



**HAL**  
open science

## Running Tree Automata on Trees and/or Dags

Barbara Fila, Siva Anantharaman

► **To cite this version:**

Barbara Fila, Siva Anantharaman. Running Tree Automata on Trees and/or Dags. 2006. hal-00080644v2

**HAL Id: hal-00080644**

**<https://hal.science/hal-00080644v2>**

Preprint submitted on 4 Aug 2006 (v2), last revised 25 Aug 2006 (v3)

**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.

# Running Tree Automata on Trees and/or Dags

Barbara Fila

*LIFO, Université d'Orléans (France), e-mail: fila@univ-orleans.fr*

Siva Anantharaman

*LIFO, Université d'Orléans (France), e-mail: siva@univ-orleans.fr*

---

## Abstract

We define tree/dag automata as extensions of (unranked) tree automata which can run indifferently on trees or dags, in a bottom-up style. The runs of such an automaton  $A$  on any tree or dag  $t$  are defined by assigning states not only to the nodes of  $t$  but also to its edges. Runs are so defined that  $A$  accepts  $t$  if and only if it accepts the tree  $\hat{t}$  obtained by unfolding  $t$ , as a tree automaton running  $\hat{t}$ , in the usual sense.

*Keywords:* Automata, Trees, Dags. Determinism.

---

## 1. Introduction

Several algorithms have been optimized in the past, by using structures over dags instead of over trees. Tree automata are widely used for querying XML documents (e.g., [7,8,12,13]); on the other hand, the notion of a compressed XML document has been introduced in [2,6,11], and a possible advantage of using dag structures for the manipulation of such documents has been brought out in [11]. It is legitimate then to investigate the possibility of using automata over dags instead of over trees, for querying compressed XML documents.

Dag automata (DA) were first introduced and studied in [3]; a DA was defined there as a natural extension of tree automaton, i.e. as a bottom-up tree automaton running on dags; and the language of a DA was defined as the set of dags that get accepted under (bottom-up) runs, defined in the usual sense; the emptiness problem for DAs was shown there to be NP-complete, and the mem-

bership problem proved to be in NP; but the problem of stability under complementation of the class of dag automata –closely linked with that of determinization– was left open. These two issues have since been settled negatively in [1], where it was observed that a *deterministic* DA runs (bottom-up) on trees or dags exactly alike, but this is no longer true for *non-deterministic* DAs; it was also shown there that the set of all terms (trees) represented by the set of dags accepted by a non-deterministic DA is not necessarily a regular tree language. A consequence is that the class of tree languages recognized by DAs as sets of accepted dags, is a *strict* superclass of the class of regular tree languages. It is well-known however, that answers to MSO-definable queries on (semi-)structured trees form regular tree languages ([15]). It is thus necessary to define the languages of DAs in a manner *different* from that of [3,1], if they are to serve as tools for analyzing and querying a document, independently of whether it is given

in a partially or fully compressed format, or as a tree. Our aim in this work is therefore to redefine the notion of the language of a DA suitably, with such an objective.

For achieving that, we first present the notion of a compressed document, as a *tree/dag* (*trdag*, for short) designating a directed acyclic graph which may be partially or fully compressed. The terminology *trdag* has been chosen to distinguish it from that of *tdag* employed in [1]; this latter term will designate in this paper a fully compressed document. A Tree/Dag automaton (TDA, for short) is then defined as an automaton which runs on trdags. The essential differences with the DAs of [1] are as follows: (i) our TDAs can be unranked, and (ii) although the transition rules of a TDA look quite like those of the DAs in [1], or those of TAs, a run of a TDA on any given *trdag*  $t$  will carry with it not only assignments of states to the nodes of  $t$ , but also to the edges of  $t$ . Runs will be so defined that a TDA accepts any given *trdag*  $t$  if and only if it accepts the tree  $\hat{t}$  obtained by uncompressing  $t$ , as a tree automaton running on the tree  $\hat{t}$ , in the usual sense. It follows that for a TDA, the emptiness problem is decidable in time  $P$ , and the uniform membership problem is decidable in time  $NP$ , with respect to its number of states.

## 2. Tree/Dag Automata

**Definition 1** A *tree/dag* (*trdag* for short) over a not necessarily ranked alphabet  $\Sigma$  is a rooted dag (directed acyclic graph)  $t = (Nodes(t), Edges(t))$ , where, for any  $u \in Nodes(t)$ :

- $u$  has a name  $name_t(u) = name(u) \in \Sigma$ ;
- the edges going out of any node are ordered;
- and if  $name(u)$  is ranked, then the number of outgoing edges at  $u$  is the rank of  $name(u)$ .

Given any node  $u$  on a *trdag*  $t$ , the notion of the sub-*trdag* of  $t$  rooted at  $u$  is defined as usual, and is denoted as  $t|_u$ . If  $v$  is any node,  $\gamma(v) = u_1 \dots u_n$  will denote the *string* of all its not necessarily distinct *children* nodes; for every  $1 \leq i \leq n$ , the  $i$ -th outgoing edge from  $v$  to its  $i$ -th *child* node  $u_i \in \gamma(v)$  will be denoted as  $e(v, i)$ ; the set of all outgoing (resp. incoming) edges at any node  $v$  will be denoted as  $Out_v$  (resp.  $In_v$ ); and for any node  $u$ , we set:

$$Parents(u) = \{v \in Nodes(t) \mid u \text{ is a child of } v\}.$$

A *trdag*  $t$  will be said to be a *tree* iff for every node  $u$  on  $t$  other than the root,  $Parents(u)$  is a singleton. For any *trdag*  $t$ , we define the set  $Pos(t)$  as the set of all the positions  $pos_t(u)$  of all its nodes  $u$ , these being defined recursively, as follows: if  $u$  is the root node on  $t$ , then  $pos_t(u) = \epsilon$ , otherwise,  $pos_t(u) = \{\alpha.i \mid \alpha \in pos_t(v), v \text{ is a parent of } u, u \text{ is an } i\text{-th child of } v\}$ . The elements of  $Pos(t)$  are words over natural integers.

The function  $name_t$  is extended naturally to the positions in  $Pos(t)$  as follows: for every  $u \in Nodes(t)$  and  $\alpha \in pos_t(u)$ , we set  $name_t(\alpha) = name_t(u)$ . Given a *trdag*  $t$ , we define its tree-equivalent as a tree  $\hat{t}$  such that:  $Pos(\hat{t}) = Pos(t)$ , and for every  $\alpha \in Pos(t)$  we have  $name_t(\alpha) = name_{\hat{t}}(\alpha)$ . A *trdag* is said to be a *tdag*, or *fully compressed*, iff for any two different nodes  $u, u'$  on  $t$ , the two sub-dags  $t|_u$  and  $t|_{u'}$  have non-isomorphic tree-equivalents; otherwise, the *trdag* is said to be *partially compressed*. For example, the tree to the left of Figure 1 is the tree-equivalent of the partially compressed *trdag* to the right, and also to the fully compressed *tdag* to the middle.

We define now the notion of a Tree/Dag automaton, first over a ranked alphabet  $\Sigma$ , to facilitate understanding. The definition is then easily extended to the unranked case.

**Definition 2** A *Tree/Dag automaton* (TDA, for short) over a ranked alphabet  $\Sigma$  is a tuple  $(\Sigma, Q, F, \Delta)$ , where  $Q$  is a finite non-empty set of states,  $F \subseteq Q$  is the set of final (or accepting) states, and  $\Delta$  is a set of transition rules of the form:  $f(q_1, \dots, q_k) \rightarrow q$ , where  $f \in \Sigma$  is of rank  $k$ , and  $q_1, \dots, q_k, q \in Q$ .

It will be convenient to write the transition rules of a TDA in a different (but equivalent) form: a transition of the form  $f(q_1, \dots, q_k) \rightarrow q$  is also written as  $(f, q_1 \dots q_k) \rightarrow q$ , where  $q_1 \dots q_k$  is seen as a word in  $Q^*$ , of length =  $rank(f)$  in the ranked case. The notion of a TDA is then extended easily to the unranked case, where the signature symbols naming the nodes are not assumed to be of fixed rank: it suffices to define the transitions to be of the form  $(f, \omega) \rightarrow q$ , where  $\omega \in Q^*$ ; we may assume wlog that  $\omega$  is a '\*'-regular expression on

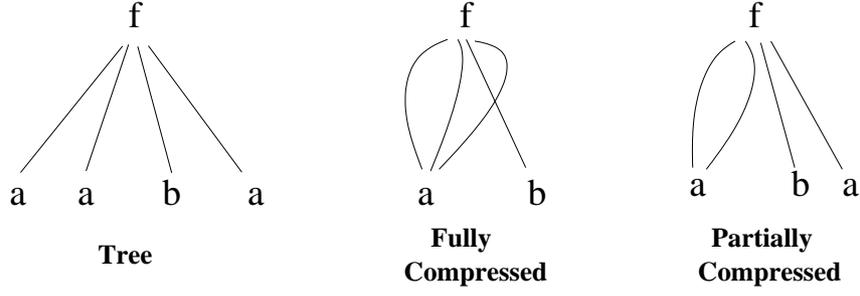


Fig. 1. A tree, tdag, trdag

$Q$  not involving ‘+’, by replacing any rule of the form  $(f, \omega + \omega') \rightarrow q$ , by the two rules  $(f, \omega) \rightarrow q, (f, \omega') \rightarrow q$ .

A TDA is said to be bottom-up *deterministic* iff whenever there are two transition rules of the form  $(f, \omega) \rightarrow q, (f, \omega') \rightarrow q'$ , with  $q \neq q'$ , we have necessarily  $\omega \cap \omega' = \emptyset$ ; otherwise it is said to be *non-deterministic*. We also agree to denote the transitions of the form  $(f, \emptyset) \rightarrow q$  simply as  $f \rightarrow q$ , and refer to them as *initial* transitions.

For defining the notion of runs of TDAs on a trdag in a bottom-up style, we need some preliminaries. Let  $\mathbf{A}$  be a TDA with state set  $Q$  and transition set  $\Delta$ . Suppose  $t$  is a trdag and assume given a map  $M : Edges(t) \rightarrow Q$ . If  $u$  is any node on  $t$  with  $u_1 \dots u_n$  as the string of all its (not necessarily distinct) children, the string  $M(\mathbf{e}(u, 1)) \dots M(\mathbf{e}(u, n))$ , formed of states assigned by  $M$  to the outgoing edges at  $u$ , will be denoted as  $M(Out_u)$ . We then define, recursively in a bottom-up style, a binary relation at  $u$  on the states of  $Q$ , with respect to (w.r.t. or wrt, for short) the given map  $M$ ; this relation, denoted as  $\triangleleft_u^M = \triangleleft_u$ , is defined as follows:

**Definition 3** Let  $\mathbf{A}, t, M$  be as above, and  $u$  any given node on the trdag  $t$ .

- If  $u$  is a leaf with  $name(u) = a$ , then  $q \triangleleft_u q'$  iff whenever  $a \rightarrow q \in \Delta$  we also have  $a \rightarrow q' \in \Delta$ ;
- otherwise  $q \triangleleft_u q'$  iff:
  - (i)  $(name(u), M(Out_u)) \rightarrow q$  is an instance of a transition rule in  $\Delta$ ; i.e.,  $\Delta$  has a rule  $(name(u), \omega) \rightarrow q$  such that  $M(Out_u)$  is in  $\omega$ ;
  - (ii) there exists a map  $\sigma_{q'} : Q \rightarrow Q$ , such that:
    - $\sigma_{q'}(q) = q'$ , and the rule  $(name(u), \sigma_{q'}(M(Out_u))) \rightarrow q'$  is also an instance of a transition rule in  $\Delta$ ;

- for any edge  $\mathbf{e} : u \xrightarrow{i} u' \in Out_u$ , we have:  $M(\mathbf{e}) \triangleleft_{u'} \sigma_{q'}(M(\mathbf{e}))$ .

**Definition 4** Let  $\mathbf{A} = (\Sigma, Q, F, \Delta)$  be any given TDA, and  $t$  any given trdag. A run of  $\mathbf{A}$  on  $t$  is a pair  $(r, M)$ , where  $r : Nodes(t) \rightarrow Q$  and  $M : Edges(t) \rightarrow Q$  are maps such that the following conditions hold, at any node  $u$  on  $t$ :

- (1) if  $name(u) = f$ , then the rule  $(f, M(Out_u)) \rightarrow r(u)$  is an instance of a transition rule in  $\Delta$ ;
- (2) there is an incoming edge  $\mathbf{e} \in In_u$  with  $M(\mathbf{e}) = r(u)$ ; and for every  $\mathbf{e}' \in In_u$  such that  $M(\mathbf{e}') = q' \neq q = r(u)$ , we have  $q \triangleleft_u^M q'$

A run  $(r, M)$  is accepting on a trdag  $t$  iff  $r(\epsilon) \in F$ , i.e.,  $r$  maps the root-node of  $t$  to an accepting state. A trdag  $t$  is accepted by a TDA iff there is an accepting run on  $t$ . The language of a TDA is the set of all trdags that it accepts, up to bisimulation.

**Remark 2.** i) Note that if  $t$  is a tree, then  $In_u$  is singleton at every non-root node  $u$  on  $t$ , so a run  $(r, M)$  of any TDA on  $t$  can be identified with its first component  $r$ ; we get then the usual notion of runs of tree automata on trees.

**Example 1.** Over the unranked signature  $\{a, f, g\}$  consider a TDA  $\mathbf{A}$ , with the following transitions:

$$\begin{aligned}
 & a \rightarrow p, \quad b \rightarrow q', \quad b \rightarrow p, \quad b \rightarrow q, \\
 & (a, p) \rightarrow q, \quad (a, q) \rightarrow p, \\
 & (g, qQ^*) \rightarrow q, \quad (g, pq) \rightarrow p, \\
 & (f, qpq) \rightarrow q_{fin}, \quad (f, pQ^*) \rightarrow q_{fin},
 \end{aligned}$$

with  $Q = \{p, q, q', q_{fin}\}$ , and  $q_{fin}$  as the unique accepting state. An accepting bottom-up run of  $\mathbf{A}$  on a tdag is depicted on the left of Figure 2, and on its right, the ‘‘same’’ run as seen on the tree equivalent of the tdag.

A few comments on the above run may be of help: we start with assigning state  $q$  to the leaf

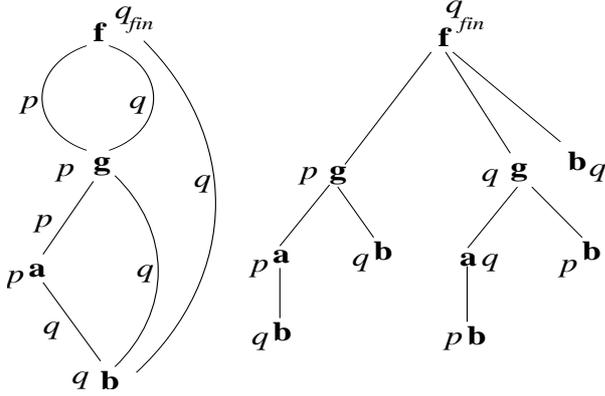


Fig. 2. A bottom-up accepting run of the TDA of Example 1 on a trdag, and the same seen on its tree equivalent.

node  $b$ , under  $r$ ; the assignments of state  $q$  under  $M$  to all the incoming edges at this node  $b$  poses no problem; we can then assign state  $p$  to node  $a$ , and subsequently also  $p$  to the node  $g$ , under  $r$ , via the transition rule  $(g, pq) \rightarrow p$ ; we then assign  $p$  under  $M$  to the first incoming edge at  $g$ ; assigning state  $q$  under  $M$  to the second incoming edge at  $g$  is valid since we have:  $q \triangleleft_a p$  and  $p \triangleleft_b q$ ; reaching  $q_{fin}$  at the root-node is trivial via the last transition rule. (Note that we could have as well assigned  $p$  under  $M$  to the second incoming edge at  $g$ , with no conditions to check, then reach  $q_{fin}$ .)

**Remark 2** (contd.) ii) Unlike the DAs of [3] or [1], the following bottom-up non-deterministic TDA:  $a \rightarrow q_1$ ,  $a \rightarrow q_2$ ,  $f(q_1, q_2) \rightarrow q_a$ , with  $q_0, q_1, q_a$  as states and  $q_a$  as the accepting state, has a non-empty language: as a TDA it accepts  $f(a, a)$ .

For a *deterministic* TDA, we have the following result (as expected):

**Proposition 1** *Let  $\mathbf{A}$  be a bottom-up deterministic TDA, and  $t$  any given trdag; then there is at most one run of  $\mathbf{A}$  on  $t$ .*

*Proof:* Let  $Q$  be the set of states of  $\mathbf{A}$ , and  $M : Edges(t) \rightarrow Q$  any given map assigning states to the edges on  $t$ . We shall show by induction that the hypothesis of determinism on  $\mathbf{A}$  implies that, at any node  $u$  on  $t$ , the binary relation  $\triangleleft_u^M = \triangleleft_u$  defined above (Definition 3), w.r.t. the map  $M$ , is the identity relation on the set  $Q$ . The proposition will then follow from conditions (1) and (2) on runs, cf. Definition 4 (we will get, in particular, that for every incoming edge  $e$  at  $u$ ,  $M(e)$  must be the same

as  $r(u)$ ; so the run can be identified with its first component  $r$ ).

The induction will be on a non-negative integer  $d_u$  –that we define at any node  $u$  of  $t$ , and refer to as its *height* on  $t$ – as the maximal number of arcs on  $t$  from  $u$  to the leaf nodes. If  $d_u = 0$ , then  $u$  is a leaf node; that  $\triangleleft_u$  is the identity relation on  $Q$  in this case is immediate, from the determinism of  $\mathbf{A}$ , and the definition of  $\triangleleft_u$ . So, assume that  $d_u > 0$ , and let  $v_1 \dots v_n$  be the string of all the children nodes of  $u$  on  $t$ . By the inductive hypothesis, for every  $i, 1 \leq i \leq n$ , the relation  $\triangleleft_{v_i}$  is identity; it follows then, from the conditions (i) and (ii) of the relation  $\triangleleft_u$ , that this latter must also be the identity relation on  $Q$ .  $\square$

We may now formulate the principal results of this paper:

**Proposition 2** *i) A TDA accepts a trdag  $t$  if and only if it accepts the tree equivalent of  $t$ .*

*ii) The emptiness problem for a TDA is decidable in time  $P$  w.r.t. its number of states.*

*iii) The uniform membership problem for a TDA is decidable in time  $NP$  (resp. time  $P$ ) w.r.t. its number of states, and the number of edges (resp. and the number of positions) on the given trdag.*

*Proof:* Let  $\hat{t}$  be the tree equivalent of the trdag  $t$ . (It is immediate that  $\hat{t}$  is uniquely determined, up to a tree isomorphism.) The ordered bisimulation relation between the sets of nodes of  $t$  and  $\hat{t}$  can actually be seen as a natural surjective map from  $Nodes(\hat{t})$  onto  $Nodes(t)$  that can be defined recursively, in a bottom-up style; in what follows, this map will be referred to as the compression map, and denote it as  $\mathbf{c}$ .

Property i): For proving the ‘only if’ part, one uses the following reasoning, coupled with induction on the height function at the nodes of  $t$  (defined in the proof of the previous proposition): Let  $(r, M)$  be an accepting run of the given TDA on the trdag  $t$ ; consider a node  $s$  on the tree equivalent  $\hat{t}$ , of which the node  $u$  on  $t$  is the image under the compression map  $\mathbf{c}$ ; let  $r(u) = q$  under the given run of the TDA on  $t$ ; then, for every state  $q'$  of the TDA such that  $q \triangleleft_u^M q'$ , one can construct a partial run of the TDA –seen as a usual tree automaton– on the tree  $\hat{t}$ , climbing up from a leaf below  $s$  on  $\hat{t}$ , to the node  $s$  (for an illustrative example, see the

tree to the right of Figure 2).

For proving the ‘if’ part of Property i), we start with a given accepting run  $\hat{r}$  of the given TDA, as a bottom-up tree automaton running in the usual sense on the tree  $\hat{t}$ ; from this run  $\hat{r}$ , we shall construct a run  $(r, M)$  of the TDA on the trdag  $t$  as follows: Consider any given node  $u$  on  $t$ , and let  $\{s_1, \dots, s_k\}$  be the set  $\mathbf{c}^{-1}(u)$  of all nodes on  $\hat{t}$  such that  $u$  is the image of each of them under the compression map  $\mathbf{c}: \text{Nodes}(\hat{t}) \rightarrow \text{Nodes}(t)$ ; the set  $\text{In}_u$ , of incoming edges at  $u$  on the trdag  $t$ , contains then exactly  $k$  edges, say  $\mathbf{e}_1, \dots, \mathbf{e}_k$  in some (arbitrarily chosen) order; we then set, for any  $i, 1 \leq i \leq k$ ,  $M(\mathbf{e}_i) = \hat{r}(s_i)$ ; and set  $r(u)$  as any (arbitrarily chosen) state among the  $\hat{r}(s_i), i, 1 \leq i \leq k$ . It is then easily checked that the pair of maps  $(r, M)$  gives an accepting run of the TDA on the trdag  $t$ .

Properties ii) and iii) follow, in the ranked case, from the proof of i) and the results of TATA ([4]), Chapter 1; in the unranked case, one can either employ a reasoning based on reduction to the ranked case as in [9], or appeal directly to the results of [10]. (Note: the number of positions on a trdag is the same as the size of its tree equivalent.)  $\square$

### 3. Conclusion

Neither the dag automata of [3,1], nor the Tree/Dag automata that we have presented in this paper, can be seen as special cases of the vertex marking or the edge marking automata on graphs, defined in [14]: the reason is that a boundedness assumption was made there on the number of *incoming* edges at the nodes; in particular, contrary to the Example 2.7 of [14], it is easy to construct a DA (resp. a TDA) whose language is the set  $L_b$  of all the dags (resp. trdags) that contain a node named with some given symbol  $b$ .

The notion of a trdag that we have defined in this paper, is none other than that of an acyclic graph obtained from a (possibly unranked) tree, by identifying –fully or partially– the nodes where the sub-trees are isomorphic. We have also shown how to run any tree automaton  $\mathbf{A}$  on any trdag  $t$ , without having to uncompress  $t$ ; the runs are defined in a bottom-up style, in such a way that  $\mathbf{A}$  accepts  $t$  if and only if it accepts the tree equivalent

of  $t$ , in the classical sense. Validating XML documents that may be in a compressed format, with respect to a specifying tree grammar is one possible area of application for the view that we have presented here; a second area is that of querying directly a compressed XML document, without having to unfold it into a tree. The interested reader may consult [5], where the results that we have already obtained in this connection are presented.

### References

- [1] S. Anantharaman, P. Narendran, M. Rusinowitch, *Closure Properties and Decision Problems of Dag Automata*, In Information Processing Letters, 94(5):231–240, 2005.
- [2] P. Buneman, M. Grohe, C. Koch, *Path queries on compressed XML*. In Proc. of the 29th Conf. on VLDB, 2003, pp. 141–152, Ed. Morgan Kaufmann.
- [3] W. Charatonik, *Automata on DAG Representations of Finite Trees*, Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany.
- [4] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, *Tree Automata Techniques and Applications*, <http://www.grappa.univ-lille3.fr/tata/>
- [5] B. Fila, S. Anantharaman, *Automata for Positive Core XPath Queries on Compressed Documents*, In Proc. of the Int. Conf. LPAR-13, November 2006, LNAI, Springer-Verlag (To appear). Full version available as part of RR-2006-03, LIFO, Université d’Orléans (Fr.). <http://www.univ-orleans.fr/lifo/prodsci/rapports/>
- [6] M. Frick, M. Grohe, C. Koch, *Query Evaluation of Compressed Trees*, In Proc. of LICS’03, pp. 188–197, IEEE,
- [7] G. Gottlob, C. Koch, *Monadic Queries over Tree-Structured Data*, In Proc. of LICS’02, pp. 189–202, IEEE.
- [8] G. Gottlob, C. Koch, *Monadic Datalog and the Expressive Power of Languages for Web Information Extraction*, In Journal of the ACM, 51(1):12–28, 2004.
- [9] G. Gottlob, C. Koch, R. Pichler, L. Segoufin, *The complexity of XPath query evaluation and XML typing* In Journal of the ACM 52(2):284–335, 2005.
- [10] W. Martens, F. Neven, *On the complexity of typechecking top-down XML transformations*, In Theoretical Computer Sc., 336(1): 153–180, 2005.
- [11] M. Marx, *XPath and Modal Logics for Finite DAGs*. In Proc. of TABLEAUX’03, pp. 150–164, LNAI 2796, 2003.

- [12] F. Neven, *Automata Theory for XML Researchers*, In SIGMOD Record 31(3), September 2002.
- [13] F. Neven, T. Schwentick, *Query automata over finite trees*, In Theoretical Computer Science, 275(1-2):633–674, 2002.
- [14] A. Potthof, S. Seibert, W. Thomas, *Nondeterminism versus determinism of finite automata over directed acyclic graphs*, In Bull. Belgian Math. Society, 1:285–298, 1994.
- [15] J.W. Thatcher, J.B. Wright, *Generalized finite automata theory with an application to a decision problem of second-order logic*, In Math. Syst. Theory, 2(1):57–81, 1968.