Efficient Inclusion Checking for Deterministic Tree Automata and DTDs

Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren

INRIA Futurs and Lille University, LIFL, Mostrare project

Abstract. We present a new algorithm for testing language inclusion $L(A) \subseteq L(B)$ between tree automata in time $O(|A| * |B|)$ where B is deterministic. We extend this algorithm for testing inclusion between automata for unranked trees A and deterministic DTDs D in time $O(|A| * |\Sigma| * |D|)$. No previous algorithms with these complexities exist.

1 Introduction

Language inclusion for tree automata is a basic decision problem that is closely related to universality and equivalence [5,14,15]. Tree automata algorithms are generally relevant for XML document processing [11,17,7,13]. Regarding inclusion checking, a typical application is inverse type checking for tree transducers [10]. Another one is schema-guided query induction [4], the motivation for the present study. There, candidate queries produced by the learning process are to be checked for consistency with deterministic DTDs, such as for HTML.

We investigate language inclusion $L(A) \subseteq L(B)$ for tree automata A and B under the assumption that B is (bottom up) deterministic, not necessarily A. Without this assumption the problem becomes DEXPTIME complete [15]. Deterministic language inclusion still subsumes universality of deterministic tree automata $L(B) = T_\Sigma$ up to a linear time reduction, as well as equivalence of two deterministic automata $L(A) = L(B)$. The converse might be false, i.e., we cannot rely on polynomial time equivalence tests, as for instance, by comparing number of solutions [14] or minimal deterministic tree automata.

In the case of standard tree automata for ranked trees, the well-known naive algorithm for inclusion goes through complementation. It first computes an automaton B^c that recognizes the complement of the language of B, and then checks whether the intersection automaton for B^c and A has a nonempty language. The problematic step is to complete B before complementing its final states, since completion might require to add rules for all possible left-hand sides. The overall running time may thus become $O(|A| * |\Sigma| * |B|^n)$, which is exponential in the maximal rank n of function symbols in the signature Σ.

It seems folklore that one can bound the maximal arity of a signature to 2. This can be done by transforming ranked trees into binary trees, and then converting automata for ranked trees into automata for binary trees correspondingly. The problem is to preserve determinism in such a construction, while the size of

automata should grow at most linearly. We show how to obtain such a transformation by stepwise tree automata [3,5]. Thereby we obtain an inclusion test in time $O(|A| * |\Sigma| * |B|^2)$. This is still too much in practice with DTDs, where A and B may be of size 500 and Σ of size 100.

Our first contribution is a more efficient algorithm for standard tree automata on binary trees that verifies inclusion in time $O(|A| * |B|)$ if B is deterministic. This bound is independent of the size of the signature, even though Σ is not fixed. As a second contribution, we show how to test inclusion between automata A for unranked trees and deterministic DTDs D in time $O(|A|*|\Sigma|*|D|)$. Determinism is required by the XML standards. Our algorithm first computes the Glushkov automata of all regular expressions of D in time $O(|\Sigma| * |D|)$, which is possible for deterministic DTDs [2]. The second step is more tedious. We would like to transform the collection of Glushkov automata to a deterministic stepwise tree automaton of the same size. Unfortunately, this seems difficult to achieve, since the usual construction of [9] eliminates ϵ-rules on the fly, which may lead to a quadratic blowup of the number of rules (not the number of states). We solve this problem by introducing factorized tree automata, which use ϵ-transitions for representing deterministic automata more compactly. We show how to adapt our inclusion test to factorized tree automata and thus to DTDs.

Related Work and Outline. Heuristic algorithms for inclusion between nondeterministic schemas that avoid the high worst-case complexity were proposed in [16]. The complexity of inclusion for various fragments of DTDs and extended DTDs was studied in [8]. The algorithms presented there assume the same types of regular expressions on both sides of the inclusion test. When applied to deterministic DTDs, the same complexity results may be obtainable. Our algorithm permits richer left-hand sides.

We reduce inclusion for the ranked case to the binary case in Section 2. The efficient algorithm for binary tree automata is given in Section 3. In Section 4, we introduce deterministic factorized tree automata and lift the algorithm for inclusion testing to them. Finally in Section 5, we apply them to testing inclusion of automata for unranked trees in deterministic DTDs.

2 Standard Tree Automata for Ranked Trees

We reduce the inclusion problem of tree automata for ranked trees [5] to the case of binary trees with a single binary function symbol.

A ranked signature Σ is a finite set of function symbols $f \in \Sigma$, each of which has an arity $n \geq 0$. A constant $a \in \Sigma$ is a function symbol of arity 0. A tree $t \in T_\Sigma$ is either a constant $a \in \Sigma$ or a tuple $f(t_1, \ldots, t_n)$ consisting of a function symbol f of arity n and n trees $t_1, \ldots, t_n \in T_\Sigma$.

A *tree automaton* (possibly with ϵ-rules) A over Σ consists of a finite set $\mathsf{s}(A)$ of states, a subset $\mathsf{final}(A) \subseteq \mathsf{s}(A)$ of final states, and a set $\mathsf{rules}(A)$ of rules of the form $f(p_1, \ldots, p_n) \to p$ or $p' \xrightarrow{\epsilon} p$ where $f \in \Sigma$ has arity n and $p_1, \ldots, p_n, p, p' \in \mathsf{s}(A)$. We write $p' \xrightarrow{\epsilon}_A p$ iff $p' \xrightarrow{\epsilon} p \in \mathsf{rules}(A)$, $\xrightarrow{\epsilon}{}^*_A$ for the

reflexive transitive closure of $\xrightarrow{\epsilon}_A$, and $\xrightarrow{\epsilon\,\leq 1}_A$ for the union of $\xrightarrow{\epsilon}_A$ and the identity relation on $\mathsf{s}(A)$.

The *size* $|A|$ of A is the sum of the cardinality of $\mathsf{s}(A)$ and the number of symbols in $\mathsf{rules}(A)$, i.e., $\sum_{f(p_1,\ldots,p_n)\to p\in\mathsf{rules}(A)}(n+2)$. The cardinality of the signature can be ignored, since our algorithms will not take unused function symbols into account. Every tree automaton A defines an evaluator $\mathsf{eval}_A : T_{\Sigma\cup\mathsf{s}(A)} \to 2^{\mathsf{s}(A)}$ such that $\mathsf{eval}_A(f(t_1,\ldots,t_n)) = \{p \mid p_1 \in \mathsf{eval}_A(t_1),\ldots,p_n \in \mathsf{eval}_A(t_n), f(p_1,\ldots,p_n) \to p' \in \mathsf{rules}(A), p' \xrightarrow{\epsilon}_A^* p\}$ and $\mathsf{eval}_A(p) = \{p\}$. A tree $t \in T_\Sigma$ is *accepted* by A if $\mathsf{final}(A) \cap \mathsf{eval}_A(t) \neq \emptyset$. The *language* $L(A)$ is the set of trees accepted by A.

A tree automaton is (bottom-up) *deterministic* if it has no ϵ-rules, and if no two rules have the same left-hand side. It is complete if there are rules for all potential left-hand sides. It is well-known that deterministic complete tree automata can be complemented in linear time, by switching the final states.

We will study the inclusion problem for tree automata, whose input consists of a ranked signature Σ, tree automata A with ϵ-rules and deterministic B, both over Σ. Its output is the truth value of $L(A) \subseteq L(B)$. We can deal with this problem by restriction to so-called stepwise signatures $\Sigma_@$, consisting of a single binary function symbol @ and a finite set of constants $a \in \Sigma$. A stepwise tree automaton [3] is a tree automaton over a stepwise signature.

Proposition 1. *The above inclusion problem for ranked trees can be reduced in linear time to the corresponding inclusion problem for stepwise tree automata over binary trees.*

We first encode ranked trees into binary trees via Currying. Given a ranked signature Σ we define the corresponding signature $\Sigma_@ = \{@\} \uplus \Sigma$ whereby all symbols of Σ become constants. Currying is defined by a function $\mathsf{curry} : T_\Sigma \to T_{\Sigma_@}$ which for all trees $t_1,\ldots,t_n \in T_\Sigma$ and $f \in \Sigma$ satisfies:

$$\mathsf{curry}(f(t_1,\ldots,t_n)) = f@\mathsf{curry}(t_1)@\ldots@\mathsf{curry}(t_n)$$

For instance, $f(a,g(a,b),c)$ is mapped to $f@a@(g@a@b)@c$ which is infix notation with left-most parenthesis for the tree $@(@(@(f,a),@(@(g,a),b)),c)$. Now we encode tree automata A over Σ into stepwise tree automata $\mathsf{step}(A)$ over $\Sigma_@$, such that the language is preserved up to Currying, i.e., such that $L(\mathsf{step}(A)) = \mathsf{curry}(L(A))$. The states of $\mathsf{step}(A)$ are the prefixes of left-hand sides of rules in A, i.e., words in $\Sigma(\mathsf{s}(A))^*$:

$$\mathsf{s}(\mathsf{step}(A)) = \{fq_1\ldots q_i \mid f(q_1,\ldots,q_n) \to q \in \mathsf{rules}(A),\ 0 \leq i \leq n\} \uplus \mathsf{s}(A)$$

Its rules extend prefixes step by step by states q_i according to the rule of A. Since constants do not need to be extended, we distinguish two cases in Fig. 1.

Lemma 1. *The encoding of tree automata A over Σ into stepwise tree automata $\mathsf{step}(A)$ over $\Sigma_@$ preserves determinism, the tree language modulo Currying, and the automata size up to a constant factor of 3.*

As a consequence, $L(A) \subseteq L(B)$ is equivalent to $L(\mathsf{step}(A)) \subseteq L(\mathsf{step}(B))$, and can be tested in this way modulo a linear time transformation. Most importantly, the determinism of B carries over to $\mathsf{step}(B)$.

$$\frac{f(q_1,\ldots,q_n)\to q\in\mathsf{rules}(A)\qquad 1\le i<n}{\begin{array}{c} f\to f\in\mathsf{rules}(\mathsf{step}(A))\\ fq_1\ldots q_{i-1}@q_i\to fq_1\ldots q_i\in\mathsf{rules}(\mathsf{step}(A))\\ fq_1\ldots q_{n-1}@q_n\to q\in\mathsf{rules}(\mathsf{step}(A))\end{array}}\qquad \frac{a\to q\in\mathsf{rules}(A)}{a\to q\in\mathsf{rules}(\mathsf{step}(A))}$$

Fig. 1. Transforming ranked tree automata into stepwise tree automata.

3 Stepwise Tree Automata for Binary Trees

We present a new inclusion test that applies to stepwise tree automata over binary trees. We first characterize inclusion into deterministic tree automata, second, express the characterization in Datalog [6] and third, turn it into an efficient algorithm. While the two first steps are easy, the last step is nontrivial.

Characterization of Inclusion. We call a state $p\in\mathsf{s}(A)$ *accessible* if there exists a tree t such that $p\in\mathsf{eval}_A(t)$. We call p *co-accessible* if there exists a tree $t\in T_{\Sigma\cup\{p\}}$ with a unique occurrence of p such that $\mathsf{eval}_A(t)\cap\mathsf{final}(A)\neq\emptyset$. A tree automaton is *productive* if all its states are accessible and co-accessible. We denote the *product* of two automata A and B with the same signature by $A\times B$. The state set of $A\times B$ is $\mathsf{s}(A)\times\mathsf{s}(B)$. For inferring its rules, we assume that B does not have ϵ-rules:

$$\frac{a\to p\in\mathsf{rules}(A)\quad a\to q\in\mathsf{rules}(B)}{a\to(p,q)}\qquad \frac{p_1@p_2\to p\in\mathsf{rules}(A)\quad q_1@q_2\to q\in\mathsf{rules}(B)}{(p_1,q_1)@(p_2,q_2)\to(p,q)}\qquad \frac{p'\xrightarrow{\epsilon}p\in\mathsf{rules}(A)\quad q\in\mathsf{s}(B)}{(p',q)\xrightarrow{\epsilon}(p,q)}$$

We do not care about final states of $A\times B$ since these are useless in our characterization of inclusion.

Proposition 2. *Inclusion $L(A)\subseteq L(B)$ for a productive stepwise tree automaton A with ϵ-rules and a deterministic stepwise tree automaton B fails iff:*

`fail`$_0$**:** *there exists a rule $a\to p\in\mathsf{rules}(A)$ but no state $q\in\mathsf{s}(B)$ such that $a\to q\in\mathsf{rules}(B)$, or*

`fail`$_1$**:** *there exist accessible states (p_1,q_1) and (p_2,q_2) of $A\times B$ and a rule $p_1@p_2\to p\in\mathsf{rules}(A)$ but no state $q\in\mathsf{s}(B)$ such that $q_1@q_2\to q\in\mathsf{rules}(B)$, or*

`fail`$_2$**:** *some accessible state (p,q) of $A\times B$ satisfies $p\in\mathsf{final}(A)$ but $q\notin\mathsf{final}(B)$.*

Proof. If one of the failure conditions holds, then failure of inclusion follows from the hypotheses that A is productive and B deterministic.

For the converse, let us consider a tree t such that $t\in L(A)$ and $t\notin L(B)$. There are two cases to be considered, depending on $\mathsf{eval}_B(t)$.

(i) Assume $\mathsf{eval}_B(t)=\emptyset$. There exists a minimal subtree t' of t such that $\mathsf{eval}_B(t')=\emptyset$, too. If $t'=a$ is a leaf then $\mathsf{eval}_A(a)\neq\emptyset$, since $t\in L(A)$, and $\mathsf{eval}_B(a)=\emptyset$, thus `fail`$_0$ holds. If $t'=t_1@t_2$, then there exist $p_1\in\mathsf{eval}_A(t_1)$, $p_2\in\mathsf{eval}_A(t_2)$ and $p_1@p_2\to p\in\mathsf{rules}(A)$, since $t\in L(A)$. Since t' is defined as a

$$(\mathrm{acc}_{/1})\ \frac{a \to p \in \mathsf{rules}(A) \quad a \to q \in \mathsf{rules}(B)}{\mathsf{acc}(p,q).} \qquad (\mathrm{acc}_{/2})\ \frac{p' \xrightarrow{\epsilon}_A p \in \quad q \in \mathsf{s}(B)}{\mathsf{acc}(p,q) :\!-\ \mathsf{acc}(p',q).}$$

$$(\mathrm{acc}_{/3})\ \frac{p_1 @ p_2 \to p \in \mathsf{rules}(A) \qquad q_1 @ q_2 \to q \in \mathsf{rules}(B)}{\mathsf{acc}(p,q) :\!-\ \mathsf{acc}(p_1,q_1), \mathsf{acc}(p_2,q_2).}$$

$$(\mathrm{frb})\ \frac{p_1 @ p_2 \to p \in \mathsf{rules}(A) \qquad \nexists q.q_1 @ q_2 \to q \in \mathsf{rules}(B)}{\begin{array}{l}\mathsf{frb}(p_2,q_2) :\!-\ \mathsf{acc}(p_1,q_1).\\ \mathsf{frb}(p_1,q_1) :\!-\ \mathsf{acc}(p_2,q_2).\end{array}}$$

$$(\mathrm{fail}_0)\ \frac{a \to p \in \mathsf{rules}(A) \qquad \nexists q.a \to q \in \mathsf{rules}(B)}{\mathtt{fail}_0.}$$

$$(\mathrm{fail}_1)\ \frac{p \in \mathsf{s}(A) \qquad q \in \mathsf{s}(B)}{\mathtt{fail}_1 :\!-\ \mathsf{acc}(p,q), \mathsf{frb}(p,q).} \qquad (\mathrm{fail}_2)\ \frac{p \in \mathsf{final}(A) \qquad q \notin \mathsf{final}(B)}{\mathtt{fail}_2 :\!-\ \mathsf{acc}(p,q).}$$

Fig. 2. Transforming tree automata A and B into a Datalog program $D_1(A,B)$.

minimal subtree and B is deterministic, $\mathsf{eval}_B(t_1) = \{q_1\}$, $\mathsf{eval}_B(t_2) = \{q_2\}$, and since $\mathsf{eval}_B(t') = \emptyset$, there is no rule $q_1 @ q_2 \to q \in \mathsf{rules}(B)$. This leads to $\mathtt{fail}_1$. **(ii)** If $\mathsf{eval}_B(t) \neq \emptyset$ then there exists $q \in \mathsf{eval}_B(t)$; B being deterministic this q is necessarily unique. Since $t \notin L(B)$, $q \notin \mathsf{final}(B)$. Moreover, since $t \in L(A)$, there exists $p \in \mathsf{eval}_A(t) \cap \mathsf{final}(A)$. This leads to $\mathtt{fail}_2$. □

Testing the Characterization. The following efficiency theorem for ground Datalog will be fundamental to all what follows. Given a Datalog program P (without negation), we write $\mathsf{lfp}(P)$ for its least fixed point semantics.

Theorem 1 (Efficiency of Ground Datalog [6]). *For every ground Datalog program P, the least fixed point semantics $\mathsf{lfp}(P)$ can be computed in linear time $O(|P|)$ where the size $|P|$ is the number of symbols in P.*

This result holds even without any bound on the arity of the relation symbols of P, which will be very useful later on. If relation symbols of higher arities are used, the number of their arguments is accounted for by the size of P.

Fig. 2 presents a Datalog program $D_1(A,B)$ that verifies the characterization of $L(A) \subseteq L(B)$ in Proposition 2. Transformation rules $(\mathrm{acc}_{/1})$, $(\mathrm{acc}_{/2})$, and $(\mathrm{acc}_{/3})$ define clauses for accessibility in $A \times B$ through predicate acc. The clauses produced by transformation rule (frb) define *forbidden states* of $A \times B$ through predicate frb. These are states that lead to $\mathtt{fail}_1$ when accessed. Transformation rules (fail_0), (fail_1), and (fail_2) define clauses for failures. The characterization of inclusion from Proposition 2 is captured in the following sense:

Proposition 3. *Let A and B be stepwise tree automata for binary trees. If A is productive and B deterministic then:*

$$L(A) \subseteq L(B) \Leftrightarrow \mathsf{lfp}(D_1(A,B)) \cap \{\mathtt{fail}_0, \mathtt{fail}_1, \mathtt{fail}_2\} = \emptyset$$

$$(\mathrm{frb}^c_{/2})\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_1 \in \mathsf{s}(B) \quad Q_2 = \{q_2 \mid q_1@q_2 \to q \in \mathsf{rules}(B)\}}{\mathsf{frb}^c(p_2, Q_2) :\!-\ \mathsf{acc}(p_1, q_1).}$$

$$(\mathrm{frb}^c_{/1})\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_2 \in \mathsf{s}(B) \quad Q_1 = \{q_1 \mid q_1@q_2 \to q \in \mathsf{rules}(B)\}}{\mathsf{frb}^c(p_1, Q_1) :\!-\ \mathsf{acc}(p_2, q_2).}$$

Fig. 3. Grouping (frb) transformations.

The sum of the sizes of the clauses defined by transformation rules ($\mathrm{acc}_{/1}$), ($\mathrm{acc}_{/2}$), ($\mathrm{acc}_{/3}$), (fail_0), (fail_1), and (fail_2) is $O(|A| * |B|)$. The sizes of the clauses defined by transformation rule (frb) sum up to $O(|A| * |\mathsf{s}(B)|^2)$. The overall size of the ground Datalog program $D_1(A, B)$ is $O(|A| * (|B| + |\mathsf{s}(B)|^2))$, which may be $O(|A| * |B|^2)$ in the worst case. Therefore, using Theorem 1, inclusion can be decided in time $O(|A| * |B|^2)$.

Efficient Algorithm. This running time is not better than that of the naive algorithm. The square factor is due to the computation of forbidden states for capturing $\mathtt{fail}_1$. Since (frb) rules cannot be inferred efficiently enough with a Datalog program, we introduce a new predicate frb^c that will group (frb) rules. Using an appropriate data structure, the frb predicates can be induced efficiently from frb^c. The semantics of the latter is given below:

$$A, B \models \mathsf{frb}^c(p, Q) \Leftrightarrow \forall q \in \mathsf{s}(B) \setminus Q,\ \ A, B \models \mathsf{frb}(p, q)$$

Formally, we impose an order $<$ on $\mathsf{s}(B)$ and consider $\mathsf{frb}^c(p, \{q_1, \ldots, q_n\})$ as $(n+1)$-ary literals $\mathsf{frb}^c(p,\ q_{i_1},\ \ldots,\ q_{i_n})$ such that $\{q_{i_1}, \ldots, q_{i_n}\} = \{q_1, \ldots, q_n\}$ and $q_{i_1} < \ldots < q_{i_n}$.

In Fig. 3, we propose two transformations ($\mathrm{frb}^c_{/1}$) and ($\mathrm{frb}^c_{/2}$) for inferring frb^c clauses, both of which group (frb) transformations. Note that for every state p, there may be several sets Q such that $\mathsf{frb}^c(p, Q)$ gets inferred. Therefore, we will have to test efficiently whether a state belongs to the union of complements of those state sets. This will be further detailed.

Let us consider the transformation of tree automata A and B into a ground Datalog program $D_2(A, B)$ defined by transformation rules ($\mathrm{acc}_{/1}$), ($\mathrm{acc}_{/2}$), ($\mathrm{acc}_{/3}$), ($\mathrm{frb}^c_{/1}$), ($\mathrm{frb}^c_{/2}$), (fail_0), and (fail_2). The clauses producing acc, $\mathtt{fail}_0$, and $\mathtt{fail}_2$ of $D_1(A, B)$ and $D_2(A, B)$ are identical and their number is in $O(|A| * |B|)$. The number of frb^c clauses introduced by rule ($\mathrm{frb}^c_{/1}$) is in $O(|A| * |\mathsf{s}(B)|)$ but the size of each such clause is $n+1$ which in the worst case could be $|\mathsf{s}(B)| + 1$, and symmetrically for ($\mathrm{frb}^c_{/2}$). The overall size of all frb^c clauses, however, is bounded by the overall number of acc clauses, which in turn is bounded by $O(|A| * |B|)$, too! To see this, we can rewrite ($\mathrm{frb}^c_{/2}$) as shown in Fig. 4, such that the corresponding ($\mathrm{acc}_{/3}$) clauses are inferred simultaneously (and these don't overlap). Therefore the overall size of $D_2(A, B)$ is in $O(|A| * |B|)$.

$$\frac{\begin{array}{rl} & q_1@q_2^1 \to q^1 \in \mathsf{rules}(B) \\ p_1@p_2 \to p \in \mathsf{rules}(A) & \vdots \\ & q_1@q_2^n \to q^n \in \mathsf{rules}(B) \end{array}\Bigg\} \text{all the rules for } q_1}{\begin{array}{c} \mathsf{frb}^c(p_2, \{q_2^1, \ldots, q_2^n\}) :\!- \ \mathsf{acc}(p_1, q_1). \\ \mathsf{acc}(p, q^1) :\!- \ \mathsf{acc}(p_1, q_1), \mathsf{acc}(p_2, q_2^1). \\ \vdots \\ \mathsf{acc}(p, q^n) :\!- \ \mathsf{acc}(p_1, q_1), \mathsf{acc}(p_2, q_2^n). \end{array}}$$

Fig. 4. Rewriting grouping rules for complexity analysis of ($\mathsf{frb}^c_{/2}$) clauses.

Inclusion Test. First it computes $\mathsf{lfp}(D_2(A,B))$ in time $O(|A|*|B|)$. If $\mathtt{fail}_0$ or $\mathtt{fail}_2$ belong to $\mathsf{lfp}(D_2(A,B))$ then inclusion does not hold, so false is returned. Otherwise, we test for $\mathtt{fail}_1$ in a second step, by checking for all states $\mathsf{acc}(p,q) \in \mathsf{lfp}(D_2(A,B))$ whether there is a $\mathsf{frb}^c(p,Q) \in \mathsf{lfp}(D_2(A,B))$ such that $q \in \mathsf{s}(B) \setminus Q$. If so, false is returned, otherwise true.

We have to prove that the second step can be done in time $O(|A| * |B|)$. For every state $p \in \mathsf{s}(A)$, there are some state sets $Q_1, \ldots, Q_m$ such that for $1 \leq j \leq m$, $\mathsf{frb}^c(p, Q_j) \in \mathsf{lfp}(D_2(A,B))$ and we have to check efficiently whether some state q is in $\bigcup_{j=1}^{j=m} \mathsf{s}(B) \setminus Q_j$. For this, we define a data structure $\mathtt{bad\_states}(p)$ as an array T of size $|\mathsf{s}(B)| + 1$. Counters are indexed by elements in $\mathsf{s}(B)$ and one counter is indexed by 0. All counter values are set to 0 initially. The initialization of all $\mathtt{bad\_states}(p)$ can be done in $O(|\mathsf{s}(A)| * |\mathsf{s}(B)|)$. For every p and every Q_j such that $\mathsf{frb}^c(p, Q_j) \in \mathsf{lfp}(D_2(A,B))$, counter $T[0]$ is incremented by 1 and counter $T[q]$ is incremented by 1 for all $q \in Q_j$, which can be done in time $O(|Q_j|)$. As the overall size of frb^c clauses in $\mathsf{lfp}(D_2(A,B))$ is in $O(|A|*|B|)$, the computation of all $\mathtt{bad\_states}(p)$ can be done in time $O(|A| * |B|)$.

	0	q_1	q_2	q_3	q_4	q_5	...
p	2	0	1	2	1	0	...

Fig. 5. Data structure $\mathtt{bad\_state}$. Here, it has been set up $\mathsf{frb}^c(p, \{q_2, q_3\})$ and $\mathsf{frb}^c(p, \{q_3, q_4\})$. When $\mathsf{frb}^c(p, \{q_2, q_3\})$ is set up, $\mathtt{bad\_state}(p)(0)$, $\mathtt{bad\_state}(p)(q_2)$ and $\mathtt{bad\_state}(p)(q_3)$ are incremented. We have for instance $\mathsf{frb}(p, q_4)$ since $\mathtt{bad\_state}(p)(q_4) = 1$ is lower than $\mathtt{bad\_state}(p)(0) = 2$. In fact, the only *not forbidden* state is (p, q_3) because q_3 belongs to the intersection of $\{q_2, q_3\}$ and $\{q_3, q_4\}$.

It remains to test for all states $\mathsf{acc}(p,q) \in \mathsf{lfp}(D_2(A,B))$ whether there exists a $\mathsf{frb}^c(p,Q) \in \mathsf{lfp}(D_2(A,B))$ such that $q \in \mathsf{s}(B) \setminus Q$. This is done by checking whether $T[q] < T[0]$ in $\mathtt{bad\_states}(p)$ (see, e.g., Fig. 5). Indeed, if $\{Q \mid \mathsf{frb}^c(p,Q) \in \mathsf{lfp}(D_2(A,B))\} = \{Q_1, \ldots, Q_m\}$ then $\mathtt{bad\_states}(p)$ is defined such that $T[q] = T[0]$ iff $q \in \bigcap_{j=1}^{j=m} Q_j$, thus $T[q] < T[0]$ iff $q \in \bigcup_{j=1}^{j=m} \mathsf{s}(B) \setminus Q_j$. Each such test costs $O(1)$ so the overall time is bounded by $O(|A| * |B|)$.

This concludes the inclusion test for stepwise tree automata, for productive A and deterministic B. Every tree automaton can be made productive in linear time. Higher arities can be reduced to 2 by Proposition 1. This yields:

Theorem 2. *Let A and B be standard tree automata for ranked trees of some signature Σ possibly with ϵ-rules. If B is deterministic, inclusion $L(A) \subseteq L(B)$ can be decided in time $O(|A| * |B|)$ independently of the size of Σ.*

4 Factorized Tree Automata

We next relax the determinism assumption on B in a controlled manner, that will be crucial to deal with DTDs. We replace B by deterministic factorized automata, that we introduce. These are automata with ϵ-rules, that represent deterministic automata in a compact manner.

Definition 1. *A* factorized tree automaton F *over a stepwise signature Σ is a stepwise tree automaton with ϵ-rules and a partition $\mathsf{s}(F) = \mathsf{s}_1(F) \uplus \mathsf{s}_2(F)$ such that if $q_1@q_2 \to q$ in $\mathsf{rules}(F)$ then $q_1 \in \mathsf{s}_1(F)$ and $q_2 \in \mathsf{s}_2(F)$.*

We say that q is of sort i in F if $q \in \mathsf{s}_i(F)$. The sort determines which states may be used in the i-th position of the binary symbol @ in rules of F.

Every factorized automaton F defines a tree automaton $\mathsf{ta}(F)$ without ϵ-rules that recognizes the same language. Both automata have the same signature and states; the rules of $\mathsf{ta}(F)$ are inferred as follows from those of F:

$$(\mathrm{E}_1)\ \frac{a \to q \in \mathsf{rules}(F)}{a \to q \in \mathsf{rules}(\mathsf{ta}(F))} \quad (\mathrm{E}_2)\ \frac{q_1 \xrightarrow{\epsilon}{}^*_F r_1 \quad q_2 \xrightarrow{\epsilon}{}^*_F r_2 \quad r_1@r_2 \to q \in \mathsf{rules}(F)}{q_1@q_2 \to q \in \mathsf{rules}(\mathsf{ta}(F))}$$

We set $\mathsf{final}(\mathsf{ta}(F)) = \{q \mid q \xrightarrow{\epsilon}{}^*_F r,\ r \in \mathsf{final}(F)\}$. Note that the size of $\mathsf{ta}(F)$ may be $O(|\mathsf{rules}(F)| * |\mathsf{s}(F)|^2)$ which is cubic in that of F in the worst case. Besides their succinctness, the truly interesting bit about factorized tree automata is their notion of determinism.

Definition 2. *A factorized tree automaton F is (bottom-up)* deterministic *if:*

d_0*: the ϵ-free part of F is (bottom-up) deterministic;*
d_1*: for all $q \in \mathsf{s}(F)$ and sorts $i \in \{1, 2\}$, there is at most one state r of sort i such that $q \xrightarrow{\epsilon}{}^*_F r$.*

Nonredundant ϵ-rules must change the sort: if $q \xrightarrow{\epsilon}_F r$ for two states of the same sort then $r = q$ by d_1 and $q \xrightarrow{\epsilon}{}^*_F q$. A similar argument shows that all proper chains of ϵ-rules are redundant so that $\xrightarrow{\epsilon}{}^*_F$ is equal to $\xrightarrow{\epsilon}{}^{\leq 1}_F$.

Proposition 4. *The tree automaton $\mathsf{ta}(F)$ represented by a deterministic factorized tree automaton F is deterministic.*

Proof. Let $B = \mathsf{ta}(F)$ which by construction is free of ϵ-rules. For every constant $a \in \Sigma$, the uniqueness of q such that $a \to q \in \mathsf{rules}(B)$ follows from d_0. For every $q_1@q_2 \to q$ in $\mathsf{rules}(B)$ we have to show that q is uniquely determined by q_1 and q_2. By d_1 there is at most one state r_1 of sort 1 such that $q_1 \xrightarrow{\epsilon}{}^*_F r_1$ at most one r_2 of sort 2 such that $q_2 \xrightarrow{\epsilon}{}^*_F r_2$. Condition d_0 implies that there exists at most one state q such that $r_1@r_2 \to q \in \mathsf{rules}(F)$. □

$$(\mathrm{acc}_{/3a})\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_1@q_2 \to q \in \mathsf{rules}(F)}{\mathsf{acc}(p,q) :\!-\ \mathsf{f.acc}(p_1,q_1), \mathsf{f.acc}(p_2,q_2).} \qquad (\mathrm{f.acc})\ \frac{p \in \mathsf{s}(A) \quad q \xrightarrow{\epsilon}{}^{\leq 1}_{F} r}{\mathsf{f.acc}(p,r) :\!-\ \mathsf{acc}(p,q).}$$

$$(\mathrm{f.frb}^c_2)\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_1 \in \mathsf{s}_1(F)}{\mathsf{f.frb}^c_2(p_2, Q^F_2(q_1)) :\!-\ \mathsf{f.acc}(p_1,q_1).} \qquad (\mathrm{f.frb}^c_1)\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_2 \in \mathsf{s}_2(F)}{\mathsf{f.frb}^c_1(p_1, Q^F_1(q_2)) :\!-\ \mathsf{f.acc}(p_2,q_2).}$$

$$(\mathrm{frb}^c_2)\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_1 \in \mathsf{s}(F)}{\mathsf{frb}^c_2(p_2, R^F_2) :\!-\ \mathsf{acc}(p_1,q_1).} \qquad (\mathrm{frb}^c_1)\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_2 \in \mathsf{s}(F)}{\mathsf{frb}^c_1(p_1, R^F_1) :\!-\ \mathsf{acc}(p_2,q_2).}$$

$$(\mathrm{frb}^c_{/1a})\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_2 \notin R^F_2}{\mathsf{frb}^c(p_1, \emptyset) :\!-\ \mathsf{acc}(p_2,q_2).} \qquad (\mathrm{frb}^c_{/2a})\ \frac{p_1@p_2 \to p \in \mathsf{rules}(A) \quad q_1 \notin R^F_1}{\mathsf{frb}^c(p_2, \emptyset) :\!-\ \mathsf{acc}(p_1,q_1).}$$

$$(\mathrm{fail}_{2a})\ \frac{p \in \mathsf{final}(A) \quad \forall r.\ q \xrightarrow{\epsilon}{}^{\leq 1}_{F} r \Rightarrow r \notin \mathsf{final}(F)}{\mathtt{fail}_2 :\!-\ \mathsf{acc}(p,q).}$$

The clauses from $(\mathrm{acc}_{/1})$, $(\mathrm{acc}_{/2})$ and (fail_0) in $D_2(A,F)$ belong to $D_3(A,F)$, too. We use sets of states $Q^F_2(q_1) = \{q' \mid q_1@q' \to q'' \in \mathsf{rules}(F)\}$, $Q^F_1(q_2)$ symmetrically, and sets of states reaching a sort $R^F_i = \{q \mid \exists r \in \mathsf{s}_i(F).\ q \xrightarrow{\epsilon}{}^{\leq 1}_{F} r\}$.

Fig. 6. Inferring clauses of Datalog program $D_3(A,F)$ simulating $D_2(A,B)$.

We fix a stepwise tree automaton A and a deterministic factorized tree automaton F, and let us $B = \mathsf{ta}(F)$. We now show how to test language inclusion $L(A) \subseteq L(B)$ from A and F without computing B. This is done by the ground Datalog program $D_3(A,F)$ of Fig. 6.

We need new predicates for properties of F in order to infer corresponding properties of B. The accessibility predicate $\mathsf{f.acc}$ for F subsumes the accessibility predicate acc for B. Subsumption may by proper as stated by the rule (f.acc) of $D_3(A,F)$. *Vice versa*, we infer accessibility in F from accessibility in B according to the rule $(\mathrm{acc}_{/3a})$. Rules $(\mathrm{acc}_{/1})$ and $(\mathrm{acc}_{/2})$ of $D_2(A,F)$ remain valid for accessibility in B, too.

Lemma 2. $\mathsf{acc}(p,q) \in \mathsf{lfp}(D_3(A,F))$ *iff* $\mathsf{acc}(p,q) \in \mathsf{lfp}(D_2(A,B))$

We need to refine predicate frb into predicates frb_1 and frb_2 that take sorts into account, and corresponding predicates $\mathsf{f.frb}_1$ and $\mathsf{f.frb}_2$ in the factorized case. Their semantics can be defined as follows, where A, B are tree automata and F is a factorized tree automaton.

$$A,B \models \mathsf{frb}_2(p_2,q_2) \Leftrightarrow \exists p,p_1,q_1.\ A,B \models \mathsf{acc}(p_1,q_1),\ p_1@p_2 \to p \in \mathsf{rules}(A),\ q_2 \notin Q^B_2(q_1)$$
$$A,F \models \mathsf{f.frb}_2(p_2,r_2) \Leftrightarrow \exists p,p_1,r_1.\ A,F \models \mathsf{f.acc}(p_1,r_1),\ p_1@p_2 \to p \in \mathsf{rules}(A),\ r_2 \notin Q^F_2(r_1)$$

The semantics of frb_1 and $\mathsf{f.frb}_1$ are symmetric. The relation to the previous predicate frb is that $A,B \models \mathsf{frb}(p,q)$ if and only if $A,B \models \mathsf{frb}_1(p,q) \vee \mathsf{frb}_2(p,q)$.

The Datalog program $D_3(A,F)$ infers for sorts $i \in \{1,2\}$ literals with predicates $\mathsf{f.frb}^c_i$ that are to be understood by grouping of $\mathsf{f.frb}_i$ literals, and similarly frb^c_i by grouping of frb_i. These grouping mechanisms account for sorts, since complementation is with respect to sorts.

$$A, F \models \mathsf{f.frb}_i^c(p, Q) \Leftrightarrow \forall q \in \mathsf{s}_i(F) \setminus Q.\ A, F \models \mathsf{f.frb}_i(p, q)$$
$$A, F \models \mathsf{frb}_i^c(p, Q) \Leftrightarrow \forall q \in \mathsf{s}_i(F) \setminus Q.\ A, F \models \mathsf{frb}_i(p, q)$$

The clauses produced by ($\mathrm{f.frb}_i^c$) and (frb_i^c) are sound with respect to this semantics for deterministic F's. This is easier to see for ($\mathrm{f.frb}_i^c$) than for (frb_i^c). We prove it by the next lemma. For states p, q, r and sorts $i \in \{1, 2\}$ we define:

$$\begin{array}{lll}
A, B \vdash \mathsf{frb}_i(p, q) & \text{iff} & \exists Q \subseteq \mathsf{s}(B) \setminus \{q\}.\ \mathsf{frb}^c(p, Q) \in \mathsf{lfp}(D_2(A, B)) \textit{ via } (\mathrm{frb}^c_{/i}) \\
A, F \vdash \mathsf{f.frb}_i(p, r) & \text{iff} & \exists R \subseteq \mathsf{s}(F) \setminus \{r\}.\ \mathsf{f.frb}_i^c(p, R) \in \mathsf{lfp}(D_3(A, F)) \\
A, F \vdash \mathsf{frb}_i(p, r) & \text{iff} & \exists R \subseteq \mathsf{s}(F) \setminus \{r\}.\ \mathsf{frb}_i^c(p, R) \in \mathsf{lfp}(D_3(A, F)) \\
 & & \quad \text{or } \mathsf{frb}^c(p, \emptyset) \in \mathsf{lfp}(D_3(A, F)) \textit{ via } (\mathrm{frb}^c_{/ia})
\end{array}$$

Lemma 3 (Core). *$A, B \vdash \mathsf{frb}_i(p, q)$ iff $A, F \vdash \mathsf{frb}_i(p, q)$ or the unique state r of sort i with $q \xrightarrow[F]{\epsilon}{}^{\leq 1} r$ exists and satisfies $A, F \vdash \mathsf{f.frb}_i(p, r)$.*

Since the size of $D_3(A, F)$ is in $O(|A| * |F|)$ we can compute the set of all $\mathsf{f.frb}_1^c(p, R)$ and $\mathsf{f.frb}_2^c(p, R)$ literals in $\mathsf{lfp}(D_3(A, F))$ in time $O(|A| * |F|)$. It remains to infer the induced literals $\mathsf{f.frb}_i(p, r)$ literals in an efficient manner. Computing all $A, F \vdash \mathsf{f.frb}_i(p, r)$ from $\mathsf{lfp}(D_3(A, F))$ can be done by the same clever algorithm as before for deducing all $A, B \vdash \mathsf{frb}(p, q)$ given $\mathsf{lfp}(D_2(A, B))$. Note, however, that we now need two different data structures for the two sorts.

Theorem 3. *For a stepwise tree automaton with ϵ-rules A and a deterministic factorized tree automaton F over the same signature, inclusion $L(A) \subseteq L(F)$ can be decided in time $O(|A| * |F|)$.*

5 Automata for Unranked Trees and DTDs

We lift our results to deterministic tree automata for unranked trees possibly with factorization, so that they become applicable to deterministic DTDs.

An unranked signature Σ is a finite set of symbols (without arity restrictions). The set T^u_Σ of unranked trees over Σ is the least set that contains all pairs $a(t_1, \ldots, t_n)$ where $a \in \Sigma$ and $(t_1, \ldots, t_n)$ is a possibly empty sequence of unranked trees in T^u_Σ. Currying carries over literally from ranked to unranked trees. This yields a bijection $\mathsf{curry} : T^u_\Sigma \to T_{\Sigma_@}$. Thus, we can reuse stepwise tree automata to recognize languages of unranked trees, and as before, we can factorize them. So, as a corollary of Theorem 3 we have:

Corollary 2. *For a stepwise tree automaton for unranked trees A and a deterministic factorized tree automaton for unranked trees F over the same signature Σ, $L(A) \subseteq L(F)$ can be decided in time $O(|A| * |F|)$ independently of $|\Sigma|$.*

Note that hedge automata [5] can be translated in linear time to stepwise tree automata with ϵ-rules [9]. The automata of [12] support factorization, too. We finally show how to convert deterministic DTDs D to deterministic factorized tree automata in time $O(|\Sigma| * |D|)$, so that we can reuse our algorithm for testing inclusion in deterministic DTDs. Factorization avoids the quadratic blowup from translating hedge to stepwise automata [9].

```
<!ELEMENT doc    (block+)>
<!ELEMENT block  (text ,( link , text ?)?
                 | link , text ?)>
<!ELEMENT text   (#PCDATA)>
<!ELEMENT link   (#PCDATA)>
```

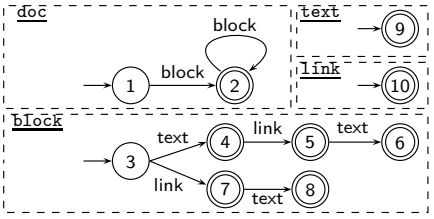

Fig. 7. An example DTD and the corresponding Glushkov automata.

A DTD D with elements in a set Σ is a function mapping letters $a \in \Sigma$ to regular expressions e over Σ, what we write $a \to_D e$ in this case. One of these elements is the distinguished start symbol. The language $L_a(D) \subseteq T^u_\Sigma$ of elements a of a DTD D is the smallest set of unranked trees such that:

$$L_a(D) = \{a(t_1, \ldots, t_n) \mid a \to_D e,\ a_1 \ldots a_n \in L(e), t_i \in L_{a_i}(D) \text{ for } 1 \leq i \leq n\}$$

The language of a DTD D is $L(D) = L_a(D)$ where a is the start symbol of D. The size of D is the total number of symbols in the regular expressions of D. An example in XML syntax is given in Fig. 7. The set of elements of D is $\Sigma = \{\texttt{doc}, \texttt{block}, \texttt{text}, \texttt{link}\}$, of which the element `doc` is the start symbol. The regular expression for `#PCDATA` recognizes only the empty word.

A DTD is deterministic if all its regular expressions are one-unambiguous [2]. This is equivalent to say that all corresponding Glushkov automata are deterministic, which is a requirement by the W3C. See Fig. 7 for the example.

Theorem 4 (Brüggemann-Klein [1]). *The collection of Glushkov automata for a deterministic DTD D over Σ can be computed in time $O(|\Sigma| * |D|)$.*

We transform a collection of Glushkov automata for a deterministic DTD D into a single factorized tree automaton F as follows. The set of states of sort 1 of F is the disjoint union of the states of the Glushkov automata. The states of sort 2 of F are the elements of D. For every element a, we connect all final states q of its Glushkov automaton to the state a, i.e., $q \xrightarrow{\epsilon} a \in \mathsf{rules}(F)$. The only final state of F is the start symbol of the DTD D. The result is a finite automaton, that represents a factorized tree automaton, as for instance in Fig. 8. This needs time of at most $O(|\Sigma| * |D|)$. Note that the size of the example automaton would grow quadratically, when eliminating ϵ-edges. For every $a \in \Sigma$, there is $a \to q \in \mathsf{rules}(F)$ for

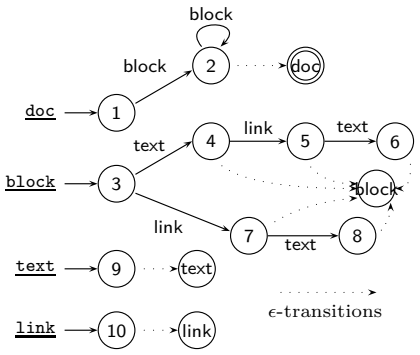

Fig. 8. The deterministic factorized tree automaton for the DTD in Fig. 7.

the unique initial state q of the Glushkov automaton of a. For every transition $q \xrightarrow{a} q'$ of one of the Glushkov automata, we add a rule $q@a \to q' \in \mathsf{rules}(F)$.

Note that F is deterministic as a factorized automaton. The ϵ-free part of F is deterministic since all Glushkov automata are: $\mathtt{d}_0$. Let q be a state of the Glushkov automaton for some letter a. The only state of sort 1 q can reach by ϵ-edges is a and the only state of sort 2 is q itself. All other states of F are elements of $a \in \Sigma$, which have no outgoing ϵ-edges: $\mathtt{d}_1$.

Theorem 5. *Deterministic DTDs D over Σ can be translated to deterministic factorized tree automata that recognize the same language in time $O(|\Sigma| * |D|)$.*

Corollary 3. *Language inclusion of hedge automata A over Σ in deterministic DTDs D with elements in Σ can be decided in time $O(|A| * |\Sigma| * |D|)$.*

References

1. A. Brüggemann-Klein. Regular expressions to finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.
2. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.
3. J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In 19th RTA, vol. 3091 of LNCS, 105 – 118. 2004.
4. J. Champavère, R. Gilleron, A. Lemay, and J. Niehren. Towards schema-guided XML query induction. In *ICML CAGI Workshop*, 2007.
5. H. Comon et al. Tree automata techniques and applications. Available on: http://tata.gforge.inria.fr, 2007.
6. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM computing surveys*, 33(3):374–425, 2001.
7. S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML type checking with macro tree transducers. In *24th PODS*, pages 283–294, 2005.
8. W. Martens, F. Neven, and T. Schwentick. Complexity and decision problems for XML Schemas and chain regular expressions. 2008. Journal extension of MFCS'04.
9. W. Martens and Joachim Niehren. On the minimization of XML schemas and tree automata for unranked trees. *J. of Comp. and Sys. Sci.*, 73(4):550–583, 2007.
10. T. Milo, D. Suciu, and V. Vianu. Type checking XML transformers. *J. of Comp. and Sys. Sci.*, 1(66):66–97, 2003.
11. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In FSTTCS, pages 134–145, 1998.
12. Stefan Raeymaekers. *Information Extraction from Web Pages Based on Tree Automata Induction.* PhD thesis, Katholieke Universiteit Leuven, 2008.
13. T. Schwentick. Automata for XML—a survey. *J. of Comp. and Sys. Sci.*, 73(3):289–315, 2007.
14. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424-437, 1990.
15. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.
16. A. Tozawa and M. Hagiya. XML schema containment checking based on semi-implicit techniques. In *Int. Conf. on Impl. and Appl. of Automata*, 2003.
17. S. Dal Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In 18th RTA, volume 2706 of LNCS, pages 246–263. 2003.