Tree Extension Algebras: Logics, Automata, and Query Languages

Michael Benedikt*
Bell Labs

Leonid Libkin§
U. Toronto

Abstract

We study relations on trees defined by first-order constraints over a vocabulary that includes the tree extension relation $T \prec T'$, holding if and only if every branch of T extends to a branch of T', unary node-tests, and a binary relation checking if the domains of two trees are equal. We show that from such a formula one can generate a tree automaton that accepts the set of tuples of trees defined by the formula, and conversely that every automaton over tree-tuples is captured by such a formula. We look at the fragment with only extension inequalities and leaf tests, and show that it corresponds to a new class of automata on tree tuples, which is strictly weaker then general tree-tuple automata. We use the automata representations to show separation and expressibility results for formulae in the logic. We then turn to relational calculi over the logic defined here: that is, from constraints we extend to queries that have second-order parameters for a finite set of tree tuples. We give normal forms for queries, and use these to get bounds on the data complexity of query evaluation, showing that while general query evaluation is unbounded within the polynomial hierarchy, generic query evaluation has very low complexity, giving strong bounds on the expressive power of relational calculi with tree extension constraints. We also give normal forms for safe queries in the calculus.

1 Introduction

Because much of computing practice involves the manipulation of tree structures, computer science abounds in formalisms for describing trees. Tree constraints and monadic-second order logic are two declarative approaches to specifying tree properties, while tree grammars, various flavors of tree automata and tree transducers are examples of more procedural formalisms. Naturally, an extensive literature exists comparing the expressive power of each of these formalisms (see [12, 33]) and for translating between declarative formalisms and their procedural implementations. In particular, work on analyzing specifications of trees plays a significant role in program analysis [3, 24], verification [15, 20, 26], logic and constraint programming [29, 30] and linguistics [14, 19].

Over the last few years, applications in information exchange have appeared that necessitate new tools for synthesizing tree-processing code from a declarative specification. These applications all revolve around XML [1, 35]. In XML, data is naturally modeled as a tree, the access methods and manipulation tools take input in the form of tree transformations or transducers, and the interface specifications give preconditions using a combination of tree grammars and tree constraints. The XML context brings issues to the fore that were not as prominent in many prior applications. Most importantly, one can no longer deal with only properties of a single tree, since databases store and manipulate large sets and relations of trees. A natural aim then is to use a tree constraint language to describe the kind of properties of interest for XML transformation and querying. XML querying could then be seen as constraint solving – a model very much in line with the traditional declarative model for database processing.

The prior literature does consider properties of tree tuples and sets of trees, both in relation to logic programming and program analysis [36, 32] and with respect to database querying [13]. Most of this work revolves around the use of equations and inequations among terms or trees. In these cases the domain of the formulas or constraints is some variation of *term*, or *feature algebra*. Rephrased in the terminology of operations on labeled trees, term algebra corresponds to merging subtrees and extending branches by a single node. Term algebra, however, does not allow one to express the vertical ordering relationships among nodes that are important for many applications. For example, a key component of several XML standards is *path expressions*, which may describe the descendant relation between nodes in a tree. An integrity constraint based on path expressions, for example, might specify that every node labeled a in a tree has a node labeled b as a descendant. Such a property

*Bell Laboratories, 2701 Lucent Lane, Lisle, IL 60532, USA. E-mail: benedikt@research.bell-labs.com.

§Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu. Research affiliation: Bell Labs.

cannot be expressed over term algebra.

In this paper we investigate tree-tuple specifications given in a constraint formalism that includes *extension* relationships between trees: $T_1 \prec T_2$ iff T_1 is an initial subtree of T_2. This is the same as the standard subsumption ordering used for *feature trees* [14, 25]: intuitively, it means that every branch of T_1 is also a branch of T_2.

We deal with the *first-order theory* of this model, as opposed to just its equational theory. One of our key criteria is that the theory be decidable. This makes it impossible to combine the ordering $\prec$ with term algebra operations, since the resulting theory is known to be undecidable [25]. Instead, we introduce operations that allow us to extend trees at the leaves, rather than combine subtrees at the root. We shall call the set of trees with the extension relation (and a number of other operations to be introduced shortly) *tree extension algebra*, and the resulting formulae *tree extension formulae*. To get an idea of the combination of multi-tree constraints and single-tree formulas, we list below several properties that can be expressed in this algebra.

- $\mathit{branch}(T_1, T_2)$: T_1 is a single branch of T_2.
- $\mathit{branch}_i^a(T_1, T_2)$: T_1 and T_2 are single branches, and T_2 extends T_1 in direction i, labeling the leaf by a.
- $ab(T)$: Every node labeled a in T is followed by a node labeled b.
- $\neg\exists T_1\ (\mathit{branch}(T_1, T) \wedge ab(T_1))$: T does not have a branch in which every node labeled a is followed by a node labeled b.

We show that the formalism allows considerable expressive power, is closed under logical operations, and is decidable. After introducing the formalism, the first part of the paper is devoted to the synthesis of automata from formulae. We show that constraints given by tree extension formulae can be solved, with a multi-tree automaton that recognizes the defined collection of tuples being generated as a result. We then consider restrictions of the tree extension algebra, possessing considerable expressive power (in fact, covering all the examples above) and yet having a simpler automaton construction. We present such an restriction, called *primal tree extension algebra*, and a corresponding class of automata, called *splitting automata*. We use these results to show separation between the two algebras.

We then examine tree extension algebras from both the model-theoretic and the complexity-theoretic point of view. What sort of combinatorial objects can be defined within the model, and how does the solution set of a formula $\varphi(\vec{x}, \vec{y})$ vary as the parameter $\vec{y}$ varies? Term algebras are *stable* in the model-theoretic sense (cf. [17]), implying that there is no definable linear order and the fibers of a formula cannot vary arbitrarily (i.e. the *VC-dimension* of definable families is bounded). We show that neither of these is true for tree extension algebra – a linear order can be defined, even in the primal case, and the VC-dimension is unbounded. We also show that conversely there are properties of tree tuples expressible over term algebra that are not expressible via tree extension formulae.

In the second part of the paper we study database-related aspects of the algebras, by looking at *queries* – formulae of the logic extended with free relational symbols. We study the complexity of evaluation of queries (that is, constraint-solving) given a collection of tree tuples as input for each symbol. We consider the complexity in terms of the size of a database (which is typically large). We first prove a quantifier-restriction result, showing that any first-order query can be expressed with quantification restricted to a finite set, definable from the database. Using this, we show that the complexity of query evaluation is essentially PH. The general worst case upper bounds do not allow us to show interesting inexpressibility results in relational calculi with tree extension constraints, but we do obtain such results by placing the complexity of evaluating *generic* queries in a much smaller class, AC^0. This gives us matching expressivity bounds for the pure relational calculus, and relational calculus with tree extension constraints, as far as generic queries are concerned. We also address the question of characterizing *safe* queries, and give a range-restricted form that captures all safe queries with tree extension constraints.

Organization. Section 2 gives notations as well as the formal definitions of the algebras. Section 3 gives the basic decidability and automata-synthesis results. Section 4 introduces a specialized multi-tree automata and gives a synthesis result for the primal tree extension algebra. Section 5 gives model-theoretic and expressibility results on the tree extension formulae. Section 6 introduces the relational algebra corresponding to these structures, and gives normal form, data-complexity, and range-restriction results for them. Section 7 gives conclusions. Due to space limitations, complete proofs are not included; a full version of this paper is available from the authors.

2 Notations

The trees we consider are based on two fixed alphabets: the alphabet for directions Δ of the form $\{1, \ldots, n\}$, and Σ for node labeling. Unless explicitly stated otherwise, we assume $n > 1$. We write $s_1 \leq s_2$ if a string s_1 is a prefix of a string s_2. A *tree domain* is a prefix-closed finite subset D of Δ^*: $s_1 \in D$ and $s_2 \leq s_1$ imply $s_2 \in D$. A *tree* is a pair $T = (D, f)$ where $D \subset \Delta^*$ is a tree domain, and f is a function from D to Σ. We use $\mathrm{dom}(T)$ to denote D.

The set of all trees over $\Delta = \{1, \ldots, n\}$ and Σ is denoted by $\mathrm{Trees}_n(\Sigma)$. Note that $\mathrm{Trees}_1(\Sigma)$ naturally corresponds to Σ^*; we shall say more about this correspondence later.

A node in a tree T is a string $s \in D = \mathrm{dom}(T)$, and $f(s)$ is its labeling. The root is the empty string ϵ, and the leaves are $s \in D$ such that s is not a prefix of any other string in D. The set of leaves of T is called the *frontier* of T and is denoted by $\mathrm{Fr}(T)$.

In the literature, quite often trees are considered over *complete* domains D; that is, for every $s \in D$, either all $s \cdot i, i \leq n$ are in D, or none are. We do *not* make this assumption here. However, we shall often consider trees with complete domains. We define a *completion* of $T = (D, f)$ with respect to a symbol $a \in \Sigma$ as $T^c_a = (D', f')$ where D' is the smallest complete domain that contains D, and $f'(s) = f(s)$ for $s \in D$, and $f'(s) = a$ for $s \in D' - D$.

We now look at the operations (functions, predicates, and constants) on trees in our algebra. The constants are $\epsilon_a, a \in \Sigma$, with domain $\{\epsilon\}$ labeled by a. Unary term construction operators are as follows. Given $i \leq n$ (direction) and $a \in \Sigma$, for $T = (D, f)$, $\mathrm{succ}^a_i(T) = (D', f')$ where $D' = D \cup \{s \cdot i \mid s \in \mathrm{Fr}(T)\}$, and f' extends f to D' by $f'(s \cdot i) = a$ for each $s \in \mathrm{Fr}(T)$.

The basic binary relation – the one that gives the name to the algebra – is the extension order. Given two trees $T = (D, f)$ and $T' = (D', f')$, we write $T \preceq T'$ (T' *extends* T) if $D \subseteq D'$ and f is the restriction of f' to D. Clearly it is a partial order. As usual $T \prec T'$ means $T \preceq T'$ and $T \neq T'$. We denote the greatest lower bound of T and T' by $T \sqcap T'$.

A tree T is called a *branch* if $\mathrm{dom}(T)$ is linearly ordered by the prefix relation, that is, for any $s, s' \in \mathrm{dom}(T)$, either $s \leq s'$ or $s' \leq s$. Sometimes we use lowercase letters to denote branches. If t is a branch and $t \preceq T$, we say that t is a branch of T. If in addition $\mathrm{Fr}(t) \subseteq \mathrm{Fr}(T)$, then t is called a maximal branch of T.

As we will see, first-order formulae over the above functions and predicates give us quite an expressive language. But to capture *all* properties of tree tuples that are implementable by tree automata, we will require an additional operation that allows us to compare trees based only on their domains, ignoring alphabet symbols. For two trees T, T', we write $T \approx_{\mathrm{dom}} T'$ iff $\mathrm{dom}(T) = \mathrm{dom}(T')$.

We now introduce the basic objects of our study. For each $n > 0$, we define the following:

Primal Tree Extension Algebra is the structure having the successor operations and the extension relation:

$$\mathfrak{T}_{\mathfrak{p}} = \langle \mathrm{Trees}_n(\Sigma), \preceq, (\mathrm{succ}^a_i)_{i \leq n, a \in \Sigma}, (\epsilon_a)_{a \in \Sigma} \rangle$$

Tree Extension Algebra is the structure that in addition allows domain comparisons:

$$\mathfrak{T} = \langle \mathrm{Trees}_n(\Sigma), \preceq, (\mathrm{succ}^a_i)_{i \leq n, a \in \Sigma}, , (\epsilon_a)_{a \in \Sigma}, \approx_{\mathrm{dom}} \rangle$$

First-order formulae over $\mathfrak{T}$ are called *tree extension formulae*.

We can show that many of basic tree operations and predicates are definable over $\mathfrak{T}_{\mathfrak{p}}$. There is a formula $\eta(t)$ saying that t is a branch: $\forall x, y\ (x \preceq t \wedge y \preceq t) \rightarrow (x \preceq y \vee y \preceq x)$. We also write $\eta(t, T)$ for $\eta(t) \wedge t \preceq T$ (t is a branch of T) and $\eta_{\max}(t, T)$ for $\eta(t, T) \wedge \neg \exists t'\ (t \prec t' \wedge \eta(t', T))$ (t is a maximal branch of T).

Completions T^c_a are definable as well. Indeed, T^c_a is the smallest, with respect to $\preceq$, tree $T' \succeq T$ such that for any nonmaximal branch t of T', and for each $i \leq n$, either $\mathrm{succ}^b_i(t) \preceq T$ for some $b \in \Sigma$, or $\mathrm{succ}^a_i(t) \preceq T'$. Clearly this is definable over $\mathfrak{T}_{\mathfrak{p}}$.

One can also see that succ^a_i could be defined in a number of different ways, for instance, as extending the leftmost branch, or the rightmost branch, or only extending branches. With each of those operations and $\preceq$ one would be able to define succ^a_i. Furthermore, we can define $T \sqcap T'$ as the greatest lower bound of T and T' in $\preceq$ (which is a tree whose domain is the largest prefix-closed subset of $\mathrm{dom}(T) \cap \mathrm{dom}(T')$ on which f and f' coincide). We can also define a predicate L_a on branches which tests if the leaf is labeled by a: $L_a(t) \equiv (t = \epsilon_a) \vee \exists t'(t' \prec t \wedge \bigvee_i t = \mathrm{succ}^a_i(t'))$.

Complete vs incomplete domains In the literature, most concepts related to regular tree languages and relations are defined for complete domains [12, 33]. To make use of them, we shall need a simple reduction of formulae concerning incomplete domains to formulae over complete domains. With this reduction, we shall continue to deal with trees over arbitrary domains, but we shall be able to use many results from the literature on trees over complete domains.

Let $\perp$ be a symbol not in Σ. Let $\mathrm{Trees}^c_n(\Sigma)$ be the set of trees of the form $T^c_\perp$, where $T \in \mathrm{Trees}_n(\Sigma)$. We extend operations succ^a_i to $\mathrm{Trees}^c_n(\Sigma)$ as follows: if $T' = T^c_\perp$, then $\mathrm{succ}^a_i(T') = (\mathrm{succ}^a_i(T))^c_\perp$. Define structures $\mathfrak{T}^c_{\mathfrak{p}}$ and $\mathfrak{T}^c$ whose universe is $\mathrm{Trees}^c_n(\Sigma)$, and the operations are the same as in $\mathfrak{T}_{\mathfrak{p}}$ and $\mathfrak{T}$, with succ interpreted as above. Then a simple inductive argument shows:

Lemma 1 *Let $\varphi(\vec{T})$ be a formula in the language of $\mathfrak{T}_{\mathfrak{p}}$ (or $\mathfrak{T}$). Then*

$$\mathfrak{T}_{\mathfrak{p}} \models \varphi(T_1, \ldots, T_k) \quad \Leftrightarrow \quad \mathfrak{T}^c_{\mathfrak{p}} \models \varphi((T_1)^c_\perp, \ldots, (T_k)^c_\perp)$$

(respectively $\mathfrak{T} \models \varphi(T_1, \ldots, T_k) \Leftrightarrow \mathfrak{T}^c \models \varphi((T_1)^c_\perp, \ldots, (T_k)^c_\perp)$ *)* □

3 Tree extension algebra and tree automata

In this section we show that sets definable by tree extension formulae are familiar objects: they are regular (recognizable) tree languages/relations. Furthermore, formulae over $\mathfrak{T}$ can be compiled into tree automata, and vice versa: this automata-theoretic characterization makes $\mathfrak{T}$ a natural model to work in.

A set of trees over complete domains is called regular if it is accepted by a tree automaton. Extending this to arbitrary domains, we say that a set $X \subseteq \mathrm{Trees}_n(\Sigma)$ is *regular* if the set $X^c_\perp = \{T^c_\perp \mid T \in X\}$ is accepted by a tree automaton.

We next define regular tree relations, that is, subsets of $\mathrm{Trees}_n(\Sigma) \times \ldots \times \mathrm{Trees}_n(\Sigma)$, following [12]. Let $\Sigma_\perp$ stand for $\Sigma \cup \{\perp\}$. Let $\vec{T} = (T_1, \ldots, T_k)$ be a tuple of trees. We represent such a tuple as a tree $[\vec{T}]$ in $\mathrm{Trees}_n(\Sigma^k_\perp)$. Let $T_i = (D_i, f_i), i \leq k$. Then $[\vec{T}] = (D, F)$ where $D = D_1 \cup \ldots \cup D_k$ and for each $s \in D$, $F(s)$ is an element of $\Sigma^k_\perp$, that is, $F(s) = (a_1, \ldots, a_k)$ in which

$$a_i = \begin{cases} f_i(s) & \text{if } s \in D_i \\ \perp & \text{otherwise} \end{cases}$$

Over complete domains, the notion of recognizability says that the set of trees $[\vec{T}]$ is accepted by a tree automata over the alphabet $\Sigma^k_\perp$. To account for incomplete domains, we say that $X \subseteq \mathrm{Trees}_n(\Sigma)^k$ is *regular* iff the set $\{[\vec{T}]^c_\perp \mid \vec{T} \in X\}$ is regular, that is, accepted by a tree automaton over the alphabet $\Sigma^k_\perp$. Here $\boldsymbol{\perp}$ stands for the k-tuple $(\perp, \ldots, \perp)$.

Theorem 1 *a) For any $k \geq 1$, a subset of $\mathrm{Trees}_n(\Sigma)^k$ is definable by a $\mathfrak{T}$ formula iff it is regular.*

b) A subset of $\mathrm{Trees}_n(\Sigma)$ is definable in $\mathfrak{T}_\mathfrak{p}$ iff it is regular.

Furthermore, for both a) and b), the translations between formulae and automata are effective.

Proof sketch. For a), we show that atomic predicates of $\mathfrak{T}$ are encoded by tree automata, and then use the closure properties. For the other direction, we show how to encode antichain logic [28] over $\mathrm{dom}([\vec{T}]^c_\perp)$ in FO over $\mathfrak{T}$. The encoding, in case of $\mathrm{dom}(T^c_\perp)$, can be done without using $\approx_{\mathrm{dom}}$, which gives us b). $\Box$

Thus, definability of sets of trees in $\mathrm{Trees}_n(\Sigma)$ is the same in $\mathfrak{T}_\mathfrak{p}$ and $\mathfrak{T}$. For relations, however, definability is different, as we shall see in the next section.

Consequences of the automata-theoretic representation First, we can show that in any formula $\varphi(\vec{T})$, quantifiers only need to range over a finite set. Given a tuple $\vec{T} \in \mathrm{Trees}_n(\Sigma)^k$, let $\mathrm{Trees}_n(\Sigma)|_{\mathrm{dom}(\vec{T})}$ be the set of all trees whose domain is a subset of $\bigcup_{T \in \vec{T}} \mathrm{dom}(T)$. By encoding the run of a tree automaton over $\mathfrak{T}$, one can see the following.

Corollary 1 *The finite set $\mathrm{Trees}_n(\Sigma)|_{\mathrm{dom}(\vec{T})}$ is definable from $\vec{T}$ over $\mathfrak{T}$. Furthermore, every formula $\varphi(\vec{T})$ over $\mathfrak{T}$ is equivalent to a formula in which quantifiers range over $\mathrm{Trees}_n(\Sigma)|_{\mathrm{dom}(\vec{T})}$.* $\Box$

The tree automata representation also gives us decidability and lower complexity bounds.

Corollary 2 *The theory of $\mathfrak{T}$ (and thus of $\mathfrak{T}_\mathfrak{p}$) is decidable. Decision procedures for both $\mathfrak{T}_\mathfrak{p}$ and $\mathfrak{T}$ have non-elementary complexity.*

Proof. Decidability follows from the automata representation; lower bounds from encoding WS1S [22]. $\Box$

4 Primal tree extension algebra and automata

The goal of this section is to compare the power of $\mathfrak{T}_\mathfrak{p}$ and $\mathfrak{T}$. Since the previous results show that all regular sets of trees can be defined in $\mathfrak{T}_\mathfrak{p}$ (assuming $n > 1$: we discuss the special case $n = 1$ later), one might ask whether the domain-comparison operator $\approx_{\mathrm{dom}}$ is in fact already definable in $\mathfrak{T}_\mathfrak{p}$. We show here that $\approx_{\mathrm{dom}}$ is not expressible in the primal tree extension algebra, and thus $\mathfrak{T}_\mathfrak{p}$ and $\mathfrak{T}$ are different. We make the difference between the two models more concrete by presenting a restricted tree-tuple automaton model that exactly captures definability in $\mathfrak{T}_\mathfrak{p}$.

Let $\vec{T} = (T_1, \ldots, T_k)$ be a tuple of trees. We say that t is a branch of $\vec{T}$ if t is a branch of one of T_is. In this case we also write $t \in \vec{T}$ for $\bigvee_i \eta(t, T_i)$. The automaton model is called a *splitting automaton*; such a device accepts or rejects a tuple $\vec{T}$ by defining a run over the set of all branches of $\vec{T}$ (as opposed to products of branches as for general tree-tuple automata). Intuitively, a splitting automaton has parallel threads moving up distinct branches of $\vec{T}$, with these threads merging at the point where the branches meet.

A *splitting-vector* is a function V that assigns to each $(i, a) \in \Delta \times \Sigma$ a finite set of integers in such a way that for any fixed i, the sets $V(i, a), a \in \Sigma$, are disjoint. The *range* of a splitting vector V is *range*$(V) = \bigcup_{(i,a) \in \Delta \times \Sigma} V(i, a)$.

For a finite set S, an *S-splitting vector* is a finite set V of tuples $(i, a, J, s) \in \Delta \times \Sigma \times \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \times S$, such that the projection on the first three components, denoted by $Subset(V) = \{(i, a, J) \mid \exists s\ (i, a, J, s) \in V\}$, is a splitting vector, and such that for every (i, a) there is exactly one $(i, a, J, s) \in V$. We let $State(V)(i, a)$ be the unique $s \in S$ such that for some J, $(i, a, J, s) \in V$. For an S-splitting vector, we define the range of V to be the range of the ordinary splitting-vector $Subset(V)$.

An *S-splitting rule* is a rule of the form

$$(I, s) \Leftarrow V,$$

where I is a finite set of integers, $s \in S$, and V is an S-splitting vector with $range(V) \subseteq I$. Intuitively, a splitting vector describes for each successor $\mathrm{succ}_i^a(t)$ of a branch t which components of $\vec{T}$ have that successor. An S-splitting vector describes the state of the machine on each of these successors of a branch, while a rule describes a bottom-up transition to a new state and new set of trees in $\vec{T}$.

An *acceptance partition* F is a function assigning to each $a \in \Sigma$ a set $J \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$, while an *$S$-acceptance partition* F is a function assigning to each $a \in \Sigma$ a pair $(J, s) \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \times S$. For such a function F, we let $Subset(F)$ and $State(F)$ be the two projection functions: $Subset(F)$ is the function mapping $a \in \Sigma$ to the J such that $(J, s) \in F(a)$, and $State(F)$ is the function that maps $a \in \Sigma$ to the s such that $(J, s) \in F(a)$.

For a branch t, let $supp(t, \vec{T})$ be $\{i \mid t \in T_i\}$. Given $\vec{T}$ and a branch t of $\vec{T}$, $v(t, \vec{T})$ is the splitting vector assigning to (i, a) the set $\{j \mid \mathrm{succ}_i^a(t) \in T_j\}$. We let $v(\emptyset, \vec{T})$ be the acceptance partition assigning to each $a \in \Sigma$ the set $\{i \mid \epsilon_a \in T_i\}$.

A *k-dimensional (bottom-up) splitting automaton* A is a tuple $(S, \delta, IR, \mathcal{F})$ where:

- S a finite set (the *states* of A).
- δ, the *transition relation*, is a finite set of S-splitting rules $(I, s) \Leftarrow V$ with $I \subseteq \{1, \ldots, k\}$, with every S-splitting vector V contained in at least one rule.
- IR, the set of *initialization rules* is a set of rules of the form $(I, s) \Leftarrow$, where $I \subseteq \{1, \ldots, k\}$, and each subset I is in at least one rule of IR.
- A collection of S-acceptance partitions $\mathcal{F}$, the *accepting partitions of* A.

A bottom-up splitting automaton is *deterministic* if there is at most one initialization rule (I, s), for each $I \subseteq \{1, \ldots, k\}$, in δ there is at most one rule with a given right-hand side.

A run r of a k-dimensional bottom-up splitting automaton A on $\vec{T}$ of size k is a function from the branches of $\vec{T}$ to the states S of A such that:

- For every frontier branch t (i.e. a branch such that no extension of t is a branch of $\vec{T}$) with $supp(t, \vec{T}) = I$, $r(t)$ is a state s such that $(I, s) \Leftarrow$ is in IR.
- For every non-frontier branch t, $r(t)$ is a state s such that $(supp(t, \vec{T}), s) \Leftarrow V$ is in δ, where $Subset(V) = v(t, \vec{T})$ and $State(V)(i, a) = r(\mathrm{succ}_i^a(t))$, whenever $\mathrm{succ}_i^a(t)$ is in $\vec{T}$.

A run is *accepting* if there is an S-acceptance partition $F \in \mathcal{F}$ with $Subset(F) = v(\emptyset, \vec{T})$ and $State(F)(a) = r(\epsilon_a)$ for each a such that ϵ_a is in $\vec{T}$.

The following is an example of a 2-splitting automaton A over alphabet $\Sigma = \{a, b\}$ $\Delta = \{1, 2\}$:

$$
\begin{array}{ll}
\mathbf{IR} = & \{(\{1\}, s_0) \Leftarrow \} \\
\delta = & \{(\{1\}, s_0) \Leftarrow \{(1, b, \{1\}, s_0), (2, b, \{1\}, s_0)\} \\
& (\{1,2\}, s_1) \Leftarrow \{(1, b, \{1\}, s_0)\} \\
& (\{1,2\}, s_1) \Leftarrow \{(1, a, \{1,2\}, s_1)\} \\
\mathcal{F} = & \{(a, \{1, 2\}, s_1)\}
\end{array}
$$

The initial rule says what A does on nodes that have no successors: these nodes must only be in the first tree, and on each such node we start in state s_0. The first rule in δ says that if a node n_0 has both of its successors n_1 and n_2 in the first tree with label b and A is in state s_0, then n_0 is only in the first tree, and A remains in state s_0 on n_0. The second rule says that on a node n_0 with only a 1-successor n_1, if n_1 is labeled with b and is only in the first tree, and A is in s_0 on n_1, then n_0 is in both trees, and A is in state s_1 on n_0. The final rule says that if n_0 has only 1-successor n_1, n_1 is in both trees, and n_1 is labeled with a, then n_0 is in both trees, and the state of A is unchanged in moving to n_0. The acceptance partition describes the requirement that the root node be common to both trees, labeled with a, and in state s_1. This automaton accepts a pair (T_1, T_2) iff T_1 consists of a binary tree labeled with b placed below a linear stem labeled with a, $T_2 \prec T_1$, and T_2 is exactly the linear stem.

In the special case of a 1-dimensional automaton, a splitting vector V is just a collection of pairs $(i, a) \in \Delta \times \Sigma$, such that for each i there is at most one a with $(i, a) \in V$. A splitting vector can thus be identified with a function from Δ into $\Sigma_\perp$. An S-splitting vector likewise corresponds to a function from Δ into $\Sigma_\perp \times S$, and the set of rules of the automaton can be identified with a partial function mapping f in $(\Sigma_\perp \times S)^\Delta$ to $s \in S$. Under this identification, a run of a splitting automaton over a single tree T corresponds to a run of a standard bottom-up tree automaton over $T_\perp^c$, and hence the set of trees accepted by a 1-dimensional splitting automaton is regular.

Theorem 2 *A subset of* $\mathrm{Trees}_n(\Sigma)^k$ *is definable by a formula of* $\mathfrak{T}_\mathfrak{p}$ *iff it is accepted by a k-dimensional splitting automaton. Furthermore, the translations from formulae to automata and vice versa are effective.*

Proof sketch. To go from a formula to an automaton, we show that atomic formulae of $\mathfrak{T}_\mathfrak{p}$ can be captured by splitting automata, and then prove the usual closure properties for splitting automata. The converse is shown by induction on the dimension: the case $k = 1$ is sketched above, and in the inductive step one shows how to code in $\mathfrak{T}_\mathfrak{p}$ the effect of a rule in which tree sequences of dimension below k are merged into a tree-sequence of size k. □

Consequences of the automaton representation for $\mathfrak{T}_\mathfrak{p}$ The main consequence is that $\mathfrak{T}_\mathfrak{p}$ and $\mathfrak{T}$ are different: $\mathfrak{T}$ defines more subsets of $\mathrm{Trees}_n(\Sigma)^k$ for any $k \geq 2$.

Corollary 3 *If* $|\Sigma| > 1$*, then the predicate* $\approx_{\mathrm{dom}}$ *is not expressible in* $\mathfrak{T}_{\mathfrak{p}}$*.*

Proof sketch. For each m we consider the pair of trees encoding strings a^m and b^m as the left-most possible path, and a pair encoding a^m and b^l, $l \neq m$. If we choose m, l so that b^m and b^l are indistinguishable by finite automata with k states, then the resulting two pairs of trees are indistinguishable by splitting automata with k or fewer states. Hence there can be no $\mathfrak{T}_{\mathfrak{p}}$ formula that uniformly separates pairs (a^m, b^m) from (a^m, b^l), while there is such a $\mathfrak{T}$ formula. □

Another consequence is that in $\mathfrak{T}_{\mathfrak{p}}$, quantification can be restricted to a finite set. For a symbol $a \in \Sigma$, let $\mathrm{Trees}_n(\Sigma)|_{\vec{T}^c_a}$ be the set of all trees T_0 such that every branch of T_0 is a branch of a tree $T^c_a, T \in \vec{T}$. Note that this set is definable from $\vec{T}$ over $\mathfrak{T}_{\mathfrak{p}}$. By encoding the run of a splitting automaton, we can show:

Corollary 4 *For any* $a \in \Sigma$*, and every* $\mathfrak{T}_{\mathfrak{p}}$ *formula* $\varphi(\vec{T})$*, there is an equivalent formula in which the quantifiers range over* $\mathrm{Trees}_n(\Sigma)|_{\vec{T}^c_a}$. □

5 Expressibility and model theory

We now study model theoretic properties and the expressive power of the tree extension algebras. We start with two results that show a sharp contrast between $\mathfrak{T}$ and $\mathfrak{T}_{\mathfrak{p}}$ on the one hand, and term algebra. As mentioned in the introduction, term algebras have a particularly well behaved model theory: they are stable (which implies that no linear order is definable), have finite VC dimension, and admit quantifier-elimination. In contrast to this, we can show:

Proposition 1 *There is a linear ordering on* $\mathrm{Trees}_n(\Sigma)$ *definable in* $\mathfrak{T}_{\mathfrak{p}}$*.*

Proof sketch. We first show how to define a linear order on branches, by comparing two branches t_1, t_2, at $t_1 \sqcap t_2$. We then show how to extend the ordering to trees. □

Since there is no linear order definable in a term algebra, the operations of $\mathfrak{T}_{\mathfrak{p}}$ clearly cannot be first-order definable in term algebra.

Let $join_a(T_1, T_2)$ be the binary tree whose root is labeled a, and whose left and right subtrees are T_1 and T_2 respectively. The structure $\langle \mathrm{Trees}_n(\Sigma), \prec, (join_a)_{a\in\Sigma}, (\epsilon_a)_{a\in\Sigma}\rangle$ corresponds to the first-order theory of $FT_{\leq}$ constraints over feature trees studied in [23, 25]. Since that theory is known to be undecidable [25], we obtain:

Proposition 2 $join_a$ *is not definable in* $\mathfrak{T}$*.* □

One can also give a direct proof, showing that with $join_a$ one can define predicates not recognizable by tree automata.

Thus, first-order logic over $\mathfrak{T}_{\mathfrak{p}}$ and $\mathfrak{T}$ is incomparable with first-order logic over term and feature algebras [25, 13]. Another consequence is that queries over this logic (see Section 6) cannot express the classes of tree set constraints used most frequently in program analysis [3, 27].

VC dimension We now show that the behavior of definable families in $\mathfrak{T}_{\mathfrak{p}}$ and $\mathfrak{T}$ formula is not as tame as in a term algebra. A standard notion of tameness for definable families is given by the concept of *VC dimension* (cf. [5]) (also known as not having the independence property [21]). Given an infinite set X and a family $\mathcal{C}$ of its subsets, X *shatters* a finite set $F \subset X$ if $\{F \cap C \mid C \in \mathcal{C}\}$ is the powerset of F. The VC dimension of $\mathcal{C}$ is the maximum size of a finite set it shatters (or ∞ if arbitrarily large finite sets are shattered). Given a structure $\mathcal{M}$ over a set U and a formula $\varphi(x_1, \ldots, x_m, y_l, \ldots, y_k)$ in the language of $\mathcal{M}$, the definable family given by φ is $\{\ \{\vec{a} \in U^m \mid \mathcal{M} \models \varphi(\vec{a}, \vec{b})\} \mid \vec{b} \in U^k\}$. We say that $\mathcal{M}$ has *finite VC dimension* if every definable family has finite VC dimension. From the learning point of view, this means that every definable family is PAC-learnable [5]. Finite VC dimension also implies strong bounds on the expressiveness of relational query languages [6, 8]. It turns out that the presence of the extension predicate $\prec$, prevents $\mathcal{M}$ from having finite VC dimension.

Proposition 3 *For any nonempty* Σ*, the structure* $\langle \mathrm{Trees}_2(\Sigma), \preceq\rangle$ *does not have finite VC dimension. Hence* $\mathfrak{T}_{\mathfrak{p}}$ *and* $\mathfrak{T}$ *do not have finite VC dimension.* □

Model theory of strings vs. model theory of trees We remarked before that if the alphabet of directions has a unique element, then trees over such alphabet are naturally associated with strings: that is, trees in $\mathrm{Trees}_1(\Sigma)$ are in 1-1 correspondence with Σ^*. What are the analogs of $\mathfrak{T}_{\mathfrak{p}}$ and $\mathfrak{T}$ in this case then? It turns out that they are known and well-studied structures. We now compare model-theoretic properties of those structures with $\mathfrak{T}$ and $\mathfrak{T}_{\mathfrak{p}}$.

First, we define analogs of $\mathfrak{T}$ functions and predicates for $n = 1$. The predicate $\prec$ becomes string prefix $<$; the successor operations become concatenation operations $f_a(s) = s \cdot a$ for $a \in \Sigma$; ϵ_a becomes definable as the smallest, with respect to $<$, image of f_a, and $\approx_{\mathrm{dom}}$ simply tests if two strings x and y have the same length, which we shall denote by $\mathrm{el}(x, y)$. Summing up, for $n = 1$, $\mathfrak{T}$ is equivalent to

$$\mathfrak{S} \;=\; \langle \Sigma^*, <, (f_a)_{a\in\Sigma}, \mathrm{el}\rangle$$

and $\mathfrak{T}_{\mathfrak{p}}$ is equivalent to its reduct $\mathfrak{S}_{\mathfrak{p}} = \langle \Sigma^*, <, (f_a)_{a\in\Sigma}, \rangle$. These structures are well-known [11, 10, 9].

Figure 1 summarizes results on $\mathfrak{T}, \mathfrak{T}_{\mathfrak{p}}$, and their string analogs $\mathfrak{S}$ [11, 10] and $\mathfrak{S}_{\mathfrak{p}}$ [9]. It turns out that model-theoretically $\mathfrak{S}$ and $\mathfrak{T}$ are rather close, but $\mathfrak{S}_{\mathfrak{p}}$ and $\mathfrak{T}_{\mathfrak{p}}$ are very different. The first line of the table talks about one-dimensional definable sets, that is, subsets of Σ^* or $\mathrm{Trees}_n(\Sigma)$. The second line is about arbitrary definable sets. The automaton construction for $\mathfrak{S}_{\mathfrak{p}}$ is a counter-free restriction of regular prefix automata of [4]. No similar restriction (either to first-order definable or star-free tree languages [34]) is possible over $\mathfrak{T}_{\mathfrak{p}}$, since even in the one-dimensional case, arbitrary regular tree languages are definable.

The third line compares VC dimension of definable families. The fourth and the fifth line compare first-order and weak monadic second-order theories; the undecidability of the latter for $\mathfrak{T}_{\mathfrak{p}}$ is shown below.

Proposition 4 *One can code arithmetic $(+, \times)$ in weak MSO over $\mathfrak{T}_{\mathfrak{p}}$. Consequently, the weak MSO theory of $\mathfrak{T}_{\mathfrak{p}}$ (and even the weak EMSO theory) is undecidable.* $\square$

6 Relational calculi with tree extension constraints

One of the motivations for tree extension algebras is to get tree constraints relevant in database (and in particular, XML) applications. In such applications, one writes queries, typically first-order, not only over trees but also over collections of trees. Using database terminology, we deal with *relational calculus* with tree extension constraints. From the logical point of view, we consider definability over $\mathfrak{T}$ and $\mathfrak{T}_{\mathfrak{p}}$ parameterized by sets or relations on trees.

In this section, after giving the basic definition for relational calculi with constraints, we obtain normal forms for queries that will allow us to classify the expressive power and complexity of query evaluation for relational calculi with tree extension constraints. We then obtain normal forms for queries that are known to produce finite output on any input.

6.1 Preliminaries

A database *schema* σ is a finite collection of relation symbols. Given an underlying structure $\mathcal{M}$, relational calculus over $\mathcal{M}$ and σ, $\mathrm{RC}(\mathcal{M}, \sigma)$, is the class of first-order queries (formulae) in the language of $\mathcal{M}$ supplemented with the symbols from σ. If σ is understood, or irrelevant, we write $\mathrm{RC}(\mathcal{M})$. For example, if σ has a single binary relation E, then the $\mathrm{RC}(\mathfrak{T}_{\mathfrak{p}}, \sigma)$ query

$$\forall x \forall y\ E(x, y) \to \big(\eta(x) \wedge \eta(y) \wedge x \prec y\big)$$

tests whether E is a subgraph of $\prec$ whose nodes are branches.

We shall always interpret σ relations as finite relations over the universe of $\mathcal{M}$ (in our case, $\mathrm{Trees}_n(\Sigma)$). The *active domain* of a σ-structure $\mathcal{A}$ is the set $adom(\mathcal{A})$ of all the elements of $\mathrm{Trees}_n(\Sigma)$ that occur in $\mathcal{A}$. Active-domain quantifiers are quantifiers of the form $\exists x \in adom$ and $\forall x \in adom$; $\exists x \in adom\ \varphi(x)$ is interpreted as the existence of an element $a \in adom(\mathcal{A})$ that satisfies $\varphi(a)$.

6.2 Normal forms

The main tools for analyzing the expressive power of $\mathrm{RC}(\mathcal{M}, \sigma)$ come in the form of normal forms for queries. The most basic one, *restricted-quantifier normal form* [6, 16, 8], states that every $\mathrm{RC}(\mathcal{M}, \sigma)$ formula is equivalent to a formula in which no σ symbol appears in the scope of a quantifier $\forall x$ or $\exists x$ (that is, they appear only in the scope of quantifiers $\forall x \in adom$ and $\exists x \in adom$). The ability to put queries in restricted-quantifier normal form implies very strong expressivity bounds on relational calculi with constraints. In particular, it gives a strong bound on *generic* queries expressible in such calculi. Recall that a query is generic if it commutes with permutations of the domain of $\mathcal{M}$. For example, *parity*, testing if the number of elements of $adom(\mathcal{A})$ is generic, as is *graph connectivity*. In fact, any query definable in a standard relational query language (relational calculus, datalog, etc.) without constraints is generic. If all queries can be put in restricted-quantifier normal form, then every generic query in $\mathrm{RC}(\mathcal{M}, \sigma)$ is definable in first-order over finite σ structures and a linear ordering on their domain. This in turn implies that queries such as parity and connectivity are not definable. However, it was shown in [8] that no structure with infinite VC dimension admits restricted-quantifier normal form. Hence,

Corollary 5 *$\mathfrak{T}_{\mathfrak{p}}$ and $\mathfrak{T}$ do not admit restricted-quantifier normal form, even if $|\Sigma| = 1$.*

We thus need to find a different way of getting bounds on the expressive power and complexity of $\mathrm{RC}(\mathfrak{T})$ and $\mathrm{RC}(\mathfrak{T}_{\mathfrak{p}})$. The main tool is a normal form result that shows how to restrict quantification to a finite extension of the active domain. From that result, we derive both complexity and expressibility bounds.

6.3 Restricting quantification in $\mathrm{RC}(\mathfrak{T})$

We show that a certain weaker restricted quantifier normal form holds for relational calculus over $\mathfrak{T}$. Recall that for a set X of trees, $\mathrm{Trees}_n(\Sigma)|_{\mathrm{dom}(X)}$ is the set of all trees T such that $\mathrm{dom}(T) \subseteq \bigcup_{T_0 \in X} \mathrm{dom}(T_0)$. Note that if X is definable by a formula of $\mathfrak{T}$, then so is $\mathrm{Trees}_n(\Sigma)|_{\mathrm{dom}(X)}$.

Model	$\mathfrak{S}_\mathfrak{p}$	$\mathfrak{T}_\mathfrak{p}$	$\mathfrak{S}$	$\mathfrak{T}$
one-dimension definable sets	*-free	regular	regular	regular
arbitrary definable sets	counter-free prefix automata	splitting automata	regular	regular
VC dimension	finite	infinite	infinite	infinite
FO theory	decidable	decidable	decidable	decidable
(weak) MSO theory	decidable	undecidable	undecidable	undecidable

Figure 1. Comparison of string and tree models

The main result of this section is that for every $\mathrm{RC}(\mathfrak{T}, \sigma)$ query, quantifiers can be restricted to range over the (finite and definable) set of trees whose domains can only contain nodes present in the domains of trees in the active domain of the finite σ-structure, and in the tuple of the free variables.

Theorem 3 *Let* $\mid \Sigma \mid > 1$. *Then any relational calculus formula* $\varphi(\vec{T})$ *in* $\mathrm{RC}(\mathfrak{T}, \sigma)$ *is equivalent to a formula in which quantifiers range over* $\mathrm{Trees}_n(\Sigma)|_{\mathrm{dom}(X)}$ *where* $X = adom(\mathcal{A}) \cup \vec{T}$.

Proof sketch. We first change the vocabulary to a purely relational one. For a σ-structure $\mathcal{A}$, we write $\mathrm{dom}(\mathcal{A})$ for $\bigcup_{T \in adom(\mathcal{A})} \mathrm{dom}(T)$. Let $\mathfrak{T}[\mathcal{A}]$ be the structure in the (new) vocabulary for $\mathfrak{T}$ plus σ whose universe is the set of all trees T with $\mathrm{dom}(T) \subseteq \mathrm{dom}(\mathcal{A})$. We then prove, in two steps, using Ehrenfeucht-Fraïssé games, that for any $k \geq 0$, there exists $m \geq 0$ such that $\mathfrak{T}[\mathcal{A}] \equiv_m \mathfrak{T}[\mathcal{B}]$ implies $(\mathfrak{T}, \mathcal{A}) \equiv_k (\mathfrak{T}, \mathcal{B})$. This suffices, since then every $\mathrm{RC}(\mathfrak{T}, \sigma)$ query of quantifier rank k is a union of rank-m types over the restriction of $\mathfrak{T}$ to trees with $\mathrm{dom}(T) \subseteq \mathrm{dom}(\mathcal{A})$, for some $m > 0$ that depends on k only, and each such type can be expressed by a query with restricted quantification. □

Theorem 3 does not hold for $|\Sigma| = 1$. However, the proof of Theorem 3 implies that in this case it suffices to restrict quantification to trees whose domains lie in the *completion* of the union of the domains of trees in the databases and in the tuple of free variables.

Remark While the result of Theorem 3 also applies to $\mathrm{RC}(\mathfrak{T}_\mathfrak{p}, \sigma)$, the set $\mathrm{Trees}_n(\Sigma)|_{\mathrm{dom}(X)}$ is not definable from X over $\mathfrak{T}_\mathfrak{p}$. We can get a finer bound for $\mathfrak{T}_\mathfrak{p}$ (using the notation of Corollary 3).

Proposition 5 *Any relational calculus formula* $\varphi(\vec{T})$ *in* $\mathrm{RC}(\mathfrak{T}_\mathfrak{p}, \sigma)$ *is equivalent to a formula in which quantifiers range over* $\mathrm{Trees}_n(\Sigma)|_{X_a^c}$, *where* $X = adom(\mathcal{A}) \cup \vec{T}$, *and* $a \in \Sigma$. □

6.4 Data complexity of relational calculi over $\mathfrak{T}$ and $\mathfrak{T}_\mathfrak{p}$

Using Theorem 3, we obtain bounds on query evaluation over $\mathfrak{T}_\mathfrak{p}$ and $\mathfrak{T}$. For relational calculi, we are interested in *data complexity* [2]: the complexity of evaluating a fixed query as databases vary. The result below says that data complexity is essentially PH (polynomial hierarchy): PH is an upper bound, and for every level of PH, there is a complete problem that can be encoded. Since the encoding can be done in $\mathrm{RC}(\mathfrak{T}_\mathfrak{p})$, this gives us matching bounds for the complexity over $\mathfrak{T}$ and the simpler algebra $\mathfrak{T}_\mathfrak{p}$.

Theorem 4 *The data complexity of* $\mathrm{RC}(\mathfrak{T})$ *(and thus* $\mathrm{RC}(\mathfrak{T}_\mathfrak{p})$*) is in PH. Furthermore, there is an infinite set* $S \subseteq \mathrm{Trees}_n(\Sigma)$ *definable in* $\mathfrak{T}_\mathfrak{p}$ *such that for every* n, *there are problems complete for* Σ_n^{p} *and* Π_n^{p} *which can be expressed in* $\mathrm{RC}(\mathfrak{T}_\mathfrak{p})$ *(and thus* $\mathrm{RC}(\mathfrak{T})$*) over databases whose active domain lies in* S.

Proof sketch. Using the quantifier-restriction of Theorem 3, we code $\mathrm{RC}(\mathfrak{T}, \sigma)$ queries in second-order logic over a (polynomially computable) expansion of σ. For the converse, we code MSO over σ in $\mathrm{RC}(\mathfrak{T}_\mathfrak{p}, \sigma)$ for σ-structures $\mathcal{A}$ with $\mathrm{dom}(T) \subset 1^*$ and labeling identically a for some fixed a, for any $T \in adom(\mathcal{A})$. □

6.5 Generic data complexity and expressivity bounds

The PH bounds on query expressivity might lead one to imagine that arbitrary NP-complete calculations on tree sets can be performed by tree extension queries. We show, however, that the complexity of *generic* queries is in fact quite low. A generic (Boolean) query is just an isomorphism type Q of σ-structures. We say that Q is expressible in $\mathrm{RC}(\mathcal{M}, \sigma)$ if there is a sentence Φ of $\mathrm{RC}(\mathcal{M}, \sigma)$ such that for any σ-structure $\mathcal{A}$ over $\mathcal{M}$, $(\mathcal{M}, \mathcal{A}) \models \Phi$ if $\mathcal{A}$ is of isomorphism type Q.

Normally, data complexity of a Boolean query Q is defined as the complexity of the language that consists of encodings of structures $\mathcal{A} \in Q$. If $\mathcal{A}$ is a relation over $\mathrm{Trees}_n(\Sigma)$, such an encoding must also encode all the trees in $adom(\mathcal{A})$. Since in generic queries it is irrelevant which trees belong to $adom(\mathcal{A})$, we can use a different encoding, where elements of the active domain, of size k, are encoded as $1, 2, \ldots, k$ in binary, just as in the case of relational calculus without any additional constraints [2]. We denote such an encoding by $enc_{\mathrm{gen}}(\mathcal{A})$, and say that *generic data complexity* of $\mathrm{RC}(\mathcal{M})$ is in a complexity class K if for any σ and any generic query Q expressible in $\mathrm{RC}(\mathcal{M}, \sigma)$, the language $\{enc_{\mathrm{gen}}(\mathcal{A}) \mid \mathcal{A} \in Q\}$ is in K.

Theorem 5 *Generic data complexity of* $\mathrm{RC}(\mathfrak{T})$ *is in (uniform)* AC^0. □

From AC^0 lower bounds (cf. [18]), we obtain:

Corollary 6 *Parity test and connectivity test are not definable in* $\mathrm{RC}(\mathfrak{T})$. □

6.6 Normal forms for safe queries

A fundamental property of database queries is their *safety*: whether for a relational calculus query $\varphi(\vec{x})$, the set $\varphi(\mathcal{A}) = \{\vec{a} \mid \mathcal{A} \models \varphi(\vec{a})\}$ is finite for all $\mathcal{A}$. It is well known that even for pure relational calculus, without additional constraints, safety is undecidable [2]. However, safe queries in pure relational calculus have *effective syntax*; that is, there is an r.e. set of safe queries such that every safe query is equivalent to one in that set. If one considers relational calculus with additional constraints, $\mathrm{RC}(\mathcal{M}, \sigma)$, the question is whether there is effective syntax for safe queries. It is known [31] that effective syntax need not exist even for some decidable $\mathcal{M}$.

The typical way of ensuring query safety is to fix some specific collection of safe queries $\langle\varphi_i\rangle$ and to explicitly bind the output of each query to lie in the output of one of these φ_i. If the sequence φ_i is sufficiently rich, then all safe queries can be captured in this way. This is the idea behind the classical database concept of *range-restriction*, which is extended to the setting of built-in functions in [7]. A query $\varphi(\vec{x})$ is range restricted by another query $\psi(\vec{x})$ if $\varphi(\mathcal{A}) \subseteq \psi(\mathcal{A})$ for every $\mathcal{A}$. For a structure $\mathcal{M}$, we say that queries over $\mathcal{M}$ have *range-restricted form* if there is an r.e. sequence of safe queries $\langle\varphi_i\rangle$ such that every safe query is range-restricted by some φ_i. If safe $\mathrm{RC}(\mathcal{M}, \sigma)$ queries have a range-restricted form, then queries of the form $\varphi \wedge \varphi_i$ provide an enumeration of all safe $\mathrm{RC}(\mathcal{M}, \sigma)$ queries; thus, safe $\mathrm{RC}(\mathcal{M}, \sigma)$ queries have effective syntax. The techniques of [16] or [13] can be used to show that queries over term algebra have a range-restricted normal form. We now extend this to our tree structures.

Theorem 6 *Safe queries in both* $\mathrm{RC}(\mathfrak{T}_{\mathfrak{p}})$ *and* $\mathrm{RC}(\mathfrak{T})$ *have a range-restricted form. In particular, safe queries for both calculi have effective syntax.*

Proof sketch. The proof is by an Ehrenfeucht-Fraïssé game argument, showing that if a tree in $\varphi(\mathcal{A})$ has a branch that is "very far" from $\mathcal{A}$, then $\varphi(\mathcal{A})$ is infinite, where "very far" depends on φ only. Then the sequence of queries φ_i returning all branches of distance at most i from $\mathcal{A}$ can be used. □

Summary We conclude this section by summarizing the results on $\mathrm{RC}(\mathfrak{T})$ and $\mathrm{RC}(\mathfrak{T}_{\mathfrak{p}})$, as well as comparing them with relational calculi over strings, $\mathrm{RC}(\mathfrak{S})$ and $\mathrm{RC}(\mathfrak{S}_{\mathfrak{p}})$. As for expressibility without database relations, $\mathfrak{T}$ happens to be close to $\mathfrak{S}$, but $\mathfrak{S}_{\mathfrak{p}}$, which is the restriction of $\mathfrak{T}_{\mathfrak{p}}$ to one-directional trees, is significantly simpler than $\mathfrak{T}_{\mathfrak{p}}$.

	$\mathrm{RC}(\mathfrak{S}_{\mathfrak{p}})$	$\mathrm{RC}(\mathfrak{T}_{\mathfrak{p}})$	$\mathrm{RC}(\mathfrak{S})$	$\mathrm{RC}(\mathfrak{T})$
restricted quantifier normal form	yes	no	no	no
data complexity	AC^0	PH	PH	PH
generic data complexity	FO(<)	AC^0	AC^0	AC^0
Syntax for safe queries	yes	yes	yes	yes

7 Conclusion

We have seen that constraints based on extension relations with additional domain comparison and leaf concatenation functions are solvable, with the resulting solution set presented by a regular tree relation. We also saw (Proposition 2) that one cannot combine these formulae with unary term equalities while remaining within the domain of regular relations.

Although the bounds presented for primal tree extension formulae are the same as for general formulae in the worst case, it remains to check to what extent primal formulae can be evaluated with greater parallelism. It is clear that splitting automata can be implemented more efficiently than general tree-tuple automata on trees with small overlap, but we currently have no formal results that capture this advantage. The bounds presented here for deciding sentences that are parameterized by predicates for tree sets are rather discouraging. Even when a primal formula φ is fixed, we have shown that the complexity of query evaluation is high. There are, however, sublogics where the data complexity is polynomial: for instance, when one restricts to formulae that operate on sequential encodings of the tree, one can obtain polynomial bounds by simply "pulling back" the corresponding results for strings.

The results here apply to tuples of finite trees. In future work, we will examine what occurs for tuples of *infinite* trees. The natural motivation for this comes from verification: given a set of state machines $S_1 \dots S_n$, one is often interested in synthesizing a state machine S such that the product of S with S_i satisfies some property. If one passes from the machine S_i to the behavior tree T_i obtained from unwinding it, this corresponds to generating a regular infinite tree T that satisfies a formula $\varphi(T, T_1 \dots T_n)$.

Acknowledgements We thank anonymous referees for their comments. Part of this work was done while Michael Benedikt was visiting the University of Toronto. Leonid Libkin is supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and from Bell University Laboratories.

References

[1] S. Abiteboul. Semistructured data: from practice to theory. In *LICS'01*, pages 379–386.

[2] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] A. Aiken. Set constraints: results, applications, and future directions. In *PPCP'94*, pages 326–335.

[4] D. Angluin, D. N. Hoover. Regular prefix relations. *Mathematical Systems Theory* 17(3),167–191,1984.

[5] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge Univ. Press, 1992.

[6] M. Benedikt, L. Libkin. Relational queries over interpreted structures. *J. ACM* 47 (2000), 644–680.

[7] M. Benedikt, L. Libkin. Safe constraint queries. *SIAM J. Comput.* 29 (2000), 1652–1682.

[8] M. Benedikt, L. Libkin, T. Schwentick, L. Segoufin. String operations in query languages. In *PODS'01*, pages 183–194.

[9] M. Benedikt, L. Libkin, T. Schwentick, L. Segoufin. A model-theoretic approach to regular string relations. In *LICS'01*, pages 431–440.

[10] A. Blumensath and E. Grädel. Automatic structures. In *LICS'00*, pages 51–62.

[11] V. Bruyère, G. Hansel, C. Michaux, R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc.* 1 (1994), 191–238.

[12] H. Comon et al. *Tree Automata: Techniques and Applications*. Available at `www.grappa.univ-lille3.fr/tata`.

[13] E. Dantsin, A. Voronkov. Expressive power and data complexity of query languages for trees and lists. In *PODS'2000*, pages 157–165.

[14] J. Dörre. Feature logics with weak subsumption constraints. In *Annual Meeting of the Assoc. of Comput. Linguistics*, 1991, pages 256–263.

[15] J. Elgaard, N. Klarlund, A. Moller. MONA 1.x: new techniques for WS1S and WS2S. In *CAV'98*, pages 516–520.

[16] J. Flum and M. Ziegler. Pseudo-finite homogeneity and saturation. *J. Symbolic Logic* 64 (1999), 1689–1699.

[17] W. Hodges. *Model Theory*. Cambridge, 1993.

[18] N. Immerman. *Descriptive Complexity*. Springer Verlag, 1999.

[19] H.-P. Kolb, U. Mönnich. *The Mathematics of Syntactic Structure: Trees and Their Logics*. Walter De Gruyter, 1999.

[20] O. Kupferman, S. Safra, M. Vardi. Relating word and tree automata. In *LICS'96*, pages 322–332.

[21] M. C. Laskowski. Vapnik-Chervonenkis classes of definable sets. *J. London Math. Soc.*, 45:377–384, 1992.

[22] A. R. Meyer. The inherent complexity of theories of ordered sets. *Proc. Int. Congress of Mathematics*, 1975, pages 477–482.

[23] M. Müller, J. Niehren. Ordering constraints over feature trees expressed in second-order monadic logic. In *RTA'98*, pages 196–210.

[24] M. Müller, J. Niehren, A. Podelski. Ordering constraints over feature trees. In *PPCP'97*, pages 297–311.

[25] M. Müller, J. Niehren, R. Treinen. The first-order theory of ordering constraints over feature trees. In *LICS'98*, pages 432–443.

[26] D. Niwinski, I. Walukiewicz. Relating hierarchies of word and tree automata. In *STACS'98*, pages 320–331.

[27] L. Pacholski, A. Podelski. Set constraints: a pearl in research on constraints. In *CP'97*, pages 549–562.

[28] A. Potthoff, W. Thomas. Regular tree languages without unary symbols are star-free. *FCT'93*, pages 396–405.

[29] G. Smolka. The Oz programming model. In *Computer Science Today*, LNCS 1000, 1995, pages 324–343.

[30] G. Smolka, R. Treinen. Records for logic programming. *J. Logic Progr.* 18 (1994), 229–258.

[31] A. Stolboushkin , M. Tsaitlin. Finite queries do not have effective syntax. *Information and Computation*, 153 (1999), 99–116.

[32] Z. Su, A. Aiken, J. Niehren, T. Priesnitz, and R. Treinen. The first-order theory of subtyping constraints. In *POPL'02*.

[33] W. Thomas. Languages, automata, and logic. *Handbook of Formal Languages, Vol. 3*, Springer, 1997.

[34] W. Thomas. Logical aspects in the study of tree languages. *CAAP'84*, pages 31–50.

[35] V. Vianu. *A Web Odyssey: from Codd to XML.* In *PODS'01*.

[36] S. Vorobyov, A. Voronkov. Complexity of nonrecursive logic programs with complex values. *PODS'98*, pages 244–253.