Automata, Logic, and XML

Frank Neven

University of Limburg
frank.neven@luc.ac.be

Abstract. We survey some recent developments in the broad area of automata and logic which are motivated by the advent of XML. In particular, we consider unranked tree automata, tree-walking automata, and automata over infinite alphabets. We focus on their connection with logic and on questions imposed by XML.

1 Introduction

Since Codd [11], databases have been modeled as first-order relational structures and database queries as mappings from relational structures to relational structures. It is, hence, not surprising that there is an intimate connection between database theory and (finite) model theory [58, 60]. As argued by Vianu, finite model theory provides the backbone for database query languages, while in turn, database theory provides a scenario for finite model theory. More precisely, database theory induces a specific measure of relevance to finite model theory questions and provides research issues that, otherwise, were unlikely to have risen independently.

Today's technology trends require us to model data that is no longer tabular. The World Wide Web Consortium has adopted a standard data exchange format for the Web, called Extended Markup Language (XML) [14], in which data is represented as labeled ordered attributed trees rather than as a table. A new data model requires new tools and new techniques. As trees have been studied in depth by theoretical computer scientists [24], it is no surprise that many of their techniques can contribute to foundational XML research. In fact, when browsing recent ICDT and PODS proceedings,[1] it becomes apparent that a new component is already added to the popular logic and databases connection: tree automata theory. Like in the cross-fertilization between logic and databases, XML imposes new challenges on the area of automata and logic, while the latter area can provide new tools and techniques for the benefit of XML research. Indeed, while logic can serve as a source of inspiration for pattern languages or query languages and as a benchmark for expressiveness of such languages, the application of automata to XML can, roughly, be divided into at least four categories:

[1] ICDT and PODS are abbreviations of *International Conference on Database Theory* and *Symposium on the Principles of Database Systems*, respectively. The following links provide more information: `http://alpha.luc.ac.be/~lucp1080/icdt/` and `http://www.acm.org/sigmod/pods/`.

– as a formal model of computation;
– as a means of evaluating query and pattern languages;
– as a formalism for describing schema's; and
– as an algorithmic toolbox.

In this paper, we survey three automata formalisms which are resurrected by recent XML research: unranked tree automata, tree-walking automata, and automata over infinite alphabets. Although none of these automata are new, their application to XML is. The first two formalism ignore attributes and text values of XML documents, and simply take finite labeled (unranked) trees as an abstraction of XML; only the last formalism deals with attributes and text values. For each of the models we discuss their relationship with XML, survey recent results, and demonstrate new research directions.

The current presentation is not meant to be exhaustive and the choice of topics is heavily biased by the author's own research. Furthermore, we only discuss XML research issues which directly motivate the use of the automata presented in this paper. For a more general discussion on database theory and XML, we suggest the survey papers by Abiteboul [1] and Vianu [61] or the book by Abiteboul, Buneman, and Suciu [2]. We do not give many proofs and the purpose of the few ones we discuss is merely to arouse interest and demonstrate underlying ideas. Finally, we mention that automata have been used in database research before: Vardi, for instance, used automata to statically analyze datalog programs [59].

The paper is further organized as follows. In Section 2, we discuss XML. In Section 3, we provide the necessary background definitions concerning trees and logic. In Section 4, we consider unranked tree automata. In brief, unranked trees are trees where every node has a finite but arbitrary number of children. In Section 5, we focus on computation by tree-walking. In Section 6, we consider such automata over infinite alphabets. We conclude in Section 7.

2 Basics of XML

We present a fairly short introduction to XML. In brief, XML is a data-exchange format whose enormous success is due to its flexibility and simplicity: almost any data format can easily be translated to XML in a transparent manner. For the purpose of this paper, the most important observation is that XML documents can be faithfully represented by labeled attributed ordered trees. Detailed information about XML can be found on the web [14] and, for instance, in the O'Reilly XML book [49].

We illustrate XML by means of an example. Consider the XML document in Figure 1 which displays some information about crew members in a spaceship. As for HTML, the building blocks of XML are *elements* delimited by *start-* and *end-tags*. A start-tag of a `crew`-element, for instance, is of the form `<crew>`, whereas the corresponding closing tag, indicating the end of the element, is `</crew>`. So, all text between and including the tags `<crew>` and `</crew>` in Figure 1, constitutes a `crew`-element. Elements can be arbitrarily nested inside other elements:

the element <name> Spock </name>, for instance, is a subelement of the outer crew-element. Elements can also have *attributes*. These are name value pairs separated by the equality sign. The value of an attribute is always atomic. That is, they cannot be nested. The attribute appears in the start-tag of the element it belongs to. For instance, <starship name="Enterprise"> indicates that the value of the name attribute of that particular starship-element is Enterprise.

```
<starship name="Enterprise">
  <crew id="a457">
    <name> Scotty </name>
    <species> Human </species>
    <job> automata </job>
  </crew>
  <crew id="a544">
    <name> Spock </name>
    <species> Vulcan </species>
    <job> logic </job>
  </crew>
</starship>
```

Fig. 1. Example of an XML document.

An XML document can be viewed as a tree in a natural way: the outermost element is the root and every element has its subelements as children. An attribute of an element is simply an attribute of the corresponding node. The tree in Figure 2, for instance, corresponds to the XML document of Figure 1. There is no unique best way to encode XML documents as trees. Another possibility is to encode attributes as child nodes of the element they belong to. In the present paper we stick to the former encoding.

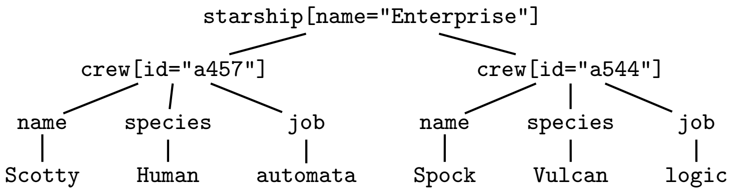


Fig. 2. Tree representation of the XML document in Figure 1

Usually, we are not interested in documents containing arbitrary elements, but only in documents that satisfy some specific constraints. One way to define such "schema's" is by means of DTDs (Document Type Definitions). DTDs are, basically, extended context-free grammars. These are context-free grammars with regular expressions as right-hand sides. In Figure 3, we give an example of a DTD describing the data type of a spaceship. The DTD specifies that `starship` is the outer most element; that every `crew` element has `name` and `species` as its first and second subelement, respectively, and `rank` or `job` as its third subelement. So, `|` and `,` denote disjunction and concatenation, respectively. `#PCDATA` indicates that the element has no subelements but consists of text only. `ATTLIST` determines which attribute belongs to which element. The attributes specified in this DTD can only have a single string value. DTDs are not the only means for representing schema's for XML. We briefly come back to this at the end of Section 4.4.

```
<!DOCTYPE starship [
    <!ELEMENT starship (crew)*>
    <!ELEMENT crew     (name,species,(rank | job))>
    <!ELEMENT name     (#PCDATA)>
    <!ELEMENT species  (#PCDATA)>
    <!ELEMENT rank     (#PCDATA)>
    <!ATTLIST starship name   CDATA>
    <!ATTLIST crew     id     CDATA>
]>
```

Fig. 3. A DTD describing the structure of the document of Figure 1.

Attributes can also be used to link nodes. For instance, the id `a457` of `Scotty` in Figure 1, can be used in a different place in the document to refer to the latter: for instance,

```
<cabin>
  <number> 988 </number>
  <inhabitant> a457 </inhabitant>
</cabin>
```

Actually, the id-attribute has a special meaning in XML but we do not discuss this as it is not important for the present paper.

As indicated above, XML documents can be faithfully represented by trees. In this respect, inner nodes correspond to elements, while leaf nodes contain in general arbitrary text. In the next sections (with exception of Section 6), we only consider the structure of XML documents and, therefore, will ignore attributes and the text in the leaf nodes. Hence, XML documents are trees over a finite alphabet where the alphabet in question is, for instance, determined by a DTD. However, such trees are unranked: nodes can have an arbitrary number of children (the DTD in Figure 3, for instance, allows an unbounded number of `crew` elements). Although ranked trees, that is, trees where the number of children of each node is bounded by a fixed constant, have been thoroughly investigated during the past 30 years [24, 57], their unranked counterparts have been rather neglected. In Section 4, we recall the definition of unranked tree automata and consider some of their basic properties. First, we introduce the necessary notation in the next section.

3 Trees and logic

3.1 Trees

For the rest of this paper, we fix a finite alphabet Σ of element names. The set of Σ-trees, denoted by $\mathcal{T}_\Sigma$, is inductively defined as follows:

(i) every $\sigma \in \Sigma$ is a Σ-tree;
(ii) if $\sigma \in \Sigma$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$, $n \geq 1$ then $\sigma(t_1, \ldots, t_n)$ is a Σ-tree.

Note that there is no a priory bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. For every tree $t \in \mathcal{T}_\Sigma$, the *set of nodes of* t, denoted by $\mathrm{Dom}(t)$, is the subset of $\mathbb{N}^*$ defined as follows: if $t = \sigma(t_1 \cdots t_n)$ with $\sigma \in \Sigma$, $n \geq 0$, and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \{iu \mid i \in \{1, \ldots, n\}, u \in \mathrm{Dom}(t_i)\}$. Thus, ε represents the root while vj represents the j-th child of v. By $\mathrm{lab}^t(u)$ we denote the label of u in t. In the following, when we say tree we always mean Σ-tree.

Next, we define our formalization of DTDs.

Definition 1. A *DTD* is a tuple (d, s_d) where d is a function that maps Σ-symbols to regular expressions over Σ and $s_d \in \Sigma$ is the start symbol. In the sequel we just say d rather than (d, s_d).

A tree t *satisfies* d iff $\mathrm{lab}^t(\varepsilon) = s_d$ and for every $u \in \mathrm{Dom}(t)$ with n children, $\mathrm{lab}^t(u1) \cdots \mathrm{lab}^t(un) \in d(\mathrm{lab}^t(u))$. Note that if u has no children ε should belong to $d(\mathrm{lab}^t(u))$.

Example 1. As an example consider the following DTD describing the XML document in Figure 1:

$$
\begin{aligned}
d(\texttt{starship}) &:= \texttt{crew}^* \\
d(\texttt{crew}) &:= \texttt{name} \cdot \texttt{species} \cdot (\texttt{rank} + \texttt{job}) \\
d(\texttt{name}) &:= \varepsilon \\
d(\texttt{species}) &:= \varepsilon \\
d(\texttt{rank}) &:= \varepsilon \\
d(\texttt{job}) &:= \varepsilon
\end{aligned}
$$

Recall that, for the moment, we are only interested in the structure of XML documents. Therefore, `name`, `species`, `rank`, and `job` are mapped to ε. In Section 6, we consider text and attribute values. □

3.2 Logic

We can also view trees as logical structures (in the sense of mathematical logic [18]). We make use of the relational vocabulary $\tau_\Sigma := \{E, <, (O_\sigma)_{\sigma\in\Sigma}\}$ where E and $<$ are binary and all the O_σ are unary relation symbols. The domain of t, viewed as a structure, equals the set of nodes of t, i.e., $\mathrm{Dom}(t)$. Further, E is the edge relation and equals the set of pairs $(v, v \cdot i)$ where $v, v \cdot i \in \mathrm{Dom}(t)$. The relation $<$ specifies the ordering of the children of a node, and equals the set of pairs $(v \cdot i, v \cdot j)$, where $i < j$ and $v \cdot j \in \mathrm{Dom}(t)$. For each σ, O_σ is the set of nodes that are labeled with a σ.

We consider first-order (FO) and monadic second-order logic (MSO) over these structures. In brief, MSO is FO extended with quantification over set variables. We refer the unfamiliar reader to, e.g., the books by Ebbinghaus and Flum [18], or the chapter by Thomas [57]. In Section 5.3, we also consider transitive closure logic.

Example 2. As an example, consider the MSO formula φ defining the set of trees where every a-labeled node always has a b-labeled descendant:

$$\varphi := \forall x(O_a(x) \rightarrow \exists y(O_b(y) \wedge \mathrm{desc}(x, y))).$$

Here, $\mathrm{desc}(x, y)$ is an abbreviation of the formula

$$\forall X\Big(\big(X(x) \wedge \forall z\forall z'(X(z) \wedge E(z, z') \rightarrow X(z'))\big) \rightarrow X(y)\Big).$$

The formula $\mathrm{desc}(x, y)$ says that any set which contains x and is closed under the edge relation, also contains y. So, it defines the pairs (x, y) where y is a descendant of x. □

4 Unranked Tree Automata

Research on unranked trees in the context of XML was initiated by Brüggemann-Klein, Murata, and Wood [8] based on early work of Pair and Quere [46] and Takahashi [55]. They considered mostly language theoretic properties like non-determinism, two-wayness, tree grammars,. . . . Since then, quite a number of applications, based on their initial ideas, have risen. We discuss these applications in Section 4.4. In Sections 4.1–4.3, we focus on non-deterministic tree automata, the connection with binary tree automata, expressiveness, and complexity.

4.1 Definition

Definition 2. A *nondeterministic tree automaton (NTA)* is a tuple $B = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a function $Q \times \Sigma \rightarrow 2^{Q^*}$ such that $\delta(q, a)$ is a regular string language over Q^* for every $a \in \Sigma$ and $q \in Q$.

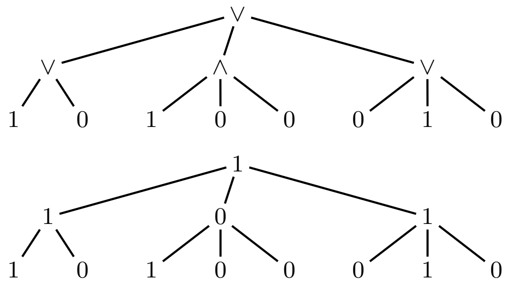


Fig. 4. A tree and an accepting run of the automaton of Example 3.

A *run* of B on a tree t is a labeling $\lambda : \mathrm{Dom}(t) \mapsto Q$ such that for every $v \in \mathrm{Dom}(t)$ with n children, $\lambda(v1) \cdots \lambda(vn) \in \delta(\lambda(v), \mathrm{lab}^t(v))$. Note that when v has no children, then the criterion reduces to $\varepsilon \in \delta(\lambda(v), \mathrm{lab}^t(v))$. A run is *accepting* iff $\lambda(\varepsilon) \in F$. A tree is accepted if there is an accepting run. The set of all accepted trees is denoted by $L(B)$. We call a set of trees *regular* when it can by recognized by an NTA.

We illustrate the above definition with an example.

Example 3. (1) Consider the alphabet $\Sigma = \{\wedge, \vee, 0, 1\}$. Suppose for ease of exposition that trees are always of the following form: 0 and 1 only appear at leaves, $\wedge$ and $\vee$ can appear everywhere except at leaves. These are all tree-shaped positive boolean circuits. We next define an automaton accepting exactly the circuits evaluating to 1. Define $B = (Q, \Sigma, \delta, F)$ with $Q = \{0, 1\}$, $F = \{1\}$, and

$$\begin{aligned}
&\delta(0,0) := \delta(1,1) := \{\varepsilon\};\\
&\delta(0,1) := \delta(1,0) := \emptyset;\\
&\delta(0,\wedge) := (0+1)^*0(0+1)^*;\\
&\delta(1,\wedge) := 1^*;\\
&\delta(0,\vee) := 0^*;\ \text{and,}\\
&\delta(1,\vee) := (0+1)^*1(0+1)^*.
\end{aligned}$$

Intuitively, B works as follows: B assigns 0 (1) to 0-labeled (1-labeled) leaves; B assigns a 1 to a $\wedge$-labeled node iff all its children are 1; B assigns a 0 to a $\vee$-labeled node iff all its children are 0. Finally, B accepts when the root is labeled with 1. In Figure 4, we give an example of a tree and an accepting run.

(2) The automaton accepting the DTD of Example 1 is defined as follows: $B = (Q, \Sigma, \delta, F)$ with

$$Q := \{\mathtt{starship}, \mathtt{crew}, \mathtt{name}, \mathtt{rank}, \mathtt{job}, \mathtt{species}\},$$

$F := \{\mathtt{starship}\}$, for all $a \in \{\mathtt{name}, \mathtt{job}, \mathtt{species}, \mathtt{rank}\}$, $\delta(a, a) = \{\varepsilon\}$, and

$$\begin{aligned}
\delta(\mathtt{starship}, \mathtt{starship}) &:= \mathtt{crew}^*;\\
\delta(\mathtt{crew}, \mathtt{crew}) &:= \mathtt{name} \cdot \mathtt{species} \cdot (\mathtt{rank} + \mathtt{job}).
\end{aligned}$$

The $\delta(q, a)$ that are not mentioned are empty. □

4.2 Connection with ranked trees

Before we start to develop a theory of regular unranked trees, it makes sense to reflect upon the relationship with regular binary trees. Unranked trees can be uniformly encoded as binary trees. We just mention one possible encoding. See, e.g., Figure 5 for an illustration. Intuitively, the first child of a node remains the first child of that node in the encoding, but it is explicitly encoded as a left child. The remaining children are right descendants of the first child. Whenever there is right child but no left child, a # is inserted. Additionally, when there is only a left child, a # is inserted for the right child.

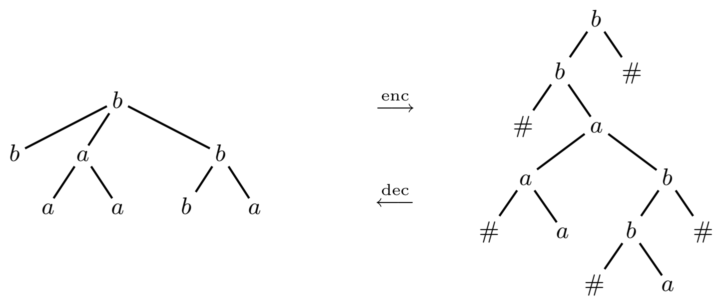


Fig. 5. An unranked tree and its binary encoding.

Using the encodings enc and dec of Figure 5 one obtains the following proposition (we represent the transition functions of NTAs by NFAs).

Proposition 1. [54]

1. *For every unranked NTA B there is a binary tree automaton A such that $L(A) = \{enc(t) \mid t \in L(B)\}$. The size of A is polynomial in the size of B.*
2. *For every binary tree automaton A there is an unranked NTA B such that $L(B) = \{dec(t) \mid t \in L(A)\}$. The size of B is polynomial in the size of A.*

The above proposition allows to transfer all closure properties of the class of ranked tree automata to the class of unranked tree automata.

4.3 Expressiveness and complexity

As enc and dec are MSO definable, Proposition 1 implies that the famous Doner-Thatcher-Wright characterization of ranked tree automata easily carries over to unranked trees [17, 56].

Corollary 1. [43] *A set of trees L is regular iff there is an MSO formula φ such that $L = \{t \mid t \models \varphi\}$.*

Although Proposition 1 provides a tool for transferring results from ranked to unranked trees, it does not deal with issues which are specific for unranked tree automata. The complexity of decision problems for NTAs, for instance, depends on the formalism used to represent the regular string languages $\delta(q, a)$ in the transition function. As there are many ways to represent regular string languages (logical formulas, automata with various forms of control, grammars, regular expressions), Proposition 1 does not seem to offer immediate help. We dwell a bit further upon this issue.

In the following $\mathcal{M}$ always denotes classes of representations of regular languages. So, NTA($\mathcal{M}$) denotes the set of NTAs where the regular languages $\delta(q, a)$ are represented by elements in $\mathcal{M}$. The translation between binary and unranked automata mentioned in Proposition 1 is polynomial for NTA(NFA)'s. For this reason, the latter class can be seen as the default for unranked tree automata. For the latter class, the complexity of the membership problem is quite tractable. The size of an automaton B is $|Q| + |\Sigma| + \sum_{q,a} |\delta(q, a)| + |F|$. By $|\delta(q, a)|$ we mean the size of the automaton accepting $\delta(q, a)$ not the size of the language.

Proposition 2. *Let $t \in \mathcal{T}_\Sigma$ and $B \in$ NTA(NFA). Testing whether $t \in L(B)$ can be done in time $\mathcal{O}(|t||B|^2)$.*

When using tree automata to obtain upper bounds on the complexity of problems related to XML, one sometimes needs to turn to more succinct formalisms. In [32], for instance, a PSPACE upper bound on the complexity of the typechecking problem for structural recursion is obtained by a reduction to the emptiness problem of NTA(2AFA)'s. Here, 2AFA stands for the class of two-way alternating string automata. In this respect, it, therefore, makes sense to explore various possibilities of $\mathcal{M}$. We mention some initial results.

We consider the following well-known decision problems:

- EMPTINESS($\mathcal{M}$): given an NTA($\mathcal{M}$) B, decide whether $L(B) = \emptyset$;
- CONTAINMENT($\mathcal{M}$): given two NTA($\mathcal{M}$)'s B_1 and B_2, decide whether $L(B_1) \subseteq L(B_2)$;
- EQUIVALENCE($\mathcal{M}$): given two NTA($\mathcal{M}$)'s B_1 and B_2, decide whether $L(B_1) = L(B_2)$;

Proposition 3. [32]

1. EMPTINESS(NFA) *is in* PTIME.
2. EMPTINESS(2AFA) *is in* PSPACE.
3. CONTAINMENT(2AFA) *is in* EXPTIME.
4. EQUIVALENCE(2AFA) *is in* EXPTIME.

Theorem 3 is optimal as even for fixed arity trees, emptiness and containment are PTIME-hard and EXPTIME-hard, respectively [12, 52]. Further, emptiness of 2DFAs is known to be hard for PSPACE.

4.4 Applications

Unranked tree automata can serve XML research in at least four different ways:

1. **as a basis of schema languages and validating of schema's.** Murata was the first to consider tree automata as a schema definition language [35]. In fact, the schema language Relax, a competitor of XML schema [15], is directly inspired upon unranked tree automata. The XDuce type system of Pierce and Hosoya [27] as well as the specialized DTDs of Papakonstantinou and Vianu [47] correspond precisely to the unranked tree languages [54]. Lee, Mani, and Murata provide a comparison of XML schema languages based on formal language theory [31].
2. **as an evaluation mechanism for pattern languages.** Several researchers defined pattern languages for unranked trees that can be implemented by unranked tree automata: Neumann and Seidl develop a μ-calculus for expressing structural and contextual conditions on forests [37].[2] They show that their formalism can be implemented by push-down forest automata. The latter are special cases of unranked tree automata. Murata defines an extension of path expressions based on regular expressions over unranked trees [36]. Brüggemann-Klein and Wood consider caterpillar expressions [9]. These are regular expressions that in addition to labels can specify movement through the tree. Neven and Schwentick define a guarded fragment ETL of MSO whose combined complexity is much more tractable than that of general MSO [22, 40, 50]. Expressiveness and complexity results on ETL are partly obtained via techniques based on unranked tree automata.
3. **as an algorithmic toolbox.** For instance, Miklau and Suciu used (binary) tree automata to obtain an algorithm for XPath containment [33]. As mentioned above, Martens and Neven obtain upper bounds on the complexity of type checking by a reduction to the emptiness test of unranked tree automata [32]. Unranked tree automata as a toolbox are hardly developed. It would be helpful to have general results on the complexity of unranked tree automata in terms of the complexity of the regular languages representing the transition functions.
4. **as a new paradigm.** Unranked tree automata use regular string languages to deal with unrankedness. The latter simple but effective paradigm found application in several formalisms.
   Neven and Schwentick define query automata [43]. These are two-way deterministic unranked tree automata that can select nodes in the tree. Query automata correspond exactly to the unary queries definable in monadic second-order logic. By a result of Gottlob and Koch they also correspond to the unary queries definable in monadic datalog [25]. In [38], an extension of the Boolean attribute grammars considered in [45] to unranked trees is defined. These also express precisely the unary queries in MSO. A translation of the region algebra, considered by Consens and Milo [13], into these attribute grammars drastically improves the complexity of the optimization

[2] A forest is a concatenation of unranked trees.

problems of the former. We refer the interested reader to [42] for a more detailed overview of pattern languages based on tree automata.

5 Tree-walking automata

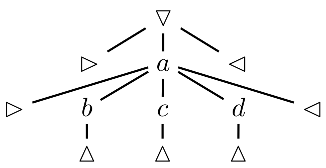
Next, we focus on computation by tree-walking. This is a well-known paradigm from formal language theory studied in the context of attribute grammars and tree-transformations [4, 7, 16]. This paradigm materialized in XML research in various ways. Indeed, a first instance of tree-walking is provided by the caterpillar expressions of Brüggemann-Klein and Wood [9]. Further, Milo, Suciu, and Vianu [34] defined a tree-walking tree-transducer model with pebbles as an abstract model for XML transformations. Segoufin and Vianu considered tree-walking automata in the context of XML streaming [51]. Finally, as argued by Bex, Maneth and Neven [6], stripped down, XSLT is essentially a tree-walking tree-transducer with registers and look-ahead. We embark on the issue of registers and look-ahead in the next section. In the present section, we consider ordinary tree-walking automata.

Admittedly, the application of tree-walking to XML is less direct than that of unranked tree automata. However, we hope that a thorough understanding of the tree-walking paradigm leads to more insight in the operation and expressiveness of languages like XSLT.

5.1 Definition

Before we give the definition of tree-walking automata, let's recall two-way deterministic finite state machines on *strings*: such devices 'walk' in two directions over a string changing state and direction depending on the current state, the current symbol and whether the current position is the left or right-delimiter. An automaton accepts if a final state is reached at some point. Analogously, a *tree*-walking automaton is a finite state device walking a *tree*. Its control is always at one node of the input tree. Based on the label of that node, its state, and its position in the tree (first or last child, root, or leaf), the automaton changes state and moves to one of the neighboring nodes (parent, first child, left or right sibling). The automaton accepts the tree when it enters a final state.

To simplify the definition of two way automata on strings, one usually delimits strings with the start and end symbols $\triangleright$ and $\triangleleft$, respectively. We do the same for trees using the extra symbols $\vartriangle$ and $\triangledown$. For instance, if t is the tree $a(bcd)$ then $\mathrm{delim}(t)$ is the tree



Definition 3. A TWA C is a tuple (Q, Σ, q_0, q_F, P) where

– Q is a finite set of states;
– $q_0 \in Q$ is the initial state and $q_F \in Q$ is the final state;
– P is a finite set of rules of the form $(q, \sigma) \to (q', d)$ where $\sigma \in \Sigma$, $q, q' \in Q$, and $d \in \{\leftarrow, \uparrow, \rightarrow, \downarrow, \text{stay}\}$

Intuitively, a transitions $(q, \sigma) \to (d, q')$ can only be applied in state q at a node labeled with σ. Further, it changes state to q' and moves in direction d where $\leftarrow, \uparrow, \rightarrow, \downarrow$, and stay mean *go to left sibling*, *go to parent*, *go to right sibling*, *go to first child*, and *stay put*, respectively. We assume that there is no transition possible from the final state.

Formally, a configuration on a tree t is a tuple $[u, q]$ where $q \in Q$ is the current state and $u \in \mathrm{Dom}(t)$ is the current node. Before we define the transition relation, we define the (partial) move function m_d for every $d \in \{\leftarrow, \rightarrow, \uparrow, \downarrow, \text{stay}\}$ as follows. For every node u, $m_{\leftarrow}(u)$, $m_{\rightarrow}(u)$, $m_{\uparrow}(u)$, $m_{\downarrow}(u)$, and $m_{\text{stay}}(u)$ equals the left sibling, the right sibling, the parent, the first child of u, and u, respectively (if they exist). Given $\gamma = [u, q]$ and $\gamma' = [u', q']$, we define the one step transition relation $\vdash$ as follows: $\gamma \vdash \gamma'$ iff there is a transition $(q, \sigma) \to (q', d) \in P$ such that $\mathrm{lab}^t(u) = \sigma$ and $m_d(u) = u'$. By $\vdash^*$ we denote the transitive closure of $\vdash$. Finally, C accepts the input tree t if $[\varepsilon, q_0] \vdash^* [\varepsilon, q_F]$.

We say that C is deterministic if there is at most one rule $(q, \sigma) \to \alpha$ in P for every $\sigma \in \Sigma$ and $q \in Q$. Denote the class of deterministic TWAs by DTWA.

We illustrate the above definition with an example.

Example 4. We construct an automaton accepting the tree defined by the XPath expression `//a[//b][//c]`. XPath is an XML pattern language employed by, for instance, XSLT [10]. We do not get into the specifics of the syntax. The present pattern selects all trees with an a-labeled node that has both a b and a c-labeled descendant. Let C be the TWA (Q, Σ, q_0, q_F, P) with $Q = \{q_a, q_b, q_c, q_{\text{root}}, q_F\}$ and $q_0 = q_b$. We give the rules in P while explaining the operation of the automaton. First C nondeterministically searches a b using the following transitions: $(q_b, \sigma) \to (q_b, d)$, for every $\sigma \in \Sigma$ and $d \in \{\leftarrow, \uparrow, \rightarrow, \downarrow, \text{stay}\}$, and $(q_b, b) \to (q_a, \text{stay})$. Then, C moves up the tree until it finds a suitable a: $(q_a, \sigma) \to (q_a, \uparrow)$, for every $\sigma \in \Sigma$ and $(q_a, a) \to (q_c, \downarrow)$. Next, C moves down in search of a c: $(q_c, \sigma) \to (q_c, d)$, for every $\sigma \in \Sigma$ and $d \in \{\rightarrow, \downarrow\}$, and $(q_c, c) \to (q_{\text{root}}, \uparrow)$. Finally, C walks to the root and accepts: $(q_{\text{root}}, \sigma) \to (\sigma, \uparrow)$, for every $\sigma \in \Sigma$ and $(q_{\text{root}}, \triangledown) \to (q_F, \text{stay})$. □

5.2 Expressiveness

Most of the recent research on TWAs focused on ranked TWAs [19, 21, 41], not on unranked ones as defined here. Ranked TWAs are defined just as unranked ones. The only difference is that there is a fixed n such that only trees of rank n are considered as inputs. In particular, a tree has rank n if every node has n or fewer children. However, as with unranked tree automata, we can transfer results between the ranked and unranked case. Let enc and dec be the encoding and decoding discussed in Section 4.

Proposition 4. *1. For every unranked (D)TWA B there is a ranked (D)TWA C such that $L(C) = \{enc(t) \mid t \in L(B)\}$. The size of C is linear in the size of B.*

2. For every ranked (D)TWA C there is an unranked (D)TWA B such that $L(B) = \{dec(t) \mid t \in L(C)\}$. The size of B is linear in the size of C.

It follows from results in [19, 21] that the language accepted by ranked TWAs is regular. From Proposition 1 and Proposition 4, it follows that unranked TWAs only define regular tree languages. Essentially, the latter is all what is known. Even the most basic questions remain unanswered:

1. Do TWAs capture the regular tree languages?
2. Are DTWAs as expressive as TWAs?
3. Are TWAs closed under complement?

It is believed that the answer to all these questions is negative. Again, using Proposition 1 and Proposition 4, it can be shown that a negative answer on the class of unranked tree implies a negative answer on the class of ranked trees (the converse is obvious).

A negative answer to question one is proved in [41] for ranked TWAs that can visit every subtree only once (and a mild generalization thereof). Engelfriet and Hoogeboom supply two tree languages as possible candidates to separate TWAs from NTAs. Fülöp and Maneth [23] showed that the domains of partial attributed tree transducers correspond to the tree-walking automata in universal acceptance mode. No lower bounds on the expressiveness of TWAs have been obtained.

Denote the class of alternating TWAs by ATWA. The next proposition is a useful characterization of the regular unranked tree languages. It immediately follows from the result by Slutzki [53] and the fact that Proposition 4 also holds for alternation.

Proposition 5. *An unranked tree language is accepted by an ATWA iff it is regular.*

TWAs can also be used as an algorithmic toolbox. An upper bound on the non-circularity test of extended attribute grammars is obtained by a reduction to the emptiness test of TWAs.

Theorem 1. [38] *The emptiness, containment and equivalence problem of TWAs are* EXPTIME*-complete.*

We finish with a remark on robustness. A striking difference between ranked and unranked tree automata is that the former can check the label of the parent of the current node by remembering the child number in the state, moving up, and moving back down to the correct child. Seemingly, unranked automata can not achieve this as the child number is unbounded. For this reason, ranked TWAs can evaluate boolean circuits with a fixed fan-in [41] while unranked TWAs, probably, can not evaluate boolean circuits with an unbounded fan-in. However, a proof of the latter would solve question one above. A solution might be to let a move of an unranked TWA depend both on the label of the current node and the label of its parent. The question remains whether this would be the right model for unranked trees.

5.3 A logical characterization

In this section, we present a logical characterization of *ranked* TWAs. We add for every $m > 0$ the unary predicate depth_m to the vocabulary of trees. In all trees, depth_m will contain all vertices the depth of which is a multiple of m.

We characterize tree-walking automata by *transitive closure logic formulas* (TC logic) of a special form. We refer the reader unfamiliar with TC logic to, e.g., [18, 29]. As we only consider TC formulas in normal form, we refrain from defining TC logic in full generality. A TC formula in normal form is an expression of the form

$$\mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon),$$

where φ is an FO formula which may make use of the predicate depth_m, for some m, in addition to E, $<$ and the O_σ. Its semantics is defined as follows, for every tree t,

$$t \models \mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon),$$

iff the pair $(\varepsilon,\varepsilon)$ is in the transitive closure of the relation

$$\{(u,v) \mid t \models \varphi[u,v]\}.$$

We use *deterministic transitive closure logic formulas* (DTC) in an analogously defined normal form to capture deterministic tree-walking automata. In particular,

$$t \models \mathrm{DTC}[\varphi(x,y)](\varepsilon,\varepsilon)$$

iff the pair $(\varepsilon,\varepsilon)$ is in the transitive closure of the relation

$$\{(u,v) \mid t \models \varphi[u,v] \wedge (\forall z)(\varphi[u,z] \rightarrow z = v)\}.$$

The latter expresses that we disregard vertices u that have multiple φ-successors.

As an example consider the formula

$$\varphi(x,y) := (E(x,y) \wedge O_a(x) \wedge O_a(y)) \vee (\mathrm{leaf}(x) \wedge y = \varepsilon).$$

Here, $\mathrm{leaf}(x)$ is a shorthand expressing that x is a leaf. Then, for all trees t, $t \models \mathrm{DTC}[\varphi(x,y)](\varepsilon,\varepsilon)$ iff there is a path containing only a's from the root to a leaf such that every non-leaf vertex on that path has precisely one a-labeled child. In contrast, $t \models \mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon)$ iff there is a path from the root to a leaf carrying only a's.

Theorem 2. *1. A ranked tree language is accepted by a nondeterministic tree-walking automaton iff it is definable by a TC formula in normal form.*

2. A ranked tree language is accepted by a deterministic tree-walking automaton iff it is definable by a DTC formula in normal form.

The simulation in TC-logic is an easy extension of a proof of Potthoff [48] who characterized two-way *string* automata by means of TC formulas in normal form. The latter direction also holds for unranked trees. To show that every TWA can evaluate a TC formula in normal form, we make use of Hanf's Theorem (see, e.g., [18]). This result intuitively says, for graphs of bounded degree, that whether a FO sentence holds depends only on the number of pairwise disjoint spheres of each isomorphism type of some fixed radius. Furthermore, the exact number is only relevant up to a certain fixed threshold, only depending on the formula. As unranked trees do not have bounded degree it is unclear whether the latter result can be extended to unranked trees.

The above result thus implies that any lower bound on (D)TC formulas in normal form is also a lower bound for (non)deterministic tree-walking automata. It is open whether the depth_m predicates are necessary. Unfortunately, proving lower bounds for the above mentioned logics does not seem much easier than the original problem as Ehrenfeucht games for DTC and TC are quite involved [18].

Engelfriet and Hoogeboom showed that tree-walking automata with pebbles correspond exactly to transitive closure logic without restrictions [20]. Hence, when allowing pebbles one can simulate nested TC operators.

6 Tree-walking and data-values

In the previous sections, we primarily focused on the tree structure of XML documents. Our abstraction ignores an important aspect of XML, namely the presence of *data values* attached to leaves of trees or to attributes, and comparison tests performed on them by XML queries. These data values make a big difference – indeed, in some cases the difference between decidability and undecidability (e.g., see [5]). As the connection to logic and automata proved very fruitful in foundational XML research, it is therefore important to extend the automata and logic formalisms to trees with data values.

6.1 Trees and logic revisited

We take a radical view when dealing with text. Indeed, we move all text occurring at leaves into the attributes. For instance, the XML document in Figure 1 can be represented as in Figure 6. Although this approach leads to awkward XML documents, it, nevertheless, remains a valid representation.

```
<starship name="Enterprise">
  <crew id="a457" name="Scotty" species="Human" job="automata"/>
  <crew id="a544" name="Spock" species="Vulcan" job="logic"/>
</starship>
```

Fig. 6. Example of an XML document with all text moved into the attributes.

Next, we add attributes to our Σ-trees. To this end, we assume an infinite domain $\mathbf{D} = \{d_1, d_2, \ldots\}$ and a finite set of attributes A.

Definition 4. An *attributed Σ-tree* is a pair $(t, (\lambda_a^t)_{a \in A})$, where $t \in \mathcal{T}_\Sigma$ and for each $a \in A$, $\lambda_a^t : \mathrm{Dom}(t) \mapsto \mathbf{D}$ is a function defining the a-attribute of nodes in t.

Of course, in real XML documents, usually, not all element types have the same set of attributes. Obviously, this is just a convenience and not a restriction. Further, XML documents can contain elements with mixed content. For instance, consider the XML document

```
<p>This is <em>not</em> a problem.</p>
```

Here, we use the special text label `T` and the attribute `text`, to represent the document by the tree. That is,

```
<p>
    <T text="This is"/>
    <em text="not"/>
    <T text="a problem."/>
</p>
```

In the following, when we say tree we always mean attributed Σ-tree.

For our logics, we make use of the extended vocabulary $\tau_{\Sigma,A} = \{E, <, \prec, (O_\sigma)_{\sigma \in \Sigma}, (\mathrm{val}_a)_{a \in A}\}$. Here, each val_a is a function, from $\mathrm{Dom}(t)$ to $\mathbf{D}$. The logic at hand is based on the logics accompanying the metafinite structures of Grädel and Gurevich [26].

An atomic formula is of the form $E(x,y)$, $x < y$, $x \prec y$, $O_\sigma(x)$, $x = y$, $\mathrm{val}_a(x) = \mathrm{val}_b(y)$ or $\mathrm{val}_a(x) = d$ where $a, b \in A$ and $d \in \mathbf{D}$. Such formulas have the obvious semantics.[3] FO* is obtained by closing the atomic formulas under the boolean connectives and first-order quantification over $\mathrm{Dom}(t)$. As an example consider the FO* sentence $\forall x(\mathrm{val}_a(x) = d \vee \mathrm{val}_a(x) = \mathrm{val}_b(x))$, expressing that the value of every a-attribute is d or is equal to the b-attribute. We stress that no quantification over $\mathbf{D}$ is possible. We get MSO* by extending FO* with set quantification over $\mathrm{Dom}(t)$. To emphasize the difference with the logics of the previous section we use a $*$ and denote the logics by FO* and MSO*, respectively.

6.2 Tree-walking automata extended with registers

To deal with data values, a tree-walking automaton is equipped with a finite number of registers. An automaton can store data-values in registers and can check whether an attribute value of the current node is equal to the content of some register. This is, essentially, the model of Kaminski and Francez who studied string automata over infinite alphabets [30]. Actually, this is also the way XSLT deals with data values. Indeed, the XSLT counterpart of registers are variables that can be passed between templates. We assume that every attribute of a delimiter $\triangleright, \triangledown, \triangleleft, \vartriangle$ contains $\perp$ where $\perp \notin \mathbf{D}$.

[3] $x \prec y$ says that y is a descendant of x.

Definition 5. A *k-register DTWA $\mathcal{B}$* is a tuple (Q, q_0, q_F, τ_0, P) where

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state and $q_F \in Q$ is the final state;
- $\tau_0 : \{1, \ldots, k\} \to \mathbf{D} \cup \{\perp\}$ is the initial register assignment; and,
- P is a finite set of rules of the form $(\sigma, q, \xi) \to \alpha$. Here, $\sigma \in \Sigma$, $q \in Q$, and ξ is a Boolean combination of atomic formulas of the form $j = b$ where $j \in \{1, \ldots, k\}$ and $b \in A$. We define α below.

Intuitively, transitions $(\sigma, q, \xi) \to \alpha$ can only be applied in state q at a node carrying a σ that satisfies ξ under the assignment interpreting j by the content of register j and b by the value of the b-attribute. The right-hand side α can be one of the following:

- (q', d) with $q' \in Q$ and $d \in \{\leftarrow, \rightarrow, \uparrow, \downarrow, \text{stay}\}$; intuitively, this means *change to state q' and move in direction d*; or,
- (q', i, a) where $q' \in Q$, $i \in \{1, \ldots, k\}$, and $a \in A$; intuitively, this means *change to state q', and replace the content of register i by the value of attribute a.*

We assume that no transition is possible from the final state. Further, we assume that the automaton never moves off the input tree.

Given a tree t, a *configuration of $\mathcal{B}$ on t* is a tuple $[u, q, \tau]$ where $u \in \text{Dom}(t)$, $q \in Q$, and $\tau : \{1, \ldots, k\} \to \mathbf{D}$. That is, u is the current node, q the current state, and τ the register content. The *initial* configuration is $\gamma_0 := [\varepsilon, q_0, \tau_0]$. A configuration $[u, q_F, \tau]$ is *accepting.* A rule (σ, p, ξ) applies to a configuration $[u, q, \tau]$ iff $\text{lab}^t(u) = \sigma$, $p = q$ and ξ holds under the interpretation induced by τ where in addition each $a \in A$ is interpreted by $\text{val}_a^t(u)$. We assume that automata are deterministic: if (σ, q, ξ_1) and (σ, q, ξ_2) appear as left-hand sides then there is never a configuration such that both ξ_1 and ξ_2 apply.

Given $\gamma = [u, q, \tau]$ and $\gamma' = [u', q', \tau']$, we define the one step transition relation $\vdash$ as follows: $\gamma \vdash \gamma'$ iff there is a transition $(\sigma, q, \xi) \to \alpha$ that applies to γ and if α is of the form (p, d) then $p = q'$, $m_d(u) = u'$, and $\tau = \tau'$; otherwise, if α is of the form (q, i, a) then $p = q'$, $u = u'$, $\tau'(i) = \text{val}_a^t(u)$, and $\tau'(j) = \tau(j)$ for all $j \neq i$. By $\vdash^*$ we denote the transitive closure of $\vdash$.

Finally, $\mathcal{B}$ accepts the input tree t if $\gamma_0 \vdash^* \gamma$ for some accepting configuration γ.

Example 5. Consider the tree-walking automaton that checks whether there is a node with the same a-attribute as the root. We assume $\Sigma = \{\sigma\}$ and $A = \{a\}$. The automaton starts by putting the value of the a-attribute of the root in the first register; subsequently, it makes a depth-first traversal of the tree; it accepts when it encounters a node with the same a-value as in the first register. Define $\mathcal{B} = (Q, q_0, Q_F, \tau_0, P)$ as the one-register automaton where $Q = \{q_0, q_{\text{down}}, q_{\text{up}}, q_F\}$, $Q_F = \{q_F\}$, $\tau(1) = \perp$ and P contains the following rules:

$$
\begin{array}{ll}
(q_0, \triangledown, \text{true}) & \to (q_0, \downarrow) \\
(q_0, \triangleright, \text{true}) & \to (q_0, \rightarrow) \\
(q_0, \sigma, \text{true}) & \to (q_{\text{down}}, 1, a) \\
\\
(q_{\text{down}}, \sigma, 1 = a) & \to (q_F, \text{stay}) \\
(q_{\text{down}}, \sigma, \neg(1 = a)) & \to (q_{\text{down}}, \downarrow) \\
(q_{\text{down}}, \triangleright, \text{true}) & \to (q_{\text{down}}, \downarrow) \\
(q_{\text{down}}, \triangle, \text{true}) & \to (q_{\text{up}}, \uparrow) \\
(q_{\text{down}}, \triangleleft, \text{true}) & \to (q_{\text{up}}, \uparrow) \\
\\
(q_{\text{up}}, \sigma, \text{true}) & \to (q_{\text{down}}, \rightarrow) \\
(q_{\text{up}}, \triangledown, \text{true}) & \to (q_{\text{up}}, \text{stay})
\end{array}
$$

□

6.3 Expressiveness

The expressiveness of k-register DTWAs behaves in a strange way: on the one hand, they can compute properties not in MSO*, while on the other hand they cannot even compute all FO*-definable properties. This is a bit awkward as, in database theory, first-order logic is generally accepted as the minimum expressiveness a query language should have, while MSO, due to its correspondence with various automata, stands for regularity and robustness.

Interestingly, the inexpressibility proof makes use of communication complexity [28]. The latter technique is inspired by a proof of Abiteboul, Herr, and Van den Bussche [3] separating the temporal query languages ETL from TS-FO. In particular, they show that every query in ETL on a special sort of databases can be evaluated by a communication protocol with a constant number of messages, whereas this is not the case for TS-FO. To simulate k-register DTWAs we need a more powerful protocol where the number of messages depends on the number of different data values in the input, but the idea is essentially the same. We sketch the argument below.

Theorem 3. [44]

1. *MSO* cannot define all properties computable by k-register DTWAs; and,*
2. *k-register DTWAs cannot compute all properties definable in FO*.*

Proof. (Sketch of (2)) We can already separate k-register DTWAs and FO* on strings as opposed to trees. In communication complexity the input string is divided in a pre-determined manner between two parties (generally referred to as I and II) that can send messages to each other according to a given protocol. A language is accepted by a protocol if for each string both parties can decide after execution of the protocol whether the string belongs to the language. Both parties have unlimited computation power on their part of the string. The protocol only restricts the way in which the parties communicate, typically by restricting the form and number of messages.

We consider strings of the form $f\#g$ where f and g encode sets of sets of **D**-symbols in a suitable way. For instance, the string $\$\sigma\sigma\$\sigma\$$ where the attribute values of the σ-symbols are a, b, and c, respectively, encodes the set $\{\{a, b\}, \{c\}\}$. The language

$$L := \{f\#g \mid f \text{ and } g \text{ represent the same set of sets}\}$$

is definable in FO*. To show that the language is not accepted by a k-register DTWA, we note that each such automaton working on strings of the form $f\#g$ can be simulated by a protocol in the following way: I is given f while II is given g. The first party simulates the automaton until this computation tries to cross the delimiter # to the right. At this point, it sends the present state q and the data values $d_1, \ldots, d_k$ currently in its registers. Hence, II gets full information about the configuration of the automaton (as the position of the symbol # is fixed). Then II sends in turn the current configuration to I. This process continues until one of the parties detects a final state. What kind of protocol can simulate such behavior? First, we need a message for every configuration. Suppose we restrict to at most N different data values in the strings $f\#g$. Then $M := |Q| \cdot N^k$ different messages are needed. Here, k is the number of registers and Q is the state set. Call a sequence of messages a dialogue. We only need to consider dialogues up to length M (as every message can only be sent once in every direction). Hence, there are only M^{2M} different dialogues on input strings consisting of N different **D**-symbols. The latter value is exponential in N. However, there are 2^{2^N} sets of sets of N different **D**-symbols. So for large enough N there must be different strings $f\#f$ and $g\#g$ with $f \neq g$ accepted by the protocol via the same dialogue. But, this means that $f\#g$ is also accepted. Hence, no such protocol can define L, which implies that no k-register DTWA accepts L. □

Actually, Neven, Schwentick, and Vianu [44] do not consider register automata on trees but on strings. However, as the separation results hold for strings they definitely hold for trees. In addition, the authors studied automata with various control mechanisms: non-determinism, alternation, one-way and two-way. In fact, the communication complexity technique sketched above can be extended to show that even *alternating* k-register automata cannot compute all FO*-definable properties. As an alternative to registers, the authors consider pebbles for dealing with data values. Every automaton is equipped with a finite number of pebbles whose use is restricted by a stack discipline. That is, pebble i can only be lifted when pebble $i + 1$ is not placed. Further, the automaton can test equality by comparing the attribute values of the pebbled symbols. It turns out that pebble automata behave much better than register automata: their expressiveness lies between FO* and MSO*, so they are neither too strong nor too weak.

6.4 Subcomputations and relational storage

In [39], we consider an extension of the register based model which is closer to XSLT. To be precise, DTWAs are extended in two ways: (i) registers can store arbitrary relations over $\mathbf{D}$ (as opposed to single $\mathbf{D}$-values); (ii) subcomputations can be started. A transition rule is of the form $(\sigma, q, \xi) \to \alpha$, where $\sigma \in \Sigma$, $q \in Q$, and ξ is an FO formula over the relational storage. So, if for instance, the relational storage contains one set X_1 and ξ is the formula $\forall x \forall y (X_1(x) \wedge X_1(y) \to x \neq y)$, then the above transition will be applied if the current symbol is σ, the current state is q and X_1 contains at most one value. The right-hand side α can determine three kinds of actions:

1. a move: $\alpha = (q', d)$ where d is a direction and q' a state;
2. a change of the relational storage: $\alpha = (q', \psi, i)$ where q' is a state, ψ an FO formula over the relational storage and the attribute values of the current node, and i is the number of a register. The intended meaning is that the content of register i is replaced by the relation defined by ψ;
3. a subcomputation: $\alpha = (q', atp(\varphi(x,y), p), i)$, where q', p are states, i is the number of a register and $\varphi(x, y)$ is an FO($\exists$) formula over the tree (extended with some other predicates). The logic FO($\exists$) functions as an abstraction of XPath. Supppose the current node is u. Intuitively, register i is replaced by the result of $\mathrm{atp}(\varphi(x, y), p)$; the latter, starts ℓ subcomputations at the nodes $\{u_1, \ldots, u_\ell\} = \{v \mid t \models \varphi(u, v)\} \subseteq \mathrm{Dom}(t)$; these computations are started in state p and with the current relational store; when they end in a final state, the contents of the first register is returned; the content of register i is then the union of the results of all subcomputations. The main thread then resumes computation at u.

We do not define this model formally but only illustrate it by means of an example.

Example 6. Assume $\Sigma = \{\sigma, \delta\}$ and $A = \{a\}$. We define an automaton that accepts a tree if for every δ-labeled node all its leaf-descendants have the same a-attribute. By leaf-descendants we do not mean nodes labeled with $\vartriangle$ but the parents of those nodes. We define a 1-register automaton where the register X_1 is a set. Let $Q = \{q_0, q_1, q_2, q_3, q_4, q_F\}$, and $\tau_0(1) = \emptyset$. P consists of the following rules:

$$(\triangledown, q_0, \mathrm{true}) \to (q_1, \mathrm{atp}(\varphi_1, q_2), 1) \tag{1}$$
$$(\triangledown, q_1, \mathrm{true}) \to (q_F, \mathrm{stay}) \tag{2}$$
$$(\delta, q_2, \mathrm{true}) \to (q_3, \mathrm{atp}(\varphi_2, q_4), 1) \tag{3}$$
$$(\delta, q_3, \xi) \to (q_F, \mathrm{stay}) \tag{4}$$
$$(\delta, q_4, \mathrm{true}) \to (q_F, x = a, 1) \tag{5}$$
$$(\sigma, q_4, \mathrm{true}) \to (q_F, x = a, 1) \tag{6}$$

where $\varphi_1 \equiv x \prec y \wedge O_\delta(y)$, $\varphi_2 \equiv \exists y_1 (x \prec y \wedge E(y, y_1) \wedge O_\vartriangle(y_1)$, and $\xi \equiv \exists x X_1(x) \wedge \forall x \forall y (X_1(x) \wedge X_1(y) \to x \neq y)$.

The automaton works as follows: (1) a subcomputation is initiated that selects all δ-labeled descendants of the root; (2) when all subcomputations return, that is, state q_1 is reached, the tree is accepted; (3) every δ-labeled node selects all leaves (recall that we work with delimited trees); (4) when the returned set is a singleton, the subcomputation accepts (otherwise, the subcomputation gets stuck and the main computation rejects); as (5) and (6) make sure that every leaf returns the value of its a attribute, the computations initiated by (3) accept in (4) iff every leaf has the same a-attribute. Note that $x = a$ is the formula that defines the set containing the value of the a-attribute of the current node. $\square$

Allthough the present model seems quite powerful, the additions still do not suffice to capture FO*:

Theorem 4. *DTWA automata extended with look-ahead and relational storage do not capture FO*.*

Again, the proof of the latter theorem is based on communication complexity. However, the protocol is no longer memory-less, as is the case in the proof of Theorem 3: both parties need a stack to process incoming messages.

The weakness of register automata is that when they leave a node, they usually cannot relocate that node. In strong contrast, if there is a fixed attribute such that for every node the value of that attribute is unique among all nodes in a tree, that is, unique ids are available, then we show that various restrictions capture natural complexity classes:

Theorem 5. *In the presence of unique ids,*

1. *DTWAs extended with single-valued registers capture* LOGSPACE*;*
2. *DTWAs extended with single-valued registers and subcomputations capture* PTIME*;*
3. *DTWAs extended with relational storage capture* PSPACE*; and,*
4. *DTWAs extended with relational storage and subcomputations capture* EXPTIME.

Actually, the above characterizations are not obtained for the mentioned standard complexity classes but for a Turing Machine model directly operating on attributed trees. It can be shown that the latter and the standard model recognize the same class of tree languages. Although the proofs of the above results are combinations of known techniques in complexity, finite model theory, and formal languages, they provide a quite complete picture of the expressiveness of query languages based on tree-walking. The most surprising might be that DTWAs extended with single-valued registers and subcomputations, which is the abstraction of XSLT defined in [6], captures in fact precisely PTIME.

7 Discussion

We considered three automata models that regained interest by the advent of XML. Our main focus was on their connection with logic and on questions motivated by XML. We hope to have convinced the reader that XML poses new challenges on the automata and logic connection. In fact, the application of automata theory in XML research has only just started.

Indeed, although unranked tree automata have found already many applications, apart from the work of Brüggemann-Klein, Murata, and Wood [8], no systematic study has been undertaken. Especially the development of their algorithmic properties deserves much more attention.

Not much is known about tree-walking automata and not that many techniques are available. Solving the questions in Section 5.2 would be an interesting starting point. However, as these questions are open for quite some time, they appear to be difficult. The connection with TC-logic made in Theorem 2 learns that the TWA–NTA problem is an "easier" instance of the open question whether (full) unary TC logic and MSO on trees are equally expressive. It is even more of a mystery, whether, and in which way, the addition of pebbles to the formalism, as in [34], increases the expressiveness. The core of XPath, for instance, can easily be expressed by DTWAs with pebbles. It is unclear how many are needed (if they are needed at all).

Tree-walking automata with registers can serve as an abstraction of transformation languages like XSLT. Characterizing their expressiveness is, hence, meaningful. However, as an algorithmic toolbox, register automata are worthless as almost all decision problems are undecidable [44]. Nevertheless, in the context of streaming [51] or typechecking [5] in the presence of datavalues, it would be interesting to find the most expressive formalism for which emptiness would remain decidable. The inexpressibility results in [44] and [39] are obtained via communication complexity. As illustrated in the proof of Theorem 3, such a proof consists of two parts: (1) showing that your formalism can be simulated by a protocol; (2) no protocol can express your property. In both case, step (2) is rather straightforward, while (1) is the most involved one (especially in [39]). It would be interesting to come up with general criteria from which a simulation lemma can be derived automatically, rather than proving the simulation lemma by hand for every new model.

Acknowledgment

I thank Thomas Schwentick and Stijn Vansummeren for their comments on a previous version of this paper.

References

1. S. Abiteboul. Semistructured data: from practice to theory. In *Proc. 16th IEEE Symposium on Logic in Computer Science (LICS 2001)*, pages 379–386, 2001.
2. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
3. S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal connectives versus explicit timestamps to query temporal databases. *Journal of Computer and System Sciences*, 58(1):54–68, 1999.
4. A. V. Aho and J. D. Ullman. Translations on a context-free grammar. *Inform. and Control*, 19:439–475, 1971.

5. N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: Typechecking revisited. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 560–572, 2001.
6. G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27(1):21–39, 2002.
7. R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1):1–50, 2000.
8. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, april 3, 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.
9. A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2(1):81–106, 2000.
10. J. Clark. XML Path Language (XPath). `http://www.w3.org/TR/xpath`.
11. E. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.
12. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.
13. M. Consens and T. Milo. Algebras for querying text regions: Expressive power and optimization. *Journal of Computer and System Sciences*, 3:272–288, 1998.
14. World Wide Web Consortium. Extensible Markup Language (XML). `http://www.w3.org/XML/`.
15. World Wide Web Consortium. XML schema. `http://www.w3.org/XML/Schema`.
16. P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definition, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer, 1988.
17. J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.
18. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.
19. J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In J. Karhumki, H. Maurer, G. Paun, and G.Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pages 72–83. Springer-Verlag, 1999.
20. J. Engelfriet and H. J. Hoogeboom. Private communication. 2002.
21. J. Engelfriet, H.J. Hoogeboom, and J.-P. van Best. Trips on trees. *Acta Cybernetica*, 14:51–64, 1999.
22. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proc. 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, 2002.
23. Z. Fülöp and S. Maneth. Domains of partial attributed tree transducers. *Information Processing Letters*, 73(5–6):175–180, 2002.
24. F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997.
25. G. Gottlob and C. Koch. Monadic datalog and the expresive power of languages for web information extraction. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 17–28. ACM Press, 2002.
26. E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998.

27. H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. In *Proceedings of 28th Symposium on Principles of Programming Languages (POPL 2001)*, pages 67–80. ACM Press, 2001.
28. J. Hromkovic. *Communication Complexity and Parallel Computing.* Texts in Theoretical Computer Science - An EATCS Series. Springer-Verlag, 2000.
29. N. Immerman. *Descriptive Complexity.* Springer, 1998.
30. M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
31. D. Lee, M. Mani, and M. Murata. Reasoning about XML schema languages using formal language theor. Technical report, IBM Almaden Research Center, 2000. Log# 95071.
32. W. Martens and F. Neven. Typechecking top-down uniform unranked tree transducers. Manuscript.
33. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 65–76, 2002.
34. T. Milo, D. Suciu, and V. Vianu. Type checking for XML transformers. In *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems*, pages 11–22. ACM Press, 2000.
35. M. Murata. Data model for document transformation and assembly. In E. V. Munson, K. Nicholas, and D. Wood, editors, *Proceedings of the workshop on Principles of Digital Document Processing*, volume 1481 of *Lecture Notes in Computer Science*, pages 140–152, 1998.
36. M. Murata. Extended path expressions for xml. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 126–137. ACM Press, 2001.
37. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 134–145. Springer, 1998.
38. F. Neven. Extensions of attribute grammars for structured document queries. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming (DBPL'99)*, volume 1949 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2000.
39. F. Neven. On the power of walking for querying tree-structured data. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 77–84. ACM Press, 2002.
40. F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proc. 19th Symposium on Principles of Database Systems (PODS 2000)*, pages 145–156, 2000.
41. F. Neven and T. Schwentick. On the power of tree-walking automata. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *International Colloquium on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2000.
42. F. Neven and T. Schwentick. Automata- and logic-based pattern languages for tree-structured data. Unpublished, 2001.
43. F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.
44. F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *Lecture Notes in Computer Science*, pages 560–572. Springer, 2001.

45. F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1), 2002.
46. C. Pair and A. Quere. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.
47. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 35–46. ACM Press, 2001.
48. A. Potthoff. *Logische Klassifizierung regulärer Baumsprachen.* Doctor's thesis, Institut f"ur Informatik u. Prakt. Math., Universit"at Kiel, 1994.
49. E. T. Ray. *Learning XML.* O'Reilly, 2001.
50. T. Schwentick. On diving in trees. In *Proceedings of 25th Mathematical Foundations of Computer Science (MFCS 2000)*, pages 660–669, 2000.
51. L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 53–64. ACM Press, 2002.
52. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.
53. G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41(2–3):305–318, 1985.
54. D. Suciu. Typechecking for semistructured data. In *Proceedings of the 8th Workshop on Data Bases and Programming Languages (DBPL 2001)*, 2001.
55. M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27(1):1–36, 1975.
56. J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.
57. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7, pages 389–456. Springer, 1997.
58. J. Van den Bussche. Applications of Alfred Tarski's ideas in database theory. In L. Fribourg, editor, *Computer Science Logic (CSL 2001)*, volume 2142 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2001.
59. M. Y. Vardi. Automata theory for database theoreticians. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 83–92. ACM Press, 1989.
60. V. Vianu. Databases and finite-model theory. In N. Immerman and Ph. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 97–148. American Mathematical Society, 1997.
61. V. Vianu. A web odyssey: From Codd to XML. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 1–15, 2001.