HAL
archives-ouvertes.fr

Querying Unranked Trees with Stepwise Tree Automata

Julien Carme, Joachim Niehren, Marc Tommasi

▶ **To cite this version:**

Julien Carme, Joachim Niehren, Marc Tommasi. Querying Unranked Trees with Stepwise Tree Automata. 19th International Conference on Rewriting Techniques and Applications, 2004, Aachen, Georgia. pp.105–118. inria-00536529

HAL Id: inria-00536529
https://hal.inria.fr/inria-00536529

Submitted on 16 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Querying Unranked Trees with Stepwise Tree Automata

Julien Carme Joachim Niehren Marc Tommasi

Mostrare project, INRIA Futurs, Lille, France

Abstract. The problem of selecting nodes in unranked trees is the most basic querying problem for XML. We propose *stepwise tree automata* for querying unranked trees. Stepwise tree automata can express the same monadic queries as monadic Datalog and monadic second-order logic. We prove this result by reduction to the ranked case, via a new systematic correspondence that relates unranked and ranked queries.

1 Introduction

Querying semi-structured documents is a base operation for information extraction from the Web or semi-structured databases. It requires expressive query languages whose queries can be answered efficiently [8]. The most widely known querying language these days is the W3C standard XPath (see e.g. [10, 9]).

Semi-structured documents in XML or HTML form *unranked trees* whose nodes may have an unbounded list of children. The most basic querying problem is to select sets of nodes in unranked trees. *Monadic queries* approach this problem declaratively. They specify sets of nodes in a tree that can then be computed by a generic algorithm.

We are interested in query languages that can describe all regular sets of nodes in trees. This property is satisfied by three classes of queries, those represented by tree automata [16, 12, 3, 13, 6], *monadic second-order logic* (MSO) [16, 8] and *monadic Datalog* [1, 7] over trees. Automata and Datalog queries can be answered in linear time. They are satisfactory in efficiency and expressiveness, in theory and practice [11].

Unranked trees are problematic in that they may be recursive in depth and breadth, in contrast to ranked trees. This additional level of recursion needs to be accounted for by recursive queries. In MSO and monadic Datalog, breadth recursion can be programmed from the next_sibling relation. Unfortunately, this relation cannot be expressed in WSωS, so that traditional results on ranked trees don't carry over for free. Selection automata [6] reduce breadth recursion to depth recursion, by operating on binary encodings of unranked trees. Encodings are problematic in that they alter locality and path properties; furthermore the close relationship to unranked Datalog queries gets lost. Hedge automata [16, 12, 3] express horizontal recursion by an extra recursion level in transition rules. This syntactic extension leads to numerous technical problems [13, 7] that one might prefer to avoid.

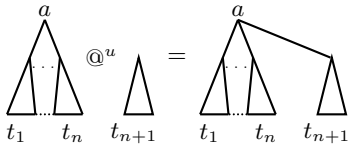


Fig. 1. Tree extension

In this paper, we propose *stepwise tree automata* for querying unranked trees. Stepwise tree automata are traditional tree automata that can either operate on unranked or ranked trees. They combine the advantages of selection and hedge automata. They model horizontal recursion by traversing siblings stepwise from the left to the right.

The algebraic approach behind stepwise tree automata yields a new systematic correspondence between queries for unranked and ranked trees. We elaborate this correspondence for monadic queries. We show that stepwise tree automata, monadic Datalog programs, and MSO can express the same monadic queries over unranked trees. We reduce this result to the case of ranked trees due to our new systematic correspondence. Specific proofs for unranked queries are not needed in contrast to [13, 7].

2 The algebras of unranked and ranked trees

An *unranked signature* Σ is a set of *symbol* ranged over by a, b. An ordered *unranked tree* or *u-tree* t over Σ satisfies the following abstract syntax:

$$t \ ::= \ a(t_1, \ldots, t_n) \qquad \text{where } n \geq 0.$$

Unordered unranked trees were investigated in [14, 15, 18]. We identify an unranked tree $a()$ with the symbol a. We write tree^u for the set of unranked trees. The extension operator $@^u : \mathsf{tree}^u \times \mathsf{tree}^u \to \mathsf{tree}^u$ for unranked trees is depicted in Fig. 1. The extended tree $t@^u t'$ is obtained from t by adjoining t' as next sibling of the last child of t:

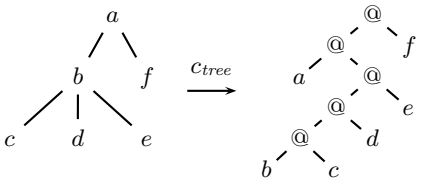
$$a(t_1, \ldots, t_n)@^u t' = a(t_1, \ldots, t_n, t')$$

Note that $a(t_1, \ldots, t_n) = a@^u t_1 @^u \ldots @^u t_n$ with parenthesis set from left to right. Tree extension $@^u$ is neither associative nor commutative.

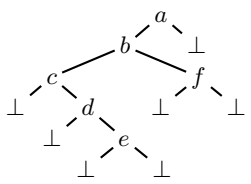
Let $\Sigma_@ = \Sigma \cup \{@\}$ be the *ranked signature* of function symbols with constants in Σ and a single binary function symbol $@$. Ranked trees over $\Sigma_@$ are ground terms over $\Sigma_@$, i.e., binary trees that satisfy the grammar:

$$t ::= a \mid t_1 @ t_2$$

We omit parenthesis as in the λ-calculus; the ranked tree $a@b@(c@b@a)$ for instance is $(a@b)@((c@b)@a)$. We write tree^r for the set of ranked trees over $\Sigma_@$.



(2-a) An unranked tree $a(b(c,d,e),f)$, its ranked construction $a@(b@c@d@e)@f$.



(2-b) First-child next-sibling encoding

Ranked trees can be constructed by the binary operator $@^r : \mathsf{tree}^r \times \mathsf{tree}^r \to \mathsf{tree}^r$ which satisfies for all ranked trees t_1, t_2:

$$t_1 @^r t_2 = t_1 @ t_2$$

Unranked trees correspond precisely to ranked constructions with respect to the function $c_{tree} : \mathsf{tree}^u \to \mathsf{tree}^r$ which satisfies for all unranked trees t_1, t_2 and symbols $a \in \Sigma$:

$$c_{tree}(t_1 @^u t_2) = c_{tree}(t_1) @^r c_{tree}(t_2) \quad \text{and} \quad c_{tree}(a) = a$$

The idea of this binary construction is known from Currying. An unranked tree describes an application of a function to a list of arguments. Its binary construction represents the Curried version of this function, receiving arguments one by one. We therefore write tree extension as function application @.

The set tree^u of unranked trees over Σ with the extension operation $@^u$ is a $\Sigma_@$ algebra, as well as the set tree^r of ranked trees over $\Sigma_@$ with the operation $@^r$.

Proposition 1. *The construction function $c_{tree} : \mathsf{tree}^u \to \mathsf{tree}^r$ is an isomorphism between $\Sigma_@$-algebras.*

Ranked and unranked trees thus have the same algebraic properties and the same finite automata [17, 5, 14].

Ranked constructions are binary representations of unranked trees. Previous approaches towards querying unranked trees rely on a different binary representation [8, 6], which encodes first-child and next-sibling relations. An example is given in Fig. 2-b. The new binary construction, however, permits to carry over traditional results from ranked to unranked trees more systematically.

3 Stepwise Tree Automata

Stepwise tree automata A over signature Σ are traditional tree automata ([4]) over the signature $\Sigma_@$. They consist of a finite set $\mathsf{states}(A)$ of *states*, a set

$$\mathsf{eval}^{\alpha}_A(a) = \{q \mid a \to q \in \mathsf{rules}(A)\}$$
$$\mathsf{eval}^{\alpha}_A(t_1 @^{\alpha} t_2) = \{q \mid q_1 \in \mathsf{eval}^{\alpha}_A(t_1),\ q_2 \in \mathsf{eval}^{\alpha}_A(t_2), q_1 @ q_2 \to q \in \mathsf{rules}(A)\}$$

Fig. 3. Ranked and unranked evaluation.

$\mathsf{final}(A) \subseteq \mathsf{states}(A)$ of *final states*, and a finite set of $\mathsf{rules}(A)$ of *transition rules* of two forms, where $a \in \Sigma$ and $q, q_1, q_2 \in \mathsf{states}(A)$:

$$a \to q \qquad \text{or} \qquad q_1 @ q_2 \to q$$

Stepwise tree automata can evaluate unranked and ranked trees, i.e. α-trees where $\alpha \in \{u, r\}$. The *α-evaluator* of A is the function $\mathsf{eval}^{\alpha}_A : \mathsf{tree}^{\alpha} \to 2^{\mathsf{states}(A)}$ defined in Fig. 3. Both evaluators only differ in the interpretation of the symbol @.

Lemma 1. $\mathsf{eval}^{u}_A(t) = \mathsf{eval}^{r}_A(c_{tree}(t))$ *for all unranked trees t.*

Stepwise tree automata A recognize all α-trees t that can be evaluated into a final state, i.e., $\mathsf{eval}^{\alpha}_A(t) \cap \mathsf{final}(A) \neq \emptyset$. The α-language $L^{\alpha}(A)$ consists of all α-trees recognized by A.

Proposition 2. *A stepwise tree automaton accepts an unranked tree if and only if it accepts its ranked construction, i.e., for all A:*

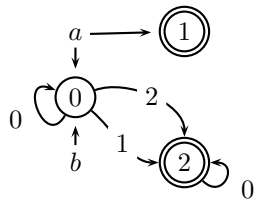
$$L^{u}(A) = c^{-1}_{tree}(L^{r}(A))$$

Proof. By Lemma 1 all unranked tree t satisfy: $t \in L^{u}(A)$ iff $\mathsf{final}_A \cap \mathsf{eval}^{u}_A(t) \neq \emptyset$ iff $\mathsf{final}_A \cap \mathsf{eval}^{r}_A(c_{tree}(t)) \neq \emptyset$ iff $c_{tree}(t) \in L^{r}(A)$.

As a consequence, stepwise tree automata inherit numerous properties from traditional tree automata, even if interpreted over unranked trees. Recognizable unranked tree languages are closed under intersection, union, and complementation. Emptiness can be checked in linear time, and membership $t \in L^{u}(A)$ in linear time $O(|t| * |A|)$.

Example. We define a stepwise tree automaton A with signature $\Sigma = \{a, b\}$ that recognizes all unranked trees with at least one a-labeled leaf.

Automaton A is illustrated to the right. It has three states 0, 1, 2 two of which are final: 1, 2. A successful run of A on an unranked tree assigns state 1 in a non deterministic way to a unique a-leaf and labels by 2 all edges visited afterwards. All other nodes and edges are labeled by 0.



a. An a-node can be the selected a-leaf: $a \to 1$.
b. Any node can be assigned to state 0: $a \to 0$, $b \to 0$, $0@0 \to 0$.
c. The edge pointing to the selected a-leaf may go into state 2: $0@1 \to 2$.
d. All edges visited later on may go into state 2, too: $2@0 \to 2$, $0@2 \to 2$.

In Fig. 4, we show the unique successful run of A on the unranked tree $a(b, a(a, b))$ and on its construction $a@b@(a@a@b)$. Both runs bisimulate each other. They evaluate the respective tree into the final state 2.

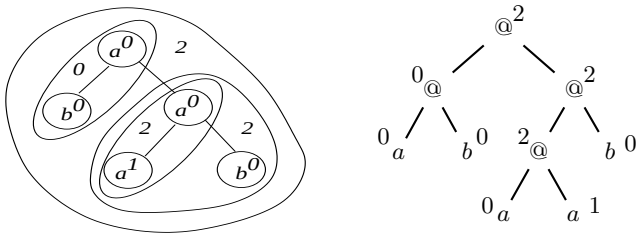


Fig. 4. An example run on an unranked tree and its construction

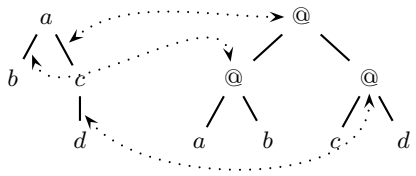


Fig. 5. Correspondence between edges and application nodes

4 Monadic Queries

Queries on unranked trees should correspond precisely to their counterparts on ranked constructions. This requires a precise correspondence between the domains of unranked trees and their ranked constructions. The domain of a ranked tree t is the set of its nodes:

$$\begin{aligned} \mathsf{dom}^r(t_1 @^r t_2) &= \mathsf{dom}^r(t_1) \uplus \mathsf{dom}^r(t_2) \uplus \{\mathsf{root}\} \\ \mathsf{dom}^r(a) &= \{\mathsf{root}\} \end{aligned}$$

Disjoint union $A \uplus B$ can be implemented by $\{1\}\times A \cup \{2\}\times B$. The ranked domain of $a@(a@a)$ is then implemented by:

$$\{\mathsf{root}, (1, \mathsf{root}), (2, \mathsf{root}), (2, (1, \mathsf{root})), (2, (2, \mathsf{root}))\}$$

As usual in mathematics, we will abstract from this implementation and talk about disjoint unions as if they were simple unions.

The nodes of unranked trees correspond to leaves in ranked constructions. But what do application nodes in construction trees correspond to? The example in Fig. 5 illustrates that they correspond precisely to edges of unranked trees. Every edge was added by some application step, and vice versa, every application step adds some edge. We therefore define the domain of an unranked tree as the union of its nodes and edges.

$$\begin{aligned} \mathsf{dom}^u(t_1 @^u t_2) &= \mathsf{dom}^u(t_1) \uplus \mathsf{dom}^u(t_2) \uplus \{\mathsf{last\_edge}\} \\ \mathsf{dom}^u(a) &= \{\mathsf{root}\} \end{aligned}$$

The $\mathsf{last\_edge}$ of $t_1 @^u t_2$ links the root of t_1 to the root of t_2. Note that all nodes of $t_1 @^u t_2$ either belong to t_1 or t_2; the root of $t_1 @^u t_2$ is that of t_1.

$$c_{dom}(a)(\mathsf{root}) = \mathsf{root}, \quad c_{dom}(t_1 @^u t_2)(\pi) = \begin{cases} \mathsf{root} & \text{if } \pi = \mathsf{last\_edge} \\ c_{dom}(t_1)(\pi) & \text{if } \pi \in \mathsf{dom}^u(t_1) \\ c_{dom}(t_2)(\pi) & \text{if } \pi \in \mathsf{dom}^u(t_2) \end{cases}$$

Fig. 6. Definition of the correspondence on domains c_{dom}.

The *correspondence* $c_{dom}(t)$ for an unranked tree t is a function between the domains of t and its ranked construction defined in Fig. 6 by recursion over the construction of t:

$$c_{dom}(t) : \mathsf{dom}^u(t) \to \mathsf{dom}^r(c_{tree}(t))$$

Definition 1. *A* monadic query *is a function q that maps trees to subsets of their domain. This definition applies to ranked and unranked trees, i.e., for both $\alpha \in \{u, r\}$. A monadic α-query q satisfies for all $t \in \mathsf{tree}^\alpha$:*

$$q(t) \subseteq \mathsf{dom}^\alpha(t)$$

We restrict ourselves to monadic queries; more general n-ary queries map to n-tuples of elements of the domain [2]. Monadic queries over unranked and ranked trees correspond. An unranked monadic query q corresponds to the ranked monadic queries $c_{query}(q)$ which satisfies for all $t \in \mathsf{tree}^r$:

$$c_{query}(q)(t) = c_{dom}(t)(q(c_{tree}^{-1}(t)))$$

All previous query notions for unranked trees [8, 6] only talk about nodes. Our extension with edges, however, is necessary to keep the symmetry to ranked queries, the reason for the simplicity of our approach.

5 Automata Queries

In the remainder of the paper, we will discuss regular monadic queries. These can be defined by tree automata, monadic second-order logic, and monadic Datalog. Here, we start with tree automata.

We next consider monadic queries as tree languages over the alphabet $\Sigma \times \mathsf{Bool}$. Such languages were already studied in [17]. Given a ranked tree t over the signature $\Sigma \times \mathsf{Bool}$ and $i \in \{1, 2\}$, let $\mathsf{proj}^i(t)$ be the ranked tree obtained by projecting all labels in t to their i'th component. For monadic queries q and ranked trees t let $\mathsf{zip}(t, q)$ be the ranked tree over the extended signature $\Sigma \times \mathsf{Bool}$ with $\mathsf{proj}^1(\mathsf{zip}(t, q)) = t$ and $\mathsf{proj}^2(\mathsf{zip}(t, q)) = q$.

Definition 2. *A monadic query q for ranked trees is* regular *if the set $\{\mathsf{zip}(t, q) \mid t \in \mathsf{tree}^r\}$ can be recognized by a tree automaton. A monadic query q for unranked trees is regular if $c_{query}(q)$ is.*

This traditional definition of regular monadic queries is simple for ranked trees but has drawbacks otherwise. First, it is not obvious how to compute such

$$\frac{a \to r(\mathsf{root}) \in \mathsf{rules}(A)}{r \in \mathsf{runs}^\alpha_A(a)} \qquad \frac{r_{|\mathsf{dom}^\alpha(t_1)} \in \mathsf{runs}^\alpha_A(t_1) \qquad r_{|\mathsf{dom}^\alpha(t_1)} \in \mathsf{runs}^\alpha_A(t_2) \quad r(\mathsf{head}^\alpha(t_1))@r(\mathsf{head}^\alpha(t_2)) \to r(\mathsf{head}(t_1@^\alpha t_2)) \in \mathsf{rules}(A)}{r \in \mathsf{runs}^\alpha_A(t_1@^\alpha t_2)}$$

Fig. 7. Runs $\mathsf{runs}^\alpha_A(t)$ of stepwise tree automata A on α-trees t.

queries efficiently, second, it is not obvious how to express them in monadic Datalog, and third, the definition of regular unranked queries depends on the correspondence to ranked queries.

Run-based queries [15, 6] with stepwise tree automata resolve these problems. We define them parametically for ranked and unranked trees. Runs of stepwise tree automaton on α-trees t associate states to all elements of the domain of t. Sequences of children in unranked trees are visited *stepwise* from the left to the right, while annotating edges to the children by states. See Fig. 4 for an example. More formally, let the *head* of a ranked tree be its root; the head of a non-constant unranked tree is $\mathsf{last\_edge}$, and the head of a constant unranked tree the root:

$$\mathsf{head}^r(t) = \mathsf{root}, \quad \mathsf{head}^u(t_1@^u t_2) = \mathsf{last\_edge}, \quad \mathsf{head}^u(a) = \mathsf{root}.$$

A *run* of a tree automaton A on an α-tree t is a function labeling elements of the domain of t by states of A:

$$r : \mathsf{dom}^\alpha(t) \to \mathsf{states}(A)$$

such that all transitions are licensed by rules of A. If $t = t_1@^u t_2$ then the restrictions of r to the domains of t_1 and t_2 must be runs and the annotation of the head of t must be justified. Furthermore, annotations of constants must be licensed. These conditions are captured by the inference rules in Fig. 7.

Lemma 2. *Let A be a stepwise tree automaton and t an α-tree, then:*

$$\mathsf{eval}^\alpha_A(t) = \{r(\mathsf{head}^\alpha(t)) \mid r \in \mathsf{runs}^\alpha_A(t)\}$$

Proof. By induction on the construction of unranked trees. If $t = a$ then $\mathsf{eval}^\alpha(a) = \{q \mid a \to q \in \mathsf{rules}(A)\} = \{r(\mathsf{root}) \mid r \in \mathsf{runs}^\alpha_A(a)\}$. For $t = t_1@^\alpha t_2$ we have $\mathsf{eval}^\alpha(t) = \{q \mid q_i \in \mathsf{eval}^\alpha_A(t_i),\ q_1@q_2 \to q \in \mathsf{rules}(A)\}$ which is equal to $\{q \mid r_i \in \mathsf{runs}^\alpha_A(t_i), r_1(\mathsf{head}^\alpha(t_1))@r_2(\mathsf{head}^\alpha(t_1)) \to q \in \mathsf{rules}(A)\}$ by induction hypothesis. The definition of runs in Fig. 7 yields $\{r(\mathsf{head}^\alpha_i(t)) \mid r \in \mathsf{runs}^\alpha_A(t)\}$.

A run r of an automaton A on an α-tree t is *successful* if $r(\mathsf{head}^\alpha(t)) \in \mathsf{final}(A)$. Let $\mathsf{succ\_runs}^\alpha_A(t)$ be the set of successful runs of A on $t \in \mathsf{tree}^\alpha$.

Definition 3. *A pair of a tree automaton A and a set $Q \subseteq \mathsf{states}(A)$ defines a monadic query which selects all elements from the domain of a tree t that are labeled by a state in Q in some successful run of A on t:*

$$\mathit{query}^\alpha_{A,Q}(t) = \{\pi \in \mathsf{dom}^\alpha(t) \mid r \in \mathsf{succ\_runs}^\alpha_A(t), r(\pi) \in Q\}$$

Selection automata [6] similarly express queries for unranked trees, but rely on universal quantification over successful runs, and use a binary encoding of unranked trees in contrast to the definition above.

Example. Reconsider the automaton A from Section 3: $\mathsf{query}^u_{A,\{1\}}$ defines the set of all a-leaves in unranked trees. Note that no automaton query with a bottom-up deterministic automaton can compute the same query, since it couldn't distinguish different a-nodes.

Proposition 3. *A monadic query on ranked trees is regular if and only if it is equal to some* $\mathit{query}^r_{A,Q}$.

The proof relies on two standard automata transformations. The idea of the transformation from regular to run based queries $\mathsf{query}^r_{A,Q}$ is memorize the Boolean values in $\mathsf{zip}(t, q)$ in automata states. In order to generalize Proposition 3 to unranked trees, we establish the correspondence between unranked and ranked run-based queries.

Theorem 1. *Queries with stepwise tree automata on ranked and unranked trees correspond:*

$$\mathit{query}^r_{A,Q} = c_{query}(\mathit{query}^u_{A,Q})$$

Proof. We first note that runs of stepwise automata on unranked trees and ranked constructions correspond. For all $t \in \mathsf{tree}^r$, we can prove:

$$\mathsf{runs}^u_A(c^{-1}_{tree}(t)) = \{r \circ c_{dom}(t) \mid r \in \mathsf{runs}^r_A(t)\}$$

The theorem follows from straightforward calculations. For all $t \in \mathsf{tree}^r$:

$$\begin{aligned} c_{query}(\mathsf{query}^u_{A,Q})(t) &= c_{dom}(t)(\mathsf{query}^u_{A,Q}(c^{-1}_{tree}(t))) \\ &= \{\pi \mid r \in \mathsf{runs}^u_A(c^{-1}_{tree}(t)),\ r(c_{dom}(t)^{-1}(\pi)) \in Q\} \\ &= \{\pi \mid r' \in \mathsf{runs}^r_A(t).\ r'(c_{dom}(t)(c_{dom}(t)^{-1}(\pi))) \in Q\} \\ &= \mathsf{query}^r_{A,Q}(t) \end{aligned}$$

□

6 Monadic Second Order Logic

We next represent regular monadic queries in ranked and unranked trees in monadic second-order logic (MSO).

The domain of the logical structure induced by an α-tree t is $\mathsf{dom}^\alpha(t)$. The signature R^r for structures of ranked trees contains the follwing relation symbols:

$$R^r = \{\mathsf{child}_1, \mathsf{child}_2, \mathsf{root}, \mathsf{leaf}\} \cup \{\mathsf{label}_a \mid a \in \Sigma_{@}\}$$

The binary relations child_1 and child_2 relate nodes to their first resp. second child. Unary relations label_a hold for all nodes labeled by $a \in \Sigma$. Furthermore, we permit the unary relations root and leaf.

Logical structures for unranked trees have the following signature:

$$R^u = \{\mathsf{first\_edge}, \mathsf{next\_edge}, \mathsf{target}, \mathsf{root}, \mathsf{leaf}\} \cup \{\mathsf{label}_a \mid a \in \Sigma_{@}\}$$

The binary relation $\mathsf{first\_edge}$ holds between a node and the edge to its first child. The $\mathsf{next\_edge}$ relation links an edge with target π to the edge whose target is the next sibling of π. The target relation holds between an edge and its target node.

Let x, y, z range over an infinite set Vars of node variables and p, q over an infinite set Preds of monadic predicates. The logics MSO^α have the following formulas:

$$\phi ::= B_n(x_1, \ldots, x_n) \mid p(x) \mid \phi \wedge \phi' \mid \neg\phi \mid \exists x\phi \mid \exists p\phi$$

where $B_n \in R^\alpha$ is a predicate with fixed tree interpretation of arity n. Note that the relations root and leaf could be expressed by the remaining relations in MSO^α. We add them anyway, as they will be needed in monadic Datalog later on.

Let ϕ be an MSO^α-formula, t an α-tree, and σ an assignment of variables into the domain of t and of predicates into the powerset of this domain. We write $t, \sigma \models_{\mathsf{MSO}^\alpha} \phi$ if ϕ becomes true in t under σ. Every formula $\phi(x)$ with a single variable x defines monadic query:

$$\mathsf{query}^\alpha_{\phi(x)}(t) \;=\; \{\sigma(x) \mid t, \sigma \models_{\mathsf{MSO}^\alpha} \phi\}$$

Theorem 2. *[Thatcher & Wright [17]] Monadic queries expressed in monadic second order logic over ranked trees* MSO^r *are regular.*

Theorem 3. *Ranked and unranked monadic queries expressed in monadic second-order logic correspond, and are thus regular; corresponding queries can be computed in linear time:*

$$\{\mathit{query}^r_{\phi(x)} \mid \phi \in \mathsf{MSO}^r\} = \{c_{query}(\mathit{query}^u_{\phi'(x)}) \mid \phi' \in \mathsf{MSO}^u\}$$

Fig. 8 presents forth and back translations between MSO^u and MSO^r. We have to show for every $\phi \in \mathsf{MSO}^u$ that $c_{query}(\mathsf{query}^u_{\phi(x)}) = \mathsf{query}^r_{[\![\phi(x)]\!]_r}$, and the analoguous property for the back translation. We proceed by structural induction over formulas. The base cases contains the difficulty, the induction step being straightforward.

We sketch the proof for formula $\mathsf{first\_edge}(x, y)$ to illustrate the principles. Consider an u-tree t and a variable assignment σ under which $\mathsf{first\_edge}(x, y)$ becomes true. There exists a u-tree $t_0 = t_1 @ t_2$ involved in the construction of t such that $\sigma(x)$ is the root of t_1 and $\sigma(y)$ is the edge from t_1 to t_2. Since this is the first edge, t_1 is a constant. Therefore, in the corresponding ranked tree, the node $c_{dom}(\sigma(x))$ is a leaf and we have $\mathsf{child}_1(c_{dom}(\sigma(y), c_{dom}(\sigma(x)))$. The converse is proved in a similar way.

The case of $\mathsf{target}(x, y)$ is more tedious as it relies on the recursive $\mathsf{lar}(x, y)$ formula, stating that x is a leaf, whose last ancestor to the right is y. This means that y denotes the up most node with $\mathsf{child}^*_1(y, x)$. A model of $\mathsf{target}(x, y)$ and its translation $[\![\phi(x)]\!]_r$ in Fig.10.

Auxiliary predicates:

$$\begin{aligned}
\mathsf{ar}'(x,p) &=_{\mathsf{def}} p(x) \wedge \forall y \forall z((\mathsf{child}_1(y,z) \wedge p(z)) \rightarrow p(y)) \\
\mathsf{ar}(x,p) &=_{\mathsf{def}} \mathsf{ar}'(x,p) \wedge \forall p'(\mathsf{ar}'(x,p') \rightarrow \mathsf{subset}(p,p')) \\
\mathsf{lar}(x,y) &=_{\mathsf{def}} \mathsf{leaf}(x) \wedge \exists p.p(y) \wedge \mathsf{ar}(x,p) \wedge (\mathsf{root}(y) \vee \exists y'\ \mathsf{child}_2(y',y))
\end{aligned}$$

Logical connectives

$$\begin{aligned}
[\![\exists x\ \psi]\!]_r &=_{\mathsf{def}} \exists x\ [\![\psi]\!]_r \\
[\![\exists p\ \psi]\!]_r &=_{\mathsf{def}} \exists p\ [\![\psi]\!]_r \\
[\![\psi \wedge \psi']\!]_r &=_{\mathsf{def}} [\![\psi]\!]_r \wedge [\![\psi']\!]_r \\
[\![\neg\psi]\!]_r &=_{\mathsf{def}} \neg[\![\psi]\!]_r \\
[\![p(x)]\!]_r &=_{\mathsf{def}} p(x)
\end{aligned}$$

Node relations

$$\begin{aligned}
[\![\mathsf{first_edge}(x,y)]\!]_r &=_{\mathsf{def}} \mathsf{child}_1(y,x) \wedge \mathsf{leaf}(x) \\
[\![\mathsf{next_edge}(x,y)]\!]_r &=_{\mathsf{def}} \exists z\ (\mathsf{child}_1(y,x) \wedge \mathsf{child}_1(x,z)) \\
[\![\mathsf{target}(x,y)]\!]_r &=_{\mathsf{def}} \exists z\ (\mathsf{child}_2(x,z) \wedge \mathsf{lar}(y,z)) \\
[\![\mathsf{label}_a(x)]\!]_r &=_{\mathsf{def}} \mathsf{label}_a(x) \quad (a \in \Sigma) \\
[\![\mathsf{last_edge}(x,y)]\!]_r &=_{\mathsf{def}} \exists z\ (\mathsf{lar}(x,y) \wedge \mathsf{child}_1(y,z)) \\
[\![\mathsf{root}(x)]\!]_r &=_{\mathsf{def}} \exists y\ (\mathsf{lar}(x,y) \wedge \mathsf{root}(y)) \\
[\![\mathsf{leaf}(x)]\!]_r &=_{\mathsf{def}} \exists y\ \mathsf{child}_2(y,x) \wedge \mathsf{leaf}(x)
\end{aligned}$$

Fig. 8. Unranked into ranked MSO

$$\begin{aligned}
[\![\exists x\ \psi]\!]_u &=_{\mathsf{def}} \exists x\ [\![\psi]\!]_u \\
[\![\exists p\ \psi]\!]_u &=_{\mathsf{def}} \exists p\ [\![\psi]\!]_u \\
[\![\psi \wedge \psi']\!]_u &=_{\mathsf{def}} [\![\psi]\!]_u \wedge [\![\psi']\!]_u \\
[\![\neg\psi]\!]_u &=_{\mathsf{def}} \neg[\![\psi]\!]_u \\
[\![p(x)]\!]_u &=_{\mathsf{def}} p(x)
\end{aligned}$$

$$\begin{aligned}
[\![\mathsf{child}_1(x,y)]\!]_u &=_{\mathsf{def}} \mathsf{first_edge}(y,x) \vee \mathsf{next_edge}(y,x) \\
[\![\mathsf{child}_2(x,y)]\!]_u &=_{\mathsf{def}} (\mathsf{target}(x,y) \wedge \mathsf{leaf}(y)) \\
&\quad \vee \exists z\ (\mathsf{target}(x,z) \wedge \mathsf{last_edge}(z,y)) \\
[\![\mathsf{label}_a(x)]\!]_u &=_{\mathsf{def}} \mathsf{label}_a(x) \quad (a \in \Sigma) \\
[\![\mathsf{root}(x)]\!]_u &=_{\mathsf{def}} (\mathsf{leaf}(x) \wedge \mathsf{root}(x)) \\
&\quad \vee \exists z\ (\mathsf{root}(z) \wedge \mathsf{last_edge}(x,z)) \\
[\![\mathsf{leaf}(x)]\!]_u &=_{\mathsf{def}} \mathsf{root}(x) \vee \exists z\ \mathsf{target}(z,x)
\end{aligned}$$

Fig. 9. Ranked into unranked MSO

7 Monadic Datalog

We next express regular monadic queries in Monadic Datalog, a logic programming language, and discuss the expressive power compared to automata and MSO queries, for ranked and unranked trees.

We consider monadic Datalog in trees without negation. The languages $\mathsf{Datalog}^\alpha$ have the same signatures as MSO^α. The programs of $\mathsf{Datalog}^\alpha$ are logic program without function symbols, predefined n-ary predicates in R^α, and free monadic predicates $p, q \in \mathsf{Preds}$. More precisely, a program $P \in \mathsf{Datalog}^\alpha$ is a finite set of rules of the form:

$$p(x) :- \mathit{Body}$$

where *Body* is a sequence of goals with n-ary predicates $B_n \in R^\alpha$:

$$\mathit{Body} ::= B_n(x_1, \ldots, x_n) \ \mid \ p(x) \ \mid \ \mathit{Body}, \mathit{Body}'$$

Every program of $\mathsf{Datalog}^\alpha$ can be seen as a formula of MSO^α; sets of clauses are conjunctions, clauses $p(x) :- \mathit{Body}$ are universally quantified implications $\forall \mathsf{Vars}.\ p(x) \leftarrow \mathit{Body}$, and bodies *Body* conjunctions of goals.

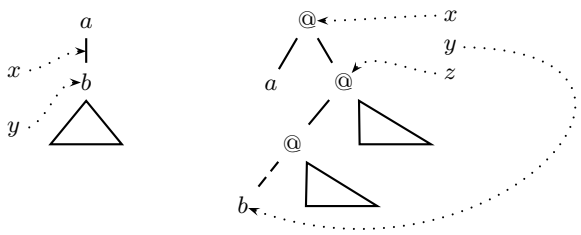


Fig. 10. A solution of $\mathsf{target}(x, y)$ on the left; a solution of its ranked translation $[\![\mathsf{target}(x, y)]\!]_r = \exists z\ (\mathsf{child}_2(x, z) \wedge \mathsf{lar}(z, y))$ on the right.

We interpret programs in $\mathsf{Datalog}^\alpha$ in the least fixed point semantics over the structures of MSO^α. For every program $P \in \mathsf{Datalog}^\alpha$, predicate $p \in \mathsf{Preds}$, and $t \in \mathsf{tree}^\alpha$ let $T^\omega_{P,t}(p)$ be the least solution of P over the tree structure of the α-tree t for predicate p. This yields a notion of monadic queries:

$$\mathsf{query}^\alpha_{P(p)}(t) = T^\omega_{P,t}(p)$$

Least fixed points can be expressed in MSO. As a consequence, every query in $\mathsf{Datalog}^\alpha$ can be expressed in linear time in MSO^α. This shows that ranked queries in $\mathsf{Datalog}^r$ are regular (Theorem 2).

Ranked monadic run-based automata queries can always be expressed in ranked monadic Datalog; the reduction is in linear time. The resulting Datalog program models the two phases of the linear time algorithm for answering automata queries: the first bottom up phase computes all states of all nodes seen in all runs of the automaton, and a top down phase selects all nodes labeled by selection states in successful runs.

Theorem 4. *Ranked and unranked monadic queries expressed in monadic Datalog correspond, and are thus regular:*

$$\{\mathit{query}^r_{P(p)} \mid P \in \mathsf{Datalog}^r\} = \{c_{query}(\mathit{query}^u_{P'(p)}) \mid P' \in \mathsf{Datalog}^u\}$$

Corresponding unranked queries can be computed from ranked queries in linear time; the converse is not true.

It suffices to encode ranked Datalog queries into corresponding unranked queries in linear time. The translation basically refines the encoding from MSO^r into MSO^u: roughly, a rule $p(x) :- \mathit{Body}$ is translated into $p(x) :- [\![\mathit{Body}]\!]_u$. Conjunctions in $[\![\mathit{Body}]\!]_u$ can be replaced by commas, existential quantifications can be ommited, i.e., replaced by implicit universal quantification in rules. Disjunctions as in the definitions of $[\![\mathsf{child}_1(x, y)]\!]_u$, $[\![\mathsf{child}_2(x, y)]\!]_u$, and $[\![\mathsf{root}(x)]\!]_u$ can expressed by multiple rules. Such a rewriting, however, spoils linear time. We can circumvent this problem following Gottlob and Koch [7]: we normalise programs of $\mathsf{Datalog}^r$ into *tree marking normal form* (TMNF) in linear time before translation. TMNF programs have of forms:

$$p(x) :- B_1(x). \qquad p(x) :- q(y), B_2(y, x).$$
$$p(x) :- p_0(x), p_1(x). \qquad p(x) :- q(y), B_2(x, y).$$

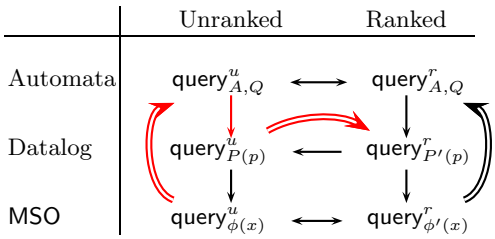


Fig. 11. Summary of reductions. Solid lines are in linear time, double lines are non elementary. Black lines are proved, red lines induced.

$$\frac{t = a(t_1, \ldots, t_n) \qquad \forall 1 \le i \le n : r_{|\mathsf{nodes}(t_i)} \in \mathsf{run}_H(t_i)}{a(L) \to r(\mathsf{root}(t)) \in \mathsf{rules}(H) \qquad r(\mathsf{root}(t_1)) \ldots r(\mathsf{root}(t_n)) \in L}$$
$$r \in \mathsf{run}_H(t)$$

Fig. 12. Runs of Hedge Automata

where B_n is a n-ary predicate of R^r. On ranked TMNF programs the reduction $[\![\_]\!]_u$ can clearly be done in linear time.

The inverse translation can be composed from our translations so far, which are summarized in Fig. 11. We first reduce unranked Datalog queries to MSO^u, then to MSO^r, move to ranked automata queries and then to ranked Datalog queries. The overall reduction has nonelementary complexity.

Note that we cannot specialize the translation $[\![\_]\!]_r$ from MSO^u to MSO^r into a translation from $\mathsf{Datalog}^u$ to $\mathsf{Datalog}^r$. The problem is that we cannot express the auxilary binary predicate $\mathsf{lar}(x, y)$. Its definition is recursive, but only monadic recursive predicates can be defined in monadic Datalog.

Corollary 1. *Stepwise tree automata, monadic Datalog, and* MSO *capture the class of regular monadic queries over unranked trees.*

This is a corollary of our correspondences between ranked and unranked queries and traditional result on ranked trees. All unranked automata queries $\mathsf{query}^u_{A,Q}$ can be expressed in linear time in $\mathsf{Datalog}^u$, by indirection over ranked automata queries and $\mathsf{Datalog}^r$. Unranked queries in MSO^u can be expressed by unranked automata queries by reduction to the ranked case.

8 Hedge Automata

We finally show how to express monadic queries with hedge automata [16, 3] in linear time with stepwise tree automata. A hedge automaton H over Σ consists of a set $\mathsf{states}(H)$ of states, a set $\mathsf{final}(H)$ of final states, and set set of transition rules of the form $a(L) \to q$ where L is a regular set of words over states.

Runs of hedge automata H on unranked trees t are functions $r : \mathsf{nodes}(t) \to \mathsf{states}(H)$ that satisfy the inference rule in Fig 12. A hedge automaton H and a

$$
\begin{aligned}
\mathsf{states}(\mathsf{step}(H)) &= \mathsf{states}(H) \uplus \biguplus_{a\in\Sigma, q\in\mathsf{states}(H)} \mathsf{states}(H_{a,q}) \\
\mathsf{rules}(\mathsf{step}(H)) &= \bigcup_{a\in\Sigma, q\in\mathsf{states}(H)} \mathsf{rules}(H_{a,q}) \\
&\quad \cup \{p \xrightarrow{\epsilon} q \mid p \in \mathsf{final}(H_{a,q}), q \in \mathsf{states}(H), a \in \Sigma\} \\
&\quad \cup \{a \to p \mid p \in \mathsf{init}(H_{a,q}), q \in \mathsf{states}(H)\} \\
\mathsf{final}(\mathsf{step}(H)) &= \mathsf{final}(H)
\end{aligned}
$$

Fig. 13. Hedge automata into stepwise tree automata

set of states $Q \subseteq \mathsf{states}(H)$ defines a monadic query for unranked trees:

$$\mathsf{query}_{H,Q}(t) = \{\pi \in \mathsf{nodes}(t) \mid r \in \mathsf{succ\_runs}_H(t), r(\pi) \in Q\}$$

For translating hedge automata into stepwise tree automata, we need to represent all regular language L in transition rules explicitly. We use a sequence of finite word automata $(H_{a,q})_{a\in\Sigma, q\in\mathsf{states}(H)}$ over the alphabet $\mathsf{states}(H)$ to do so.

Proposition 4. *Queries by hedge automata can be translated in linear time to queries by stepwise tree automata.*

Proof. Given an hedge automaton H we define a stepwise tree automaton $\mathsf{step}(H)$ by unifying all subautomata $H_{a,q}$ into a single finite automaton. We then add all states of $\mathsf{states}(A)$ to this automaton, and link them to all final states p of $H_{a,q}$ through ϵ-transitions $p \xrightarrow{\epsilon} q$. We add rules $a \to q'$ for all initial states q' of some $H_{a,q}$. The final states of $\mathsf{step}(H)$ are those in $\mathsf{final}(H)$, not those in $\mathsf{final}(H_{a,q})$. The complete construction is detailed in Fig. 13. It remains to show that every run of H can be simulated by a run of $\mathsf{step}(H)$. □

Acknowledgments: We would like to thank our collegues from the Mostrare project for their continuous support, particularly R. Gilleron and S. Tison, and the anonymous referees for hinting us towards Currification.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *The Very Large Data Bases Journal*, pages 119–128, 2001.
2. A. Berlea and H. Seidl. Binary queries. In *Proceedings of Extreme Markup Languages*, Montreal, 2002.
3. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical report, 2001.
4. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Online book, 450 pages. Available at: http://www.grappa.univ-lille3.fr/tata, 1997.
5. B. Courcelle. On recognizable sets and tree automata. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Algebraic Techniques*, volume 1, chapter 3, pages 93–126. Academic Press, 1989.
6. M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees. In *Proceedings of the IEEE Symposium on Logic In Computer Sciences*, Ottawa, 2003.

7. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. In *Proceedings of the ACM Symposium on Principle of Databases Systems*, pages 17–28, 2002.
8. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proceedings of the IEEE Symposium on Logic In Computer Sciences*, Copenhagen, 2002.
9. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPATH query evaluation. In *Proceedings of the ACM Symposium on Principle of Databases Systems*, pages 179–190. 2003.
10. G. Gottlob, C. Koch, and R. Pichler. XPATH processing in a nutshell. *ACM SIGMOD Record*, 32(2):21–27, 2003.
11. C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *Proceedings of the International Conference on Very Large Data Bases*, 2003.
12. A. Neumann, H. Seidl. Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science*, pages 134–145, 1998.
13. F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.
14. J. Niehren and A. Podelski. Feature automata and recognizable sets of feature trees. In *Proceedings of TAPSOFT'93*, volume 668 of *LNCS*, pages 356–375, 1993.
15. H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *Proc. of the IEEE Symposium on Principles of Database Systems*, pages 155–166, 2003.
16. J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of automata theory. *J. of Comp. and Syst. Sci.*, 1:317–322, 1967.
17. J. W. Thatcher, J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Math. System Theory*, 2:57–82, 1968.
18. S. D. Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In R. Nieuwenhuis, editor, *Proceedings of RTA*, volume 2706 of *LNCS*, pages 246–263. 2003.