boolean operations; in particular, we need to use implications rather than identities.

Theorem 1

A regular tree language is definable by boolean combination of clauses of the form “the tree contains label a ” if and only if it satisfies the identities

$$pq \sim qp \quad pp \sim p .$$

Note that although the identities are tested for a tree language, the variables p and q quantify over contexts that may have several roots. This will be true later on in the paper as well: when a forest or context is not explicitly restricted, then it can have any possible number of roots.

Proof

The “only if” part, as is usual in such results, is clear, since both sides of each identity have the same set of node labels. For the “if” part, we will show that the identities above imply that any tree t is equivalent to one in a normal form, which only depends on the set of labels in t . The first step on the way to the normal form is that each forest is equivalent to one of the form $a_1 + \dots + a_n$. The proof is by induction on the depth of the forest, with the induction step:

$$at = (a\Box)(\Box + t)0 \sim (\Box + t)(a\Box)0 = a + t .$$

In a similar way, the root of a tree can be swapped with any other node. Since $pq \sim qp$ implies $s + t \sim t + s$, by taking $p = \Box + s$ and $q = \Box + t$, the labels can be ordered in any way. Finally, duplicates can be removed by $pp \sim p$. ■

Before we proceed to the next class, we remark on how the identities give an effective characterization. Take for instance the identity $pq \sim qp$. How do we know if this identity is satisfied by a regular tree language L ? We cannot test all contexts p, q , since there are infinitely many of them.

The solution is that we only need to test one context in each equivalence class, since the syntactic equivalence is a congruence. Therefore, such an identity can be tested in polynomial time based on the syntactic congruence.

The second toy example is the definite languages. A language L is called *definite* if there is some threshold $k \in \mathbb{N}$ such that membership $t \in L$ depends only on the nodes of t that have depth at most k . Note that trees are unranked here, so a definite tree language can have an infinite number of trees of any given depth. For instance, the language “the root has an even number of children” is definite.

Theorem 2

A regular tree language is definite if and only if it satisfies the identity

$$p^\omega 0 \sim p^\omega t \quad \text{if } p \text{ has the hole not at the root.} \quad (1)$$

The characterization above introduces a new feature into the identities, the ω power. Informally, the ω power should be substituted for “a large number”. For instance, identity (1) says that if a context is repeated many times, then its argument is deep in the tree and therefore irrelevant.

The formal definition of the ω power involves idempotence. It is not difficult to show that for any regular language L , there is some number n , such that $p^n \sim_L p^n p^n$ holds for any context p . This number n is denoted by ω_L , with the subscript omitted when L is clear from the context. Note that ω also works for forest concatenation, since the ω -fold concatenation of t is the same as $(t + \square)^\omega 0$.

Proof

If a language is definite, then the identity (1) has to be satisfied. Otherwise, for some context q , and any k , only one of the two trees $qp^{\omega-k}0$ and $qp^{\omega-k}t$, which agree up to depth k , would belong to the language.

For the converse implication, take a tree language that satisfies identity (1). We will show that for some sufficiently large depth $k \in \mathbb{N}$, any modification of a tree below depth k does not affect membership in the language. In other words, we need to show that if a context p has the hole at depth at least k , then $pt \sim p0$ holds for any forest t . Let then p be a context with the hole at depth at least k . The result will follow from the identity in the statement if we manage to decompose this context as $p \sim q_1 q^\omega q_2$. By assumption on p , there is a decomposition $p = p_1 \cdots p_k$, such that none of the contexts p_i has the hole at the root. Since there is a finite number of equivalence classes under \sim , Ramsey’s theorem says that if k is sufficiently large, then there are some $i < j < l$ in $1, \dots, k$ with

$$p_1 \cdots p_{j-1} \sim p_j \cdots p_{l-1} \sim p_1 \cdots p_{l-1} .$$

Let $q = p_1 \cdots p_{j-1}$. By the above, we have

$$q \sim p_1 \cdots p_{l-1} = q \cdot p_{j-1} \cdots p_{l-1} \sim qq .$$

In particular, we have $q \sim q^\omega$. ■

Apart from introducing the ω power, identity (1) also restricts the contexts to a certain subclass, in this particular case contexts that do not have the hole at the root. Does this pose a problem for effectiveness of the characterization? Fortunately, even such extended identities can be effectively checked: using the multiplication table and the mapping α described in Section 2, we can use a fix-point algorithm to determine the equivalence classes that contain a context

with the hole not at the root. So the characterization given by (1) is an effective characterization.

The last toy example concerns tree languages with a finite number of trees. In this case, we cannot hope for characterizations stated in terms of identities, because the class is not closed under complementation. The reason is that any tree language has the same syntactic congruence as its complement, and therefore also satisfies the same identities.

Theorem 3

A regular tree language is finite if and only if it contains no tree of the form $qp^\omega s$, with p nonempty.

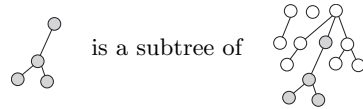
The proof of the above theorem follows the same lines as in Theorem 2. Note that a nonempty context may have the hole at the root, this type of context is needed to prove the above theorem for languages with unbounded node outdegree.

3. LOOKING FOR PATTERNS IN TREES

In this section, we give effective characterizations for three classes of languages. In each case, the language class is given by boolean combinations of clauses which examine if the tree contains a given pattern. The patterns discussed are: a subtree, a piece and a maximal path.

Frontier testable languages.

Previously, we have talked about definite languages, which only depend on the prefix of the tree up to some given depth $n \in \mathbb{N}$. What about suffixes? It is not obvious what the suffix of a tree is. One idea is to look at the n -frontier, which is the set of subtrees with at most n nodes. A *subtree* is a tree obtained by picking a new root, and removing nodes not below that new root.



A language is called *frontier testable* if for some $n \in \mathbb{N}$, membership in the language depends only on the n -frontier of a tree. Note that we do not count the subtrees in the frontier, as a multiset would, but just test which ones are present, and which ones are not. An equivalent definition is that a language is frontier testable if it is a finite boolean combination of languages of the form “trees that contain s as a subtree”.

Any finite tree language is frontier testable. For instance, the (singleton) tree language “one node with label a ” is the same as “no subtree with at least two nodes” intersected with “at least one subtree with label a ”.

Frontier testable languages have the distinction of being the first non-trivial class of tree languages to get an effective characterization, in a paper of Wilke [32]. The characterization of [32] was stated for ranked trees, while below we present the adaptation to unranked trees.

Theorem 4

A regular tree language is frontier testable if and only if it satisfies the following identities.

$$a(s + p^\omega t) \sim s + p^\omega t \quad (2)$$

$$p^\omega t + s + u \sim u + s + p^\omega t \quad (3)$$

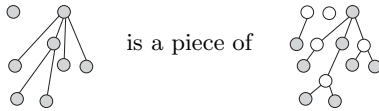
$$p^\omega t + s \sim p^\omega t + s + s \quad (4)$$

In the above, the context p must be nonempty, while a is a context with an a -labelled root and the hole below.

We only roughly sketch the proof of the theorem above. The first identity says a node a with many descendants (a forest $p^\omega t$ has many nodes) can be removed without affecting the frontier. The second identity says that the order of trees in a forest with a lot of nodes is not important. The third identity says that duplicates of trees can be added or removed in any forest with a lot of nodes. The key point in the proof is that for any fixed n , any two forests with the same frontier can be transformed into each other by repeatedly applying the above identities.

Piecewise testable languages.

We now move to a different type of pattern, called a *piece*. We say a forest s is a piece of a forest t , written $s \preceq t$ if s can be obtained from t by removing nodes. In other words, there is an injective mapping from nodes of s to nodes of t that preserves the lexicographic and descendant ordering and the labels.



A tree language is called *piecewise testable* if it is a finite boolean combination of languages of the form “trees that contain forest s as a piece”.

Over words, the piece relation corresponds to taking a (not necessarily connected) subword. Piecewise testable word languages were studied by Simon [26], who discovered that a word language is piecewise testable if and only if its syntactic monoid is \mathcal{J} -trivial. This is one of the fundamental results in algebraic language theory.

The theorem below extends Simon’s result to trees. The statement uses the piece relation on contexts, which is defined as for trees, but with the added requirement that the injective mapping preserves the hole.

Theorem 5 ([6])

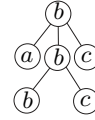
A regular tree language is piecewise testable if and only if it satisfies the identity

$$qp^\omega \sim p^\omega \sim p^\omega q \quad \text{for } q \preceq p \quad (5)$$

The idea in the “only if” is that for any fixed $n \in \mathbb{N}$, the contexts qp^ω , p^ω and $p^\omega q$ all have the same pieces of size n , and can therefore be used interchangeably. The “if” proof requires a complicated combinatorial argument.

Path testable languages.

The last type of pattern we consider, paths, makes no sense for words and is unique to trees. The idea is to associate with each (maximal) path in a tree the sequence of labels on the path (say, from root to leaf). For instance, the maximal paths in



are ba, bbb, bbc and ba . A tree language over an alphabet A is called *path testable* if it is a finite boolean combination of languages “trees where some maximal path in the tree belongs to the (regular) word language $K \subseteq A^*$ ”.

For instance, the tree language “some node has label a ” is path testable, since a tree belongs to this language if and only if some maximal path belongs to the language A^*aA^* . However, the forest language “the forest has two nodes with label a ” is not path testable, since the forests a and $a + a$ both have the same maximal paths, but only one belongs to the language.

Theorem 6 ([8])

A regular tree language is path testable if and only if it satisfies the identities

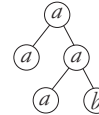
$$s + t \sim t + s \quad s + s \sim s \quad (6)$$

$$p(s + t) \sim ps + pt \quad (7)$$

Proof

The “only if” part is clear, since both sides in each identity have the same maximal paths. For the converse implication, one shows that using the above identities, any two trees can be transformed into each other as long as they have the same maximal paths. ■

Path testable languages are especially interesting for binary trees, so for a moment we make an exception and consider binary trees instead of unranked ones. For binary trees, we modify the notion of path a bit: the path not only says for each node what its label is, but it also has a bit indicating if a node is a right child. For instance, in the tree



the three maximal paths are

$$(a, 0)(a, 0) \quad (a, 0)(a, 1)(a, 0) \quad (a, 0)(a, 1)(b, 1) .$$

Therefore, a property of a path in a binary tree can be identified with a word language L over an alphabet $A \times \{0, 1\}$. Any finite set of binary trees is path testable, since the set of paths (with the child number indicators) uniquely determines a tree; this not the case for the languages in Theorem 6, since any nonempty language that satisfies the identity $s + s \sim s$ is infinite.

Why do we include the child number for binary trees? The reason is that these path-testable languages are closely connected to top-down deterministic tree automata. One can easily show that regular language of binary trees is recognized by a top-down deterministic tree automaton if and only if it is universally path testable, i.e. a language of the form “all maximal paths belong to K ”, for some regular word language K over $A \times \{0, 1\}$. In particular, a regular language L of binary trees is recognized by a top-down deterministic tree automaton if and only if L is the same as the language

$$L' = \{t : \text{each maximal path in } t \text{ appears in some tree } s \in L\} .$$

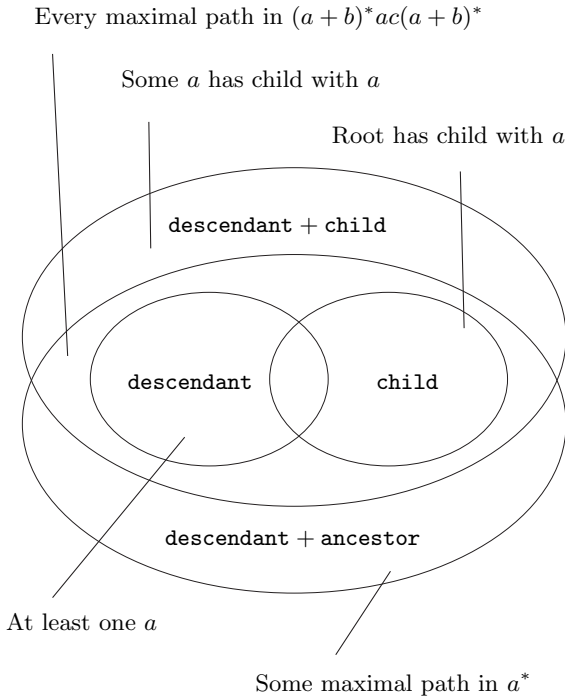


Figure 1: Fragments of CoreXPath

This observation gives an algorithm deciding if a tree language is recognized by some deterministic top-down automaton: compute L' then test if $L = L'$. Is there a similar effective criterion for boolean combinations of such automata, i.e. arbitrary path testable languages? This is an open problem:

Problem 1 Decide if a given regular language L of binary trees is path testable with child numbers or, equivalently, if it is a finite boolean combination of languages recognized by deterministic top-down automata.

We end our digression into binary trees. From now on all trees will be unranked again.

4. CORE XPATH AND FRAGMENTS

Some of the most successful work on effective characterizations has been devoted to fragments of Core XPath, introduced in [14]. The statements in the papers cited below are usually in terms of temporal logic (e.g. EF is written instead of descendant), but here we rephrase the results in the language of Core XPath to provide a common framework. We will omit the word Core and simply say XPath from now on.

The basis of XPath is a set of eight *axes*, which describe spatial relationships between nodes in a tree. The first four axes are

child parent nextSibling prevSibling .

In a given tree, an axis specifies a binary relation on tree nodes in the natural way (the next sibling is the next one to the right; there is at most one next sibling). The remaining four axes are obtained by applying transitive closure to the first four:

descendant ancestor right left .

The transitive closure above is without reflexivity, so for instance descendant is for proper descendants. XPath is usually given with more axes, but the others can be defined in terms of the above eight.

The axes are used to inductively build formulas of XPath. The base formulas of XPath are label tests, i.e. each label $a \in A$ is treated as a formula that selects nodes with label a . Furthermore, if X is an axis and φ is a formula of XPath, then $X\varphi$ is a formula of XPath, which selects nodes that can be connected to a node satisfying φ via the axis X . For instance, the formula child a selects nodes with an a child. Finally, formulas of XPath admit boolean combinations. For instance the formula

$$b \wedge \text{right } a \wedge \text{left } a$$

selects a node that has label b , but all its other siblings have label a .

We say a formula of XPath is true in a tree if it selects the root. Unfortunately, we do not know if there is an effective characterization of tree languages that can be defined in XPath.

Problem 2 Decide if a tree language can be defined in XPath (with all axes).

However, there are effective characterizations for fragments of XPath with reduced sets of axes. The rest of this section is devoted to these fragments.

Child.

As a warm-up, we begin with the fragment that is only allowed to use the child axis. This fragment is better known as *modal logic*. In modal logic, child φ is written as $\diamond\varphi$ and $\neg\text{child}\neg\varphi$ is written as $\square\varphi$.

With only the child axis, we can only recognize definite languages, since a formula with k nested child operators can only inspect nodes up to depth k . However, not all definite languages can be defined. A simple induction proof shows that languages definable with only the child must be *bisimulation invariant*, i.e. satisfy the identities

$$s + s \sim s \quad s + t \sim t + s. \quad (8)$$

It is not difficult to show that this is the only requirement:

Theorem 7

A regular tree language is definable in the fragment of XPath with the child axis (i.e. modal logic) if and only if it is bisimulation invariant and definite.

Descendant.

We now move to a more interesting fragment, where instead of the child axis we can only use the descendant axis. This fragment, also known as EF, or Gödel-Löb logic, can look arbitrarily deep into the tree. For instance, the formula $a \vee \text{descendant } a$ holds in trees with at least one node labeled by a . For the moment, we only allow descendant and not child; the fragment with both axes will be considered later on.

Theorem 8

A regular tree language is definable in the fragment of XPath with the descendant axis if and only if it is bisimulation

invariant and satisfies the identity

$$ps \sim ps + s . \quad (9)$$

What is remarkable about the above theorem is that none of the identities use the ω power. This is unlike the word case, where the identity is

$$u(wv)^\omega \sim uv(wv)^\omega ,$$

for u a nonempty word, see [33].

Actually, the identity $s + s \sim s$ in bisimulation invariance is redundant, since it follows from (9) by taking p to be the empty context. The first version of Theorem 8 was proved in [7] for binary trees. However, since binary trees do not lend themselves easily to expressing properties like $s + s \sim s$, both the statement and proof in [7] were cumbersome. Once languages of unranked trees are considered, the situation is greatly simplified, see [8] for a proof.

Note that the fragment above uses the proper descendant. Using the non-proper version—i.e. descendant or self—gives less expressive power. Note that this does not say anything about the difficulty of giving an effective characterization. A less expressive formalism may be harder to characterize. Effective characterizations of XPath with non-proper descendants were given independently by [12] and [34] (both results are for ranked trees).

Child and descendant.

We now move to the fragment of XPath where both the axes **child** and **descendant** can be used together.

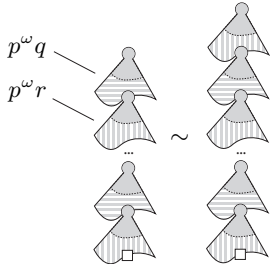
Theorem 9

A tree language is definable in the fragment of XPath with the **child** and **descendant** axes if and only if it is bisimulation invariant and satisfies the identity

$$(p^\omega qp^\omega r)^\omega \sim p^\omega r(p^\omega qp^\omega r)^\omega \quad (10)$$

for p a context where the hole is not at the root.

In the above identity, $p^\omega q$ and $p^\omega r$ should be interpreted as two contexts that have a very long common prefix. The identity says that if two such contexts are alternated sufficiently often, it is no longer relevant which one is chosen as the root.



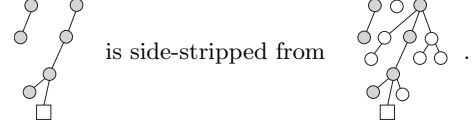
The result above was presented in [7] for ranked trees. The proof for unranked trees follows along similar lines.

Descendant and ancestor.

The axes **child** and **descendant** used in the three fragments above can go only down into the tree. In particular, only the subtree below x is relevant when determining if node x is selected by a formula. This will no longer be true in

the fragment below, where we use the **ancestor** axis. Recall that when defining a tree language, a formula is evaluated at the root, so it makes no sense to consider the fragment of XPath with only the **ancestor** axis.

In the theorem below we use the notion of side-stripping. This is the process of removing subtrees in a context that are siblings of an ancestor of the hole, or siblings of the hole itself. For instance,



Formally, *side-stripping* is defined by the following rules. For any two forests s, t , the context $s + \square$ (respectively, $\square + s$) is side-stripped from a context of the form $t + s + \square$ (respectively, $\square + s + t$). If p' is side-stripped from p , then for any context q , the context $p'q$ (respectively, qp') is side-stripped from pq (respectively, qp).

Theorem 10 ([4])

A regular tree language is definable in the fragment of XPath with the **descendant** and **ancestor** axes if and only if it is bisimulation invariant and satisfies the identities

$$(pq)^\omega \sim (pq)^\omega p(pq)^\omega \quad (11)$$

$$(pq)^\omega \sim (pq)^\omega (p'q')^\omega \quad (12)$$

if p' (resp. q') is side-stripped from p (resp. q).

5. FIRST- AND SECOND-ORDER LOGICS

We now move from CoreXPath to first-order and second-order logics. This is the setting where probably the best known and most important classes of regular languages can be found.

The most general logic used here is *monadic second-order logic* (MSO). A formula of this logic can quantify over individual nodes of the tree (first-order quantification), and over sets of nodes in the tree (monadic second-order quantification). It uses predicates (the logic name for axes) to test the labels of nodes and compare their spatial relationships. For instance, the formula

$$\begin{aligned} \exists x x \in X \wedge a(x) \\ \exists X \quad \forall x \forall y \text{ child}(x, y) \wedge x \in X \Rightarrow y \in X \\ \forall x x \in X \Rightarrow a(x) \vee b(x) \end{aligned}$$

says that there is some set of nodes X (capital letters are used for set variables), which contains a node x with label a (lower-case letters are used for node variables), is closed under children, contains an a label and only contains a, b labels. Equivalently, there is some node with label a that only has b descendants (this property is not equivalent to $\exists x a(x)$ when the alphabet is a, b, c). By using the descendant predicate $>$ instead of the child relation, we could express the above without quantifying over sets

$$\exists x a(x) \wedge \forall y y > x \Rightarrow b(y) .$$

A formula that does not quantify over sets, such as the one just above, is called a formula of *first-order logic*. This section is devoted to studying fragments of monadic second-order logic and fragments of first-order logic.

Of course the expressive power of a logic depends on the predicates (i.e. axes) used. We will have a *label predicate* $a(x)$ for each letter a of the alphabet, this predicate holds if the label of node x is a . The other predicates that will appear are the same as the axes in XPath, and the lexicographic order (which is usually also considered an axis in XPath, under the name of document order). Since all logics below use label predicates and node equality, we will not mention these when specifying the allowed predicates. For instance first-order logic with the ancestor relation refers to formulas that can quantify over nodes (but not sets of nodes), and can use the label predicates, node equality, and the descendant predicate.

Before we present the results on trees, we do a brief detour into word languages. We cite the Schützenberger theorem, and use it to argue that first-order logic is the most important class of regular languages, albeit one which still does not have an effective characterization for trees.

The reason why we talk about monadic second-order logic is that it captures exactly the regular languages. This fundamental observation was made (for word languages) independently by Büchi [9], Elgot [11] and Trakhtenbrot [31].

Theorem 11

A word language is regular if and only if it can be defined in monadic second-order logic, with the successor predicate.

Note that we could also use the order predicate instead of the successor, since a position x is after position y if and only if x is contained in every set that contains y and is closed under successors.

Theorem 11 sets the stage for numerous classes of regular languages. These classes are obtained from monadic second-order logic by restricting the patterns of quantification, the predicates allowed, or both.

The gold standard for effective characterizations is Schützenberger’s Theorem [25], which says that star-free word languages correspond to aperiodic monoids. We present this theorem below in an expanded version, which includes two other theorems, due to McNaughton and Papert [19] (equivalence of first-order logic and star-free languages) and Kamp [16] (equivalence of LTL and first-order logic).

Theorem 12

The following are equivalent for a regular word language L .

1. L is definable in first-order logic with the order predicate.
2. L is definable in linear temporal logic LTL.
3. L is star-free, i.e. can be defined by a regular expression without the Kleene star, but with complementation.
4. The syntactic monoid of L is aperiodic, i.e. the identity $v^\omega \sim v^{\omega+1}$ holds for every word v .

The heart of the theorem is the equivalence with item 4. Since one can effectively calculate the syntactic monoid of a regular word language, and then check if it is aperiodic, it follows that all the first three language classes have an effective characterization.

Note that equivalence of aperiodic monoids with first-order logic follows quickly from Schützenberger’s Theorem: a star-free expression can easily be captured by first-order

logic, while an Ehrenfeucht–Fraïssé game shows that a first-order definable language must have an aperiodic syntactic monoid. Bringing LTL into the loop requires more work, basically showing that languages definable in LTL are closed under concatenation.

One of the principal open problems in the study of tree logics is finding a tree generalization of the above theorem. A first question, of course, is what are the tree counterparts of the language classes described above. Probably the most canonical extension is for first-order logic. Here we will be most interested in first-order logic with the descendant predicate, but one can also consider other sets of predicates.

There is evidence supporting the choice of first-order logic. A notion of star-free regular expression can be developed for trees that matches the expressive power of first-order logic [3]. First-order logic and the star-free expressions are also matched by a branching extension of LTL, similar to CTL*, which was first introduced for binary trees in [15], and generalized to unranked trees in [1].

But what about the last condition in Theorem 12, on aperiodic monoids, which makes the characterization effective? Aperiodicity on its own is a necessary, but not sufficient condition. Finding the correct condition is the principal open problem mentioned in this paper:

Problem 3 Decide if a given regular language can be defined in first-order logic with the descendant order.

This is just one of the versions of the problem, others are obtained by modifying the set of predicates allowed (but keeping the descendant), or passing to a temporal logic syntax like CTL*.

In the rest of this section, we discuss in more detail the various fragments of monadic second-order logic, and what is known about their effective characterizations.

Second-order logics on trees.

This section is devoted to fragments of monadic second-order logic, which still require use of set variables.

Theorem 11 extends to unranked trees, just as it extends to other objects (e.g. infinite words and infinite trees).

Theorem 13

A tree language is regular if and only if it can be defined in monadic second-order logic, with the child and next-sibling predicates.

Again, there is some freedom in the choice of predicates. The child predicate could be replaced by the descendant predicate, and the next-sibling by any of the other axes that move to siblings.

In Theorem 13, the logic uses both the child and next-sibling predicates. What if drop the next-sibling predicate? If we only have the child predicate, we can no longer reason about the order between siblings, so the languages defined will be commutative, i.e. satisfy $s + t \sim t + s$. Does the converse implication hold? The answer is no: the language “the root has an even number of children” is clearly commutative, but cannot be defined in monadic second-order logic with only the child predicate. What do we need to add to monadic second-order logic with the child predicate to get exactly the commutative tree languages? The answer is modulo counting, which is defined as a new quantifier

$\exists^n x. \varphi(x)$ that makes a formula true if the number of nodes x satisfying $\varphi(x)$ is divisible by n . Note that the formula $\varphi(x)$ may have free variables (possibly set variables) other than just x , which can be used as parameters.

Theorem 14 ([10])

A regular tree language is definable in monadic second-order logic with the child predicate and modulo counting if and only if it satisfies the identity

$$t + s \sim s + t .$$

Furthermore, if the language also satisfies

$$\omega \cdot t \sim \omega \cdot t + t ,$$

then modulo counting is not needed.

Monadic second-order logic can quantify over arbitrary sets of nodes. What if we restrict quantification to sets of a special form? There are two important logics of this type. The first is *antichain logic*, where set quantification is restricted to *antichains*, i.e. sets of nodes pairwise incomparable with respect to the descendant predicate. The second is *chain logic*, where set quantification is restricted to *chains*, i.e. sets of nodes pairwise comparable with respect to the descendant predicate.

We begin our discussion with antichain logic.

Over trees where each non-leaf node has at least two children, antichain quantification has the same power as full set quantification, so antichain logic captures exactly the regular languages, as shown in [23]. The general idea is to encode a non-leaf node x as the rightmost leaf below the leftmost child of x . This gives a one-to-one correspondence between non-leaf nodes and leaves, so any set of nodes can be encoded as two sets of leaves (which are antichains): one for leaf nodes, and one for non-leaf nodes.

Over unary trees—trees where each non-leaf node has exactly one child—quantifying over antichains is the same as quantifying over single nodes, so antichain logic has the same expressive power as first-order logic.

What about arbitrary unranked trees, which can contain some nodes with one child and some nodes with more than one child? It turns out that the two observations above can be combined: a language of unranked trees can be defined in antichain logic if and only if it behaves like first-order logic on the unary parts of the tree.

Theorem 15

A regular tree language is definable in antichain logic if and only if it satisfies the identity

$$p^\omega \sim p^{\omega+1} \quad \text{for } p \text{ a unary context.} \quad (13)$$

In the above, a unary context is one where all nodes except the hole have exactly one child.

We now turn to chain logic. Chain logic was introduced in [29]. We focus here on chain logic that has only the descendant predicate, although the discussion is similar for most other choices of predicates.

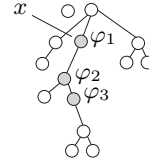
Chain logic properly extends first-order logic. For instance, any path-testable language (such as “some path has an even number of a ’s”) can be expressed in chain logic. The formula uses chains to describe the run of a word automaton on some path.

Chain logic is also strictly weaker than monadic second-order logic (i.e. it does not capture all regular tree languages). Tree languages that cannot be defined in chain logic include “trees with an even number of a ’s”, or the language with boolean expressions described in the next section. Unfortunately, we do not have an effective characterization of chain logic.

Problem 4 Decide if a given regular tree language can be defined in chain logic.

A word can be seen as a unary tree. In this interpretation, any set of nodes is a chain, so chain logic captures all of monadic second-order logic. This way, chain logic can be viewed as one of the tree logics that collapse to monadic second-order logic on words (of course monadic second-order logic on trees is another). There is other evidence supporting the importance of chain logic, such as a characterization in terms of regular expressions [3], or its equivalence with the temporal logic PDL. We briefly present PDL logic here, since it sheds some light on the type of languages that can be defined in chain logic.

As in XPath, a formula of PDL selects a set of nodes in a tree. The inductive definition is as follows. Any label a is a PDL formula, which selects nodes with label a . PDL formulas allow boolean combinations. Finally, PDL formulas allow testing paths against regular word properties, and these word properties may nest smaller PDL formulas. More formally, if Γ is a finite set of PDL formulas, and $L \subseteq \Gamma^*$ is a regular language, then $\mathbf{E} L$ is also PDL formula. This formula selects a node x if there is a path $x_1 x_2 \dots x_k$ with $x_1 = x$ and a word $\varphi_1 \varphi_2 \dots \varphi_k \in \Gamma^*$ such that each node x_i is selected by φ_i . (Recall that consecutive nodes in the path are connected by the child relation.) For instance, the node x below is selected by $\mathbf{E} L$ if the word $\varphi_1 \varphi_2 \varphi_3$ belongs to L .



In the semantics, it is not necessary for the path to reach a leaf, but a PDL formula can test if a node is a leaf, using the formula $\neg \mathbf{E} \text{ true} \cdot \text{ true}$.

It is not difficult to show that for every PDL formula there is an equivalent formula of chain logic. The converse does not hold, because languages definable in PDL are bisimulation invariant, unlike those definable in chain logic. But this is only a superficial difference, which disappears once we extend PDL by operators of the form “at least k children are selected by φ ”, for all $k \in \mathbb{N}$. In particular, over binary trees PDL is equivalent to chain logic [15], since the new operators are redundant for binary trees.

First-order logics.

In this section, we talk about formulas of first-order logic, where variables can quantify only over individual nodes, and not sets of nodes.

Using a standard Ehrenfeucht-Fraïsse game, one can show that no formula of first-order logic with descendant can define the language “some leaf has even depth”. The idea is that a formula with n quantifiers cannot tell the difference

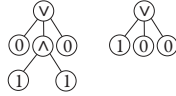
between two unary trees of depths 2^n and $2^n + 1$, respectively. Therefore, first-order logic forms a proper subclass of all regular languages. This subclass is very robust, and may well be considered the most important subclass of regular tree languages. As we already mentioned, first-order logic is not known to have an effective characterization, for any set of predicates that includes the descendant.

Proposition 16 A regular tree language definable in first-order logic must satisfy the identity

$$p^\omega \sim p^{\omega+1} . \quad (14)$$

The above result holds for any set of predicates taken from axes in CoreXPath. Identity (14) is equivalent to saying that the syntactic equivalence on contexts gives an aperiodic monoid. In particular, if the language contains only unary trees, then the above condition is also necessary.

However, in general, condition (14) is only necessary, and may not be sufficient. Consider, for instance, the set of trees over an alphabet $\vee, \wedge, 0$ and 1 that contains the well formed boolean expressions that evaluate to 1 , such as



This set of trees is not definable in first-order logic, although it satisfies a stronger identity than (14), namely $pp \sim p$.

We now give another example. If a language is definable in first-order logic with the descendant order, then it is commutative, i.e. satisfies $s + t \sim t + s$. The reason is that the logic can only compare nodes vertically, so it has no way of comparing the order between siblings. Now take a language L that is defined in a richer signature, say the descendant order and the next-sibling, which also happens to be commutative. Is L necessarily definable by using only the descendant order? The answer to this question is negative.

Proposition 17 A commutative tree language definable in first-order logic with descendant and next-sibling need not be definable with just the descendant.

Proof

Let L be the set of binary (i.e. each node has two or zero children) trees where some leaf has even depth. This language is clearly commutative. Using the zigzag trick described in the introduction, it can be defined in first-order logic using the descendant order and next-sibling predicates (the next-sibling is used to talk about left and right children). Using an Ehrenfeucht-Fraïsse game, one can show that a first-order formula with n quantifiers that only uses the descendant predicate cannot tell the difference between two complete binary trees, one of depth 2^n and the other of depth $2^n + 1$. ■

The language in the proof above also falsifies another conjecture, as shown below.

Proposition 18 A tree language definable in chain logic that satisfies (14) need not be definable in first-order logic with descendant.

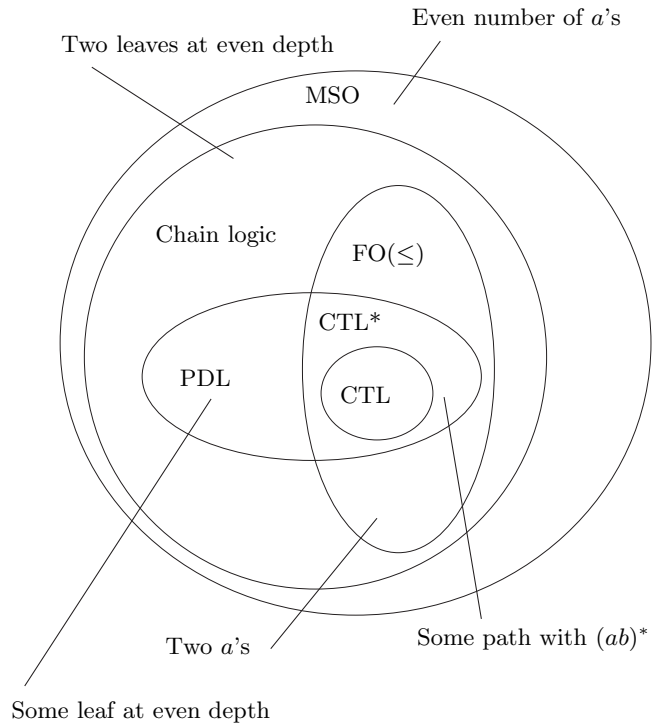


Figure 2: Expressive power of tree logics.

Proof

We take the same language L as in Proposition 17. The only non-obvious part is that L satisfies (14), which seems unlikely for a language involving parity. The catch is that the trees need to be binary. Take then a context p . The interesting case is when the hole in p is at odd depth, say it is a child of the root. Either the hole has a sibling, or there are two root nodes in p , since otherwise p^2, p^3, \dots would be equivalent as “error” contexts. (An “error” context is one that cannot appear inside any tree in the language, in this case due to a node with only one child.) But then the contexts p^2, p^3, \dots would all be equivalent, since each would have leaves at both odd and even depth. ■

First-order logic with child.

While an effective characterization of first-order logic with descendant is a widely open problem, there has been much more success for logics without the descendant predicate. The most notable example is first-order logic with the child predicate.

If only the child predicate is allowed, then first-order logic can only express “local” properties. By the Gaifman Locality Theorem [13], first-order logic with the child predicate captures exactly the *locally threshold testable* languages. A locally threshold testable language is a finite boolean combination of languages of the form “the root is selected by φ ” or “at least k nodes are selected by φ ”, where the *threshold* k is a natural number, and the *local property* φ is a formula of XPath using only the `child` axis. A typical property that can be defined in this logic is “at least five nodes have label a and exactly three children”. On the other hand, the property “some leaf has all ancestors with label a ” cannot be defined, at least as long as the alphabet is a, b .

Benedikt and Segoufin [2] give an effective characterization of first-order logic with the child predicate, or equivalently, of the locally threshold testable languages. The idea behind their identities is that locality ensures that parts of the tree can be swapped, as long the swapping preserves the local neighborhoods. In the theorem below, we say that two forests agree up to depth n if they are equal once nodes of depth greater than n are removed. A similar definition is used for contexts.

Theorem 19

A regular tree language can be defined in first-order logic with the child predicate if and only if there is some $n \in \mathbb{N}$ such that the following identities hold.

$$p^\omega \sim p^{\omega+1} \tag{15}$$

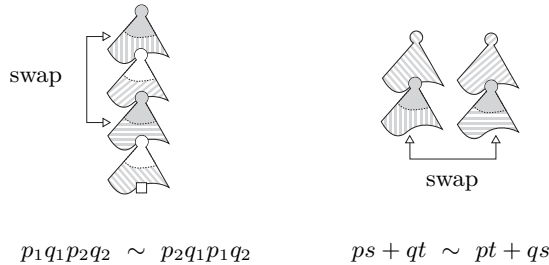
$$s + t \sim t + s \tag{16}$$


$$ps + qt \sim pt + qs \tag{17}$$

$$p_1q_1p_2q_2 \sim p_2q_1p_1q_2, \tag{18}$$

In (17), the pair of forests s, t must agree up to depth n . Likewise in (18), for the contexts pairs p_1, p_2 and q_1, q_2 .

The identities (17) and (18), which account for the swapping, are illustrated below.



 large common prefixes

We try to justify identity (18), which is on the right in the above illustration. We only focus on its correctness, i.e. proving that a tree language L definable in first-order logic with the child predicate must satisfy (18). As mentioned above, the language must be locally threshold testable. Let now $n \in \mathbb{N}$ be the size of the largest local property φ in the locally threshold testable expression defining L . Assume also that the common prefix of the forests s and t , depicted as the grey part in the illustration, has depth at least n . If we swap the two forests s and t , then none of the local properties will be affected, so swapping does not affect membership in L .

An open problem is finding an effective characterization for first-order logic with both the child and next-sibling predicates.

Low levels of the quantifier alternation hierarchy.

So far we have considered logics that were obtained by either limiting the type of predicates available, or the type of sets available in second-order quantification. In this section we will limit the patterns of quantification.

In the *quantifier alternation hierarchy*, each level counts the alterations between \forall and \exists quantifiers in a first-order formula in prenex normal form (i.e. when all quantifiers are pulled to the beginning of the formula). Formulas with $n - 1$ alternations are called Σ_n if they begin with \exists , and Π_n if they begin with \forall . For instance, the property “at least two nodes” can be defined by a Σ_1 formula $\exists x \exists y. x \neq y$, while the property “root has label a ” can be defined by the Σ_2 formula $\exists x \forall y. a(x) \wedge (y \geq x)$. Note that consecutive quantifiers of the same type do not contribute to the complexity, like the two existential quantifiers $\exists x \exists y$ in the first example.

The quantifier alternation hierarchy can be considered for signatures with different predicates, but we focus here on just the descendant (for trees), or the left-to-right order (for words). Note that although the child (respectively, successor) relation can be defined in terms of the descendant (respectively, order), the definition requires a \forall quantifier, which may affect the position in the hierarchy.

The quantifier alternation hierarchy has been intensively studied for word languages. There is a corresponding hierarchy of star-free regular expressions on words, which matches the quantifier alternation hierarchy level by level [28]. There are effective characterizations for levels leading up to Σ_2 and Π_2 , including two intermediate levels: finite boolean combinations of Σ_1 , and the intersection $\Sigma_2 \cap \Pi_2$. Each of these levels has several—sometimes surprising—alternative descriptions. For instance, languages that belong to both Σ_2 and Π_2 are the same as those definable in two-variable first-order logic with the order predicate, and also the same as those recognized by two-way deterministic ordered automata. A more detailed study of the intersection $\Sigma_2 \cap \Pi_2$ can be found in [27], while [21] offers a broader look on the quantifier alternation hierarchy and other logics for words.

The quantifier alternation hierarchy has also been studied for tree languages. Just like word languages, tree languages also have a regular expression hierarchy that corresponds level by level to the quantifier alternation hierarchy [3]. We also have effective characterizations for some of the lower levels, which are cited below.

In the discussion below, we focus on formulas where only the descendant predicate is allowed, although results have been presented for other sets of predicates, too.

We begin with level Σ_1 . Since this level is not closed under boolean combinations, we cannot hope for identities, but only for an implication. The characterization is straightforward: a tree language is definable in Σ_1 if and only if it is commutative—since we only have the descendant—and closed under adding new nodes.

Theorem 20

A regular tree language is definable by a formula of Σ_1 with descendant if and only if it is commutative and satisfies the implication:

$$ps \in L \Rightarrow pqs \in L.$$

Proof

The “only if” part is obvious. For the “if” part, take a commutative language L that satisfies the above implication. Let S be the set of minimal (with respect to \preceq) trees in L . By upward closure under \preceq , a tree belongs to L if and only if it contains a piece from S . The latter property can be expressed in Σ_1 , since S is commutative and finite (by a pumping argument). ■

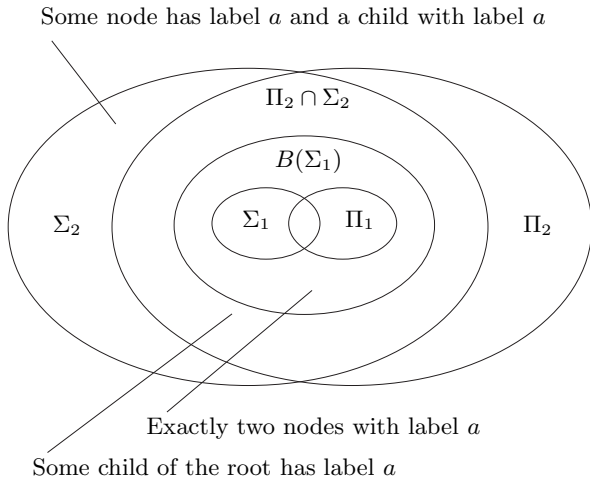


Figure 3: Low levels of the alternation hierarchy.

The characterization for Π_1 is obtained by reversing the implication above. Between levels 1 and 2 of the quantifier alternation hierarchy we find the two intermediate mentioned previously.

It is easy to see that the finite boolean combinations of Σ_1 sentences are the same as the piecewise testable tree languages, which were characterized in Theorem 5.

As mentioned above, for word languages the intersection of Σ_2 and Π_2 is a remarkably robust class. The situation for tree languages is less clear, since some of the descriptions that are equivalent for words diverge over trees (e.g. two-variable first-order logic is no longer the same as the intersection of Σ_2 and Π_2). However, we do have an effective characterization.

Theorem 21 ([5])

A regular tree language is definable in both Σ_2 and Π_2 with descendant if and only if it is commutative and satisfies the identity

$$p^\omega \sim p^\omega q p^\omega \quad \text{for } q \preceq p \quad (19)$$

We still do not have an effective characterization of levels Σ_2 or Π_2 . One conjecture is that these classes are captured by replacing the identity in (19) with a one-way implication (left-to-right for Σ_2 and right-to-left for Π_2). An effective characterization of any level above Σ_2 or Π_2 is a widely open problem, even for word languages.

6. AN UNDECIDABILITY RESULT

We conclude this paper with a negative result.

After seeing a long line of logics, each one with a different set of identities, the reader may be tempted to ask: is there a more general approach? Is there an algorithm, which inputs a definition of a logic \mathcal{L} and a language L , and decides if the language L can be defined in the logic \mathcal{L} ? In this section, we show that such a general algorithm does not exist, even for a fairly weak way of describing logics.

We will show that a general algorithm is impossible already for fragments of XPath with the **descendant** axis. To define the notion of a general algorithm, we need a way of representing logics on the input of an algorithm. Below, we will give an abstract definition of a “one-way oper-

ator”. The general idea is as follows: an operator, such as **descendant** φ , takes the set of nodes in a tree that satisfy the formula φ , and says if the root of the tree is selected by the operator. If the operator takes two arguments, such as $E\varphi U\psi$ of CTL (which says “some node is selected by ψ and has all ancestors selected by φ ”), then two sets of nodes—one for φ and one for ψ —are needed to give the result. These two sets can be given by labeling each tree node with a bit-vector $(a_\varphi, a_\psi) \in \{0, 1\}^2$, with a_φ indicating if φ holds, and a_ψ indicating if ψ holds. An *n-ary one-way operator* is represented as a tree language over the alphabet $\{0, 1\}^n$.

For instance, the language representing the unary one-way operator **child** φ is “some child of the root has label 1”, while the language representing to the binary one-way operator $E\varphi U\psi$ is “some node x has label $(0, 1)$ or $(1, 1)$, and all ancestors of x have label $(1, 0)$ or $(1, 1)$ ”.

We will only be interested in operators where the tree languages are regular. There is also a definition of a two-way operator, which can capture e.g. **parent** φ , but we omit it for lack of space.

We now proceed to define the temporal logic induced by a set of one-way operators. Fix an alphabet A . Let L_1, \dots, L_k be tree languages over A and let t be a tree over A . The *characteristic tree* of t under L_1, \dots, L_k is obtained from t by changing the label of each node x of t to a bit vector in $\{0, 1\}^k$, whose i -th bit indicates whether or not the subtree of t rooted in x belongs to L_i . With L_1, \dots, L_k as above, let L be a language representing a k -ary one-way operator. The *nesting of L_1, \dots, L_k inside the operator L* is the tree language over A that contains trees whose characteristic tree under L_1, \dots, L_k belongs to L . Given a set \mathcal{L} of representations of one-way operators, the *temporal logic induced* by \mathcal{L} is the smallest language class that contains all languages of the form “the root has label a ”, and is closed under boolean combinations and nesting inside operators from \mathcal{L} .

With this general definition, one could hope there is an algorithm that inputs a set of representations of one-way temporal operators and a target language L , and answers if L can be defined using the given operators. Unfortunately, such an algorithm does not exist, even if we only use operators that can be defined in XPath with only the **descendant** axis:

Theorem 22 ([24])

The following problem is undecidable:

*Input: Tree languages L , and L_1, \dots, L_n , all definable in XPath with only the **descendant** axis.*

Question: Is L definable in the temporal logic induced by L_1, \dots, L_n ?

In the above problem there is even a choice of operators L_1, \dots, L_n such that the problem stays undecidable with only one input, L . In light of the above result, it seems that we are resigned to giving a characterization for each logic on a case-by-case basis.

Finally, while we are on the subject of composing one-way operators, we mention that no finite set of one-way temporal operators can capture all the path testable languages.

Theorem 23

No finite set of one-way operators induces a logic that defines all the languages $L_n = \text{“some path in } (a^n b)^ \text{”}$, for $n \in \mathbb{N}$.*

One consequence of the above theorem is that first-order logic, which can define all the languages L_n , cannot be induced by a finite set of one-way operators. This is in contrast with words, where the one-way operator $\varphi U \psi$ of LTL captures all of first-order logic. Note that Theorem 23 does not say that first-order logic cannot be captured with a finite number of operators, it only says that it cannot be captured by a finite number of one-way operators. As shown by Marx [18], first-order logic is captured by adding extending CoreXPath with a form of transitive closure.

7. REFERENCES

- [1] Pablo Barceló and Leonid Libkin. Temporal logics over unranked trees. In *LICS*, pages 31–40, 2005.
- [2] Michael Benedikt and Luc Segoufin. Regular tree languages definable in FO. In *STACS*, pages 327–339, 2005. A corrected version can be found on <http://www-rocq.inria.fr/segoufin/PAPIERS>.
- [3] Miłkołaj Bojańczyk. Forest expressions. In *CSL*, pages 146–160, 2007.
- [4] Miłkołaj Bojańczyk. Two-way unary temporal logic over trees. In *LICS*, pages 121–130, 2007.
- [5] Miłkołaj Bojańczyk and Luc Segoufin. Tree languages definable with one quantifier alternation. Submitted.
- [6] Miłkołaj Bojańczyk, Luc Segoufin, and Howard Straubing. Piecewise testable tree languages. *LICS*, 2008.
- [7] Miłkołaj Bojańczyk and Igor Walukiewicz. Characterizing EF and EX tree logics. *Theor. Comput. Sci.*, 358(2-3):255–272, 2006.
- [8] Miłkołaj Bojańczyk and Igor Walukiewicz. Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107 – 132. Amsterdam University Press, 2007.
- [9] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.
- [10] Bruno Courcelle. The monadic second-order logic of graphs X: Linear orderings. *Theor. Comput. Sci.*, 160(1&2):87–143, 1996.
- [11] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98:21–52, 1961.
- [12] Zoltán Ésik and Szabocs Ivan. Some varieties of finite tree automata related to restricted temporal logics. *Fundamenta Informaticae*, 82(1-2), 2008.
- [13] Haim Gaifman. On local and non-local properties. In *Proceedings of the Herbrand Symposium, Logic Colloquium '81*, 1982.
- [14] Georg Gottlob and Christoph Koch. Monadic queries over tree-structured data. In *LICS*, pages 189–202, 2002.
- [15] Thilo Hafer and Wolfgang Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *ICALP*, pages 269–279, 1987.
- [16] J. A. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Univ. of California, Los Angeles, 1968.
- [17] Leonid Libkin. Logics for unranked trees: An overview. *Logical Methods in Computer Science*, 2(3), 2006.
- [18] Maarten Marx. Conditional XPath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005.
- [19] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65)*. The MIT Press, 1971.
- [20] Jean-Éric Pin. Logic, semigroups and automata on words. *Ann. Math. Artif. Intell.*, 16:343–384, 1996.
- [21] Jean-Éric Pin. Logic on words. In *Current Trends in Theoretical Computer Science*, pages 254–273. 2001.
- [22] Andreas Potthoff. First-order logic on finite trees. In *TAPSOFT*, pages 125–139, 1995.
- [23] Andreas Potthoff and Wolfgang Thomas. Regular tree languages without unary symbols are star-free. In *FCT*, pages 396–405, 1993.
- [24] Mefodie Rață. A formal reduction of the general problem of expressibility of formulas in the Gödel-Löb provability logic. *Discrete Mathematics and Applications*, 12(3):279–290, 2002.
- [25] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
- [26] Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, 1975.
- [27] P. Tesson and D. Thérien. Diamonds are forever: the variety DA. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500, 2002.
- [28] Wolfgang Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.
- [29] Wolfgang Thomas. Logical aspects in the study of tree languages. In *CAAP*, pages 31–50, 1984.
- [30] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.
- [31] B. A. Trakthenbrot. Finite automata and the logic of monadic second order predicates(russian). *Doklady Akademii Nauk SSSR*, 140:326–329, 1961.
- [32] Thomas Wilke. An algebraic characterization of frontier testable tree languages. *Theor. Comput. Sci.*, 154(1):85–106, 1996.
- [33] Thomas Wilke. Classifying discrete temporal properties. In *STACS*, pages 32–46, 1999.
- [34] Zhilin Wu. A note on the characterization of TL[EF]. *Inf. Process. Lett.*, 102(2-3):48–54, 2007.