Semistructured data, Logic, and Automata (course notes)

Diego Figueira
CNRS & LaBRI

Abstract

This is a brief overview of the material covered in a lecture given at EPIT'19 Spring School.[1]

Summary: *Semistructured data is an umbrella term encompassing data models which are not logically organized into tables (i.e., the relational data model) but rather in hierarchical structures using markers such as tags to separate semantic elements and data fields in a 'self-describing' way. In this lecture we survey some of the multiple connections between formal language theory and semi-structured data, in particular concerning the XML format. We will cover ranked and unranked tree automata, and its connections to Monadic Second Order logic, First Order logic, and XPath. The aim is to take a glimpse at the landscape of closure properties, algorithms and expressiveness results for these formalisms.*

Keywords and phrases XML, tree automata, MSO, FO, XPath

Acknowledgement

This course is inspired by course slides from Wim Martens and Stijn Vansummeren [29], and Thomas Schwentick [37], and by many surveys on the subject, in particular due to Frank Neven [33, 34], Leonid Libkin [28], Thomas Schwentick [38, 39], Luc Segoufin [40, 41] —and this is not an exhaustive list. Thanks to them!

1 XML

An XML document is basically a data-exchange format which is very flexible and simple. XML documents can be seen:

- As graphs, more precisely as finite unranked ordered trees. Nodes correspond to elements, while leaf nodes contain in general arbitrary text. We will often work with the *structure* of the document and thus ignore the text in leaf nodes.
- As terms, especially when the tree is ranked.
- As a string with well-formed opening and closing 'tags' (which leads to consider models such as Visibly Pushdown Automata [1], which are essentially equivalent to the tree automata on which we focus).

On these documents we are interested in:

- Extracting information though query languages (*e.g.*, XPath and XQuery).
- Expressing constraints: sometimes the document structure needs to be restricted (for example to disallow documents which would not lead to a correct interpretation). These constraint languages are usually called *schema languages*. For example DTD's, XML Schema, Relax NG are some schema languages.

[1] Spring School on Theoretical Computer Science EPIT (*Ecole de Printemps d'Informatique Théorique*) on Databases, Logic and Automata. 8–12 April 2019. CIRM, Luminy, France. `https://conferences.cirm-math.fr/1934.html`

Why study automata and logics on trees?

- Because XML documents are, essentially, trees.
- Because tree automata are a formal principled model of computation on trees.
- Because logics and automata provide the foundations of query design, evaluation, and optimization.
- Because these formalisms can be used to describe and reason on schema languages.
- Because they offer a toolbox of fundamental algorithms, complexity bounds, and they are suitable for studying the expressive power of query languages. For example, schema validation (whether a document validates a schema) can be seen as whether a finite tree verifies an MSO property.

2 Tree Automata

What's a tree? A class of trees can be classified as
ordered if every node has a *sequence* of children trees (*a.k.a. hedge*), or
unordered if every node has a *multiset* of children trees (*a.k.a. forest*); and
k-ranked if every node of every tree has at most k children, or
unranked if there is no bound on the number of children of nodes.

There are many definitions of automata on trees, some of them equivalent.

- They can walk the tree or work on different branches "in parallel",
- work the computation bottom-up or top-down,
- be deterministic or non-deterministic,
- be used as "language acceptors" or as "querying devices" by selecting some nodes as output.

We give a small summary of the main definitions, properties and algorithms for tree automata. For more details on tree automata, see [12].

2.1 Ranked tree automata

A **non-deterministic ranked tree automaton** (NTA) is a tuple consisting of a finite set of states Q, a finite alphabet $\mathbb{A}$, a distinguished subset $F \subseteq Q$, and a finite set of transitions $\delta \subseteq Q^* \times \mathbb{A} \times Q$. An accepting run on a tree t is a mapping from nodes of t to Q so that (1) the root has a state from F, and (2) for every every node with label a and state q having k children with states $q_1, \ldots, q_k$ we have $(q_1 \cdots q_k, a, q) \in \delta$. Note that since δ is finite, the tree language accepted by such an automaton is always ranked. Note also that one can think of a run as being top-down having F as initial states and $S = \{q \mid (\varepsilon, a, q) \in \delta\}$ as final states, or as being bottom-up, having S as initial states and F as final states. The class of languages recognized by NTA are called **regular tree languages**.

2.1.1 Pumping lemma

A tree with one distinguished leaf is called a **context**. Given a context C and a tree t the application of C to t, denoted by $C[t]$, is the tree resulting when replacing the distinguished leaf of C with the tree t. In particular if C has just one (distinguished) node, $C[t] = t$, in which case we say that C is the **empty context**. The application can be composed or iterated, *e.g.*, $C^3[t] = C[C[C[t]]]$.

▶ **Lemma 1** (Pumping Lemma). *For every regular tree language L there is $k \in \mathbb{N}$ so that for every tree $t \in L$ of height $\geq k$ there is a tree t' and non-empty contexts C_1, C_2 where*
1. $t = C_1[C_2[t']]$ *and*
2. $C_1[C_2^n[t']] \in L$ *for every* $n \in \mathbb{N}$.

2.1.2 Myhill-Nerode characterization

For any tree language L, we write $t \equiv_L t'$ if for every context C, $C[t] \in L$ iff $C[t'] \in L$.

▶ **Lemma 2** (Myhill-Nerode). *L is a regular tree language if and only if $\equiv_L$ has finite index (i.e., finitely many equivalence classes).*

Therefore, if L is regular, it is the union of some (finite number of) equivalence classes of $\equiv_L$. The minimal automaton can be then defined by taking one distinct state per $\equiv_L$ equivalence class.

2.1.3 Determinism

Two forms of determinism a tree automaton can be:

deterministic bottom-up (DTA) if it has no two transitions $(\gamma, a, q), (\gamma', a', q')$ with $\gamma = \gamma'$ and $a = a'$. In expressive power, NTA = Deterministic Bottom-up TA (DTA) = "regular tree languages".
Determinizing a NTA takes exponential time through the powerset construction: From a NTA $(\mathbb{A}, Q, F, \delta)$ construct $(\mathbb{A}, 2^Q, \{S : S \cap F \neq \emptyset\}, \delta')$ with $\delta' = \{(S_1 \cdots S_n, a, \{s : (s_1 \cdots s_n, a, s) \in \delta \text{ for } s_1 \in S_1, \ldots, s_n \in S_n\}) : S_1, \ldots, S_n \subseteq Q, a \in \mathbb{A}\}$.

deterministic top-down if it has no two transitions $(\gamma, a, q), (\gamma', a', q')$ with $q = q'$, $a = a'$ and $|\gamma| = |\gamma'|$. In expressive power, Deterministic top-down TA $\subsetneq$ NTA. For example, the language "there is a leaf with letter a" cannot be expressed. Their weakness comes from the fact in this model there is no way to take information coming from two different paths in order to define its behavior. Basically, it can only express whether a regular property holds on all its branches (extended with information about being a left or right sibling) [49].[2] As a consequence, they are not closed under complement.

2.1.4 Closure properties

NTA languages are closed under intersection, union and complement.

Intersection in PTime. Like for word automata, from two NTA $(\mathbb{A}, Q_1, F_1, \delta_1)$ and $(\mathbb{A}, Q_2, F_2, \delta_2)$ we can produce, in quadratic time, a NTA recognizing the languages intersection: $(\mathbb{A}, Q_1 \times Q_2, F_1 \times F_2, \delta)$, where $\delta = \{(\gamma_1 \otimes \gamma_2, a, (q_1, q_2)) : |\gamma_1| = |\gamma_2| \text{ and } (\gamma_i, a, q_i) \in \delta_i \text{ for each } i\}$.[3]

Union in PTime. This time we produce $(\mathbb{A}, Q_1 \dot\cup Q_2, F_1 \dot\cup F_2, \delta_1 \cup \delta_2)$.

Complement in ExpTime. Idea:
1. Determinize it (state blowup);
2. complete it (transition blowup), by adding a new non-final state $q_\perp$ and all transitions
$$(Q \cup \{q_\perp\})^{\leq k} \times \mathbb{A} \times \{q_\perp\} \setminus \{(\gamma, a, q_\perp) : \exists q \in Q \text{ s.t. } (\gamma, a, q) \in \delta\};$$
3. switch final and non-final states.

On the other hand, complementing DTA is in PTime, since the last two items are computable in PTime (the rank k is fixed).

[2] This is also true for unranked trees [14].
[3] We write $\otimes$ for the convolution operator, defined as $q_1 \cdots q_n \otimes p_1 \cdots p_n = (q_1, p_1) \cdots (q_n, p_n)$.

2.1.5 Decision problems

(Uniform) Membership Given a NTA [resp. DTA] $\mathcal{A}$ and a tree t, is $t \in \mathcal{L}(\mathcal{A})$? It is in PTIME [resp. in linear time].

More precisely, for NTA it is logCFL-complete: log-space inter-reducible to context-free grammars; and for DTA it is in logDCFL (the precise complexity is unknown). This is because it can be solved in logarithmic space and polynomial time if we have access to a stack (which is not subject to the logarithmic bound). This is known to be a characterization of logCFL [45].

Emptiness Given a NTA [resp. DTA] $\mathcal{A}$, is $\mathcal{L}(\mathcal{A}) = \emptyset$? It is PTIME-complete for both models. Idea for upper bound: saturation algorithm on the set of reachable states. Idea for lower bound: reduction from "Path Systems", which is a well-studied PTIME-complete problem [13] of whether some proposition is 'provable' from some set of axioms and rules, which resemble a lot to leaf states and binary transitions, respectively.

Finiteness Given a NTA [resp. DTA] $\mathcal{A}$, is $|\mathcal{L}(\mathcal{A})| = \infty$? It is PTIME-complete for both models. Idea for upper bound: we find useful states (states that appear in accepting runs) in PTIME, and we see if there is a loop on useful states in NL. Lower bound: reduce from emptiness.

Universality For a fix rank k and alphabet $\mathbb{A}$, given a NTA [resp. DTA] $\mathcal{A}$, is $\mathcal{L}(\mathcal{A}) =$ the set of all k-ranked trees on $\mathbb{A}$? It is EXPTIME-complete [resp. in PTIME]. For the EXPSPACE lower bound: reduction from halting of alternating linear space bounded Turing machines, where the tree branching takes care of alternation. The language consists of all trees which are not correct encodings of halting runs, *i.e.*, the automaton tests whether there is an 'error' in the computation or in the encoding. The result follows since APSPACE = EXPTIME [11]. For the upper bound, complement and test for emptiness.

Containment Given two NTA [resp. DTA] $\mathcal{A}, \mathcal{B}$ is $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$? Again, it is EXPTIME-complete [resp. in PTIME], hardness comes from universality. For the upper bound, we test $L(\mathcal{A}) \cap \overline{L(\mathcal{B})} = \emptyset$. Lower bound: from universality.

Equivalence Given two NTA [resp. DTA] $\mathcal{A}, \mathcal{B}$ is $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$? Again, it is EXPTIME-complete [resp. in PTIME], hardness comes from universality, membership from containment.

Emptiness of intersection Given NTA [resp. DTA] $\mathcal{A}_1, \ldots, \mathcal{A}_n$, is $\mathcal{L}(A_1) \cap \cdots \cap \mathcal{L}(A_n) = \emptyset$? It is EXPTIME-complete for both models. Upper bound: construct n-fold product automaton and test for emptiness. For the lower bound again we reduce from simulating an alternating n-space Turing machine. The idea is that $\mathcal{A}_i$ ensures that the simulation is correct in the neighborhood of the i-th position of the tape.

Minimization Not really a decision problem... It is the task of, given a NTA [resp. DTA], producing an equivalent minimal automaton. It can be done in EXPTIME [resp. in PTIME] by computing the equivalence class on the statespace corresponding to the Myhill-Nerode equivalence class $\equiv_L$ of the language L recognized by the automaton. From a complete and reduced DTA[4], we start with the equivalence relation $(\sim_0) = F^2 \cup (Q \setminus F)^2$ and we iteratively refine it $(\sim_0) \supsetneq (\sim_1) \supsetneq \cdots$ until we reach some n (bounded by the number of states) so that $(\sim_n) = (\sim_{n+1})$. We then replace each state with its $\sim_n$-equivalence class and we obtain the minimal automaton. The refinement is

[4] A DTA is reduced if every state is reachable, that is, for each state there is a tree whose partial run maps the root to that state.

defined, at each step, as $p \sim_{i+1} q$ if: (1) $p \sim_i q$, and (2) for every $\gamma, \gamma' \in Q^*$, $r_1, r_2 \in Q$, and $a \in \mathbb{A}$ so that $(\gamma p \gamma', a, r_1), (\gamma q \gamma', a, r_2) \in \delta$ we have $r_1 \sim_i r_2$.

2.1.6 Alternation, two-wayness

As for finite word automata, there is an alternating and a two-way version of NTA, which does not increase its expressive power (but its succinctness). See *e.g.*, [48].

2.1.7 A different perspective: try walking

Tree walking automata (TWA) are another, perhaps even more natural, generalization of finite automata on words to trees. On binary trees it is defined as a non-deterministic word automaton with a transition set

$$\delta \subseteq Q \times \{\textit{the root}, \textit{not the root}\} \times \{\textit{a leaf}, \textit{not a leaf}\} \times \mathbb{A} \times \{\textit{left child}, \textit{right child}, \textit{parent}\} \times Q,$$

with the expected semantics: a transition (q, r, l, a, d, q') says "*If I'm in state q on a node which is r and l and has letter a, then go to d with state q'*". There are deterministic and non-deterministic versions in the usual sense. In terms of expressive power,

$$\text{det. TWA} \subsetneq_{[3]} \text{non-det. TWA} \subsetneq_{[4],[8]} \text{nested TWA} \subsetneq_{[10]} \text{regular tree languages.}$$

A nested TWA is parameterized by a rank $k \in \mathbb{N}$, and it is, intuitively, "a non-deterministic TWA $\mathcal{A}$ that has finitely many sub-automata of rank less than k, and such that each transition of $\mathcal{A}$ may be conditional on whether some of the sub-automata do or do not have an accepting run from the current node, either in general or within the subtree rooted at the current node." [10] We do not give the definition here.

With respect to closure properties, all these models are closed under union and intersection. On the other hand, while deterministic TWA are closed under complement, it is not known whether the same holds for non-deterministic TWA. Finally, nested TWA are (trivially) closed under complement.

It is easy to see that alternating extension of TWA define, exactly, the class of tree regular languages. So we conclude that adding non-determinism and/or alternation to TWA increases its expressive power.

2.2 Unranked tree automata

But wait a second, XML documents are *unranked* trees! Most commonly occurring XML documents are in fact shallow.

2.2.1 Definition

A **non-deterministic unranked tree automaton** is defined similarly as on ranked trees: a tuple consisting of a finite set of states Q, a finite alphabet $\mathbb{A}$, a distinguished subset $F \subseteq Q$, and a set of transitions $\delta \subseteq Q^* \times \mathbb{A} \times Q$, which can now be *infinite*, given as a finite set of triples (L, a, q), where $L \subseteq Q^*$ is regular —usually represented as an NFA on words. An accepting run is defined as before.

2.2.2 Deterministic version

If we define the deterministic version as for ranked automata (*i.e.*, there are no two transitions $(\gamma, a, q), (\gamma, a, q') \in \delta$ with $q \neq q'$), we obtain the class of **blockwise deterministic** unranked tree automata. However, note that this definition does not imply that the languages L of the triples (L, a, q) of δ are defined by deterministic automata, and in general they don't admit a canonical minimal automaton. There exists another, more restrictive version of determinism, the **stepwise determinism**, which is closer to the intuition of determinism (and which we're not defining here, but you can find the definition in [12]).

2.2.3 Bridging with ranked tree automata

There are many ways to encode unranked tree as binary tree. One possibility is the first-child/next-sibling (fcns) encoding, in which any tree is represented as the binary tree whose left child is corresponds to the leftmost child relation, and the right child corresponds to the next-sibling relation. Note that the fcns encoding is injective (*i.e.*, for any two trees t, t', $\mathit{fcns}(t) = \mathit{fcns}(t')$ implies $t = t'$). This encoding allows to easily transfer results back and forth thanks to this lemma:

▶ **Lemma 3.** *[44]*

- *For every unranked NTA $\mathcal{A}$ there is a binary NTA $\mathcal{B}$ computable in linear time so that $L(\mathcal{B}) = \{\mathit{fcns}(t) : t \in L(\mathcal{A})\}$;*
- *For every binary NTA $\mathcal{B}$ there is an unranked NTA $\mathcal{A}$ computable in linear time so that $L(\mathcal{B}) = \{\mathit{fcns}(t) : t \in L(\mathcal{A})\}$.*

This fact, together with the fact that unranked tree automata are closed under union, intersection and complement has as immediate corollary:

▶ **Corollary 4.** *For non-deterministic unranked tree automata, the same complexities hold as for non-deterministic tree automata for the problems of: membership, emptiness, finiteness, universality, containment, equivalence, emptiness of intersection.*

2.3 Querying with automata

A **query automaton** [35] is a NTA together with some subset of $X \subseteq Q \times \mathbb{A}$. A node of a tree t is output by the automaton if there is an accepting run[5] on t containing a node with symbol a and state q so that $(q, a) \in X$. A deterministic version exists, but it needs two passes.

3 Logic

There are two usual yardsticks: first-order logic (FO) and monadic second-order logic (MSO). For this, we consider the four basic binary relations inherited from finite trees: the child relation, the descendant relation, the next-sibling relation (*i.e.*, whether two nodes sharing the same parent are one next to the other), and the following-sibling relation (the transitive closure of the next-sibling relation). Henceforward, FO and MSO stand for these logics over finite trees with unary relations $a(\cdot)$ for each $a \in \mathbb{A}$ and the four binary relations just mentioned, interpreted in the expected way. Note that any sentence φ from a logic on trees defines the tree language consisting of all trees verifying φ.

[5] If instead we use a universal semantics (*i.e.*, "*...for every accepting run...*") we obtain an equi-expressive formalism [35].

3.1 Monadic Second Order logic

Monadic Second Order logic (MSO) is the fragment of second-order logic where the second-order quantification is limited to quantification over sets. That is we have first order quantification of variables ranging over nodes of the tree, and second-order quantification over sets of nodes.

We can, for example, state that a node x is the root

$$root(x) = \neg\exists y \quad (x \text{ is child of } y)$$

that the sets X and Y alternate

$$alt(X,Y) = \forall x, y \quad (x \text{ is child of } y) \Rightarrow (x \in X \Leftrightarrow y \in Y)$$

or that x is an a-labelled node at even depth

$$a(x) \ \wedge \ \forall X, Y \quad x \in X \wedge alt(X,Y) \Rightarrow (\exists y \quad y \in Y \wedge root(y)).$$

Similar to what happens in the case of words, MSO and NTA are equivalent in expressive power.

▶ **Lemma 5.** *MSO sentences characterize the class of regular tree languages [9]. Further, unary MSO formulas are equi-expressive to query automata, both on ranked and unranked trees [35].*

The idea is that one can encode the run of a NTA with statespace $\{q_1, \ldots, q_n\}$ as a formula $\exists Q_1, \ldots, Q_n \ \varphi$, where φ is an FO formula expressing that the guessed sets represent an accepting run: every node has exactly one state, the root has a final state, for every node and its children sequence there is a transition in conformity.

On the other hand, every MSO sentence can be encoded in a NTA. In order to do this, each formula $\varphi(X_1, \ldots, X_n, x_{n+1}, \ldots, x_m)$ can be seen as defining a tree language over an extended alphabet $\mathbb{A} \times \{0,1\}^m$ (nodes having a 1 in the i-th bit stand for nodes which are in the interpretation of the i-th free variable). One can then show that every formula is definable by a NTA by structural induction on the formula. The base cases are the atoms: $a(x_i)$, $x_i \in X_j$, x_i child of x_j, x_i next sibling of x_j are easily definable by NTA. For example, for $x_i \in X_j$ the automaton checks that the i-th bit is 1 in exactly one node, which happens to have the j-th bit equal to 1; for x_i child of x_j the automaton checks that there is exactly one node n_1 having the i-th bit in 1, and exactly one node n_2 having the j-th bit in 1, and that n_1 is a child of n_2. Conjunction, disjunction and negation follow from closure properties of NTA. Finally, existential quantification $\exists X_i$ and $\exists x_i$ follow by projecting the alphabet of the language recognized by the NTA by dropping the i-th bit. Note that this last projection operation yields always a non-deterministic automaton.

Note that as a consequence, there is a normal form for MSO sentences: every MSO sentence φ is equivalent to one of the form $\varphi \equiv \exists X_1, \ldots, X_n \ \psi$ where ψ is a FO formula —here n is related to the number of states needed in a NTA recognizing φ.

But then, *is MSO a good candidate for a query language?* MSO sentences can be evaluated in linear time in the tree, by transforming it to a tree automata, but the transformation is non-elementary!

▶ **Lemma 6.** *For every $n \in \mathbb{N}$ there is an MSO sentence with n quantifiers of polynomial size in n so that every NTA $\mathcal{A}$ that recognizes the same language has size*

$$|\mathcal{A}| \geq \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n+1 \text{ times}}.$$

What's more, there is no elementary algorithm to solve the satisfiability problem. Further, this is even true for FO on words. [42, 36]

3.2 Monadic Datalog

A Monadic Datalog program is a sequence of rules of the form

$$H \text{ :- } P_1, \ldots, P_k$$

where H, $P_1, \ldots, P_k$ are atoms and H is monadic (*i.e.*, of the form H(x)). Predicates that appear on the left of some rule are called *intensional*, all other are called *extensional*. Monadic Datalog over extensional monadic predicates *Leaf*, *LastChild*, *Root*, extensional binary predicates *next-sibling* and *first-child* is equivalent in expressive power to unary MSO queries [23]. Further, Monadic Datalog queries can be evaluated in linear time both in the program and the tree.

3.3 First Order logic

First-order logic (FO), on the other hand, is closer to XPath, the most popular node-selecting language for XML documents.

Core-XPath [25] is the navigational core of XPath 1.0. We don't define its syntax and semantics here, we refer the reader to [25]. The satisfiability problem is in general ExpTime-complete [30] and the evaluation problem is linear in the tree and polynomial in the query [7]. As it turns out, node expressions of XPath are equivalent in expressive power ('equi-expressive' for short) to formulas of the two-variable fragment of FO having one free variable. As opposed to XPath, satisfiability for FO^2 is ExpSpace-complete [2].

▶ **Lemma 7.** *Unary* core-XPath *is equi-expressive to unary* FO^2 *—the two-variable fragment of* FO *[31].*

On the other hand, XPath *path expressions* (denoting binary relations) are not closed under intersection nor complement.

In order to get the full expressive power of FO one needs to add some sort of Until to the language. More precisely, path expressions of the form $(\mathsf{child}[\psi])^*$, interpreted as the transitive closure of $\mathsf{child} :: n[\psi]$, *i.e.*, the existence of a descending path whose every node verifies an XPath node expression ψ. We call this extension Conditional XPath, or CXPath for short.

▶ **Lemma 8.** CXPath *and* FO *are equi-expressive [30] (for both unary and binary expressions).*

This result can be seen as the analogue of the expressive completeness of the relational algebra with respect to FO on finite relational structures.

If we extend this further by having a Kleene star on any path expression, we obtain Regular XPath (regXPath for short). And if we also add subtree relativization (W), we obtain a formalisms which is equi-expressive to FO(MTC), the extension of First-order logic with monadic transitive closure, that is, FO extended with binary formulas $TC^{\varphi}_{x,y}(u,v)$ for each formula φ having x, y free. Such formulas are interpreted as the reflexive-transitive closure of the binary relation denoted by φ on x, y:

$$\forall X.(u \in X \wedge \forall xy(x \in X \wedge \varphi \Rightarrow y \in X) \Rightarrow v \in X).$$

▶ **Lemma 9.** regXPath(W), FO(MTC), *and nested* TWA *are equi-expressive [10].*

The restriction to transitive-closure formulas $\mathrm{TC}^{\varphi}_{x,y}(u,v)$ where φ has exactly x, y as free variables yields FO*. If we extend regXPath with loop(α) meaning that the path loops —or, equivalently, with path equalities $\langle \alpha \approx \beta \rangle$— we obtain regXPath$^{\approx}$ [46].

▶ **Lemma 10.** regXPath$^{\approx}$ *and* FO* *are equi-expressive [46].*

In terms of expressive power, FO(MTC) $\subsetneq$ MSO [10] over trees,[6] but it is not known whether FO(MTC) = FO*, or equivalently, if regXPath$^{\approx}$ = regXPath(W). Even further, it is not known whether regXPath = regXPath(W).

Note that by the above characterizations CXPath, regXPath$^{\approx}$ and regXPath(W) are, contrary to XPath, closed under intersection and complement both on node and path expressions. It is not known whether regXPath is closed under complement and intersection of path expressions.

3.4 Complexity of decision problems

Logic	Evaluation problem	Satisfiability pb.
MSO	PSpace-c [43, 47], in time $O(2^{\lvert t\rvert \cdot \lvert\varphi\rvert})$ or $O(\mathit{tower}(\lvert\varphi\rvert)\cdot\lvert t\rvert)$	non-elementary [42]
FO	PSpace-c [43, 47], in time $O(\lvert t\rvert^{\lvert\varphi\rvert})$ or $O(\mathit{tower}(\lvert\varphi\rvert)\cdot\lvert t\rvert)$	non-elementary [42]
FO^2	PTime-c (in fact, this holds for every fragment FO^k, even on arbitrary relational structures)	ExpSpace-c [2]
XPath	PTime-c [24], in time $O(\lvert\varphi\rvert^2\cdot\lvert t\rvert)$ [7]	ExpTime-c [30]
regXPath(W)	PTime-c [24, 10]	2ExpTime-c [10]

Note that on arbitrary relational structures, the satisfiability problem for FO and MSO is undecidable. Also, on arbitrary structures, although the evaluation problem is still PSpace-complete, unless $P = NP$ there is no algorithm solving it in time $O(f(\lvert\varphi\rvert)\cdot p(\lvert \mathit{Structure}\rvert))$ for some computable f and polynomial p (we say that it is not *fixed-parameter tractable*), and this is true even for conjunctive queries (the $\{\exists, \wedge\}$-fragment of FO).[7]

3.5 Patterns

Another way of querying trees is through the use of patterns, which can be seen as a positive fragment of XPath with no disjunction. This is akin to Conjunctive Queries for relational databases. The use of patterns a fundamental tool for node selection and querying trees, appearing in many specification formalisms. Paradigmatic decision problems for patterns are, unsurprisingly, better behaved than NTA, in particular evaluation is polynomial time [24], their containment is coNP-complete [32], and its minimization is Σ^p_2-complete [15].

4 Data values

XML documents have not only 'labels' but also 'data values' associated with some nodes (attributes or `PCDATA` values).

[6] As opposed to what happens on strings, where FO(MTC)=MSO.

[7] This is because these problems are hard for the parameterized complexity class 'W[1]'. Parameterized complexity theory provides a framework for a refined analysis of hard algorithmic problems, see [22].

Schemas describe a class of XML documents, where
- structure constraints can be captured by regular tree languages and
- *data constraints can be specified through uniqueness, keys, and foreign keys constraints.*

XPath can
- specify navigational queries, but it also
- *allows comparisons between data.*

The model of tree can be extended to **data trees**, which are finite trees with one data value (*i.e.*, an element from an infinite domain) per node. Specification mechanisms (automata and logics) can be extended in one way or another to have the ability to compare data values (for equality, or other domain-specific relations).

4.1 Automata

Register Tree Automata We can add registers to tree automata. An automaton can store data values in registers and during the computation check whether the data value stored in the current node is equal or not to the content of some given register. For this automata model, non-emptiness is decidable (PSPACE [16] or NP [27] depending on two equi-expressive variants of the model) Containment decidable as long as the bigger automaton has at most two registers [27]. Universality is undecidable even for one register, even for words [27]. It is closed under union and intersection, but not under complement. Variants: 1-way or 2-way; deterministic or non-deterministic; alternating (decidable both on words [16] and trees [26] when restricted to one register).

4.2 Logic

XPath Evaluation is in linear time in the tree and polynomial in the query [7], but satisfiability is undecidable. Some navigational fragments are decidable: the downward (having child and descendant) [18], the forward (having child, descendant, next-sibling and following-sibling) [17], the vertical (having child, descendant, next-sibling, and following-sibling) [21], and the transitive (having descendant, following-sibling, and its inverse) [19]. See [20] for a survey.

FO^2 Remember, all the navigational part of XPath can be captured by two-variable FO. Then, what if we add an equal-data relation? Satisfiability is unknown (only variant decidable: only successor relation available [6]). On data words, it is decidable but non-elementary (equivalent to the reachability problem for VAS) [5].

The general landscape when adding data is that there is no *silver bullet*:
- Each formalism has its own specific features and it is expressive-incomparable with others.
- There is no candidate for what could be a robust class of "regular data tree languages".
- Static analysis tasks (*e.g.*, equivalence,...) become quickly undecidable or intractable.

References

1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Symposium on Theory of Computing (STOC)*, pages 202–211. ACM Press, 2004. doi:10.1145/1007352.1007390.

2 Saguy Benaim, Michael Benedikt, Witold Charatonik, Emanuel Kieronski, Rastislav Lenhardt, Filip Mazowiecki, and James Worrell. Complexity of two-variable logic on finite trees. *ACM Transactions on Computational Logic*, 17(4):32:1–32:38, 2016.

3 Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata cannot be determinized. *Theoretical Computer Science*, 350(2-3):164–173, 2006. doi:10.1016/j.tcs.2005.10.031.

4 Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular languages. *SIAM J. Comput.*, 38(2):658–701, 2008. doi:10.1137/050645427.

5 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 2010.

6 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3):1–48, 2009. doi:10.1145/1516512.1516515.

7 Mikołaj Bojańczyk and Paweł Parys. XPath evaluation in linear time. *Journal of the ACM*, 58(4):17:1–17:33, 2011. doi:10.1145/1989727.1989731.

8 Mikołaj Bojańczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive power of pebble automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4051 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2006. doi:10.1007/11786986_15.

9 J Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

10 Balder ten Cate and Luc Segoufin. Transitive closure logic, nested tree walking automata, and XPath. *Journal of the ACM*, 57(3):18, 2010.

11 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

12 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications (TATA). Freely available at http://tata.gforge.inria.fr, 2007.

13 Stephen A. Cook. An observation on time-storage trade off. *Journal of Computer and System Sciences (JCSS)*, 9(3):308–316, 1974. doi:10.1016/S0022-0000(74)80046-2.

14 Julien Cristau, Christof Löding, and Wolfgang Thomas. Deterministic automata on unranked trees. In *Fundamentals of Computation Theory*, volume 3623 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2005. doi:10.1007/11537311_7.

15 Wojciech Czerwiński, Wim Martens, Matthias Niewerth, and Paweł Parys. Minimization of tree patterns. *Journal of the ACM*, 65(4):26:1–26:46, 2018. doi:10.1145/3180281.

16 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009. doi:10.1145/1507244.1507246.

17 Diego Figueira. Alternating register automata on finite data words and trees. *Logical Methods in Computer Science (LMCS)*, 8(1), 2012. doi:10.2168/LMCS-8(1:22)2012.

18 Diego Figueira. Decidability of downward XPath. *ACM Transactions on Computational Logic*, 13(4), 2012. doi:10.1145/2362355.2362362.

19 Diego Figueira. On XPath with transitive axes and data tests. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 249–260. ACM Press, 2013. doi:10.1145/2463664.2463675.

20 Diego Figueira. Satisfiability of XPath on data trees. *SIGLOG News*, 5(2):4–16, 2018. URL: https://dl.acm.org/citation.cfm?id=3212021.

21 Diego Figueira and Luc Segoufin. Bottom-up automata on data trees and vertical XPath. *Logical Methods in Computer Science (LMCS)*, 13(4), 2017. doi:10.23638/LMCS-13(4:5)2017.

22 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

23 Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for Web information extraction. *Journal of the ACM*, 51(1):74–113, 2004. doi:10.1145/962446.962450.

24 Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems (TODS)*, 30(2):444–491, 2005. doi:10.1145/1071610.1071614.

25 Georg Gottlob, Christoph Koch, Reinhard Pichler, and Luc Segoufin. The complexity of XPath query evaluation and XML typing. *Journal of the ACM*, 52(2):284–335, 2005. doi:10.1145/1059513.1059520.

26 Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *ACM Transactions on Computational Logic*, 12(3):19, 2011. doi:10.1145/1929954.1929956.

27 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

28 Leonid Libkin. Logics for unranked trees: An overview. *Logical Methods in Computer Science (LMCS)*, 2(3), 2006. doi:10.2168/LMCS-2(3:2)2006.

29 Wim Martens and Stijn Vansummeren. Automata and logic on trees. ESSLLI 2017 summer school. Course slides. Available at https://www.theoinf.uni-bayreuth.de/pool/documents/Other/ESSLLI07.zip.

30 Maarten Marx. Conditional XPath. *ACM Transactions on Database Systems (TODS)*, 30(4):929–959, 2005. doi:10.1145/1114244.1114247.

31 Maarten Marx and Maarten de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46, 2005.

32 Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004. doi:10.1145/962446.962448.

33 Frank Neven. Automata, logic, and XML. In *EACSL Annual Conference on Computer Science Logic (CSL)*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002. doi:10.1007/3-540-45793-3_2.

34 Frank Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002. doi:10.1145/601858.601869.

35 Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.

36 Klaus Reinhardt. The complexity of translating logic to finite automata. In *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, pages 231–238. Springer, 2001. doi:10.1007/3-540-36387-4_13.

37 Thomas Schwentick. Formal methods for XML: Algorithms and complexity. EDBT 2004 summer school. Course slides. Available at https://ls1-www.cs.tu-dortmund.de/de/vortraege-thomas-schwentick.

38 Thomas Schwentick. Automata for XML – A survey. *Journal of Computer and System Sciences (JCSS)*, 73(3):289–315, 2007. doi:10.1016/j.jcss.2006.10.003.

39 Thomas Schwentick. Foundations of XML based on logic and automata: A snapshot. In *Foundations of Information and Knowledge Systems (FoIKS)*, volume 7153 of *Lecture Notes in Computer Science*, pages 23–33. Springer, 2012. doi:10.1007/978-3-642-28472-4_2.

40 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *EACSL Annual Conference on Computer Science Logic (CSL)*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006.

41 Luc Segoufin. Static analysis of XML processing with data values. *SIGMOD Record*, 36(1):31–38, 2007. doi:10.1145/1276301.1276308.

42 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *Symposium on Theory of Computing (STOC)*, pages 1–9. ACM Press, 1973. doi:10.1145/800125.804029.

43 Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, Massachusetts Institute of Technology, 1974.

44 Dan Suciu. Typechecking for semistructured data. In *Database Programming Languages (DBPL)*, volume 2397 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2001. doi:10.1007/3-540-46093-4_1.

45 Ivan Hal Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978. doi:10.1145/322077.322083.

46 Balder ten Cate. The expressivity of XPath with transitive closure. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 328–337. ACM Press, 2006. doi:10.1145/1142351.1142398.

47 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Symposium on Theory of Computing (STOC)*, pages 137–146. ACM Press, 1982. doi:10.1145/800070.802186.

48 Moshe Y. Vardi. Reasoning about the past with two-way automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998. doi:10.1007/BFb0055090.

49 János Virágh. Deterministic ascending tree automata I. *Acta Cybernetica*, 5(1):33–42, 1980.