Queries on XML Streams with Bounded Delay and Concurrency

Olivier Gauwin, Joachim Niehren, Sophie Tison

Queries on XML Streams with Bounded Delay and Concurrency

Olivier Gauwin [1,2,3], Joachim Niehren [1,3], Sophie Tison [2,3]

Abstract

Query answering algorithms on XML streams check answer candidates on the fly in order to avoid the unnecessary buffering whenever possible. The delay and concurrency of a query are two measures for the degree of their streamability. They count the maximal number of stream elements during the life time for some query answer, and respectively, the maximal number of simultaneously alive answer candidates of a query. We study queries defined by deterministic nested word automata, which subsume large streamable fragments of XPath subject to schema restrictions by DTDs modulo P-time translations. We show that bounded and k-bounded delay and concurrency of such automata-defined queries are all decidable in polynomial time in the size of the automaton. Our results are obtained by P-time reduction to the bounded valuedness problem for recognizable relations between unranked trees, a problem that we show to be decidable in P-time.

Keywords: streaming, tree automata, XML, databases, XPath.

1. Introduction

Streaming algorithms are relevant for XML databases and data exchange, whenever large data collections that cannot be stored in main memory are to be processed. Instead data is communicated over streams and processed on the fly. Recently, XML streaming algorithms were proposed for schema validation [49] (membership in tree languages), one-pass typing [36] (annotating nodes of trees by types), and query answering [33, 22, 8, 9, 25].

An XML streaming algorithm inputs a linearization of an XML document (as a series of events) on an external stream, computes its output incrementally, and writes it to an external output device. The document on the stream is processed in a single reading pass. A central quality criterion of streaming algorithms is memory efficiency. In the best case, the required space should be independent of the size of the input document. Furthermore, the time complexity should be polynomial in the size of the query, in the space that is required, and in the size of the output that is produced. We refer to [26] for a discussion of streamability notions for query languages that capture all these aspects.

[1]INRIA, Lille
[2]University of Lille 1
[3]Mostrare project, INRIA & LIFL (CNRS UMR8022)

Streaming algorithms for a given XML query compute the answer set of the query on the fly. They can be understood as some kind of pushdown transducers that operate on linearizations of XML documents. Their space complexity depends on the number of states, the size of the pushdown which of is often bounded by the depth of the tree, and the maximal number of answer candidates stored in main memory simultaneously.

Bounded Concurrency and Delay

The concurrency of an n-ary node-selection query is the maximal number of simultaneously alive answer candidates while processing the stream [5]. An answer candidate is *alive* at an event of the input stream, if it can still be selected in some continuation of the stream and rejected in some other. Hence the selection and rejection of this candidate cannot be decided at this event. Note that the notions of aliveness and concurrency do only depend on the query and not on the particular query answering algorithm. The concurrency of a monadic query is a lower bound for the space requirement of any streaming algorithm that answers the query on realistic machine [25, 26], or on unrestricted machine models for restricted query languages [5].

The delay of a query is a lower bound for waiting times on query answers, *i.e.*, the maximal number of events between visiting and selecting an answer element on the stream. Queries with bounded delay permit high quality query answering algorithms, where the waiting time for query answers is bounded. We will see that bounded delay implies bounded concurrency for monadic node selection queries (but not for binary queries).

Relevance for XPath Queries.

We illustrate the relevance of bounded and k-bounded delay and concurrency in practice at a sample of XPath queries on XML documents. These documents represent bibliographies satisfying the following DTD:

```
B -> P*    P -> T A*    T -> #PCDATA   A -> #PCDATA
```

This schema definition states that a bibliography (`B`) is a list of publications (`P`), that a publication is a pair of a title (`T`) and a list of authors (`A`), and that titles and authors are data values. Publications can thus be abstracted into flat unranked trees such as for instance:

```
P(T('found. of databases'),A('abiteboul'),A('hull'),A('vianu'))
```

The corresponding XML stream is a list of opening tags, closing tags, and data values, *i.e.* the following word:

```
<P> <T> 'found. of databases' </T> <A> 'abiteboul' </A>
    <A> 'hull' </A> <A> vianu </A> </P>
```

We start with three queries (`Q1`, `Q2`, `Q3`) with increasing bounded delay and then present two queries with infinite delay (`Q4` and `Q5`). The first query `Q1` selects all title nodes of publications. `Q1` satisfies $delay(t) = concur(t) = 0$ for all bibliographies t:

```
Q1:   //P/T
```

A streaming algorithm for `Q1` can select all `T`-nodes of bibliographies t immediately at opening time (*i.e.* when reading the opening tag `<T>`), so with delay 0. All other nodes can be immediately rejected at opening time, so the concurrency is 0 too.

The next query `Q2` selects all publications whose title contains the substring `'XML'`. `Q2` satisfies $delay(t) \leq 2$ and $concur(t) \leq 1$ for all bibliographies t:

```
Q2: //P[T[contains(.,'XML')]]
```

In order to decide the selection of a `P`-node of a bibliography t, the subsequent `T`-node must be opened, and then the subsequent data value must be read. The delay of t is thus bounded by 2. At any time point at most one title node must be stored so the concurrency is at most 1.

Next, we consider the query `Q3` that selects all co-authors of `'abiteboul'` in some publication with at most 10 authors. It satisfies for all bibliographies t that $delay(t) \leq 28$ and $concur(t) \leq 9$:

```
Q3: //P[count(A)<=10 and A[text()='abiteboul']]/
       A[not(text()='abiteboul')]
```

In order to decide the selection of an `A`-node in a bibliography t, in the worst case one might have to inspect the data values of 9 follow-up `A`-nodes. The delay thus contains at most 9 opening `A`-nodes, 9 closing `A`-nodes, and 10 data value nodes, and it thus bounded by 28. The number of candidates that are to be stored at the same time point may be at most 9 since at most 9 A-nodes may be undecided simultaneously, so the concurrency is bounded by 9.

We now consider the query `Q4` that selects all co-authors of `'abiteboul'` in some publication. Here both the delay and concurrency are unbounded for varying bibliographies, *i.e.*, $\sup_t delay(t) = \sup_t concur(t) = \infty$:

```
Q4: //P[A[text()='abiteboul']]/A[not(text()='abiteboul')]
```

Since the number of authors of a publication may be unbounded a priori, the delay of selection may be unbounded and also the concurrency. This illustrates that many practically streamable queries do not have bounded concurrency. In this case however, it is often possible to rewrite the query as done for `Q4` in `Q3` or else the schema in order to restrict node selection to documents admissible in practice.

Query `Q5` selects all publications whose last author is `'abiteboul'`. It satisfies $\sup_t delay(t) = \infty$, even though $concur(t) \leq 1$ for all bibliographies t.

```
Q5: //P[A[text()='abiteboul' and not(following-sibling::A)]]
```

This is an example of a query with a unbounded delay but small concurrency. The converse is false for monadic queries, where bounded delay always implies bounded concurrency. This fact may be relevant in cases where deciding bounded delay is more costly than deciding bounded concurrency.

All queries above except for `Q3` and `Q5` belong to the finitely streamable fragment of Forward XPath distinguished in [25, 26] where the branching width of the path expression is bounded by 3 (modulo considering data values as tags) and can thus be compiled in polynomial time to deterministic nested word automata (dNWAs) [4]. What is needed to capture `Q3` and `Q5` in addition is a streamable extension of these fragments of XPath with next-sibling axis.

It should also be noticed that flat unranked trees are essentially words since they contain no relevant nesting structure so that one can remove all closing tags. Therefore it is natural to start our study of streaming algorithms for queries defined by deterministic finite automata (dFAs) on words, and only to move to queries on nested words (trees) defined by dNWAs in a second step.

Contributions.

In this paper, we show that bounded delay and bounded concurrency are both decidable in P-time for n-ary queries and schemas defined by dNWAs [4, 2, 3]. Queries by dNWAs subsume large streamable fragments of XPath [25, 26] with schema restrictions defined by dNWAs, and thus with extended DTDs with restrained competition [38, 36].

It should also be noticed that schema restrictions may be essential to reduce delay and concurrency, as they permit to restrict the set of possible continuations of XML streams. The choice of dNWAs as automata notion for unranked trees is equally justified by their generality: dNWAs subsume bottom-up deterministic tree automata that operate on bottom-up binary encodings of unranked trees (currying) and on top-down deterministic tree automata operating on top-down encodings of unranked trees (first-child next-sibling encoding). These relationships are worked out in the appendix in more details.

In the subcase of dFA queries on words (or flat unranked trees), we can reduce bounded delay and concurrency of dFA-queries on words in P-time to bounded ambiguity of finite automata, which can be decided in cubic time [57, 1]. The algorithm for dFA queries on words, however, cannot be lifted to dNWA queries on trees in any straightforward manner, mainly since dNWAs have stacks in contrast to dFAs. In particular, it seems not sufficient to replace bounded ambiguity of nFAs by bounded ambiguity of NWAs or of tree automata on ranked trees. Instead, we propose another solution by reduction to bounded valuedness of bottom-up tree transducers for ranked trees.

A central idea of our reduction is to define aliveness of automata queries on unranked trees by recognizable relations between unranked trees [11]. More precisely, we define the aliveness for a dNWA query A by a possibly non-deterministic NWA B of size polynomial in the size of A. The most surprising point here is polynomiality, in particular since the direct construction of a dNWA for aliveness from [29] produced dNWAs of exponential size. Instead, we propose to construct a non-deterministic NWA B by an existential FO-logic formula over basic recognizable relations depending on A. The restriction to existential FO-formula is crucial to obtain a definition in P-time. Indeed, we conjecture that B cannot always be constructed deterministically from A without growing exponentially.

As an intermediate result of its own interest, we show that bounded valuedness of recognizable relations between unranked trees is decidable in P-time. This result can be reduced in P-time to the analogous result for recognizable relations between ranked trees [56] via binary encoding of unranked trees. We then present a further P-time reduction to bounded valuedness of bottom-up tree transducers for ranked trees, which can be decided in P-time [52].

We also show for fixed k that k-boundedness of NWA-recognizable relations can be decided in P-time. Our decision procedure relies on direct automata construction. Note that it is open whether the more general problem of k-bounded valuedness of bottom-up tree transducers is decidable in P-time. The best algorithm so far is in coNP-time [53].

We added some new results to the LATA conference version [28] of the present article. First, we showed that k-bounded delay and concurrency for fixed k can be decided in P-time (Theorem 6), by proving that k-bounded valuedness of binary recognizable relations is decidable in P-time (Theorem 7). Second, we proved the EXPTIME-hardness of deciding whether a binary recognizable relation has k-bounded valuedness, when k is variable (Theorem 9). Third, we added the result that bounded delay implies bounded concurrency for monadic queries (Proposition 6). Fourth, we now show in the case of words, how to compute the delay efficiently, and thus how to decide k-bounded delay efficiently (Theorem 4). Examples and complete proofs were added.

Related Work

While usual evaluation algorithms store the whole XML document in main memory [31, 43], on-the-fly evaluation algorithms on XML streams start processing input data before it is completely received [49, 7, 18, 44, 34, 58]. The ideal of on-the-fly algorithms with optimal memory management [6] lead to the idea of earliest query answering (EQA), whose objective is to keep only alive answer candidates in main memory [12, 5, 29]. The memory consumption of EQA algorithms thus directly depends on the concurrency of the query.

Many evaluation algorithms for XPath fragments on XML streams subscribe to the idea of EQA [5, 42, 32, 41, 46]. However, it turned out recently [9, 25], that EQA is infeasible, even for small fragments of XPath, for which satisfiability or universality cannot be decided in P-time. As a consequence, Benedikt *et al.* [9] propose streaming algorithms for a restricted fragment of XPath, where queries can never look forward, so that node selection can always be decided immediately (*i.e.*, the delay is zero).

Gauwin *et al.* [29] present an EQA algorithm for queries defined by DNWAs or equivalently, by deterministic visibly pushdown automata [3], streaming tree automata [27], or pushdown forest automata [39]. Berlea's EQA algorithm for queries defined by pushdown forest automata [12] assumes infinite alphabets, which removes the algorithmic hardness of EQA for non-deterministic automata, but limits its relevance at the same time.

Recognizable relations for ranked trees [56, 19] were applied for instance, in order to decide the first-order theory of ground tree transducers [21]. The closure operations of tree automata correspond to the first-order closure properties of recognizable relations. Recognizable relations for unranked trees were introduced in [10, 11]. They showed that all recognizable relations between unranked trees can be defined in the first-order logic of unranked trees with two extension operators, downwards and to the right. Note however, that their results are restricted to recognizable relations between trees over the same signature, while we need different signatures in the current article. Furthermore, the problem of bounded valuedness of recognizable relations is not studied there.

Outline.

We recall needed known boundedness results for word or tree automata and transducers in Section 2. In Section 3, we recall how to map queries on relational structures to canonical languages of annotated elements of the domain of the structure. This mapping is well-known from the relationship between logic and automata for words or trees. In Section 4, we discuss streaming

algorithms for query answering and recall the notions of concurrency and delay. In Section 5, we show how to decide bounded delay and concurrency for dFA-queries in words (aka flat unranked trees) by reduction to bounded ambiguity of nFAs. This section is not essential for our main results on more general dNWA queries on unranked trees, but illustrates the nature of the problems and provides some stronger results for this special case. In Section 6, we formulate our main results for dNWA-defined queries (Theorem 6). In Section 7, we discuss bounded valuedness problems for recognizable relations between unranked trees. In Section 8, we show how to define delay and concurrency as recognizable relations in P-time.

2. Bounded Ambiguity and Valuedness

We recall some results from the literature that we will make use of later on. They concern the bounded ambiguity for automata on words and trees and the bounded valuedness of tree transducers. The classes of trees considered there are all ranked, but these results will be used to prove properties of relations over unranked trees.

2.1. Bounded Ambiguity of Finite Automata

A *finite automaton* (nFA) over Σ is a tuple $A = (stat, init, rul, fin)$ where $init$, fin and $stat$ are finite sets with $init, fin \subseteq stat$, and $rul \subseteq stat^2 \times (\Sigma \cup \{\epsilon\})$ contains rules that we write as $q \xrightarrow{a} q'$ or $q \xrightarrow{\epsilon} q'$ where $q, q' \in stat$ and $a \in \Sigma$. Whenever necessary, we will index the components of A by A. Let the size of A count all states and rules, *i.e.* $|A| = |stat_A| + |rul_A|$. We also sometimes use the notation $A[init{=}I]$ (resp. $A[fin{=}I]$) for the automaton obtained from A by setting its initial (resp. final) states to I.

Let $eve(w) = \{0\} \cup dom(w)$ be the set of all positions of w and the start event 0. A *run* of A on a word w is a function $r : eve(w) \to stat_A$ so that $r(0) \in init_A$ and $r(\pi{-}1) \xrightarrow{\epsilon}{}^{*} \xrightarrow{a} \xrightarrow{\epsilon}{}^{*} r(\pi)$ is justified by rul for all $\pi \in dom(w)$ with $a = lab^w(\pi)$. A run is successful if $r(|w|) \in fin_A$. The *language* $L(A) \subseteq \Sigma^*$ is the set of all words that permit a successful run by A.

An nFA is called *productive*, if all its states are used in some successful run. This is the case if all states are reachable from some initial state, and if for all states, some final state can be reached.

An nFA A is *deterministic* or a dFA if it has at most one initial state, no epsilon rules, and for every pair (q, a) there exists at most one rule $q \xrightarrow{a} q' \in rul_A$. Note that for every word w there exists at most one run by a dFA A.

The *ambiguity* $amb_A(w)$ is the number of successful runs of A on w. This measures the degree of non-determinism of nFAs A. Clearly, $amb_A(w) \leq 1$ for all $w \in \Sigma^*$ if A is a dFA. Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_0$ the set of non-negative integers. Given $k \in \mathbb{N}$, we call the ambiguity of A *k-bounded* if $amb_A(w) \leq k$ for all $w \in \Sigma^*$. It is *bounded*, if it is bounded by some k.

Theorem 1 (Stearns and Hunt 85, Weber and Seidl 86)**.** *For nFAs bounded ambiguity and k-bounded ambiguity with fixed k can be decided in P-time.*

Decidability of k-bounded ambiguity for fixed k is shown by Theorem 4.1 of Stearns and Hunt [55]. They don't report the precise polynomial though. The result on bounded ambiguity was shown by Weber and Seidl [57]. Here we can

obtain a decent polynomial as follows. They define $p \xrightarrow{w} q$ *by* A if there exists a run of $A[init{=}\{p\}]$ on w that ends in q and show that an nFA A has unbounded ambiguity iff there exists a word $w \in \Sigma^+$ and distinct states $p \neq q$ such that $p \xrightarrow{w} p$, $p \xrightarrow{w} q$, and $q \xrightarrow{w} q$ by A. This can be tested in $O(|A|^3)$ as shown very recently by [1].

In the case of words, we will see that we can reduce (k)-bounded delay and concurrency of queries defined by dFAs to (k)-bounded ambiguity of nFAs.

2.2. Bounded Valuedness of Tree Transducers

In a first step, we generalize the results on bounded ambiguity to standard tree automata for binary trees [19], and in a second to bounded valuedness of tree transducers.

Let $\Sigma_r = \Sigma_0 \uplus \Sigma_2$ be a ranked alphabet with constants in Σ_0 and binary function symbols in Σ_2. The set of binary trees $\mathcal{T}^{bin}_{\Sigma_r}$ is the least set of unranked trees over Σ_r that contains all constants $c \in \Sigma_0$ and pairs $f(t_1, t_2)$ where $f \in \Sigma_2$ and $t_1, t_2 \in \mathcal{T}^{bin}_{\Sigma_r}$. The nodes of a ranked tree t are defined by: $nod(c) = \{\epsilon\}$ and $nod(f(t_1, t_2)) = \{\epsilon\} \cup \{1{\cdot}\pi \mid \pi \in nod(t_1)\} \cup \{2{\cdot}\pi \mid \pi \in nod(t_2)\}$.

A *tree automaton (n*TA*)* with signature Σ_r is a tuple of three finite sets $A = (stat, fin, rul)$ such that $fin \subseteq stat$ and $rul \subseteq \cup_{i\in\{0,2\}} stat^{i+1} \times \Sigma_i$. We denote rules in rul as $f(q_1, q_2) \to q$ and $c \to q$ where $q_1, q_2, q \in stat$, $f \in \Sigma_2$ and $c \in \Sigma_0$. A run of A on a tree $t \in \mathcal{T}^{bin}_{\Sigma_r}$ is a function $r : nod(t) \to stat$ mapping nodes to states, such that for all $\pi \in nod(t)$, if $lab^t(\pi) = f \in \Sigma_2$ then rule $f(r(\pi{\cdot}1), r(\pi{\cdot}2)) \to r(\pi)$ belongs to rul, and if $lab^t(\pi) = c \in \Sigma_0$ then rule $c \to r(\pi)$ belongs to rul. A run r is successful if $r(\epsilon) \in fin$. Automaton A recognizes the language of binary trees $L^{bin}(A) \subseteq \mathcal{T}^{bin}_{\Sigma_r}$ that permit a successful run by A.

An nTA is called *bottom-up deterministic* or a dTA if no two of its rules have the same left-hand side. We call it *top-down deterministic* or a d$^{\downarrow}$nTA if the set of final states fin contains at most one element, and if for every $f \in \Sigma_2$ and state $q \in stat$ there is at most one rule matching $f(q_1, q_2) \to q$ in rul.

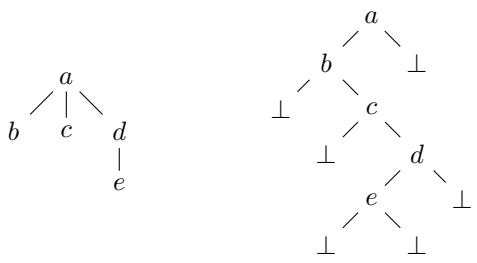
The *ambiguity* $amb_A(t)$ is the number of successful runs of A on t. Clearly, $amb_A(t) \leq 1$ for all $t \in \mathcal{T}^{bin}_{\Sigma_r}$ if A is bottom-up or top-down deterministic. Bounded and k-bounded ambiguity of nTAs are defined as for dFAs, and were proved to be decidable in P-time by Seidl in [51] and [50] respectively (for fixed k).

Theorem 2 (Seidl 89)**.** *Bounded ambiguity and k-bounded ambiguity for fixed k of tree automata on ranked trees (n*TA*s) in P-time.*

These results for trees generalize those for words in a straightforward manner. However we don't know, in the case of trees, how to reduce (k)-bounded delay and concurrency of queries defined by dTAs to (k)-bounded ambiguity of nTAs. Instead, we will need even stronger results on bounded valuedness of tree transducers.

Given a binary relation $R \subseteq S \times S'$ and an element $s \in S$, let $\#R(s)$ be the number of $s' \in S'$ such that $(s, s') \in R$. The *valuedness* of R is the maximal such number $val(R) = \max_{s\in S} \#R(s)$. We call R *k-bounded* if $val(R) \leq k$, and *bounded* if it is k-bounded for some $k \in \mathbb{N}_0$.

Let $\Sigma_r = \Sigma_2 \cup \Sigma_0$ and $\Delta_r = \Delta_2 \cup \Delta_0$ be binary signatures. A bottom-up tree transducer with input signature Σ_r and output signature Δ_r is a triple



(a) A unranked tree $t \in \mathcal{T}_\Sigma$ (b) The binary tree $\mathit{fcns}(t) \in \mathcal{T}^{bin}_{\Sigma_\perp}$

Figure 1: Top-down binary encoding

$T = (\mathit{stat}, \mathit{fin}, \mathit{rul})$ that consists of a finite set of states stat, a subset of final state $\mathit{fin} \subseteq \mathit{stat}$, and a finite set of rules rul of the following forms:

$$f(q_1(x_1), q_2(x_2)) \to q(g(x_1, x_2)) \qquad \text{or} \qquad a \to q(b)$$

where x_1, x_2 are two fixed variables, $q_1, q_2, q \in \mathit{stat}$, $a \in \Sigma_0$, $f \in \Sigma_2$, $b \in \Delta_0$ and $g \in \Delta_2$. Note that tree transducers usually admit more general rules [52], but these simpler rules suffice for our usage of tree transducers. The size of T is the sum of the numbers of its rules and states. Its semantics is a binary relation $[\![T]\!] \subseteq \mathcal{T}^{bin}_{\Sigma_r} \times \mathcal{T}^{bin}_{\Delta_r}$ defined in the obvious manner.

Theorem 3 (Seidl 92)**.** *Bounded valuedness of relations $[\![T]\!]$ between ranked trees defined by bottom-up tree transducers T can be decided in P-time in the size of T.*

This result was proved by Seidl in [52]. Note that it is not known for fixed k whether k-valuedness can be decided in P-time. The best existing algorithm is in coNP-time (Theorem 2.2 of [53]).

Note also that every dTA A can be mapped to some bottom-up tree transducer T in linear time, such that $[\![T]\!]$ relates trees $t \in L^{bin}(A)$ to successful runs by A on t. It then holds that T is k-valued if and only if A is k-unambiguous. This shows that deciding k-ambiguity of nTAs can be reduced to k-valuedness of bottom-up tree transducers, but not necessarily vice versa.

2.3. Translation to Unranked Trees

Since dealing with XML we are mostly interested in unranked trees. We will use the top-down encoding that is also called first-child next-sibling encoding [35] or Rabin's encoding [45] in order to lift boundedness results from ranked to unranked trees. In the appendix, we will also use a bottom-up encoding called Currying [15], in order to illustrate the generality of our results.

We define the set $\mathcal{T}_\Sigma$ of *unranked trees over* Σ to be the least set that contains all pairs $a(t_1, \ldots, t_m)$ consisting of a letter $a \in \Sigma$ and a tuple $(t_1, \ldots, t_m)$ of unranked trees in $\mathcal{T}_\Sigma$ where $m \geq 0$. Clearly, $\mathcal{T}^{bin}_{\Sigma_r} \subseteq \mathcal{T}_{\Sigma_r}$ for every binary signature Σ_r.

Given an unranked alphabet Σ, let $\Sigma_\perp = \Sigma \uplus \{\perp\}$ be the ranked alphabet where all symbols of Σ are binary and $\perp$ is the unique constant. The encoding is defined by a function $\mathit{fcns} : \mathcal{T}_\Sigma \to \mathcal{T}^{bin}_{\Sigma_\perp}$ is illustrated in Figure 1(b). nTA A over $\Sigma_\perp$ defines a language of unranked trees modulo the fcns encoding $L(A) = \{t \in \mathcal{T}_\Sigma \mid \mathit{fcns}(t) \in L^{bin}(A)\}$.

3. Queries and Canonical Languages

We abstract databases into relational structures in order to reason about words and trees in a common framework. In particular, we recall the notion of canonical languages for n-ary queries, first-order queries, and n-ary queries in words and trees.

3.1. Queries for Relational Structures

We define queries for relational structures by canonical languages of annotated relational structures. In the case of words and trees, this will enable definitions of queries by deterministic finite automata.

A *relational signature* Δ consists of a finite set of relation symbols $r \in \Delta$ each with a fixed arity $ar(r) \in \mathbb{N}_0$. A relational *structure* s over Δ consists of a non-empty finite set $dom(s)$ called the domain of s and relations $r^s \subseteq dom(s)^{ar(r)}$ interpreting all symbols $r \in \Delta$. We write $\mathcal{S}_\Delta$ for the set of structures over Δ.

Definition 1. *Let Δ be a relational signature and $n \in \mathbb{N}_0$. A* schema over Δ *is a subset $S \subseteq \mathcal{S}_\Delta$. An* n-ary query with schema S *is a function Q with domain $dom(Q) = S$, which maps all structures $s \in S$ to a set of tuples of elements $Q(s) \subseteq dom(s)^n$. A* Boolean query *is a query of arity* 0.

Below, we will define queries in words, where the schema $dom(Q)$ is a class of relational structures of words in Σ^*, and queries in unranked trees where the schema is a class of relational structures of unranked trees $\mathcal{T}_\Sigma$. Further restrictions on these domains can be defined by automata or XML schemas.

3.2. Canonical Languages

Boolean queries with domain $\mathcal{S}_\Delta$ can be identified with schemas $L_Q = \{s \mid () \in Q(s)\}$. But how can we define languages of structures for more general n-ary queries?

In order to do so, we fix an ordered set of distinct variables $\mathcal{V}_n = \{x_1, \ldots, x_n\}$ and define extended relation signatures $\Delta_n = \Delta \cup \mathcal{V}_n$ such that every variable becomes a monadic relation symbol. For every structure $s \in \mathcal{S}_\Delta$ and tuple $\tau = (\pi_1, \ldots, \pi_n) \in dom(s)^n$ we define an *annotated structure* $s * \tau \in \mathcal{S}_{\Delta_n}$ as follows:

$$\begin{array}{ll} dom(s * \tau) = dom(s) & \\ r^{s*\tau} = r^s & \text{for all } r \in \Delta \\ x_i^{s*\tau} = \{\pi_i\} & \text{for all } 1 \leq i \leq n \end{array}$$

We call a structure $\tilde{s} \in \mathcal{S}_{\Delta_n}$ *canonical* if $x^{\tilde{s}}$ is a singleton for all $x \in \mathcal{V}_n$. Clearly, all annotated structures $s*\tau$ are canonical. Conversely, every canonical structure $\tilde{s}$ is equal to some annotated structure $s * \tau$. We therefore define the canonical language of n-ary queries as follows:

Definition 2. *The canonical language L_Q of an n-ary query Q is the following set of annotated structures:*

$$L_Q = \{s * \tau \mid \tau \in Q(s)\}$$

Note that the canonical language of a Boolean query indeed coincides with the schema $L_Q = \{s \mid () \in Q(s)\}$. Note however, that the domain of a query is only partially specified by the canonical language. What is missing in the canonical language is the difference between structures s verifying $s \notin dom(Q)$ and structures s such that $Q(s) = \emptyset$. In order to fix this problem, we identify a query Q by the pair $(L_Q, dom(Q))$ of its canonical language and its domain.

3.3. First-Order Queries

We define logical operations for n-ary queries Q, Q' with the same schema S: conjunction $Q \wedge Q'$, negation $\neg Q$, existential quantification $\exists x_i.Q$ for all $1 \leq i \leq n$, and cylindrification $c_\theta Q$ for functions $\theta : \{1,\ldots,m\} \to \{1,\ldots,m\}$ with $\{1,\ldots,n\} \subseteq \theta(\{1,\ldots,m\})$. All these queries have the same domain, say S, and satisfy for all structures $s \in S$:

conjunction	$Q \wedge Q'(s) = Q(s) \cap Q'(s)$
negation	$\neg Q(s) = dom(s)^n - Q(s)$
quantification	$\exists x_i.Q(s) = \{(\pi_1,\ldots,\pi_{i-1},\pi_{i+1},\ldots,\pi_n) \mid \exists \pi_i.\ (\pi_1,\ldots,\pi_n) \in Q(s)\}$
cylindrification	$c_\theta Q(s) = \{(\pi_{\theta(1)},\ldots,\pi_{\theta(m)}) \in dom(s)^m \mid (\pi_1,\ldots,\pi_n) \in Q(s)\}$

Note that $\exists x_i.Q$ is a query of arity $n-1$, and $c_\theta Q$ of arity $m \geq n$, while all others have arity n. While projection deletes component in all tuples, cylindrification permits extension by new components, plus copying and permutation, but no deletion.

We next define formulas of first-order logics over a relational signature Δ as usual, where y ranges of an infinite vocabulary of variables $\mathcal{V}$, $r \in \Delta$, and $m = ar(r)$:

$$\phi ::= r(y_1,\ldots,y_m) \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists y.\phi$$

Every FO formula ϕ with at most m free variables $\tilde{y} = (y_1,\ldots,y_m) \in \mathcal{V}^m$ defines an m-ary query $Q_{\phi(\tilde{y})}$ whose domain contains all Δ-structures.

$$Q_{\phi_1\wedge\phi_2(\tilde{y})} = Q_{\phi_1(\tilde{y})} \wedge Q_{\phi_2(\tilde{y})} \qquad Q_{\neg\phi(\tilde{y})} = \neg Q_{\phi(\tilde{y})}$$
$$Q_{(\exists z.\phi)(\tilde{y})} = \exists z.Q_{\phi(\tilde{y},z)} \qquad Q_{r(y_1,\ldots,y_n)(y_{\theta(1)},\ldots,y_{\theta(m)})} = c_\theta r$$

Here, we identify relation symbol r with the query of arity $ar(r)$ that satisfies $r(s) = r^s$ for all structures $s \in \mathcal{S}$.

3.4. Queries on Words

Non-empty words over a finite alphabet Σ can be identified with relational structures. We write $w{\cdot}w' \in \Sigma^*$ for the concatenation of two words $w, w' \in \Sigma^*$, and $\epsilon \in \Sigma^*$ for the empty word. The domain of the structure of a word $w = a_1{\cdot}\ldots{\cdot}a_m$ is the set of its positions:

$$dom(w) = pos(w) = \{1,\ldots,m\}$$

Note that the domain of all non-empty words is non-empty. Indeed, we define structures only for non-empty words in Σ^+ in order to avoid anomalies later on.

The structure of a word $w \in \Sigma^+$ has signature $\Delta = \{lab_a \mid a \in \Sigma\} \cup \{\leq\}$ with the following interpretations where $w = a_1{\cdot}\ldots{\cdot}a_m$:

$$lab_a^w = \{i \mid a_i = a,\ 1 \leq i \leq m\}$$
$$\leq^w = \{(i,j) \mid 1 \leq i \leq j \leq m\}$$

An n-ary query Q in words has some schema $dom(Q) \subseteq \Sigma^+$ and selects n-tuples of positions in words in $dom(Q)$. Suppose that we fix $dom(Q) = \Sigma^+$. We can then define a monadic query by the following FO formula with a single free variable x_1:

$$\phi(x_1) =_{\text{df}} \exists x_2(x_1 \leq x_2 \wedge lab_a(x_2))$$

For every word w in the schema, the query $Q_{\phi(x_1)}$ defined by this formula selects all positions before some a-labeled positions.

Given a word $w = a_1 \cdot \ldots \cdot a_m \in \Sigma^*$ and a tuple $\tau = (\pi_1, \ldots, \pi_n) \in dom(w)^n$, we can identify the annotated structure $w * \tau$ with the following annotated word over $\Sigma \times 2^{\mathcal{V}_n}$:

$$(a_1, \{x_i \mid \pi_i = 1\}) \cdot \ldots \cdot (a_m, \{x_i \mid \pi_i = m\})$$

For instance we identify $(a \cdot a \cdot b) * (2, 1)$ with the word $(a, \{x_2\}) \cdot (a, \{x_1\}) \cdot (b, \emptyset)$. The canonical language of an n-ary query Q in words over Δ thus can be identified with a language L_Q of annotated words with alphabet $\Sigma \times 2^{\mathcal{V}_n}$.

3.5. Queries on Unranked Trees

We can identify unranked trees in $\mathcal{T}_\Sigma$ with structures over the relational signature $\Delta = \{lab_a \mid a \in \Sigma\} \cup \{ch^*, ns^*\}$, where all labeling relations are monadic and all others binary. The domain of the structure of $t = b(t_1, \ldots, t_m) \in \mathcal{T}_\Sigma$ is the set of its nodes:

$$dom(t) = nod(t) = \{\epsilon\} \cup \{i \cdot \pi \mid \pi \in nod(t_i)\}$$

The relations of the structure of t are defined as follows where $a \in \Sigma$:

$$\begin{aligned} lab_a^t &= \{\epsilon \mid a = b\} \cup \{i \cdot \pi \mid \pi \in lab_a^{t_i},\ 1 \leq i \leq m\} \\ (ch^*)^t &= \{(\pi, \pi \cdot \pi') \mid \pi \cdot \pi' \in nod(t)\} \\ (ns^*)^t &= \{(\pi \cdot i, \pi \cdot j) \mid 1 \leq i \leq j,\ \pi \cdot j \in nod(t)\} \cup \{(\epsilon, \epsilon)\} \end{aligned}$$

The word $w = a \cdot b \cdot a \cdot c \cdot a$, for instance, can be encoded by the tree $t = d(a, b, a, c, a)$, where $d \in \Sigma$ is an arbitrary symbol. Note that $nod(t) = \{\epsilon\} \cup dom(w)$.

Queries Q in unranked trees of $\mathcal{T}_\Sigma$ are queries with some domain $dom(Q) \subseteq \mathcal{T}_\Sigma$. They select tuples of nodes $Q(t) \subseteq nod(t)^n$ for all trees $t \in dom(Q)$. For instance, if we fix the schema to $\mathcal{T}_\Sigma$ then we can define a query for all trees that selects all nodes with a-labeled descendants by the following FO formula with one free variable x_1:

$$\phi(x_1) =_{\text{df}} \exists x_2\ (ch^*(x_1, x_2) \wedge lab_a(x_2))$$

In analogy to the case of words, the canonical language of an n-ary query Q in unranked trees over Σ can be identified with a language of unranked trees over the alphabet $\Sigma \times 2^{\mathcal{V}_n}$ where $\mathcal{V}_n = \{x_1, \ldots, x_n\}$.

4. Streaming Algorithms for Query Answering

We discuss fundamental concepts of streaming algorithms for query answering. In particular, we introduce the notion of bounded delay and bounded concurrency and discuss their relevance for streamability.

4.1. Linearizations of Structures

A streaming algorithm that answers a query Q in some class of structures $\mathcal{S}$ reads a linearization of a structure $s \in \mathcal{S}$ from the input stream, and computes a collection of answers $Q(s)$ incrementally.

Since words are linear structures, they can be put onto a stream in the naive manner. A streaming algorithm for words in Σ^+ can be understood as a one-way automaton, that reads the letters of a word $w \in \Sigma^+$ from the left to the

right. It assigns states to all events $eve(w) = \{0\} \cup dom(w)$, *i.e.*, to all positions of w and to the start event 0. The set of events is totally ordered with least element 0. We write $dom_\eta(w) = \{1, \ldots, \eta\}$ for the set of positions of w visited before the event η.

Unranked trees need linearization in order to be put onto a stream. For every set *Set*, we define a set of tagged opening and closing parenthesis:

$$\widehat{Set} = \{\mathtt{op}, \mathtt{cl}\} \times Set$$

An opening parenthesis $(\mathtt{op}, a)$ corresponds to the XML tag `<a>` and a closing parenthesis $(\mathtt{cl}, a)$ to the XML tag `</a>`. For every tree $t \in \mathcal{T}_\Sigma$ we define the *nested word* $nw(t) \in \hat{\Sigma}$ by linearization as follows:

$$nw(a(t_1, \ldots, t_n)) = (\mathtt{op}, a) \cdot nw(t_1) \cdot \ldots \cdot nw(t_n) \cdot (\mathtt{cl}, a)$$

This word is well-nested in that every opening parenthesis is properly closed. For instance, if $t = a(b, c(d), f)$ then $nw(t) = (\mathtt{op}, a) \cdot (\mathtt{op}, b) \cdot (\mathtt{cl}, b) \cdot (\mathtt{op}, c) \cdot (\mathtt{op}, d) \cdot (\mathtt{cl}, d) \cdot (\mathtt{cl}, c) \cdot (\mathtt{op}, f) \cdot (\mathtt{cl}, f) \cdot (\mathtt{cl}, a)$. The events of the nested word $nw(t)$ can be identified with element of the following set:

$$eve(t) = \{\mathtt{start}\} \cup \widehat{nod(t)}$$

Let $\leq$ be the total order on $eve(t)$ corresponding to the total order of $eve(nw(t))$ and $pred(e) \in eve(t)$ be the immediate predecessor of an event $\eta \in \widehat{nod(t)}$. For instance, $pred(\mathtt{op}, 2{\cdot}1) = (\mathtt{cl}, 1)$ for the tree $t = a(b, c(d), f)$. We write $dom_\eta(t) = \{\pi \in nod(t) \mid (\mathtt{op}, \pi) \leq \eta\}$ for the set of all nodes visited until event η.

4.2. Earliest Selection and Bounded Delay

The delay of a query is the maximal life time of some query answer. In order to define the delay formally, we must say what it means that an event is sufficient for the selection of an answer candidate, or equivalently, we must define the earliest event that is sufficient for its selection.

We consider the cases of words and trees in simultaneously, where either $\mathcal{S} = \Sigma^*$ is the set of all words or $\mathcal{S} = \mathcal{T}_\Sigma$ the set of all unranked tree over Σ. Let Q be an n-ary query in $\mathcal{S}$, $s \in \mathcal{S}$ a structure, and $\eta \in eve(s)$ an event of s. A *complete candidate* until event η is a tuple $\tau \in dom_\eta(s)^n$. Given two structures $s_1, s_2 \in \mathcal{S}$ and an event $\eta \in eve(s_1) \cup eve(s_2)$, we say that the prefixes of the linearizations of s_1 and s_2 until η coincide, if:

$$eq_\eta(s_1, s_2) \Leftrightarrow \begin{cases} dom_\eta(s_1) = dom_\eta(s_2) \wedge \\ \forall a \in \Sigma.\ \forall \pi \in dom_\eta(s_1).\ (lab_a^{s_1}(\pi) \Leftrightarrow lab_a^{s_2}(\pi)) \end{cases}$$

Definition 3. *We call an event η sufficient for selection of a complete candidate τ until η in structure s by query Q, and write $(\tau, \eta) \in sel_Q(s)$, if τ will be selected by Q in all possible continuations of the stream beyond η:*

$$(\tau, \eta) \in sel_Q(s) \;\Leftrightarrow\; \tau \in dom_\eta(s)^n \wedge \forall s' \in dom(Q).\ eq_\eta(s, s') \Rightarrow \tau \in Q(s')$$

Note that allowed continuations are only those that extend the current prefix of the linearization of the structure to a member of $dom(Q)$. Note also that the

start event may be sufficient to select the empty tuple () in Boolean queries where $n = 0$, while it is never sufficient for selection if $n \geq 1$ since otherwise $\tau \notin dom_\eta(s)^n$. Let

$$latest((\pi_1, \ldots, \pi_n)) = \min\{\eta \in eve(s) \mid \pi_1, \ldots, \pi_n \in dom_\eta(s)\}$$

be the minimal event, where all elements of the tuple have been visited. The delay of an n-ary query Q for a tuple $\tau \in dom(s)$ is the number of events η following $latest(\tau)$ such that η is insufficient for selection, *i.e.* $(\tau, \eta) \notin sel_Q(s)$.

$$delay_Q(s, \tau) = |\{\eta \in eve(s) \mid \ latest(\tau) \leq \eta,\ (\tau, \eta) \notin sel_Q(s)\}|$$

A query Q has *k-bounded delay* if $delay_Q(s, \tau) \leq k$ for all $s \in dom(Q)$ and $\tau \in Q(s)$. It has *bounded delay* if it has k-bounded delay for some $k \geq 0$. Having bounded delay means that every EQA algorithm will output selected tuples a constant time after completion. This is a guarantee on the quality of service.

4.3. Earliest Rejection and Bounded Concurrency

The concurrency of a query is the maximal number of concurrently alive answer candidates at every time point [5]. In order to define the concurrency formally, we have to define aliveness of answer candidates. Intuitively, a candidate is alive at an event at which it can neither be safely rejected nor selected.

Recall that the concepts of earliest selection and rejection are closely related to the idea of EQA. An EQA algorithm for a query Q is a streaming algorithm that inputs a linearization of a structure on the stream, and decides selection and rejection of answer candidates at every time point (without knowing the rest of the stream). This way, it needs to keep in main memory only alive candidates, which are neither safe for selection nor rejection. As an example, consider the monadic query Q_2 that selects all positions in words $w \in \{a, b, c\}^*$ that are labeled by a and followed by $b{\cdot}b$. When applied to $w_2 = a{\cdot}a{\cdot}b{\cdot}b{\cdot}a{\cdot}b{\cdot}b{\cdot}c{\cdot}a{\cdot}b{\cdot}a{\cdot}b$, this query returns $Q_2(w_2) = \{(2), (5)\}$. A streaming algorithm can enumerate these answers by using a sliding window of length 3. Position 1 for instance can be rejected when having seen the labels of positions 1 and 2, while position 2 can be selected when having seen the labels of positions 2, 3, and 4.

In order to formalize the concept of earliest rejection for n-ary queries, we have to deal with *partial* answer candidates for a given structure s. We fix a constant $\bullet$ that represents unknown components, and define partial tuples τ of positions until $\eta \in eve(s)$ as members of $(dom_\eta(s) \uplus \{\bullet\})^n$. So far, we have only studied *complete* answer candidates, which do not contain any unknown component. We write $compl(\tau, s, \eta)$ for the set of complete candidates, in which all unknown components of τ have been instantiated with elements $\pi \in dom(s) - dom_\eta(s)$.

Definition 4. *We call a partial candidate τ* rejected *at event $\eta \in eve(s)$, if no completion of τ by nodes in the future of η can be selected by Q.*

$$(\tau, \eta) \in rej_Q(s) \Leftrightarrow \begin{cases} \tau \in (dom_\eta(s) \uplus \{\bullet\})^n \ \wedge \\ \forall s' \in dom(Q).\ eq_\eta(s, s') \Rightarrow \forall \tau' \in compl(\tau, s', \eta).\ \tau' \notin Q(s') \end{cases}$$

We call a partial candidate $\tau \in (dom_\eta(s) \cup \{\bullet\})^n$ *alive* at η if τ is neither rejected nor selected at η.

$$(\tau, \eta) \in alive_Q(s) \Leftrightarrow \begin{cases} \tau \in (dom_\eta(s) \cup \{\bullet\})^n \text{ and} \\ (\tau, \eta) \notin rej_Q(s) \text{ and } (\tau, \eta) \notin sel_Q(s) \end{cases}$$

The concurrency of a query Q on a structure $s \in dom(Q)$ at event $\eta \in eve(s)$ is the number of alive partial candidates τ until η, so that η is neither sufficient for selection or rejection of τ.

$$concur_Q(s, \eta) = |\{\tau \in (dom_\eta(s) \cup \{\bullet\})^n \mid (\tau, \eta) \in alive_Q(s)\}|$$

We choose to include the empty tuple $\bullet^n$ in the concurrency, because in our algorithm it is processed like any other partial tuple[4]. Hence Boolean queries have concurrency at most 1, and for monadic queries it is at most $|dom(s)| + 1$. Concurrency 0 means that the query is empty.

Lemma 5. *For all monadic queries Q, structures $s \in dom(Q)$, and events $\eta \in eve(s)$:*

$$concur_Q(s, \eta) \leq \sup_{s' \in dom(Q), \tau \in Q(s')} delay_Q(s', \tau) + 1$$

The lemma fails for queries of higher arities, where the delay between the tuple components may be unbounded even though the delay of selection of complete tuples is bounded. In this case, the set of alive partial tuples may grow without bound, even though the set of alive complete tuples is bounded. For instance consider the query Q with $Q(t) = nod(t)^2$ for all trees $t \in \mathcal{T}_\Sigma$. This query has delay 0, since every pair of nodes can be selected immediately, once its last component has been visited. Nevertheless, all partial tuples $(\pi, \bullet)$ with $\pi \in dom_\eta(t)$ are alive at all events η, so that the concurrency of this query is not bounded.

Proof. Let $s' \in \mathcal{S}$ and $k \in \mathbb{N}_0 \cup \{\infty\}$. In the case of words (where $\mathcal{S} = \Sigma^*$), we define $dom^k_\eta(s')$ by $\{\pi' \mid \eta - k \leq \pi' \leq \eta\}$, and in the case of trees (where $\mathcal{S} = \mathcal{T}_\Sigma$), we define $dom^k_\eta(s')$ as $\{\pi' \mid pred^k(\eta) \leq (\mathtt{op}, \pi') \leq \eta\}$.

Let Q be a monadic query. Let $d = sup_{s' \in dom(Q), \tau \in Q(s')} delay_Q(s', \tau)$ be the number in the lemma, and $s \in dom(Q)$ be a structure with event $\eta \in eve(s)$. We claim for all $\pi \in dom(s)$ that:

$$\pi \notin dom^d_\eta(s) \Rightarrow ((\pi), \eta) \notin alive_Q(s)$$

To see this, we first note that if $\pi \notin dom_\eta(s)$ then π is not alive at η. Now let us consider $\pi \in dom_\eta(s) - dom^d_\eta(s)$. We distinguish two cases.

1. In the first case, there exists a continuation $s' \in dom(Q)$ with $eq_\eta(s, s')$ such that $(\pi) \in Q(s')$. This continuation s' satisfies $delay_Q(s', (\pi)) \leq d$, so that $\pi \in dom_\eta(s) - dom^d_\eta(s)$ yields $((\pi), \eta) \in sel(s)$. This contradicts aliveness.

[4]Note that the notion of concurrency used in the introduction did not take the empty tuple into account. This just differs the concurrency from 1.

2. Otherwise, all continuations s' of s beyond η satisfy $(\pi) \notin Q(s')$, so that $((\pi), \eta) \in \mathit{rej}(s)$. This equally implies non-aliveness.

This proves the claim, which yields for all partial tuples τ:

$$(\tau, \eta) \in \mathit{alive}_Q(s) \Rightarrow \tau \in \mathit{dom}^d_\eta(s) \cup \{\bullet\}$$

Hence, $\mathit{concur}_Q(s, \eta) \leq d + 1$ by definition of concurrency. □

We say that the concurrency of a query Q is bounded if there exists $k \geq 0$ such that $\mathit{concur}_Q(s, \eta) \leq k$ for all structures $s \in \mathit{dom}(Q)$ and $\eta \in \mathit{eve}(s)$. Note that queries with unbounded concurrency cannot be processed in streaming manner with bounded memory.

Proposition 6. *A monadic query with k-bounded delay has $(k+1)$-bounded concurrency.*

Proof. This is an immediate consequence of Lemma 5. □

The converse does not hold. As a counter example, consider the monadic query which selects the first letter of all words whose last letter is a b. This query has concurrency bounded by 1, since the first letter is the only alive candidate before the end, but unbounded delay.

4.4. Schema Elimination

Domain restrictions of queries by schemas are relevant to EQA, since they constrain the possible continuations of a stream. As an example, reconsider the monadic query Q_2 that selects all positions in words w that are labeled by a and followed by $b{\cdot}b$, but now with domain restricted by schema $(a|b)^*{\cdot}c{\cdot}(a{\cdot}b)^*$. If the word w on the stream is assumed to satisfy the schema, then no position that follows a c symbol on the stream can be selected.

EQA for queries Q with schema restriction can be reduced to EQA for queries without. In order to do so, we define a query σ_Q with $\mathit{dom}(\sigma_Q) = \mathcal{S}$ such that $\mathit{sel}_Q = \mathit{sel}_{\sigma_Q}$. We set $\sigma_Q(s) = Q(s)$ if $s \in \mathit{dom}(Q)$ and $\mathit{dom}(s)^n$ otherwise. Similarly for rejection, we define a query ρ_Q with $\mathit{dom}(\rho_Q) = \mathcal{S}$ by $\rho_Q(s) = Q(s)$ if $s \in \mathit{dom}(Q)$ and $\emptyset$ otherwise.

Lemma 7. *$\mathit{sel}_Q = \mathit{sel}_{\sigma_Q}$ and $\mathit{rej}_Q = \mathit{rej}_{\rho_Q}$.*

Proof. Straightforward from the definitions. Let $\tau \in \mathit{dom}_\eta(s)^n$:

$$\begin{array}{lll}
(\tau, \eta) \in \mathit{sel}_{\sigma_Q} & \text{iff} & \tau \in \mathit{dom}_\eta(s)^n \ \wedge\ \forall s' \in \mathcal{S}.\ \mathit{eq}_\eta(s, s') \Rightarrow \tau \in \sigma_Q(s') \\
& \text{iff} & \tau \in \mathit{dom}_\eta(s)^n \ \wedge\ \forall s' \in \mathit{dom}(Q).\ \mathit{eq}_\eta(s, s') \Rightarrow \tau \in Q(s') \\
(\tau, \eta) \in \mathit{rej}_{\rho_Q} & \text{iff} & \left\{ \begin{array}{l} \tau \in (\mathit{dom}_\eta(s) \cup \{\bullet\})^n \ \wedge \\ \forall s' \in \mathcal{S}.\ \mathit{eq}_\eta(s, s') \Rightarrow \forall \tau' \in \mathit{compl}(\tau, s', \eta).\ \tau' \notin \rho_Q(s') \end{array} \right. \\
& \text{iff} & \left\{ \begin{array}{l} \tau \in (\mathit{dom}_\eta(s) \cup \{\bullet\})^n \ \wedge \\ \forall s' \in \mathit{dom}(Q).\ \mathit{eq}_\eta(s, s') \Rightarrow \forall \tau' \in \mathit{compl}(\tau, s', \eta).\ \tau' \notin Q(s') \end{array} \right.
\end{array}$$

□

However, given automata A and B with $L(A) = L(Q)$ and $L(B) = S$, we cannot build an automaton recognizing σ_Q or ρ_Q without a blowup in $O(2^n)$ in the general case, since we have to extend the alphabet of B from Σ to $\Sigma \times 2^{\mathcal{V}_n}$.

5. Bounded Delay and Concurrency for Word Automata

We consider the case, where queries in words are defined by two deterministic finite automata, that recognize the canonical language of the query and its schema respectively. We obtain P-time decision procedures for bounded delay and concurrency by reduction to bounded ambiguity of non-deterministic finite automata.

5.1. Defining n-ary Queries

We can define queries by two automata, one for the canonical language and another for the schema. We call an nFA canonical if and only if its language is.

Definition 8. *Let A be a canonical n*FA *with alphabet $\Sigma \times 2^{\mathcal{V}_n}$ and B an n*FA *with alphabet Σ, such that $w \in L(B)$ for all $w * \tau \in L(A)$. The query $Q_{(A,B)}$ defined by the pair (A, B) is the unique n-ary query with domain $L(B)$ and canonical language $L(A)$. If $L(B) = \Sigma^+$ then we write Q_A instead of $Q_{(A,B)}$.*

Automaton B is needed in order to distinguish those words on which the query is not defined from those where the query returns the empty set. Note that if $Q_{(A,B)}(w) \neq \emptyset$ then $w \in L(B)$.

Let the type of a word w with alphabet $\Sigma \times 2^{\mathcal{V}_n}$ be a function $type_w : \mathcal{V}_n \to \mathbb{N}_0$ that counts how many times a variable appears in labels, *i.e.*, for $x \in \mathcal{V}_n$:

$$type_w(x) = |\{\pi \in dom(w) \mid lab^w_{(a,V)}(\pi) \text{ with } x \in V\}|$$

We say that a word w has type $1^{\mathcal{V}_n}$ if $type_w(x) = 1$ for all $x \in \mathcal{V}_n$. All words over $\Sigma \times 2^{\mathcal{V}_n}$ of type $1^{\mathcal{V}_n}$ have the form $w * \tau$, and vice versa. We next show that all states of productive canonical nFAs have unique types. This was already noticed in Lemma 3 of [14]:

Lemma 9. *If A is a productive canonical n*FA *and $q \in stat_A$ then all words recognized by $A[fin = \{q\}]$ have the same type.*

Proof. Since A is productive, there exists a word $w \in L(A[init = \{q\}])$. Assume that there exist words $w_1, w_2 \in L(A[fin = \{q\}])$ with different types. Hence, the words $w_1 \cdot w$ and $w_2 \cdot w$ must have different types, since $type_{w_1 \cdot w} = type_{w_1} + type_w \neq type_{w_2} + type_w = type_{w_2 \cdot w}$. This is impossible, though, since $L(A)$ is canonical, so that $type_{w_2 \cdot w}(x) = type_{w_1 \cdot w}(x) = 1$ for all $x \in \mathcal{V}_n$ □

We can thus define the type of a state q of a productive canonical nFA in a unique manner, via the type of some word w that evaluates to this state. $type(q)$ will denote this type. Furthermore, as the automaton is canonical and productive, this type is determined by the set $\{x \in \mathcal{V}_n \mid type_w(x) = 1\}$. So we can identify the type of a state with a subset of $\mathcal{V}_n$.

Reconsider the query $Q_{\phi(x_1)}$ in words with alphabet $\{a, b\}$ from Section 3.4, which selects all positions labeled by a or eventually succeeded by an a. In Figure 2, we illustrate an automaton for the canonical language of this query graphically. Its states have the following types: $\emptyset$ for q_0 (no variables seen before entering in this state), and $\{x_1\}$ for q_1 and q_2 (x_1 seen before entering in these states).

Query answering for dFAs is the algorithmic problem that receives as input two dFAs A and B defining an n-ary query and a word $w \in L(B)$ defining a valid

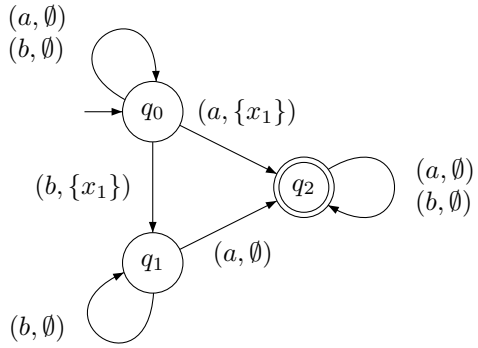


Figure 2: A dFA for the canonical language of $Q_{\phi(x_1)}$ where $\phi = \exists x_2.\, (x_1 \leq x_2 \wedge lab_a(x_2))$.

database, and returns as output $Q_{(A,B)}(w)$. The objective is to find all tuples τ of positions in w such that $w * \tau \in L(A)$. The naive algorithm enumerates all tuples $\tau \in dom(w)^n$ and runs A deterministically on $w * \tau$. This algorithm first resolves the choice of τ non-deterministically, before running the deterministic automaton A.

Determinism for canonical automata will turn out to be essential for P-time streaming algorithms and decision complexity (e.g. the safety property below). It should be noticed that canonical nFAs can always be determinized without changing the query they define. This would fail when defining queries by selection automata, *i.e.* nFAs over Σ with a set of selection states as considered in [24, 40].

5.2. Computing Delays of Queries

We show how to decide whether a query has bounded delay and how to compute this delay in polynomial time. We consider the case with schemas, since subsequent schema elimination as proposed in Section 4.4 would yield polynomial bounds only for queries with fixed arity n. Moreover, these bounds for fixed n would be larger than those obtained by the following construction.

For every language $L \subseteq \Sigma^+$ we define a language of annotated words $L \otimes \emptyset$ with alphabet $\Sigma \times 2^{\mathcal{V}_n}$ such that all letters of words in L are annotated by $\emptyset$, *i.e.*, $L \otimes \emptyset = \{(a_1,\emptyset)\cdot \ldots \cdot (a_k,\emptyset) \mid a_1 \cdot \ldots \cdot a_k \in L\}$

Definition 10. *If dFAs A and B define a query then we call a state $(p,q) \in stat_A \times stat_B$* safe for selection *by $Q_{(A,B)}$ if $L(B[init{=}\{q\}]) \otimes \emptyset \subseteq L(A[init{=}\{p\}])$.*

Figure 3 illustrates an automaton for the query that selects all a-nodes that are succeeded by $b{\cdot}b$. In this example, we assume the universal schema B with a single state, so that A is isomorphic to $P(A,B)$. The types and safety properties of all states are indicated in the figure.

We next show that safe states capture sufficiency for selection. In order to do so, we construct a dFA $P(A,B)$ which runs A and B in parallel. Its alphabet is $\Sigma \times 2^{\mathcal{V}_n}$ as for A, while B has alphabet Σ.

$$\begin{aligned} stat_{P(A,B)} &= stat_A \times stat_B \\ init_{P(A,B)} &= init_A \times init_B \\ fin_{P(A,B)} &= fin_A \times fin_B \end{aligned} \qquad \frac{p \overset{(a,V)}{\to} p' \in rul_A \qquad q \overset{a}{\to} q' \in rul_B}{(p,q) \overset{(a,V)}{\to} (p',q') \in rul_{P(A,B)}}$$

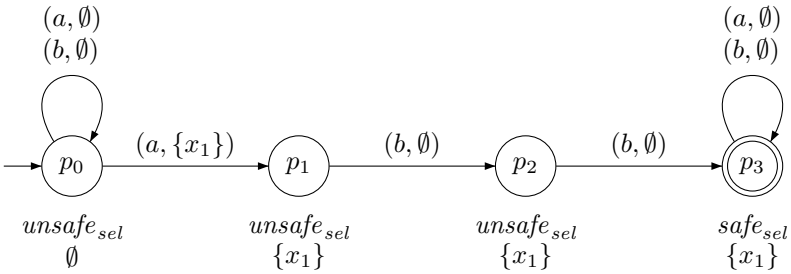


Figure 3: Automaton A for the query selecting a-nodes followed by $b{\cdot}b$. There are two reachable unsafe states of type $\{x_1\} = \mathcal{V}_1$, p_1 and p_2. The restriction of A to these two states is acyclic, so the selection delay of Q_A is bounded. It is bounded by 2, since the longest path in this part of the automaton has 2 nodes.

Building $P(A, B)$ requires time in $O((|\Sigma|+n){\cdot}|A|{\cdot}|B|)$, if we suppose for instance that variables in V are stored in a vector of n bits.

Lemma 11. *Let A and B be productive dFAs that define a query, and r a run of $P(A, B)$ on $w * \tau$ and $\eta \in eve(w)$. Then state $r(\eta)$ is safe for selection by $Q_{(A,B)}$ if and only if $(\tau, \eta) \in sel_{Q_{(A,B)}}(w)$.*

Proof. Sufficiency for selection $(\tau, \eta) \in sel_{Q_{(A,B)}}(w)$ is equivalent to $\tau \in dom_\eta(w)^n$ and $\forall w' \in L(B) : \ eq_\eta(w, w') \Rightarrow w' * \tau \in L(A)$. Let $w = w_0{\cdot}w_1$ such that $|w_0| = \eta$. Since $\tau \in dom_\eta(w)^n$, we have $w * \tau = (w_0 * \tau){\cdot}(w_1 \otimes \emptyset)$. Furthermore, $eq_\eta(w, w')$ is equivalent to $\exists w'_1.\ w' = w_0{\cdot}w'_1$. Now $r(\eta)$ is the state that the unique run of $P(A, B)$ on $w_0 * \tau$ reaches (determinism). For $(p, q) = r(\eta)$ we have:

$$
\begin{array}{ll}
 & \forall w' \in L(B) : \ eq_\eta(w, w') \Rightarrow w' * \tau \in L(A) \\
\Leftrightarrow & \forall w'_1.\ w_0{\cdot}w'_1 \in L(B) \Rightarrow (w_0 * \tau){\cdot}(w'_1 \otimes \emptyset) \in L(A) \\
\Leftrightarrow & \forall w'_1.\ w'_1 \in L(B[init = \{q\}]) \Rightarrow w'_1 \otimes \emptyset \in L(A[init = \{p\}]) \quad \text{(determinism)} \\
\Leftrightarrow & L(B[init = \{q\}]) \otimes \emptyset \subseteq L(A[init = \{p\}]) \\
\Leftrightarrow & r(\eta) \text{ safe for selection by } Q_{(A,B)}
\end{array}
$$

Conversely, assume that $r(\eta) = (p, q)$ is safe for selection by $Q_{(A,B)}$. Since we assumed A and B to be productive, this implies that $type(p) = \mathcal{V}_n$, so that $\tau \in dom_\eta(w)^n$. We can thus decompose $w = w_0{\cdot}w_1$ such that $|w_0| = \eta$ as above, and apply the above equivalence, in order to conclude from safety for selection, that $\forall w' \in L(B) : \ eq_\eta(w, w') \Rightarrow w' * \tau \in L(A)$, and thus sufficiency for selection. □

The parallel automaton $P(A, B)$ is canonical, since $L(A) = L(P(A, B))$, but may contain non-productive states, even if A and B are productive. For instance, consider productive automata A and B that define the query Q with $dom(Q) = \{a, a{\cdot}a\}$, $Q(a) = \{1\}$ and $Q(a{\cdot}a) = \emptyset$. We will be interested only in the productive part of the canonical automaton $P(A, B)$, for which unique types exist.

Lemma 12. *If A and B are productive, then all safe states of $Q_{(A,B)}$ that are reachable in $P(A, B)$ are productive and have type $\mathcal{V}_n$.*

Proof. To see this, suppose that (p, q) is safe and reachable. Since B is productive, there exists a word $w \in L(B[init{=}\{q\}])$. Safety proves that $w \otimes \emptyset \in$

$L(A[init = \{p\}])$. Thus, $w \in P(A,B)[init = \{(p,q)\}]$, so that (p,q) is productive. Since A is canonical, $P(A,B)$ is canonical, so that $type(p) \uplus type(w \otimes \emptyset) = \mathcal{V}_n$. □

Proposition 13. *Let $Q_{(A,B)}$ be defined by productive dFAs A and B, and let P^u be the restriction of nFA $P(A,B)$ to productive unsafe states of type $\mathcal{V}_n$.*

1. *The delay of $Q_{(A,B)}$ is bounded if and only if the digraph of nFA P^u is acyclic.*
2. *In this case, the delay of $Q_{(A,B)}$ is equal to the length of the longest path in P^u.*

Proof. Let $P = P(A,B)$ and P^u the restriction of P to productive unsafe states of type $\mathcal{V}_n$. Let q be a state of P^u for which a cycle exists. Since all states of P^u are productive in P, there exists a word $v_1 \in L(P[fin = \{q\}])$. Since P^u has a cycle, there exists a nonempty word $v_2 \in L(P[init = \{q\}, fin = \{q\}])$. Again, since P is productive, there exists a word $v_3 \in L(P[init = \{q\}])$. It follows for all $m \geq 0$, that $v = v_1 \cdot (v_2)^m \cdot v_3 \in L(P)$. Since $L(P) = L(A)$, word v has the form $w * \tau$ for some word $w \in \Sigma^*$ and $\tau \in dom(w)^n$. By Lemma 11, none of the events in $|v_2|^m$ is sufficient for the selection of τ in w since the run of P on v maps all of them to unsafe states. This shows that the selection delay of τ in v is at least m and thus unbounded.

For the converse, we suppose that P^u is acyclic and show that the delay of $Q_{(A,B)}$ is bounded by the length of the longest path in $stat_{P^u}$. Let w and τ be such that $w * \tau \in L(A)$ and r be the successful run of A that accepts this word. Let η be an arbitrary event that contributes to the delay of τ, *i.e.*, an event with $\tau \in dom_\eta(w)$ and $(\tau,\eta) \notin sel_{Q_{(A,B)}}(w)$. The first condition yields that $type(r(\eta)) = 1^{\mathcal{V}_n}$ and the second condition that $r(\eta)$ is unsafe for selection by Lemma 11. Thus, $r(\eta) \in stat_{P^u}$. Since P^u is acyclic, it follows that states $r(\eta)$ are distinct for distinct events η that contribute to the delay. Furthermore, all these states belong to the same path of P^u, such that $delay_{Q_{(A,B)}}(w,\tau)$ is bounded by the length of the longest path in P^u.

If P^u is acyclic, let r a longest path in P^u and let w a word such that $w * \emptyset$ labels r. Since all states of P are reachable and productive, there exists $w_1 * \tau$ which reaches in P the first state of r; similarly, there exists a word w_2 such that $w_2 * \emptyset$ labels a path from the last state of r to a final state of P. Then $delay_Q(w_1 \cdot w \cdot w_2, \tau)$ is the length (here, the number of states) of r.

□

Hence we get a first algorithm for determining the delay. The remaining question is the complexity of this computation. To obtain an efficient algorithm, we rely on the following characterization of unsafe states.

Lemma 14. *Let A, B be productive dFAs that define a query. A reachable state (p_0,q_0) of $P(A,B)$ is unsafe for selection by $Q_{(A,B)}$ if and only a state (p,q) can be reached from (p_0,q_0) such that:*

(U1) either $p \notin fin_A$ and $q \in fin_B$,

(U2) or there exists a transition $q \xrightarrow{a} q' \in rul_B$ but no transition $p \xrightarrow{(a,\emptyset)} p' \in rul_A$ for all $p' \in stat_A$.

Proof. Let $P = P(A,B)$. We start with a claim about propagation of unsafety.

Claim 15. *Reachable states of P that can reach unsafe states are unsafe.*

To see this, let (p_1, q_1) be a reachable state and (p_2, q_2) be an unsafe state that is reached from (p_1, q_1) by some word v_1, *i.e.* $v_1 \in P[init = \{(p_1, q_1)\}, fin = \{(p_2, q_2)\}]$. Since (p_2, q_2) is unsafe, there exists a word $w \in L(B[init = \{q_2\}])$ such that $w \otimes \emptyset \notin L(A[init = \{p_2\}])$. We distinguish two cases.

1. If v_1 matches $w_1 \otimes \emptyset$ then $w_1 \cdot w \in L(B[init = \{q_1\}])$ and $(w_1 \cdot w) \otimes \emptyset \notin L(A[init = \{p_2\}])$, so that (p_1, q_1) is unsafe.
2. If v_1 does not match $w_1 \otimes \emptyset$ then $type(p_1) \neq \mathcal{V}_n$ so that (p_1, q_1) is unsafe by Lemma 12, since (p_1, q_1) is reachable in P and since A and B are productive.

Based on this claim, we can now show both directions of the lemma.

"$\Leftarrow$" By Claim 15 it is sufficient to show that all states (p, q) satisfying (U1) or (U2) are unsafe. In case of (U1) where $p \notin fin_A$ and $q \in fin_B$, the empty word contradicts the safety of (p, q), since $\epsilon \in L(B[init = \{q\}])$ but $\epsilon \otimes \emptyset \notin L(A[init = \{p\}])$. In case of (U2), there exists some transition $q \xrightarrow{a} q' \in rul_B$ but no transition $p \xrightarrow{(a,\emptyset)} p' \in rul_A$ for all $p' \in stat_A$. Since B is productive, there exists a word $w \in L(B[init = \{q_2\}])$. The word $a \cdot w$ now contradicts safety of (p, q) since $a \cdot w \in L(B[init = \{p\}])$ but $(a \cdot w) \otimes \emptyset \notin L(A[init = \{q\}])$.

"$\Rightarrow$" We show that all unsafe states (p_0, q_0) can reach some state (p, q) that satisfies (U1) or (U2). If (p_0, q_0) is unsafe then there exists a word $w \in \Sigma^*$ such that $w \in L(B[init = \{q_0\}])$ and $w \otimes \emptyset \notin L(A[init = \{p_0\}])$. Let w_0 be the longest prefix of w such that there exists a run of $P[init = \{(p_0, q_0)\}]$ on w_0. Let (p, q) be the state reached by this run after reading w_0, and let w_1 be the suffix of w such that $w = w_0 \cdot w_1$. State (p, q) is thus reached from (p_0, q_0). It remains to show that (p, q) satisfies (U1) or (U2).

1. If $w_1 = \epsilon$ then $p \in fin_B$ and $q \notin fin_A$, so that (p, q) satisfies (U1).
2. If w_1 matches $a \cdot w_2$ then there cannot exist any transition $p \xrightarrow{(a,\emptyset)} p'$ since w_0 was chosen of maximal length. There exists a transition $q \xrightarrow{a} q'$ for some q' though. Hence, (p, q) satisfies (U2).

□

Lemma 16. *The set of reachable safe states for selection for an n-ary query $Q_{(A,B)}$ can be computed in time $O((|\Sigma| + n) \cdot |A| \cdot |B|)$ from dFAs A and B.*

Proof. Instead of the set of reachable safe states, we compute the set of reachable unsafe states. A Datalog program testing the reachability of states satisfying (U1) or (U2), which characterizes unsafety for reachable states by Lemma 14, can be defined as follows:

$$\frac{p' \notin fin_A \qquad q' \in fin_B}{unsafe_{sel}(p, q).} \qquad \frac{\forall p'.p \xrightarrow{(a,\emptyset)} p' \notin rul_A \qquad q \xrightarrow{a} q' \in rul_B}{unsafe_{sel}(p, q).}$$

$$\frac{(p, q) \xrightarrow{(a,V)} (p', q') \in rul_{P(A,B)}}{unsafe_{sel}(p, q) \text{ :- } unsafe_{sel}(p', q').}$$

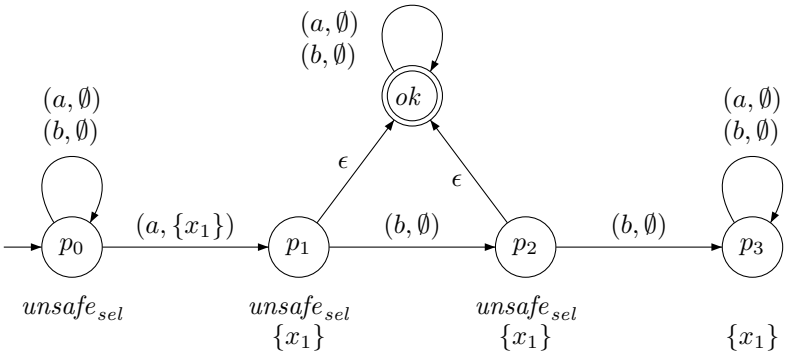


Figure 4: nFA $D(A,B)$ for the dFA A in Figure 3 with trivial universal B. The ambiguity of $D(A,B)$ is 2 (on word $(a,\{x_1\})\cdot(b,\emptyset)$ for instance), such as the delay of $Q_{(A,B)}$.

This program P can be computed in time $O((|\Sigma|+n)\cdot|A|\cdot|B|)$, while being of size $O(|A|\cdot|B|)$. It is a ground Datalog program, so its least fixed point $lfp(P)$ can be computed in time $O(|A|\cdot|B|)$ (see Proposition 36 in the appendix). □

Theorem 4. *The delay of queries $Q_{(A,B)}$ in words with alphabet Σ and arity $n\in\mathbb{N}_0$ defined by dFAs A and B can be computed in time $O((|\Sigma|+n)\cdot|A|\cdot|B|)$.*

In particular, we can decide in the same time, whether a query $Q_{(A,B)}$ has bounded delay or k-bounded delay, even if k belongs to the input.

Proof. We first render B productive and construct the dFA $P(A,B)$. Second, we compute all reachable safe states by Lemma 16 and derive the sub-automaton P^u, that restricts $P(A,B)$ to productive unsafe states of type $\mathcal{V}_n$. By Proposition 13, the delay of $Q_{(A,B)}$ is ∞ if and only if P^u contains a cycle. Otherwise, we compute the delay by counting the length of the longest path of P^u. All of these operations can be performed in time $O((|\Sigma|+n)\cdot|A|\cdot|B|)$. □

5.3. Reduction to Bounded Ambiguity

There exist an alternative method by which to decide bounded delay, which is by reduction to bounded ambiguity of nFAs. The interest is more general than the concrete constructions above, in that it can also be applied to bounded concurrency.

The idea is to turn the dFA $P(A,B)$ it into an nFA $D(A,B)$ such that $amb_{D(A,B)}(w*\tau)=delay_{Q_{(A,B)}}(w,\tau)$ for all $\tau\in Q_{(A,B)}(w)$.

We construct $D(A,B)$ from $P(A,B)$ by adding a new state ok and ϵ-transitions from all unsafe states of type $\mathcal{V}_n$ to ok. Figure 4 presents the result of this operation on the automaton in Figure 3.

$$stat_{D(A,B)}=stat_{P(A,B)}\uplus\{ok\},\ init_{D(A,B)}=init_{P(A,B)},\ fin_{D(A,B)}=\{ok\}$$

$$\frac{r\in rul_{P(A,B)}}{r\in rul_{D(A,B)}}\qquad\frac{unsafe_{sel}(p,q)\qquad p\text{ has type }\mathcal{V}_n}{(p,q)\xrightarrow{\epsilon}ok\in rul_{D(A,B)}}\qquad\frac{a\in\Sigma}{ok\xrightarrow{(a,\emptyset)}ok\in rul_{D(A,B)}}$$

Proposition 17. *For all $\tau\in Q_{(A,B)}(w)$: $delay_{Q_{(A,B)}}(w,\tau)=amb_{D(A,B)}(w*\tau)$.*

Proof. Consider a run r of $D(A,B)$ on a canonical word $w * \tau$ with $\tau \in Q(w)$. We can show inductively on r that the ambiguity of $D(A,B)$ on w is exactly the number of states used in r that are not safe for selection. The initial state is unique as A is deterministic, so at the beginning the ambiguity is 1. When reading a new letter, if the associated state q is not unsafe or has not type $\mathcal{V}_n$, then there is only one way to continue the run, via a rule of $P(A,B)$. If it is unsafe with type $\mathcal{V}_n$, then there are two possibilities: either by using the run of $P(A,B)$, or by firing the ϵ-transition. Both runs will succeed (as *ok* is universal), so in this case the ambiguity is increased by one. Hence $amb_{D(A,B)}(w * \tau)$ is the number of unsafe states used in the run of $P(A,B)$, and also of A, on $w * \tau$. From the definitions of delay (here the type $\mathcal{V}_n$ ensures that we start counting at $latest(\tau)$), safe states and by Lemma 11, this is exactly $delay_{Q_{(A,B)}}(w,\tau)$. □

Proposition 17 provides a P-time reduction from bounded delay to bounded ambiguity and from k-bounded delay to k-bounded ambiguity. The results from the literature reported in Theorem 1 thus show that all these problems can be decided in P-time under the assumption that k is fixed.

It should be noticed that Theorem 4 obtained by a direct automaton construction is slightly stronger. First, it allows to compute the optimal bound in P-time, second, does not require to k in order to decide k-boundedness in P-time, and third yields small polynomials.

5.4. Deciding Bounded Concurrency

We show how to reduce in P-time bounded concurrency to bounded ambiguity and k-bounded concurrency to k-bounded ambiguity. We notice that we do not know how to obtain any more direct algorithm in this case.

The concurrency of a query counts the number of simultaneously alive partial candidates. In addition to sufficiency for selection, aliveness depends on sufficiency for rejection. We thus need a notion of safe states for rejection.

Definition 18. *A pair of states (p,q) of $P(A,B)$ is* safe for rejection by $Q_{(A,B)}$ *if no final state can be reached from (p,q), i.e., if $L(P(A,B)[init = \{(p,q)\}]) = \emptyset$.*

We saw in the proof of Theorem 4 how to compute safe states for selection, so now we need a method to compute safe states for rejection.

Lemma 19. *The set of safe states for rejection by $Q_{(A,B)}$ for* nFA*s A and B can be computed in time $O(|A| \cdot |B|)$.*

Proof. We compute the set of all unsafe states for rejection. In order to do so, it is sufficient to compute the set of all states of $P(A,B)$ from which some final state can be reached. This can be done by the following ground Datalog program:

$$\frac{p' \in fin_A \qquad q' \in fin_B}{unsafe_{rej}(p,q).} \qquad \frac{p \overset{(a,V)}{\to} p' \in rul_A \qquad q \overset{a}{\to} q' \in rul_B}{unsafe_{rej}(p,q) \text{ :- } unsafe_{rej}(p',q').}$$

This program can be constructed in time $O(|A| \cdot |B|)$ from A and B. By Proposition 36, the $lfp(P)$ can be computed in time $O(|A| \cdot |B|)$. □

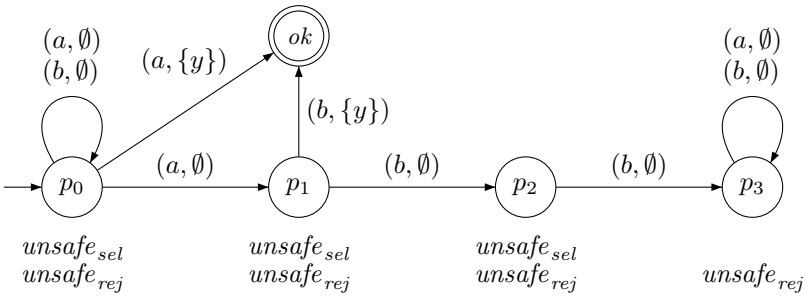


Figure 5: nFA $C(A,B)$ for query dFA A in Figure 3 and trivial universal B. Even though non-deterministic, the ambiguity of $C(A,B)$ is 1, equally to the concurrency of $Q_{(A,B)}$.

We define an nFA $C(A,B)$ such that $amb_{C(A,B)}(w*\eta) = concur_Q(w,\eta)$. The situation is a little different than for $D(A,B)$, in that $C(A,B)$ runs on words annotated by events rather than tuples. We fix a new variable $y \notin \mathcal{V}_n$ that will denote the event of interest, and define the alphabet of $C(A,B)$ to be $\Sigma \times 2^{\{y\}}$. The idea of nFA $C(A,B)$ is to guess a partial candidate τ, until the event marker y comes, and to test whether τ is alive at that event, and to accept in case of success.

$$stat_{C(A,B)} = stat_A \times stat_B \uplus \{ok\}$$
$$init_{C(A,B)} = init_A \times init_B$$
$$fin_{C(A,B)} = \{ok\}$$

$$\frac{(p,q) \overset{(a,V)}{\to} (p',q') \in rul_{P(A,B)}}{(p,q) \overset{(a,\emptyset)}{\to} (p',q') \in rul_{C(A,B)}}$$

$$\frac{(p,q) \overset{(a,V)}{\to} (p_1,q_1) \in rul_{P(A,B)} \quad unsafe_{sel}(p_1,q_1) \quad unsafe_{rej}(p_1,q_1)}{(p,q) \overset{(a,\{y\})}{\to} ok \in rul_{C(A,B)}}$$

Both rules guess a set of variables V and check that the current position is the denotation of all variables in V, by running automaton A with V in the input letter. The second rule inputs the event marker, and goes into the *ok*-state, if automaton $P(A,B)$ could move to states that are unsafe for both selection and rejection, so that the current partial candidate is alive. Note that, using Lemmas 16 and 19, $C(A,B)$ can be computed in polynomial time. For illustration, consider Figure 5 which shows the automaton $C(A,B)$ obtained from the automaton A in Figure 3 and the trivial universal automaton B.

Given a word $w = a_1 \cdot \ldots \cdot a_m$ and a position $1 \leq \eta \leq m$ we write $w|\eta$ for the word $(a_1,\emptyset)\cdot(a_{\eta-1},\emptyset)\cdot(a_\eta,\{y\})$.

Proposition 20. *$concur_{Q_{(A,B)}}(w,\eta) = amb_{C(A,B)}(w|\eta)$, for all $w \in L(B)$ and $\eta \in dom(w)$.*

Proof. Let $w \in L(B)$ and $\eta \in dom(w)$. Suppose that τ_1 and τ_2 are different partial tuples that are alive at η. Let r_1 and r_2 be the runs of A on the prefixes of $w*\tau_1$ resp. $w*\tau_2$ until η. Since τ_1 and τ_2 are different, there exists a position i such that the prefixes of length $i < \eta$ of $w*\tau_1$ and $w*\tau_2$ have different types. Since A is canonical, this implies that both runs assign states of different types to position i, so that $r_1(i) \neq r_2(i)$.

Let $a_1 \cdot \ldots \cdot a_\eta$ be the prefix of w until position η. By construction of $C(A,B)$, both runs r_i restricted to $\{1,\ldots,\eta{-}1\}$ are also runs of $C(A,B)$ on word $v =$

$(a_1 \cdot \ldots \cdot a_{\eta-1}) \otimes \emptyset$. These runs can be extended to successful runs of $C(A,B)$ on $w|\eta = v \cdot (a_\eta, \{y\})$ by mapping position η to *ok*, since both tuples τ_i are alive at event η (and thus neither safe for selection nor rejection). Both runs are different, since runs r_1 and r_2 differ at some position $i < \eta$. Hence $concur_{Q_{(A,B)}}(w,\eta) \leq amb_{C(A,B)}(w|\eta)$.

For the converse, consider two different runs r_1 and r_2 of $C(A,B)$ on $w|\eta$. We now build two partial tuples τ_1 and τ_2 and the corresponding runs r_1' and r_2' of A on the prefixes of $w * \tau_1$ and $w * \tau_2$ until η. These are hidden in the rules applied for producing runs r_1 and r_2 by $C(A,B)$. Since the states which are permitted to move to *ok* are alive, the runs r_1' and r_2' can be extended into an alive state at η. This shows that both tuples τ_1 and τ_2 are alive. They are different, since produced from distinct runs r_1 and r_2. This shows that $amb_{C(A,B)}(w|\eta) \leq concur_{Q_{A,B}}(w,\eta)$. □

Theorem 5. *Bounded and k-bounded concurrency for queries and schemas defined by canonical* dFA*s can be decided in P-time for any fixed $k \geq 0$.*

Proof. From Lemmas 16 and 19, $C(A,B)$ can be constructed in P-time from A and B. Before the construction, we need to make A and B productive, which can be done in time $O(|A| + |B|)$. By Proposition 20, it remains to decide the finite (resp. k-bounded) ambiguity of $C(A,B)$. Since k is fixed by assumption, this can be done in P-time according to Theorem 1. □

It should be noticed that our results for concurrency problems are weaker than those for delay problems. In particular, we don't know whether the concurrency of a dFA query can be computed in P-time and whether k-bounded concurrency can be decided in P-time for variable k (where k becomes part of the input of the problem). The reason is that we could not come up with a direct automaton construction in the case of concurrency, and that we don't know neither whether k-bounded ambiguity of nFAs is in P-time for variable k.

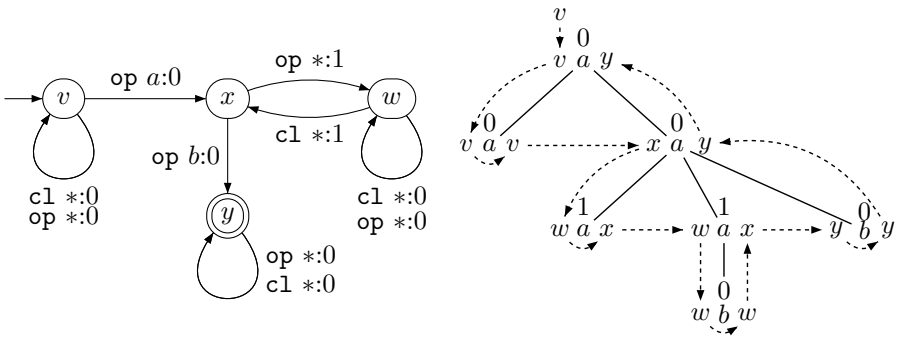
6. Queries on Unranked Trees by Nested Word Automata

We state our main results on deciding bounded delay and concurrency for queries on unranked trees that are defined by dNWAs [4, 2]. These automata were called deterministic streaming tree automata in [27], since they define deterministic streaming algorithms in a natural way. As shown there, the notion of deterministic forest pushdown automata [39] is equivalent to dNWAs too.

Alternatively, we could have decided to work with bottom-up or top-down deterministic tree automata on ranked trees modulo some binary encoding. It is well-known that the same queries can be expressed this way but possibly in a less succinct manner. Intuitively, the notion of determinism of dNWAs is advantageous since corresponding naturally to the notion of determinism of streaming algorithms. For this reason it is possible to compile XPath queries in large streamable fragments of XPath to dNWAs [25, 26] in polynomial time.

6.1. Nested Word Automata

We start by recalling the notion of dNWAs. We consider nested words as linearizations of unranked trees. The nested word for the tree $a(b(c),d)$ for instance, is $(\texttt{op},a)\cdot(\texttt{op},b)\cdot(\texttt{op},c)\cdot(\texttt{cl},c)\cdot\ (\texttt{cl},b)\cdot(\texttt{op},d)\cdot(\texttt{cl},d)\cdot(\texttt{cl},a)$ or in XML



(a) NWA A with wildcards for letters (b) A successful run of A

Figure 6: An NWA accepting all trees with some a-node with b-child, *i.e.*, for the FO formula $\exists x.(lab_a(x) \wedge \exists y.(ch(x,y) \wedge lab_b(y)))$.

syntax `<a><b><c></c></b><d></d></a>`. In order to avoid further encodings, we will interpret NWAs directly on unranked trees, *i.e.*, as streaming tree automata [27].

Definition 21. *An* NWA *is a tuple* $A = (\Sigma, \Gamma, \mathit{stat}, \mathit{init}, \mathit{fin}, \mathit{rul})$ *where* Γ *is a finite set of stack symbols,* stat *a finite set of states,* $\mathit{init}, \mathit{fin} \subseteq \mathit{stat}$ *and* $\mathit{rul} \subseteq \mathit{stat}^2 \times \Gamma \times \hat{\Sigma}$*, that we denote as* $q_1 \xrightarrow{\alpha\ a:\gamma} q_2$ *where* $q_1, q_2 \in \mathit{stat}$*,* $\alpha \in \{\texttt{op}, \texttt{cl}\}$*,* $\gamma \in \Gamma$*, and* $a \in \Sigma$*.*

Figure 6(a) presents an NWA that accepts all trees with labels in $\{a, b\}$ that contain some node labeled by a with a child labeled by b. This is the language of the Boolean query defined by the FO formula $\exists x.(lab_a(x) \wedge \exists y.(ch(x,y) \wedge lab_b(y)))$ without free variables.

An NWA traverses a tree t in pre-order. It visits every node of t twice, once when entering (open event) and once when exiting (close event) the subtree. An NWA associates a state to each event of t, and a stack symbol to each node of t, as shown in Figure 6(b). This corresponds to operating on the stream of events produced by a SAX parser on the XML linearization. At every time point, the configuration of an NWA maintains a node, a current state, and a current stack, which is the lists of states annotated to the path to the current node. The height of the stack is thus equal to the depth of the current node.

More formally, a run of an NWA on a tree t is a pair of functions (r_e, r_n) with types $r_e : \mathit{eve}(t) \to \mathit{stat}$ and $r_n : \mathit{nod}(t) \to \Gamma$ which map events to states and nodes to stack symbols, such that $r_e(\texttt{start}) \in \mathit{init}$ and the rule

$$r_e(\mathit{pred}((\alpha, \pi))) \xrightarrow{\alpha\ a:r_n(\pi)} r_e(\alpha, \pi)$$

belongs to rul for all $\pi \in \mathit{nod}(t)$ with $a = \mathit{lab}^t(\pi)$, and actions $\alpha \in \{\texttt{op}, \texttt{cl}\}$. The language $L(A)$ is the set of all unranked trees $t \in \mathcal{T}_\Sigma$ that permit a successful run by A, *i.e.*, $r_e((\texttt{cl}, \epsilon)) \in \mathit{fin}$.

An NWA is *deterministic* or equivalently a DNWA, if it has a single initial state, no two `op` rules for the same letter use the same event state on the left, and no two `cl` rules for the same letter use the same stack symbol and the same event state on the left. DNWAs can perform one-pass typing for extended DTDs [13] with restrained competition [36] as well as for earliest query answering [29].

6.2. Closure Properties

Language recognizable by NWAs enjoy the usual closure properties under Boolean operations, projection, and cylindrification.

We will consider projections $\Pi_i : \mathcal{T}_{\Sigma_1 \times \ldots \times \Sigma_m} \to \mathcal{T}_{\Sigma_i}$ for all $1 \leq i \leq m$, such that all $\Pi_i(t)$ relabels all nodes $\pi \in nod(t)$ to the i-th component of its label. We write $t = t_1 * \cdots * t_m$ if $\wedge_{i=1}^m \Pi_i(t) = t_i$. We will also use more general projections operations $\Pi_I : \mathcal{T}_{\Sigma_1 \times \ldots \times \Sigma_m} \to \mathcal{T}_{\Sigma_{i_1} \times \ldots \times \Sigma_{i_n}}$ that preserve a subset of components $I = \{i_1, \ldots, i_n\}$ where $1 \leq i_1 < \ldots < i_n \leq m$ by $\Pi_I(t_1 * \ldots * t_m) = t_{i_1} * \ldots * t_{i_n}$. Projections can be lifted to languages of trees $L \subseteq \mathcal{T}_{\Sigma_1 \times \ldots \times \Sigma_m}$ by $\Pi_I(L) = \{\Pi_I(t) \mid t \in L\}$.

We also need cylindrification operations on tree languages, which may add, copy, and exchange components of tuple trees, but not delete them. We formalize unsorted cylindrification operations that apply to trees $L \subseteq \mathcal{T}_{\Sigma^n}$, where all components have the same signature Σ. For functions $\theta : \{1, \ldots, m\} \to \{1, \ldots, m\}$ with $\{1, \ldots, n\} \subseteq \theta(\{1, \ldots, m\})$ we define:

$$c_\theta L = \{t_{\theta(1)} * \ldots * t_{\theta(m)} \in \mathcal{T}_{\Sigma^m} \mid t_1 * \ldots * t_n \in L\}$$

Note that all newly added components have signature Σ. Sorted cylindrification operations, that add components of particular types, can be obtained from unsorted cylindrification and intersection.

Proposition 22. *Languages of unranked trees recognizable by* NWA*s are closed under Boolean operations, projection and cylindrification. The corresponding operations on* NWA*s all preserve determinism except for projection and can be performed in P-time except for complementation non-deterministic* NWA*s. Furthermore every* NWA *can be made deterministic (in time* $O(2^{|A|^2})$*).*

The analogous results are well-known for bottom-up deterministic tree automata on ranked trees. Proposition 22 can be established for dNWAs in analogy, or else by compiling NWAs to standard tree automata that operate on curried binary encodings of unranked trees (see Appendix B).

It should be noticed that the cylindrification operations c_θ here are a little richer than the usual ones c_i as in [19] that insert a single new component at position i. In addition, operators c_θ can copy components, which can be tested by intersection with deterministic NWAs that recognize the set $\{t * t \mid t \in \mathcal{T}_\Sigma\}$. They can also permute components. While operation c_θ can be implemented in P-time for every fixed θ by computing intersections with a fixed number of tree automata (but not for flexible θ).

Note also cylindrification cannot delete components in contrast to projection since the latter may spoil determinism in contrast to the former.

6.3. Querying Unranked Trees by dNWAs

Total n-ary queries Q_A on unranked trees in $\mathcal{T}_\Sigma$ can be defined by NWAs A over $\Sigma \times \mathbb{B}^n$ that recognize the canonical language of Q. Partial queries $Q_{(A,B)}$ are defined by adding automaton B over Σ that recognizes the domain of the query.

The closure properties of NWAs in combination with determinization as stated in Proposition 22 imply that all MSO definable n-ary queries on unranked trees are definable by dNWAs. Nevertheless, we are able to prove the following theorem:

Theorem 6 (Main)**.** *Bounded delay is decidable in P-time for n-ary queries defined by deterministic nested word automata (d*NWA*s) where n may be variable. Bounded concurrency is decidable in P-time for fixed n. For fixed k and n, k-bounded delay and concurrency are decidable in P-time.*

Unfortunately, our P-time algorithms for dFA-queries on words cannot be lifted to dNWA-queries trees in any straightforward manner. The problem is that the notion of safe states of dFAs must be generalized to safe configuration of dNWAs where a configuration depends on the current stack in addition. Nevertheless it was shown in [29] how to extend some of the automata constructions to dNWA queries. These constructions, however, produce deterministic NWAs of exponential size rather than non-deterministic NWAs of polynomial size.

In order to solve these problems, we present another proof by reduction to bounded resp. k-bounded valuedness of recognizable relations between unranked trees. We first show that bounded and k-bounded valuedness of NWA-recognized relations between unranked trees can be decided in P-time (Section 7) and then that delay and aliveness of dNWA-queries can be defined by NWAs of polynomial size (Section 8).

7. Recognizable Relations between Unranked Trees

We study recognizable relations between trees in the ranked [19] and unranked case [11]. A recognizable relation is a set of tuples of trees such that the set of overlays of these tuples is recognizable by a tree automaton. Similarly to transducers, a notion of valuedness can be associated with binary recognizable relations. We will use valuedness of recognizable tree relations to capture the delay and concurrency of queries.

We first define recognizable relations over unranked trees and provide a key example of such relations. We then present a procedure for deciding the bounded valuedness of binary recognizable relations in polynomial time. The proof is by reduction to bounded valuedness of tree transducers [52]. We then recall a standard method to define recognizable relations in FO logic from a set of basic recognizable relations, while relying on closure properties of tree automata. Finally, we prove that k-bounded valuedness of binary recognizable relations is decidable in P-time, by reduction to emptiness of tree automata.

An alternative way of proving the P-time decidability of bounded valuedness of recognizable relations could be to use *visibly pushdown transducers* (VPTs) [47] instead of bottom-up tree transducers. Indeed, VPTs directly operate on unranked trees. Functionality (*i.e.* 1-bounded valuedness) of VPTs was proven decidable in P-time recently [23], but it is still open whether k-bounded valuedness is decidable in P-time (it is known to be in NP-time [23]). Neither exists there any decidability result for bounded valuedness of VPTs.

In this paper, we use NWAs as underlying class of tree automata. We could have chosen another class $\mathcal{A}$ of unranked tree automata, provided that a) it can be translated in P-time to NWAs, and b) $\mathcal{A}$ is closed under intersection, complementation, cylindrification and projection modulo P-time transformations, that preserve determinism except for projection. In particular, this is the case for two classes: top-down nTAs operating on the *fcns* encoding of trees, and bottom-up nTAs over the *curry* encoding of trees (see Appendix B for more details). Note however, that hedge automata with dFAs for horizontal languages

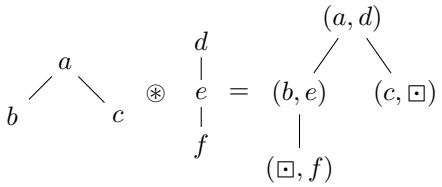


Figure 7: Example for overlays

[19] fail to satisfy these requirements, since deterministic hedge automata cannot be complemented in P-time.

7.1. Recognizable Relations

The *overlay* of k unranked trees $t_i \in \mathcal{T}_{\Sigma^i}$ is the unranked tree $t_1 \circledast \ldots \circledast t_k$ in $\mathcal{T}_{\Sigma^1_\boxdot \times \ldots \times \Sigma^k_\boxdot}$ obtained by superposing these k trees top-down and left-to-right; the $\boxdot$ symbol represents missing children where the structures of the trees differ. This is illustrated in Figure 7 and formally defined by:

$$a(t_1,\ldots,t_k) \circledast b(t'_1,\ldots,t'_l) = \begin{cases} (a,b)(t_1 \circledast t'_1,\ \ldots,\ t_l \circledast t'_l,\ t_{l+1} \circledast \boxdot,\ \ldots,\ t_k \circledast \boxdot) & \text{if } l \leq k \\ (a,b)(t_1 \circledast t'_1,\ \ldots,\ t_k \circledast t'_k,\ \boxdot \circledast t_{k+1},\ \ldots,\ \boxdot \circledast t_l) & \text{otherwise} \end{cases}$$

Overlays of ranked trees can be obtained this way too [19], except that overlayed symbols need to inherit the maximal arity. A k-ary relation R between unranked trees is *recognizable* iff the language of its overlays $ovl(R) = \{t_1 \circledast \ldots \circledast t_k \mid (t_1,\ldots,t_k) \in R\}$ is recognizable by an NWA. We say that R is recognized by the NWA A if $ovl(R) = L(A)$.

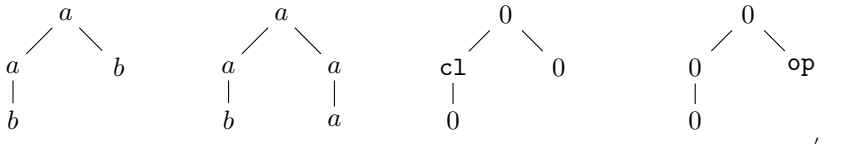
7.2. Example: Tree Equality until Some Event

For illustration, we consider the relation $eq_\eta(t,u)$ between unranked trees that expresses the equality of their sequentializations until some event. This holds if t and u have the same structure and labels until event η. The idea now is to represent an event η by some unranked tree, so that equality until some event can be represented by a ternary relation between trees.

More precisely, we represent an event η of t by the unranked tree $ren^\eta(t)$ with signature $\{0, \mathtt{op}, \mathtt{cl}\}$, whose domain coincides with that of t. If the event $\eta = (\alpha, \pi)$ is an opening event, then its node π is renamed to $\mathtt{op}$, and in analogy if it is a closing event. All other nodes are renamed to 0. See Figure 8 for a graphical illustration. More formally, let $ren^{(\alpha,\pi)}(t) \in \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}}$ be obtained by renaming the label of π to α and the labels of all other nodes of t to 0. We then define the ternary relation $Eq \subseteq \mathcal{T}_\Sigma \times \mathcal{T}_\Sigma \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}}$ such that for all events $\eta \in eve(t)$ and trees $t \in \mathcal{T}_\Sigma$:

$$(t, u, ren^\eta(t)) \in Eq \Leftrightarrow_{df} eq_\eta(t,u)$$

Lemma 23. *For every signature Σ we can compute a d*NWA *in time $O(|\Sigma|^2)$, that recognizes the relation $Eq \subseteq \mathcal{T}_\Sigma \times \mathcal{T}_\Sigma \times \mathcal{T}_{\{0,op,cl\}}$.*



(a) Tree t (b) Tree u (c) Tree $ren^{\eta}(t)$ with $\eta = (1, \mathtt{cl})$. (d) Tree $ren^{\eta'}(t)$ with $\eta' = (2, \mathtt{op})$.

Figure 8: $(t, u, ren^{\eta}(t)) \in Eq$ but $(t, u, ren^{\eta'}(t)) \notin Eq$

Proof. We define a dNWA A on $\Sigma_{\boxdot} \times \Sigma_{\boxdot} \times \{0, \mathtt{op}, \mathtt{cl}\}_{\boxdot}$ such that $L(A) = ovl(Eq)$. We use two states $stat_A = \{before, after\}$, where $init_A = \{before\}$ and $fin_A = \{after\}$. We use a single dummy stack symbol $\Gamma = \{\_\}$. The rules are given by the following inference schema:

$$\frac{\alpha \in \{\mathtt{op}, \mathtt{cl}\} \qquad a \in \Sigma \qquad b \in \Sigma_{\boxdot}}{\begin{array}{ll} before \xrightarrow{\alpha\ (a,a,0):\_} before & before \xrightarrow{\mathtt{op}\ (a,a,\mathtt{cl}):\_} before \\ before \xrightarrow{\alpha\ (a,a,\alpha):\_} after & after \xrightarrow{\alpha\ (a,b,0):\_} after \\ after \xrightarrow{\mathtt{cl}\ (a,b,\mathtt{op}):\_} after & after \xrightarrow{\alpha\ (\boxdot,a,\boxdot):\_} after \end{array}}$$

Note that the rule $before \xrightarrow{\mathtt{op}\ (a,a,\mathtt{cl}):\_} before$ is used to check the equality below a node π if prefix equality has to be checked until $(\mathtt{cl}, \pi)$. The NWA A has size $O(|\Sigma^2|)$ and can be computed in this time. □

Other examples of recognizable relations [11] are the tree extension relations $\leq_{\downarrow}, \leq_{\rightarrow} \subseteq \mathcal{T}_{\Sigma} \times \mathcal{T}_{\Sigma}$, such that $t \leq_{\downarrow} t'$ if t' is obtained by repeatedly adding children to leaves of t, and $t \leq_{\rightarrow} t'$ if t' is obtained by repeatedly adding next-siblings to right most children of t.

7.3. Bounded Valuedness

Like bottom-up tree transducers on ranked trees, bounded valuedness of recognizable relations over unranked trees is decidable in P-time.

Theorem 7. *For every* NWA A *recognizing a binary relation* R *between unranked trees,* $val(R) < \infty$ *can be decided in P-time in* $|A|$.

Note that A may be non-deterministic. The deterministic case is easier since then the valuedness of R is equal to the ambiguity of the projection of A to the input signature of R. So the problem is that the non-determinism introduced by projection must be properly separated from any non-determinism of A.

Proof. We prove Theorem 7 by a reduction to Theorem 3, *i.e.* the P-time decision procedure for bounded valuedness of bottom-up (ranked) tree transducers.

Our first reduction is to the analogous problem for the ranked case. Here we use the top-down encoding (*fcns*) and show that it preserves the valuedness of relations (Proposition 24). Moreover, we can compute in P-time an NTA A' recognizing the *fcns* encoding R' of R (see Proposition 39).

The second reduction transforms binary relations on ranked tree R to relabeling relation $Relab_R$ with the same valuedness. A *relabeling relation* $R \subseteq$

$\mathcal{T}_{\Sigma^1} \times \ldots \times \mathcal{T}_{\Sigma^n}$ is a relation between trees of the same structure, *i.e.* whenever $(t_1, \ldots, t_n) \in R$ then $nod(t_1) = \ldots = nod(t_n)$. In other words, the overlays in $ovl(R)$ do not contain any place holder $\boxdot$. Proposition 25 shows that $val(R) = val(Relab_R)$, and provides a P-time procedure to compute an nTA A' recognizing $Relab_R$, from an nTA A recognizing R.

Finally, the third reduction reduces the bounded valuedness problem of recognizable relabeling relations to bounded valuedness of bottom-up tree transducers (see Proposition 26). □

Let *ren* be the morphism on binary trees that renames constants $(\boxdot, \ldots, \boxdot)$ to $\boxdot$ and preserves the trees otherwise. This morphism is linear and one-to-one, so it preserves regularity in both directions: L is recognizable iff $ren(L)$ is recognizable. Overlays of unranked and ranked trees are related in the following way:

$$fcns(t_1 \circledast \ldots \circledast t_n) = ren(fcns(t_1) \circledast \ldots \circledast fcns(t_n)) \quad (1)$$

The following proposition shows that valuedness is preserved by the *fcns* encoding. Let $fcns(R) = \{(fcns(t_1), fcns(t_2)) \mid (t_1, t_2) \in R\}$.

Proposition 24. *A binary relation R between unranked trees is recognizable iff the corresponding relation between binary trees $fcns(R)$ is, and $val(fcns(R)) = val(R)$.*

Proof. By definition $fcns(R) = \{(fcns(t_1), fcns(t_2)) \mid (t_1, t_2) \in R\}$. Equation (1) yields $fcns(ovl(R)) = ren(ovl(fcns(R)))$. The morphism *ren* preserves recognizability back and forth. Thus, $fcns(R)$ is a recognizable relation iff $ovl(fcns(R))$ is recognizable language of binary trees iff $ren(ovl(fcns(R)))$ is a recognizable language of binary trees iff $fcns(ovl(R))$ is a recognizable language of binary trees iff $ovl(R)$ is a recognizable language of unranked trees iff R is a recognizable relation of unranked trees. □

We show how to convert recognizable relations over binary trees into recognizable relabelings, while preserving valuedness. Let R be a recognizable relation over $\mathcal{T}^{bin}_{\Sigma^1} \times \mathcal{T}^{bin}_{\Sigma^2}$. We define a recognizable relabeling $Relab_R \in \mathcal{T}^{bin}_{\Sigma^1_\boxdot \times \Sigma^2_\boxdot}$, where we have 2 symbols $(\boxdot, \boxdot)$ with arities 0 and 2 respectively. The idea is to expand both trees in pairs $(t_1, t_2) \in R$ to trees $(t'_1, t'_2) \in Relab_R$ of the same structure, by repeatedly adding $\boxdot$-children to leaves of t_1 or t_2. Expansion $ex_i(t, t')$ holds for two trees $t \in \mathcal{T}^{bin}_{\Sigma^i}$ and $t' \in \mathcal{T}^{bin}_{\Sigma^i_\boxdot}$ if $nod(t) \subseteq nod(t')$, both trees have the same labels on common nodes, and all new nodes of t' are labeled by $\boxdot$. We define the relabeling $Relab_R$ by:

$$Relab_R = \{(t'_1, t'_2) \in \mathcal{T}_{\Sigma^1_\boxdot} \times \mathcal{T}_{\Sigma^2_\boxdot} \mid (t_1, t_2) \in R,\ ex_1(t_1, t'_1), ex_2(t_2, t'_2),\ nod(t'_1) = nod(t'_2)\}$$

An example is given in Figure 9. While the relation R there is finite, the corresponding relabeling $Relab_R$ is infinite, since there may be infinitely many witnesses for every pair of R.

Proposition 25. *$Relab_R$ and R have the same valuedness. Moreover, an nTA A' recognizing $Relab_R$ can be computed from an nTA A recognizing R in time $O(|A|)$.*

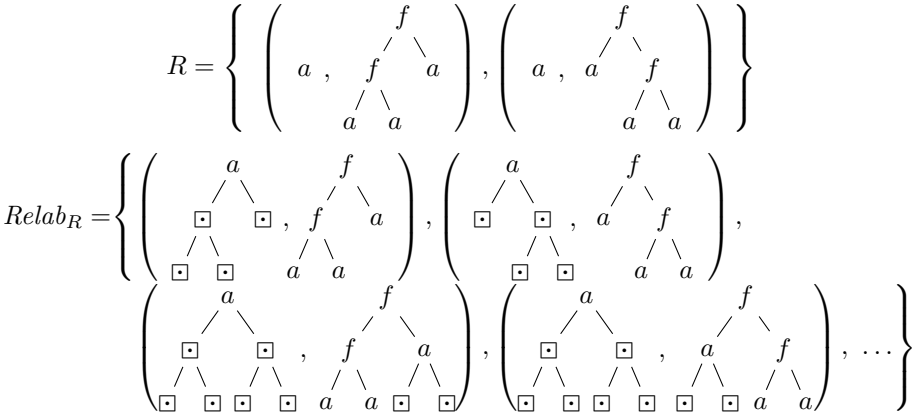


Figure 9: A recognizable relation R and the relabeling $Relab_R$ with the same valuedness.

Proof. The nTA A' is obtained by adding one more state to A, so that $stat(A') = stat(A) \cup \{q_\boxdot\}$ and $fin(A) = fin(A')$. Automaton A' runs A top-down, until $\boxdot$ occurs, and then checks for equal domains:

$$(\boxdot, \boxdot) \to q_\boxdot \in rul_{A'} \qquad (\boxdot, \boxdot)(q_\boxdot, q_\boxdot) \to q_\boxdot \in rul_{A'} \qquad \frac{(a,b) \to q \in rul_A}{(a,b)(q_\boxdot, q_\boxdot) \to q \in rul_{A'} \quad (a,b)(q_\boxdot) \to q \in rul_{A'}}$$

Valuedness preservation is checked in Lemma 40 of Appendix C. □

For relabelings over binary trees, bounded valuedness can be tested efficiently.

Proposition 26. *The finite valuedness of a binary relabeling recognizable relation R over binary trees can be decided in P-time in $|A|$, when given an* nTA *A recognizing R.*

Proof. Let $R \subseteq \mathcal{T}^{bin}_{\Sigma_1} \times \mathcal{T}^{bin}_{\Sigma_2}$ be a relabeling relation for binary signatures, and A an nTA for trees in $\mathcal{T}^{bin}_{\Sigma_1 \times \Sigma_2}$ that recognizes R, *i.e.* $L(A) = ovl(R)$. We transform A into a bottom-up tree transducer T defining the relation R. The rules of T are inferred as follows where x_1, x_2 are variables:

$$\frac{(f,g)(q_1,q_2) \to q \in rul_A}{f(q_1(x_1), q_2(x_2)) \to q(g(x_1,x_2)) \in rul_T} \qquad \frac{(a,b) \to q \in rul_A}{a \to q(b) \in rul_T}$$

This transducer T has the same valuedness as R. Theorem 3 shows that it can be decided in polynomial time whether T is finite-valued, *i.e.* whether R is bounded. □

The above construction of bottom-up transducers cannot be lifted to recognizable relations beyond relabelings. This is why we introduced $Relab_R$. Even though testing bounded valuedness of tree transducers is known to be in P-time [52], the complexity of known polynomial time algorithms is much higher than for testing bounded ambiguity of tree automata [48].

Note that if we add the condition that A is deterministic, then a similar construction could have been done using automata instead of transducers. If

A' is the automaton on Σ_2 obtained from A by projecting the Σ_1 components, then $amb(A') = val(R)$, and ambiguity (and k-ambiguity) of A' can be obtained in P-time [51, 50]. However, we will use relations defined by existential first-order formulas over relation symbols, whose corresponding automata are non-deterministic.

7.4. Sorted FO Logic

Just as logics on words give alternative means of specifying regular languages, FO logic on unranked trees gives a means of specifying recognizable relations [11]. Since we have to deal with relations of trees over various signatures we will use a sorted FO logic, so that signatures can be mapped to sorts.

A *sorted relational signature* is a relational signature $\Delta = \Omega \uplus \Re$, that consists of a set of monadic symbols $\omega \in \Omega$ called *sorts* and a set of relation symbols $r \in \Re$, each of which has a sort $sort(r) \in \Omega^{ar(r)}$.

We fix an infinite set $\mathcal{V}$ of variables. A *sorted* FO *formula* ϕ over Δ has the following abstract syntax, where $r \in \Re$ is a relation symbol of arity n, $X_1, \ldots, X_n, X \in \mathcal{V}$ and $\omega \in \Omega$:

$$\phi \ ::= \ r(X_1, \ldots, X_n) \mid \phi \wedge \phi' \mid \neg\phi \mid \exists X{:}\omega.\ \phi$$

Every sorted FO formula can be understood as a unsorted FO formula, obtained by mapping sort bounded quantifiers $\exists x{:}\omega.\phi$ to unsorted quantifiers $\exists x.(\omega(x) \wedge \phi)$; here sorts $\omega \in \Omega$ are used as monadic predicate symbols.

A *sorted relational structure* s over $\Delta = \Omega \uplus \Re$ is a relational structure such that: $dom(s) = \cup_{\omega\in\Omega}\omega^s$ and for every relation symbol $r \in \Re$ of arity m:

$$sort(r) = (\omega_1, \ldots, \omega_m) \Rightarrow r^s \subseteq \omega_1^s \times \ldots \times \omega_m^s$$

Every sorted FO formula ϕ over Δ with at most m free sorted variables defines an m-ary relation for every sorted relational structure s over Δ:

$$\mathcal{R}_{\phi(X_1:\omega_1,\ldots,X_m:\omega_m)}(s) = Q_{\phi(X_1,\ldots,X_m)}(s) \cap \omega_1^s \times \ldots \times \omega_m^s$$

Note that the sorted formula ϕ in $Q_{\phi(X_1,\ldots,X_m)}$ is considered as an unsorted FO formula in order to reuse the definition of queries in this new context.

7.5. Sorted FO Logic of Recognizable Relations

In what follows let Ω be a fixed collection of unranked alphabets. Note that only the simpler case with a single signature was treated in [11]. For every sorted relation signature $\Delta = \Omega \uplus \Re$, we will write $\text{FO}_\exists[\Re]$ for the set of sorted Δ-formulas in prenex normal form where quantifiers are existential.

Definition 27. *A* sorted structure of tree relations *s over $\Re$ is a sorted relational structure over $\Omega \uplus \Re$ in which all alphabets $\Sigma \in \Omega$ are interpreted as the set of unranked Σ-trees $\Sigma^s = \mathcal{T}_\Sigma$. We call a sorted structure of tree relations s* recognizable *if all tree relations r^s with $r \in \Re$ are recognizable.*

A sorted structure of recognizable tree relations over $\Re$ can be defined by a collection of NWAs $\{A_r\}_{r\in\Re}$ defining a recognizable relation between unranked trees for every relation symbol. While overloading notion, we will write $s = \{A_r\}_{r\in\Re}$ for the sorted structure of tree relations defined by these automata.

Every sorted FO formula ϕ with at most m free sorted variables defines an m-ary relation:

$$\mathcal{R}_{\phi(X_1:\Sigma_1,\ldots,X_m:\Sigma_m)}(s) \subseteq \mathcal{T}_{\Sigma_1} \times \cdots \times \mathcal{T}_{\Sigma_m}$$

The closure properties of NWAs under Boolean operations, cylindrification, and projection ensure that all such relations are recognizable. Furthermore, under some assumptions, an NWA recognizing the relation defined by an $\text{FO}_\exists[\Re]$ formula can be computed efficiently from the NWAs $\{A_r\}_{r\in\Re}$.

Proposition 28. *Let ϕ be a fixed formula in $\text{FO}_\exists[\Re]$ with at most m free sorted variables $X_1{:}\Sigma_1, \ldots, X_m{:}\Sigma_m$ where $X_i \in \mathcal{V}$ and $\Sigma_i \in \Omega$. Then there exists a polynomial p such that for all sorted structures of recognizable tree relations $s = \{A_r\}_{r\in\Re}$ defined by NWAs such that A_r is deterministic if r occurs in the scope of a negation in ϕ, one can compute in time $p(\sum_{r\in\Re} |A_r|)$ an NWA that recognizes the relation $\mathcal{R}_{\phi(X_1:\Sigma_1,\ldots,X_m:\Sigma_m)}(s)$. The computed NWA is deterministic, if all automata are deterministic and ϕ is free of existential quantifiers.*

Proof. The proposition relies of the closure properties of NWAs. The proof, which is by induction on the structure of formulas in $\text{FO}_\exists[\Re]$, is mostly standard and presented in detail in Appendix C. Note that all automata in the construction remain deterministic, except for those capturing outermost existential quantification. □

In the remainder of this section, we prove that k-bounded valuedness of NWA-recognized relations is decidable in P-time too. There, we will use the above proposition for the first time. Further applications will follow in Section 8, in order to construct NWAs of polynomial size, that recognize relations capturing delay and concurrency of dNWA defined queries.

7.6. k-Bounded Valuedness

In this section we study the problem of deciding whether a binary recognizable relation has k-bounded valuedness. We first for all fixed k that k-bounded valuedness is decidable in P-time. Subsequently we consider the problem for variable k, and prove that it becomes EXPTIME-hard.

Note that we cannot obtain a P-time decision for k-bounded valuedness by using transducers, in contrast to Proposition 26, since known algorithms for deciding k-boundedness of transducers are in coNP-time (Theorem 2.2 of [53]).

We can neither reduce the problem to deciding k-ambiguity of tree automaton. We will need to measure the valuedness of relations that capture delay and concurrency, but in general, $amb(A)$ and $val(R)$ may be uncomparable even for tree automata A recognizing relation R.

Theorem 8. *Let Σ_1 and Σ_2 be two alphabets and $k \in \mathbb{N}_0$ fixed. There exists a polynomial p such that for every relation $R \subseteq \mathcal{T}_{\Sigma_1} \times \mathcal{T}_{\Sigma_2}$ recognized by a possibly non-deterministic NWA A, $val(R) \leq k$ can be decided in time $p(|A|)$.*

Proof. We consider the tree relation $SameTree = \{(t,t) \mid t \in \mathcal{T}_{\Sigma_2}\}$ which is recognizable by an NWA of size $O(|\Sigma_2|^2)$. We fix a binary relation symbol r and a relation symbol $SameTree$. For every relation R we define a structures s_R such that $r^{s_R} = R$ and $SameTree^{s_R}$ is the relations with the same name. We define a formula $val_{>k}$ with $k+2$ free variables in the logic of recognizable relations in

FO$\exists[r, SameTree]$, such that $\mathcal{R}_{val_{>k}(X:\mathcal{T}_{\Sigma_1},X_1:\mathcal{T}_{\Sigma_2},\ldots,X_{k+1}:\mathcal{T}_{\Sigma_2})}(s_R) = \emptyset$ if and only if $val(R) > k$:

$$val_{>k} =_{df} \bigwedge_{1\leq i\leq k+1} r(X, X_i) \wedge \bigwedge_{1\leq i<j\leq k+1} \neg SameTree(X_i, X_j)$$

An NWA recognizing relation $\mathcal{R}_{val_{>k}(X:\Sigma_1,X_1:\Sigma_2,\ldots,X_{k+1}:\Sigma_2)}(s_R) = \emptyset$ can be computed in polynomial time from the NWA A, where the polynomial depends on the fixed parameters $|\Sigma_1|$, $|\Sigma_2|$ and k. This follows from Proposition 28 since relation symbol r does not occur below negation in formula $val_{>k}$. Emptiness of the language of this automaton can be tested in linear time. Hence, there exists a polynomial p (depending on the fixed parameters k, Σ_1, and Σ_2), such that we can check $val(R) > k$ in polynomial time $O(p(|A|))$ from an NWA A recognizing R. □

Theorem 8 provides a P-time decision procedure for k-bounded valuedness, under the assumption that k is fixed. The proof relies on an NWA of size $O(|A|^{k+1})$ and a complexity bound of $O(p(|A|^{k+1}))$ for some polynomial p. Without bounding k, however, this algorithm can only be shown to be in EXPTIME.

Theorem 9. *The problem that inputs $k \in \mathbb{N}_0$ and an NWA A recognizing a binary relation R between unranked trees, and outputs the truth value of $val(R) \leq k$ is* EXPTIME*-complete.*

Proof. As argued above, the algorithm proving Theorem 8 runs in EXPTIME if not bounding k. For the hardness part, we will reduce emptiness of intersection of deterministic NWAs to this problem [4, 54]. Let $Int(S)$ be the problem that inputs S, a finite sequence of dNWAs, and outputs "yes" if and only if there is at least one tree recognized by each automaton of the sequence. For each NWA $A \in S$, consider the binary relation R_A that associates trees t with accepting runs of A on t: $R_A = \{(t, r) \mid r \text{ is a run of } A \text{ on } t\}$. From $A \in S$, we can build in polynomial time an NWA recognizing R_A. So, from S – w.l.o.g. we suppose that the set of states are disjoint – we construct in polynomial time an NWA A_S for the binary relation $\vee_{A\in S} R_A$. As the automata are deterministic, A_S will be $(|S|-1)$-bounded iff there isn't any tree recognized by each automaton of the sequence. The conclusion follows, because emptiness of intersection of dNWAs is EXPTIME-hard [4, 54]. □

Using the above constructions and Theorem 5.5 of [53], we can build an algorithm for computing the exact value of $val(R)$, if it exists. The overall complexity is a fixed number of exponentials in $|A|$.

8. Deciding Bounded Delay and Concurrency

We prove our main Theorem 6 on deciding bounded delay and concurrency for queries defined by dNWAs by reduction to bounded valuedness of recognizable relations.

8.1. Basic Recognizable Relations

We start by defining various relations between trees by dNWAs, that we will use later on for defining the delay and concurrency of dNWA-defined queries by recognizable relations between trees.

In this section, we fix the alphabets $\Omega = \{\Sigma, \{0, \mathtt{op}, \mathtt{cl}\}\} \cup \bigcup_n \{2^{\mathcal{V}_n}\}$ and consider the sorted relational signature $\Delta = \Omega \uplus \Re$ with relation symbols $\Re = \{\mathit{Can}, \mathit{Bef\&Can}, \mathit{Bef\&NotCan}, \mathit{Bef}_\bullet, S, \mathit{Eq}_\Sigma, \mathit{SameTuples}\} \cup \bigcup_n \{\mathit{Eq}_{2^{\mathcal{V}_n}}, \mathit{Type1}^n\}$. Given an n-ary query Q, we define in the sequel the sorted structure of tree relations s_Q over $\Re$ by providing interpretations of the symbols in $\Re$. We denote the interpretations by indexing them with Q. For instance, S is always mapped to the schema associated with the query, so that $S_Q = \mathit{dom}(Q)$.

The relation $\mathit{Eq} \subseteq \mathcal{T}_\Sigma \times \mathcal{T}_\Sigma \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}}$ has been introduced in Section 7.2. We use it on alphabets Σ and $2^{\mathcal{V}_n}$, hence defining Eq_Σ and $\mathit{Eq}_{2^{\mathcal{V}_n}}$ respectively.

8.1.1. Type1^n

Since states of canonical NWAs need no more to be typed (the type of a tree recognized in a state may vary with the current stack content), we consider unary relations $\mathit{Type1}^n \subseteq \mathcal{T}_{2^{\mathcal{V}_n}}$ that are equal to the set of trees of $\mathcal{T}_{2^{\mathcal{V}_n}}$ of type $1^{\mathcal{V}_n}$.

Lemma 29. *A dNWA recognizing $\mathit{Type1}^n$ can be computed in time $O(3^n)$.*

Proof. Here we just have to collect variables in states at opening tags, and read only variables that have not been seen so far.

$$\mathit{stat}{=}2^{\mathcal{V}_n} \qquad \mathit{init}{=}\{\emptyset\} \qquad \mathit{fin}{=}\{\mathcal{V}_n\} \qquad \Gamma{=}\{\_\}$$

$$\frac{V, V' \subseteq \mathcal{V}_n \qquad V \cap V' = \emptyset}{V \xrightarrow{\mathtt{op}\ V' : \_} V \cup V' \in \mathit{rul}} \qquad \frac{V' \subseteq V \subseteq \mathcal{V}_n}{V \xrightarrow{\mathtt{cl}\ V' : \_} V \in \mathit{rul}}$$

This dNWA can be computed in time $O(3^n)$: For opening rules, choosing V and V' consists in determining for each variable $x \in \mathcal{V}_n$ whether $x \in V - V'$, $x \in V' - V$ or $x \notin V \cup V'$. Similarly, for closing rules, we have to choose whether $x \in V - V'$, $x \in V'$, or $x \notin V \cup V'$. □

8.1.2. Can

The next kind of tree relations express canonical languages of queries. Given a tree $t \in \mathcal{T}_\Sigma$ and a complete tuple $\tau \in \mathit{dom}(t)^n$, we define a tree $\mathit{prune}^\tau(t) \in \mathcal{T}_{2^{\mathcal{V}_n}}$ as follows. Let t' be the prefix of t with domain $\mathit{dom}_{\mathit{latest}(\tau)}(t)$. We set $\mathit{prune}^\tau(t) = \Pi_2(t' * \tau)$.

For every n-ary query Q, we define a recognizable relation $\mathit{Can}_Q \subseteq \mathcal{T}_\Sigma \times \mathcal{T}_{2^{\mathcal{V}_n}}$, which relates trees $t \in \mathcal{T}_\Sigma$ with tuples $\tau \in Q(t)$:

$$\mathit{Can}_Q = \{(t, \mathit{prune}^\tau(t)) \mid \tau \in Q(t)\}$$

Lemma 30. *Let A and B be dNWAs that define an n-ary query $Q = Q_{(A,B)}$. Then we can compute a dNWA from A in time $O(|A| \cdot 3^n)$ that recognizes Can_Q.*

Proof. The dNWA A recognizes the relation $\{(t, \Pi_2(t * \tau)) \mid \tau \in Q(t)\}$. Hence only pruning of the second component is missing. This is obtained by doing the product of A with a modified version of the dNWA recognizing $\mathit{Type1}^n$ (Lemma 29), where opening actions are forbidden after reaching state $\mathcal{V}_n$. □

8.1.3. Bef&Can

The relation $Bef = \{(t, prune^\tau(t), ren^\eta(t)) \mid \tau \in dom_\eta(t)^n\}$ is the subset of $\mathcal{T}_\Sigma \times \mathcal{T}_{2^{\mathcal{V}_n}} \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}}$ that captures all n-tuples of nodes of t (on its second component) that contain only nodes opened before an event η provided by the third component. *Bef* is recognizable by a dTA of size $O(3^n)$, so we cannot use this relation for P-time algorithms without fixing n. The problem can be circumvented by using the following relation $Bef\&Can_Q$ which can be recognized while using the states of the canonical automaton for $L(Q)$ for checking types:

$$Bef\&Can_Q = \{(t, ren^\tau(t), ren^\eta(t)) \in \mathcal{T}_\Sigma \times \mathcal{T}_{2^{\mathcal{V}_n}} \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}} \mid \tau \in Q(t) \ \wedge \ \tau \in dom_\eta(t)^n\}$$

where $ren^\tau(u) \in \mathcal{T}_{2^{\mathcal{V}_n}}$ is the projection of $u * \tau$ to $2^{\mathcal{V}_n}$, *i.e.*, $nod(ren^\tau(u)) = nod(u)$ and $lab^{ren^\tau(u)}(\pi) = V$ if $lab^u(\pi) = (a, V)$ for some $a \in \Sigma$, and all $\pi \in nod(u)$.

Lemma 31. *We can compute a d*NWA $A_{B\&C}$ *recognizing* $Bef\&Can_{Q(A,B)}$ *in time* $O(|A|)$.

Proof. We define $stat_{A_{B\&C}} = stat_A \times \mathbb{B}$, in order to control by a Boolean, whether the third component has been seen before. We define initial states by $init_{A_{B\&C}} = init_A \times \{0\}$, final states by $fin_{A_{B\&C}} = fin_A \times \{1\}$, and stack symbols by $\Gamma_{A_{B\&C}} = \Gamma_A$. Before reaching event η, we run A on $t * \tau$:

$$\frac{q_0 \xrightarrow{\alpha\ (a,V):\gamma} q_1 \in rul_A \qquad \alpha' \neq \alpha}{(q_0, 0) \xrightarrow{\alpha\ (a,V,\alpha'):\gamma} (q_1, 0) \in rul_{A_{B\&C}}}$$

When reaching η, we change the Boolean value:

$$\frac{q_0 \xrightarrow{\alpha\ (a,V):\gamma} q_1 \in rul_A}{(q_0, 0) \xrightarrow{\alpha\ (a,V,\alpha):\gamma} (q_1, 1) \in rul_{A_{B\&C}}}$$

After η, we only allow empty annotations at opening (*i.e.*, no variables):

$$\frac{q_0 \xrightarrow{\alpha\ (a,V):\gamma} q_1 \in rul_A \qquad \alpha' \neq \alpha \qquad \alpha = \mathtt{op} \Rightarrow V = \emptyset}{(q_0, 1) \xrightarrow{\alpha\ (a,V,\alpha'):\gamma} (q_1, 1) \in rul_{A_{B\&C}}}$$

□

8.1.4. Bef&NotCan

We also need a *negated* version of *Bef&Can*. The relation

$$Bef\&NotCan_Q = \{(t, ren^\tau(t), ren^\eta(u)) \in \mathcal{T}_\Sigma \times \mathcal{T}_{2^{\mathcal{V}_n}} \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}} \mid \tau \notin Q(t) \ \wedge \ \tau \in dom_\eta(t)^n \ \wedge \ dom_\eta(t) = dom_\eta(u)\}$$

is the subset of $\mathcal{T}_\Sigma \times \mathcal{T}_{2^{\mathcal{V}_n}} \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}}$ that captures all tuples τ (on its second component) that contain only nodes opened before an event η provided by the third component, such that $\tau \notin Q(t)$.

Lemma 32. *We can compute a d*NWA $A_{B\&NC}$ *recognizing* $Bef\&NotCan_{Q(A,B)}$ *in time* $O(|A|)$.

Proof. The dNWA $A_{B\&NC}$ is similar to $A_{B\&C}$, but final states are inverted, and the structure of the η-component may differ from the first two ones after η. This second feature requires to be able to stop and resume the run of A on the first two components. We use the stack to store the current state when a run of A is stopped, and add a state (and stack symbol) to go through subtrees belonging only to the η-component. The automaton $A_{B\&NC}$ also has to check that all descendants and right-siblings of $\boxdot$-labelled nodes must be labelled by $\boxdot$ on the corresponding component. The full construction is technical but not difficult, and omitted for clarity. Every step can be performed in $O(|A|)$, as only empty annotations are allowed after reading η. □

8.1.5. SameTuples

The relation *SameTuples* checks whether two tuples are equal until a given event, the first one being selected by the query:

$$\begin{aligned} \mathit{SameTuples}_Q = \{(t, \mathit{ren}^\tau(t), u_{\tau'}, \mathit{ren}^\eta(t)) \in \mathcal{T}_\Sigma \times \mathcal{T}_{2^{\mathcal{V}_n}} \times \mathcal{T}_{2^{\mathcal{V}_n}} \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}} \mid \\ \tau \in Q(t) \text{ and } \mathit{Eq}_{2^{\mathcal{V}_n}}(\mathit{ren}^\tau(t), u_{\tau'}, \mathit{ren}^\eta(t)) \\ \text{and nodes of } u_{\tau'} \text{ opened after } \eta \text{ are labeled by } \emptyset\} \end{aligned}$$

Lemma 33. *A* dNWA $A_{\mathit{SameTuples}}$ *recognizing* $\mathit{SameTuples}_{Q(A,B)}$ *can be computed in time* $O(|A|)$.

Proof. The dNWA $A_{\mathit{SameTuples}}$ recognizing *SameTuples* can be obtained by running A on $t * \tau$ and $t * \tau'$ simultaneously until η, and then allowing the τ'-component to have a different structure, labeled only with $\emptyset$. Hence the proof uses the same technique as proof of Lemma 32. □

8.1.6. Bef$_\bullet$

We define a variant of *Bef* for partial tuples, called *Bef*$_\bullet$. Here, we do not try to avoid the blow-up for two reasons. First, *Bef*$_\bullet$ will be used with *Type1*^{n}, and a blow-up is necessary to recognize *Type1*^{n}. Second, separating the relations permits to clarify the definition of the formula capturing concurrency.

The relation $\mathit{Bef}_\bullet = \{(\mathit{ren}^\tau(t), \mathit{ren}^\eta(t)) \mid \exists t \in \mathcal{T}_\Sigma.\ \tau \in (\mathit{dom}_\eta(t) \cup \{\bullet\})^n\}$ is a subset of $\mathcal{T}_{2^{\mathcal{V}_n}} \times \mathcal{T}_{\{0,\mathtt{op},\mathtt{cl}\}}$ that relates annotations of trees with tuples τ and events η, such that $\mathit{latest}(\tau) \leq \eta$.

Lemma 34. *A* dNWA *recognizing* *Bef*$_\bullet$ *can be computed in time* $O(3^n)$.

Proof. The following dNWA recognizes the relation *Bef*$_\bullet$. In the states, we collect (at opening) variables corresponding to the components of τ that have been encountered. We also add a Boolean, that indicates whether the event η has been read. Note that on the second component, we can read values different from 0 when we are not at η. For instance if $\eta = (\mathtt{op}, \pi)$, we will read "op" on the second component when we go through $(\mathtt{cl}, \pi)$.

$$\mathit{stat}{=}2^{\mathcal{V}_n} \times \mathbb{B} \qquad \mathit{init}{=}\{(\emptyset, 0)\} \qquad \mathit{fin}{=}2^{\mathcal{V}_n} \times \{1\} \qquad \Gamma{=}\{\_\}$$

Rules are defined by the following inference schemas. At opening, we check canonicity if η has not been reached; otherwise we forbid variables in the first component. When η is reached, we still allow to read variables, and change the

Boolean.

$$\frac{\alpha \in \{0, \mathtt{cl}\} \qquad V, V' \subseteq \mathcal{V}_n \qquad V \cap V' = \emptyset}{\begin{array}{c}(V,0) \xrightarrow{\mathtt{op}\ (V',\alpha):\_} (V \cup V', 0) \in \mathit{rul} \\ (V,0) \xrightarrow{\mathtt{op}\ (V',\mathtt{op}):\_} (V \cup V', 1) \in \mathit{rul} \\ (V,1) \xrightarrow{\mathtt{op}\ (\emptyset,0):\_} (V,1) \in \mathit{rul}\end{array}}$$

At closing, we do not check anything. We just change the Boolean when η is reached.

$$\frac{\delta \in \mathbb{B} \qquad \alpha \in \{0, \mathtt{op}\} \qquad V' \subseteq V \subseteq \mathcal{V}_n}{\begin{array}{c}(V,0) \xrightarrow{\mathtt{cl}\ (V',\mathtt{cl}):\_} (V,1) \in \mathit{rul} \\ (V,\delta) \xrightarrow{\mathtt{cl}\ (V',\alpha):\_} (V,\delta) \in \mathit{rul}\end{array}}$$

The complexity comes from the same argument as Lemma 29. □

8.2. Bounded Delay

Our objective is to define the formulas delay_Q and concur_Q in the logic FO$\exists[\mathit{Eq}, \mathit{Can}, S, \mathit{Bef}, \mathit{Bef\&Can}, \mathit{Bef\&NotCan}, \mathit{SameTuples}]$ preferably without using *Can* and *Bef* (to avoid a 3^n blowup). We start with the definition of the relation $\mathit{Sel}_Q = \{(t, \mathit{prune}^\tau(t), \mathit{ren}^\eta(t)) \mid (\tau, \eta) \in \mathit{sel}_Q(t)\}$ by an FO formula *Sel* with three free variables, such that $\mathit{Sel}_Q = \mathcal{R}_{\mathit{Sel}(X_t:\Sigma, X_\tau:2^{\mathcal{V}_n}, X_\eta:\{0,\mathtt{op},\mathtt{cl}\})}(s_Q)$:

$$\begin{array}{rl}\mathit{Sel} =_{df} & S(X_t) \ \wedge\ \mathit{Bef}(X_t, X_\tau, X_\eta) \\ & \wedge\ \forall X'_t{:}\Sigma.\ (S(X'_t) \ \wedge\ \mathit{Eq}_\Sigma(X_t, X'_t, X_\eta)) \ \Rightarrow\ \mathit{Can}(X'_t, X_\tau)\end{array}$$

Note that entailment of $\mathit{Can}(X'_t, X_\tau)$ is correct only since we prune trees using *Bef*: if (t', t, η) belongs to relation $\mathcal{R}_{\mathit{Eq}(X_t:\Sigma, X'_t:\Sigma, X_\eta:\{0,\mathtt{op},\mathtt{cl}\})}(s_Q)$ then t and t' may have different domains beyond η. Given DNWAs A and B defining $Q = Q_{(A,B)}$ we can thus define a DNWA recognizing $\mathit{Sel}_Q(X_t, X_\tau, X_\eta)$. Unfortunately, we cannot construct this DNWA in P-time yet, since formula *Sel* does not belong to the existential fragments of FO and uses relations *Can* and *Bef*. Nevertheless, we obtain an algorithm for deciding judgments $(\tau, \eta) \in \mathit{sel}_Q(t)$.

We define the relation $\mathit{Delay}_Q = \{(t, \mathit{ren}^\tau(t), \mathit{ren}^\eta(t)) \mid \eta \in \mathit{delay}_Q(t, \tau)\}$ by the following formula of FO$\exists[\mathit{Eq}, \mathit{Bef\&Can}, \mathit{Bef\&NotCan}, S, \mathit{SameTuples}]$, that expresses that η is an event increasing the delay if the nodes of $\tau \in Q(t)$ are before η in t, and there is a tree t' that equals t until η but with $\tau \notin Q(t')$. The formula has 3 free variables such that $\mathit{Delay}_Q = \mathcal{R}_{\mathit{Delay}(X_t:\Sigma, X_\tau:2^{\mathcal{V}_n}, X_\eta:\{0,\mathtt{op},\mathtt{cl}\})}(s_Q)$.

$$\begin{array}{rll}\mathit{Delay} & =_{df} & \exists X'_t{:}\Sigma.\ \exists X'_\tau{:}2^{\mathcal{V}_n}. \\ & & S(X_t) \ \wedge\ \mathit{Bef\&Can}(X_t, X_\tau, X_\eta) \ \wedge \\ & & S(X'_t) \ \wedge\ \mathit{Bef\&NotCan}(X'_t, X'_\tau, X_\eta) \ \wedge \\ & & \mathit{Eq}_\Sigma(X_t, X'_t, X_\eta) \ \wedge\ \mathit{SameTuples}(X_t, X_\tau, X'_\tau, X_\eta)\end{array}$$

All base relations can be defined by DNWAs computed in P-time when leaving n variable (since we do not need the relations *Can* and *Bef* here, and by Lemmas 23, 31, 32 and 33). Given deterministic automata A and B, we can thus define a possibly non-deterministic automaton recognizing $\mathit{Delay}_{Q_{(A,B)}}(X_t, X_\tau, X_\eta)$ in P-time from A and B, by Proposition 28. Let $\mathit{2Delay}_Q = \{(t \circledast u_\tau, u_\eta) \mid (t, u_\tau, u_\eta) \in \mathit{Delay}_Q\}$. Both relations are recognized by the same automaton. This relation exactly captures the delay:

$$\mathit{val}(\mathit{2Delay}_Q) = \max_{\tau \in Q(t)} \mathit{delay}_Q(t, \tau)$$

Thus we can decide bounded delay and k-bounded delay of Q for a fixed k in P-time by Theorems 7 and 8.

8.3. Bounded Concurrency

For concurrency, we proceed in a similar manner, except that we have to work with partial (incomplete) tuples. Completion raises new difficulties, since after the event η, we allow non-empty annotations on the $2^{\mathcal{V}_n}$ component, which requires a blowup wrt. n.

Proposition 35. *If arity $n \in \mathbb{N}$ is fixed, then for every n-ary query $Q = Q_{(A,B)}$ defined by dNWAs A and B, we can compute in P-time a possibly non-deterministic* NWA *that recognizes the relation $Alive_Q = \{(t, ren^\tau(t), ren^\eta(t)) \mid (\tau, \eta) \in alive_Q(t)\}$.*

Proof. We define $Alive_Q$ by a formula of $\mathrm{FO}\exists[S, Can, Eq_\Sigma, Eq_{2^{\mathcal{V}_n}}, Type1^n, Bef_\bullet]$, such that $Alive_Q = \mathcal{R}_{Alive(X_t:\mathcal{T}_\Sigma, X_\tau:\mathcal{T}_{2^{\mathcal{V}_n}}, X_\eta:\mathcal{T}_{\{0,\mathrm{op},\mathrm{c1}\}})}(s_Q)$. Here we use the relation Eq with two different alphabets: Σ and $2^{\mathcal{V}_n}$. The latter permits to express completions of tuples.

$$
\begin{aligned}
&Alive(X_t,X_\tau,X_\eta) =_{df} \exists X'_t \in \mathcal{T}_\Sigma.\ \exists X''_t \in \mathcal{T}_\Sigma.\ \exists X'_\tau \in \mathcal{T}_{2^{\mathcal{V}_n}}.\ \exists X''_\tau \in \mathcal{T}_{2^{\mathcal{V}_n}}.\\
&\quad S(X'_t)\ \wedge\ S(X''_t)\\
&\wedge\ Can_Q(X'_t, X'_\tau)\ \wedge\ Eq_\Sigma(X_t, X'_t, X_\eta) \wedge Eq_{2^{\mathcal{V}_n}}(X_\tau, X'_\tau, X_\eta)\ \wedge\ Bef_\bullet(X_\tau, X_\eta)\\
&\wedge\ \neg Can_Q(X''_t,X''_\tau)\ \wedge\ Eq_\Sigma(X_t,X''_t,X_\eta)\ \wedge\ Eq_{2^{\mathcal{V}_n}}(X_\tau,X''_\tau,X_\eta)\ \wedge\ Type1^n(X''_\tau)
\end{aligned}
$$

This formula expresses that τ is alive at η of $t \in \mathcal{T}_\Sigma$ if there exists continuations $t', t'' \in \mathcal{T}_\Sigma$ of t beyond η and two completions τ', τ'' of τ beyond η such that $\tau' \in Q(t')$ but $\tau'' \notin Q(t'')$. $Bef_\bullet$ checks whether $latest(\tau) \leq \eta$. $Type1^n$ verifies that X''_τ is canonical, as this is not done by $\neg Can_Q(X''_t,X''_\tau)$. All relations used in the formula are recognizable by dNWAs that can be computed in P-time by Lemmas 23, 30, 34 and 29, so that an NWA for $Alive_Q$ is obtained in P-time from Proposition 28 (since A is deterministic). Indeed, this result remains true if B is non-deterministic, since relation symbol S does not occur below negation. □

Note that we cannot integrate the canonicity control for X''_t into the negated relation $\neg Can(X''_t,X''_\tau)$. The deeper problem is that automata A for canonical languages of queries $Q_{(A,B)}$ do not have a notion of safe states *in the case of trees*, since safety depends also on the current stack content.

Let $2Alive_Q$ be the binary version of $Alive_Q$, *i.e.*, $2Alive_Q = \{(t \circledast u_\eta, u_\tau) \mid (t, u_\eta, u_\tau) \in Alive_Q\}$, then:

$$val(2Alive_Q) = \max_{t \in dom(Q), \eta \in eve(t)} concur_Q(t, \eta)$$

Since we can define relation $2Alive_Q$ by automata that we can compute in P-time for fixed n from A and B by Proposition 28 we can decide bounded and k-bounded concurrency of Q for fixed n and k in P-time by Theorems 7 and 8.

Conclusion.

In this paper we proved that bounded delay and concurrency are decidable in P-time for queries in words defined by dFAs and for queries in unranked trees defined by dNWAs. We obtained analogous results for k-bounded delay and k-bounded concurrency for fixed k.

We presented a direct P-time algorithm that computes the delay of queries in words, and P-time reductions of various boundedness problems for queries in words to ambiguity of nFAs, which are know to be decidable in P-time. In the case of trees, we presented P-time reductions for the boundedness problems of queries to bounded valuedness of recognizable relations between trees. We then show how to decide k-boundedness of recognizable relations for fixed k by reduction to emptiness of tree automata, and that k-boundedness for flexible k becomes EXPTIME-hard. We proved that bounded valuedness of recognizable relations is decidable in P-time by reduction to bounded valuedness of tree transducers [52].

In follow-up work, we have proposed streaming models based on the notion of bounded concurrency and obtained coNP-hardness results for boundedness problems for fragments of XPath, and distinguished a fragment of Forward XPath with bounded concurrency [29]. An open question is how to obtain a direct algorithm for deciding bounded valuedness of recognizable relations without applying results on tree transducers. This could help to lower the polynomials in our complexity results. In future work, we would like to study lower bounds based on concurrency in sufficient generality.

Acknowledgments.

We thank the reviewers for their helpful comments on the revision of this article. This work was partially supported by the Enumeration project ANR-07-blanc and the project CODEX of ANR-08-Defis.

Bibliography

[1] Cyril Allauzen, Mehryar Mohri, and Ashish Rastogi. General algorithms for testing the ambiguity of finite automata. In *Developments in Language Theory, 12th International Conference*, volume 5257 of *Lecture Notes in Computer Science*, pages 108–120. Springer Verlag, 2008.

[2] Rajeev Alur. Marrying words and trees. In *26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–242. ACM-Press, 2007.

[3] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *36th ACM Symposium on Theory of Computing*, pages 202–211. ACM-Press, 2004.

[4] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):1–43, 2009.

[5] Ziv Bar-Yossef, Marcus Fontoura, and Vanja Josifovski. Buffering in query evaluation over XML streams. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 216–227. ACM-Press, 2005.

[6] Ziv Bar-Yossef, Marcus Fontoura, and Vanja Josifovski. On the memory requirements of XPath evaluation over XML streams. *Journal of Computer and System Science*, 73(3):391–441, 2007.

[7] Charles Barton, Philippe Charles, Deepak Goyal, Mukund Raghavachari, Marcus Fontoura, and Vanja Josifovski. Streaming XPath Processing with Forward and Backward Axes. In *19th International Conference on Data Engineering*, pages 455–466, 2003.

[8] Michael Benedikt and Alan Jeffrey. Efficient and expressive tree filters. In *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 461–472. Springer Verlag, 2007.

[9] Michael Benedikt, Alan Jeffrey, and Ruy Ley-Wild. Stream Firewalling of XML Constraints. In *ACM SIGMOD International Conference on Management of Data*, pages 487–498. ACM-Press, 2008.

[10] Michael Benedikt and Leonid Libkin. Tree extension algebras: Logics, automata, and query languages. In *Proceeding of the 9^{th} Logic in Computer Science Conference*, pages 203–214. IEEE Comp. Soc. Press, 2002.

[11] Michael Benedikt, Leonid Libkin, and Frank Neven. Logical definability and query languages over ranked and unranked trees. *ACM Transactions on Computational Logics*, 8(2), April 2007.

[12] Alexandru Berlea. Online evaluation of regular tree queries. *Nordic Journal of Computing*, 13(4):1–26, 2006.

[13] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml) 1.0 (fourth edition), November 2006. `http://www.w3.org/TR/2006/REC-xml-20060816`.

[14] Julien Carme, Aurélien Lemay, and Joachim Niehren. Learning node selecting tree transducer from completely annotated examples. In *7th International Colloquium on Grammatical Inference*, volume 3264 of *Lecture Notes in Artificial Intelligence*, pages 91–102. Springer Verlag, 2004.

[15] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 105–118. Springer Verlag, 2004.

[16] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Trans. on Know. Data Eng.*, 1(1):146–166, March 1989.

[17] Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Efficient inclusion checking for deterministic tree automata and XML schemas. *Information and Computation*, 207(11):1181–1208, november 2009. doi:10.1016/j.ic.2009.03.003.

[18] Yi Chen, Susan B. Davidson, and Yifeng Zheng. An Efficient XPath Query Processor for XML Streams. In *22nd International Conference on Data Engineering*, page 79. IEEE Computer Society, 2006.

[19] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available online since 1997: `http://tata.gforge.inria.fr`, October 2007. Revised October, 12th 2007.

[20] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM computing surveys*, 33(3):374–425, September 2001.

[21] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *IEEE Conference on Logic in Computer Science*, pages 242–248, 1990.

[22] Mary Fernandez, Philippe Michiels, Jérôme Siméon, and Michael Stark. XQuery streaming à la carte. In *23nd International Conference on Data Engineering*, pages 256–265, 2007.

[23] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Properties of visibly pushdown transducers. In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*. Springer Verlag, 2010.

[24] Markus Frick, Martin Grohe, and Christoph Koch. Query evaluation on compressed trees. In *18th IEEE Symposium on Logic in Computer Science*, pages 188–197, 2003.

[25] Olivier Gauwin. *Streaming Tree Automata and XPath*. PhD thesis, Université Lille 1, 2009.

[26] Olivier Gauwin and Joachim Niehren. Streamable fragments of Forward XPath. 2010.

[27] Olivier Gauwin, Joachim Niehren, and Yves Roos. Streaming tree automata. *Information Processing Letters*, 109(1):13–17, December 2008.

[28] Olivier Gauwin, Joachim Niehren, and Sophie Tison. Bounded delay and concurrency for earliest query answering. In *3rd International Conference on Language and Automata Theory and Applications*, volume 5457 of *Lecture Notes in Computer Science*, pages 350–361. Springer Verlag, 2009.

[29] Olivier Gauwin, Joachim Niehren, and Sophie Tison. Earliest query answering for deterministic nested word automata. In *17th International Symposium on Fundamentals of Computer Theory*, volume 5699 of *Lecture Notes in Computer Science*, pages 121–132. Springer Verlag, 2009.

[30] Georg Gottlob, Erich Gr"adel, and Helmut Veith. Datalog LITE: a Deductive Query Language with Linear Time Model Checking. *ACM Transactions on Computational Logics*, 3(1):42–79, January 2002.

[31] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

[32] Gang Gou and Rada Chirkova. Efficient algorithms for evaluating XPath over streams. In *36th ACM SIGMOD International Conference on Management of Data*, pages 269–280. ACM-Press, 2007.

[33] Ashish Kumar Gupta and Dan Suciu. Stream processing of XPath queries with predicates. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 419–430. ACM-Press, 2003.

[34] Vanja Josifovski, Marcus Fontoura, and Attila Barta. Querying XML streams. *The VLDB Journal*, 14(2):197–210, 2005.

[35] Christoph Koch. Efficient processing of expressive node-selecting queries on xml data in secondary storage: A tree automata-based approach. In *Proc. VLDB 2003*, 2003.

[36] Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML schema. *ACM Transactions of Database Systems*, 31(3):770–813, 2006.

[37] Wim Martens and Joachim Niehren. On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Science*, 73(4):550–583, 2007.

[38] M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.

[39] Andreas Neumann and Helmut Seidl. Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science*, pages 134–145, 1998.

[40] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison. N-ary queries by tree automata. In *10th International Symposium on Database Programming Languages*, volume 3774 of *Lecture Notes in Computer Science*, pages 217–231. Springer Verlag, September 2005.

[41] Abdul Nizar and Sreenivasa Kumar. Efficient Evaluation of Forward XPath Axes over XML Streams. In *14th International Conference on Management of Data (COMAD)*, pages 222–233, 2008.

[42] Dan Olteanu. SPEX: Streamed and progressive evaluation of XPath. *IEEE Trans. on Know. Data Eng.*, 19(7):934–949, 2007.

[43] Pawel Parys. XPath evaluation in linear time with polynomial combined complexity. In *28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 55–64. ACM-Press, 2009.

[44] Feng Peng and Sudarshan S. Chawathe. XSQ: A streaming XPath engine. *ACM Transactions on Database Systems*, 30(2):577–623, 2005.

[45] M.O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[46] Prakash Ramanan. Worst-case optimal algorithm for XPath evaluation over XML streams. *Journal of Computer and System Science*, 75:465–485, 2009.

[47] Jean-François Raskin and Frédéric Servais. Visibly pushdown transducers. In *Automata, Languages and Programming, 35th International Colloquium*, volume 5126 of *Lecture Notes in Computer Science*, pages 386–397. Springer Verlag, 2008.

[48] Jacques Sakarovitch and Rodrigo de Souza. Decidability of bounded valuedness for transducers. In *Mathematical Foundations of Computer Science*, 2008.

[49] Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 53–64, 2002.

[50] Helmut Seidl. Deciding equivalence of finite tree automata. In *Annual Symposium on Theoretical Aspects of Computer Science*, volume 349 of *Lecture Notes in Computer Science*, pages 480–492. Springer Verlag, 1989.

[51] Helmut Seidl. On the finite degree of ambiguity of finite tree automata. *Acta Informatica*, 26(6):527–542, 1989.

[52] Helmut Seidl. Single-valuedness of tree transducers is decidable in polynomial time. *Theoretical Computer Science*, 106:135–181, 1992.

[53] Helmut Seidl. Equivalence of finite-valued tree transducers is decidable. *Mathematical System Theory*, 27:285–346, 1994.

[54] Helmut Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.

[55] R. E. Stearns and H. B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985.

[56] S. Tison. *Automates Comme Outils de Décision dans les Arbres.* Thèse d'habilitation, Laboratoire d'Informatique Fondamentale de Lille, 1990.

[57] Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. In *Mathematical Foundations of Computer Science*, volume 233 of *Lecture Notes in Computer Science*, pages 620–629. Springer Verlag, 1986.

[58] Xiaoying Wu and Dimitri Theodoratos. Evaluating Partial Tree-Pattern Queries on XML Streams. In *17th ACM International Conference on Information and Knowledge Management (CIKM'08)*, pages 1409–1410. ACM Press, 2008.

Appendix A. Ground Datalog

We recall some folklore definitions and results about ground Datalog, frequently used in this paper. For more details, we invite the reader to refer to the following papers [16, 20, 30].

Let Λ be a ranked signature containing constants $c \in \Lambda$ and predicates $p \in \Lambda$, where all predicates have an arity $ar(p) \in \mathbb{N}_0$. We call a term $p(c_1, \ldots, c_{ar(p)})$ a *literal*, and denote the set of literals over Λ by $lit(\Lambda)$. A *clause* is a pair in $lit(\Lambda) \times lit(\Lambda)^k$ (with $k \in \mathbb{N}_0$) that we write L :- $L_1, \ldots, L_k$. as usual. A *ground Datalog program* P is a finite set of clauses over Λ. Its size $|P|$ is the total number of symbols appearing in all its clauses.

The *least fixed point* $lfp(P)$ of P is the least set of literals over Λ that satisfies that for all clauses L :- $L_1, \ldots, L_k$. of P, if $L_1, \ldots, L_k \in lfp(P)$ then $L \in lfp(P)$. As no negation is allowed, every ground Datalog program P has a unique least fixed point, and this one is finite. For ground Datalog, this least fixed point can be efficiently computed.

Proposition 36. *For every signature Λ and every ground Datalog program P over Λ, the least fixed point of P can be computed in time $O(|P|)$.*

Proof. A program P can be seen as an hypergraph: vertices are literals, and edges are tuples $(L, L_1, \ldots, L_k)$ such that L :- $L_1, \ldots, L_k$. is a clause of P. Then the least fixed point of P is exactly the set of accessible literals in this hypergraph. This can be computed in linear time, modulo the ability of testing whether $L = L'$ in time $O(1)$. This is trivial if all literals are constants, but we have to prove it for arbitrary signatures.

We can do this by assigning a number to each literal in P. This is done in $O(|P|)$ by parsing P and using a prefix tree that stores the assigned numbers of all encountered literals. For instance the prefix tree $\#(p_2(c_2(c_1(1), c_2(2))))$ (where $\#$ is just an extra symbol) indicates that $p_2(c_2, c_1)$ is assigned to 1, and $p_2(c_2, c_2)$ to 2. □

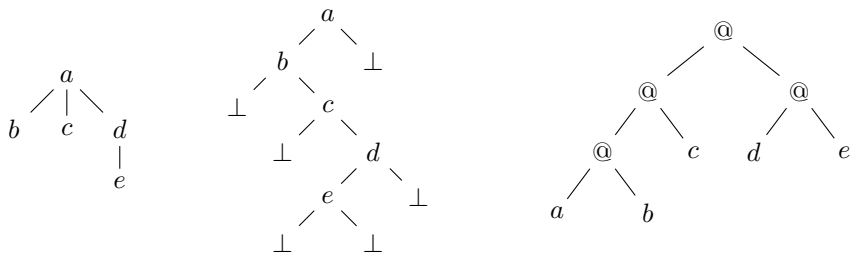
Appendix B. NWAs versus Standard Tree Automata

In this section, we relate NWAs on unranked trees to standard tree automata on ranked trees with respect to expressiveness. We also show that the notion of determinism of dNWAs generalizes over both bottom-up and top-deterministism of standard tree automata in a succinct manner. This implies that Theorem 6 does equally apply to queries defined by top-down deterministic tree automata on top-down encodings of unranked trees (*fcns*), or by bottom-up deterministic tree automata on bottom-up encodings (*curry*).

Appendix B.1. Binary Encodings

The top-down encoding is defined by a function $fcns : \mathcal{T}_\Sigma \to \mathcal{T}^{bin}_{\Sigma_\perp}$ and is illustrated in Figure 10(b). Its definition is based on an encoding of hedges, $fcns' : \cup_{n \geq 0} (\mathcal{T}_\Sigma)^n \to \mathcal{T}^{bin}_{\Sigma_\perp}$, *i.e.*, sequences of unranked trees (where $(t) \neq t$).

$$fcns(t) = fcns'((t))$$
$$fcns'((a(s_1, \ldots, s_l), t_2, \ldots, t_k)) = a(fcns'((s_1, \ldots, s_l)), fcns'((t_2, \ldots, t_k)))$$
$$fcns'(()) = \perp$$



(a) A tree $t \in \mathcal{T}_\Sigma$ (b) $fcns(t)$ (c) $curry(t)$

Figure B.10: Binary encodings

Every nTA A over $\Sigma_\perp$ defines a language of unranked trees modulo the *fcns* encoding $L(A) = \{t \in \mathcal{T}_\Sigma \mid fcns(t) \in L^{bin}(A)\}$. Deterministic DTDs can be recognized by top-down deterministic nTAs this way (see [17] for precise translations).

The bottom-up encoding is based on currying. Let $\Sigma_@ = \Sigma \uplus \{@\}$ be the ranked alphabet in which all symbols from Σ are constants, and @ is the only binary symbol. Function $curry : \mathcal{T}_\Sigma \to \mathcal{T}^{bin}_{\Sigma_@}$ maps unranked tree over Σ to binary trees over the ranked signature $\Sigma_@$:

$$curry(a(t_1, \ldots, t_k)) = \begin{cases} a & \text{if } k = 0 \\ @(curry(a(t_1, \ldots, t_{k-1})), curry(t_k)) & \text{otherwise} \end{cases}$$

A stepwise tree automaton [15] is a nTA over $\Sigma_@$. Every stepwise tree automaton B defines a language of unranked trees modulo currying $L(B) = \{t \in \mathcal{T}_\Sigma \mid curry(t) \in L^{bin}(B)\}$. The notions of bottom-up determinism of nTAs on binary trees induce a notion of bottom-up and left-to-right determinism for stepwise tree automata on unranked trees (see [37] for discussion).

Appendix B.2. Relation to NWAs

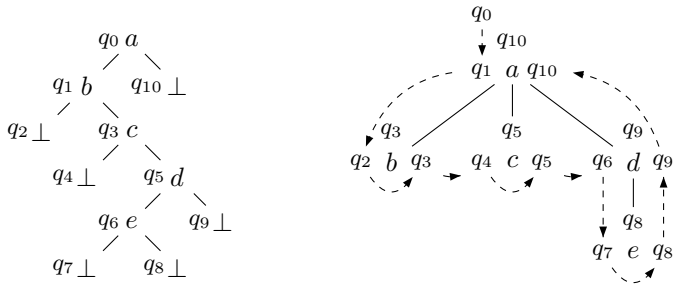
Furthermore, deterministic stepwise tree automata [15] can be converted in dNWAs in P-time.

Lemma 37. *Every nTA over $\Sigma_\perp$ can be transformed to a NWA in P-time, such that the language of unranked trees is preserved modulo the fcns-encoding. This transformation preserves top-down determinism, in that it maps $d^\downarrow$nTAs to d*NWA*s.*

Proof. Let A be a nTA recognizing binary *fcns* encodings in $\mathcal{T}_{\Sigma_\perp}$. We define an NWA A' over Σ such that $L(A) = L(A')$. This is illustrated by Figure B.11, with runs of A on $fcns(t)$ and A' on t.

$$\begin{array}{r} stat_{A'} = stat_A \\ init_{A'} = fin_A \\ fin_{A'} = stat_A \end{array} \qquad \frac{a(q_1, q_2) \to q \in rul_A}{q \xrightarrow{\texttt{op}\ a:q_2} q_1 \in rul_{A'}} \qquad \frac{\perp \to q_1 \in rul_A \quad a \in \Sigma \quad q_2 \in stat_A}{q_1 \xrightarrow{\texttt{cl}\ a:q_2} q_2 \in rul_{A'}}$$

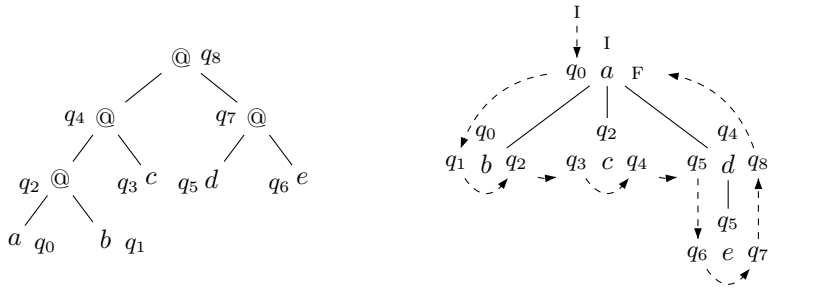
This transformation maps d$^\downarrow$nTAs to dNWAs. Its correctness can be proved by showing the following invariant: If $h = (t_1, \ldots, t_k)$ is a hedge over Σ, then there is a run r of A on $fcns'(h)$ iff there is a run r' of A' on h, and if such runs exist, then, if π' is the root of t_1 and π the corresponding node in $fcns'(h)$ we have: $r'_e((\texttt{op}, \pi')) = r(\pi{\cdot}1)$ and $r'_n(\pi') = r(\pi{\cdot}2)$. □



(a) A run of a NTA A on $t \in \mathcal{T}_{\Sigma_\bot}$.

(b) A run of the corresponding NWA A' on $fcns^{-1}(t)$.

Figure B.11: Example of runs for the translation of NTAs modulo *fcns* encoding to NWAs.



(a) A run of a NTA A on $t \in \mathcal{T}_{\Sigma_@}$.

(b) A run of the corresponding NWA A' on $curry^{-1}(t)$.

Figure B.12: Example runs for the translation of NTAs for *curry* encoding to NWAs.

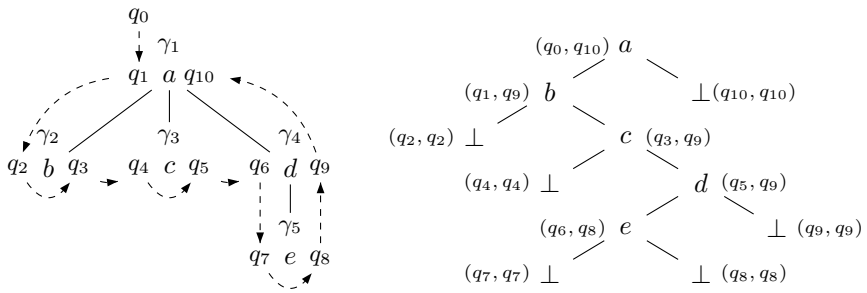
Lemma 38. *Stepwise tree automata can be transformed in P-time to* NWA*s with the same language of unranked trees modulo currying. This transformation preserves bottom-up determinism, i.e. it maps d*TA*s to d*NWA*s.*

Proof. Modulo currying, stepwise tree automata can be seen as a weaker form of NWAs: a stepwise tree automaton evaluates hedges from left to right. The difference with NWAs is that when evaluating a new tree of the hedge, the state resulting from the evaluation of the beginning of the hedge is unknown. The translation of a NTA A to an NWA A' is detailed and proved below, and illustrated by Figure B.12. The key idea here is to translate an @-rule by a closing rule, that uses the stack to know how the hedge of preceding siblings of the current node was evaluated, and the current state to know what is the state for the subtree rooted at the current node. Labels are only used at opening.

$$stat_{A'} = stat_A \uplus \{\mathrm{I}, \mathrm{F}\} \qquad init_{A'} = \{\mathrm{I}\} \qquad fin_{A'} = \{\mathrm{F}\}$$

$$\frac{@(q_0, q_1) \to q_2 \in rul_A}{q_1 \xrightarrow{\texttt{cl}\ a:q_0} q_2 \in rul_{A'}} \qquad \frac{a \to q_1 \in rul_A \qquad q_0 \in stat_A \cup \{\mathrm{I}\}}{q_0 \xrightarrow{\texttt{op}\ a:q_0} q_1 \in rul_{A'}} \qquad \frac{q \in fin_A \qquad a \in \Sigma}{q \xrightarrow{\texttt{cl}\ a:\mathrm{I}} \mathrm{F}}$$

Correctness relies on the following property, that can be proved by induction on the structure of $t \in \mathcal{T}_{\Sigma_@}$: a run r of A on t exists iff there is a run r' of A' on $curry^{-1}(t)$, and $r(\epsilon) = r'((\texttt{cl}, k))$ if the root of $curry^{-1}(t)$ has k children, and $r'((\texttt{op}, \epsilon)) = r(\pi_\epsilon)$ where π_ϵ is the first leaf of t in pre-order. □



(a) A run of the NWA A on $t \in \mathcal{T}_\Sigma$. (b) A run of the corresponding nTA A on $fcns(t)$.

Figure B.13: Example runs for the translation of NWAs to nTAs wrt. *fcns* encoding.

The next proposition states that all three classes of automata have the same expressiveness. They capture monadic second-order definable languages of unranked trees (see e.g. [15] for stepwise tree automata).

Proposition 39 (Same expressiveness)**.** *The three classes of tree automata over unranked trees that we consider (n*TA*s wrt. fcns and curry encodings, and* NWA*s) permit determinization and recognize the same languages of unranked trees modulo P-time automata translations (not always preserving determinism).*

Proof. For standard tree automata, determinization is standard. For NWAs, it is well-known too; it can be obtained from determinization of stepwise tree automata and the translations discussed here. We have already shown how to convert nTAs modulo both encoding to NWAs (Lemmas 37 and 38). It remains to provide inverse encodings.

NWA*s to n*TA*s wrt. fcns encoding.* Let A be an NWA over the alphabet Σ. We define the nTA A' over $\Sigma_\bot$ such that $L(A') = L(A)$:

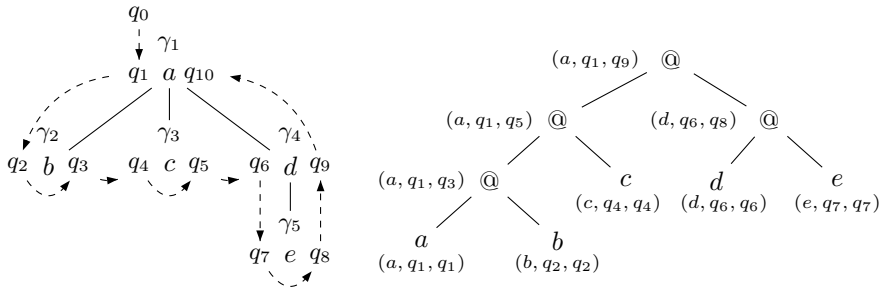
$$stat_{A'} = stat_A \times stat_A \qquad fin_{A'} = init_A \times fin_A$$

$$\frac{q_0 \xrightarrow{\texttt{op}\ a:\gamma} q_1 \in rul_A \qquad q_2 \xrightarrow{\texttt{cl}\ a:\gamma} q_3 \in rul_A \qquad q_4 \in stat_A}{a((q_1,q_2),(q_3,q_4)) \to (q_0,q_4) \in rul_{A'}} \qquad \frac{q \in stat_A}{\bot \to (q,q) \in rul_{A'}}$$

Figure B.13 illustrates this translation. The following property is easy to prove by induction on the structure of t, and gives the main idea of the construction: There is a run r' of A' on t iff there is a run r of A on the hedge $fcns^{-1}(t)$, and if such runs exist then $r'(\epsilon) = (q_0, q_1)$ iff there is a run of A on $fcns^{-1}(t)$ starting in q_0 and ending in q_1.

NWA*s to n*TA*s over curry encoding.* We exhibit a translation from an NWA A to a nTA recognizing the language of corresponding *curry* encodings of trees. This time the translation is more intricate, as NWAs allow to send the current state from one node to its right sibling, but nTAs over *curry* encoding do not. This is why we have to guess this state, and then to check whether this guess corresponds to the state reached when closing the previous sibling. The construction is shown above and illustrated by Figure B.14.

$$stat_{A'} = \Sigma \times stat_A \times stat_A \qquad \frac{q_0 \xrightarrow{\texttt{op}\ a:\gamma} q_1 \in rul_A}{a \to (a, q_1, q_1) \in rul_{A'}}$$



(a) A run of the NWA A on $t \in \mathcal{T}_\Sigma$. (b) A run of the corresponding NTA A' on $curry(t)$.

Figure B.14: Example runs for the translation of NWAs to NTAs modulo *curry* encoding.

$$\frac{q_0 \xrightarrow{\texttt{op } a:\gamma} q_1 \in rul_A \qquad q_2 \xrightarrow{\texttt{cl } a:\gamma} q_3 \in rul_A \qquad q_0 \in init_A \qquad q_3 \in fin_A}{(a, q_1, q_2) \in fin_{A'}}$$

$$\frac{q_0 \xrightarrow{\texttt{op } b:\gamma} q_1 \in rul_A \qquad q_2 \xrightarrow{\texttt{cl } b:\gamma} q_3 \in rul_A \qquad q_4, q_5 \in stat_A \qquad a \in \Sigma}{@((a, q_4, q_0), (b, q_1, q_2)) \to (a, q_4, q_9) \in rul_{A'}}$$

The following invariant can be proved inductively on the structure of $t \in \mathcal{T}_{\Sigma_@}$: There is a run r' of A' on t such that $r'(\epsilon) = (a, q_0, q_1)$ iff the root of $curry^{-1}(t)$ is labeled by a, there is a run r of A on $curry^{-1}(t)$ such that $r((\texttt{op}, \epsilon)) = q_0$ and $r((\texttt{cl}, k)) = q_1$ where k is the last child of the root. □

Appendix C. Recognizable Relations between Unranked Trees

Lemma 40. *$Relab_R$ and R have the same valuedness.*

Proof. If $ex_i(t, t')$ holds for $(t, t') \in \mathcal{T}_{\Sigma^i} \times \mathcal{T}_{\Sigma^i_\boxdot}$, then we write $clean_i(t') = t$, which is well-defined as t is unique for a given t'. It is easy to check that:

- if $u \in \mathcal{T}_{\Sigma^1_\boxdot \times \Sigma^2_\boxdot}$ then $u \in ovl(Relab_R)$ iff $(clean_1(\Pi_1(u)), clean_2(\Pi_2(u))) \in R$
- $(t_1, t_2) \in Relab_R$ iff $(clean_1(t_1), clean_2(t_2)) \in R$ and $nod(t_1) = nod(t_2)$.

First, let us prove that the valuedness of $Relab_R$ is at least the valuedness of R. Let t in $\mathcal{T}_{\Sigma^1}$ such that there exists at least k distinct t_i with $(t, t_i) \in R$. Let $D = nod(t) \cup \cup_{i=1}^k nod(t_i)$. For a tree u and a set of nodes D such that $nod(t) \subseteq D$, we define the completion of u w.r.t. D as the tree u^D defined by $nod(u^D) = D$ and $lab^{u^D}(\pi) = lab^u(\pi)$ if p belongs to $nod(u)$, $lab^{u^D}(\pi) = \boxdot$ otherwise. As $nod(t^D) = nod(t_i^D)$ and $clean_1(t^D) = t$, $clean_2(t_i^D) = t_i$, we have $(t^D, t_i^D) \in Relab_R$, $1 \leq i \leq n$. As the t_i, $1 \leq i \leq n$, are distinct, so are the t_i^D, $1 \leq i \leq n$: the valuedness of $Relab_R$ is at least the valuedness of R.

Now, let us prove that the valuedness of $Relab_R$ is at most the valuedness of R. Let u in $\mathcal{T}_{\Sigma^1_\boxdot}$ such that there exists at least k distinct v_i with $Relab_R(u, v_i)$. Let $t = clean_1(u), t_i = clean_2(v_i)$: we have $(t, t_i) \in R$. It remains to prove that the t_i are all distinct.

Let $1 \leq i < j \leq n$: as $v_i \neq v_j$ there exists a position π such that $lab^{v_i}(\pi) \neq lab^{v_j}(\pi)$:

- either $lab^{v_i}(\pi) \neq \boxdot$ and $lab^{v_j}(\pi) \neq \boxdot$: then π belongs to $nod(t_i)$ and to $nod(t_j)$ and $lab^{t_i}(\pi) \neq lab^{t_j}(\pi)$.
- either $lab^{v_i}(\pi) \neq \boxdot$ and $lab^{v_j}(\pi) = \boxdot$: then π belongs to $nod(t_i)$ and π does not belong to $nod(t_j)$.
- either $lab^{v_j}(\pi) \neq \boxdot$ and $lab^{v_i}(\pi) = \boxdot$: similar to the precedent case.

So, there exists $t \in \mathcal{T}_{\Sigma^1}$ such that there exists at least k distinct t_i with $(t, t_i) \in R$. □

Proposition 28. *Let ϕ be a fixed formula in $\text{FO}_\exists[\Re]$ with at most m free sorted variables $X_1{:}\Sigma_1, \ldots, X_m{:}\Sigma_m$ where $X_i \in \mathcal{V}$ and $\Sigma_i \in \Omega$. Then there exists a polynomial p such that for all structures of recognizable relations $s = \{A_r\}_{r\in\Re}$ defined by tree automata such that A_r is deterministic if r occurs in the scope of a negation in ϕ, one can compute in time $p(\sum_{r\in\Re} |A_r|)$ an automaton that recognizes the relation $\mathcal{R}_{\phi(X_1:\Sigma_1,\ldots,X_m:\Sigma_m)}(s)$. The computed automaton is deterministic, if all automata are deterministic and ϕ is free of existential quantifiers.*

Proof. The proof follows from two claims, that relate operations on tree relations to operations on tree languages to closure properties of tree automata.

Claim 41. *For all $Q \subseteq \mathcal{T}_{\Sigma_1} \times \ldots \times \mathcal{T}_{\Sigma_m}$, $\mathcal{V}_m = \{X_1, \ldots, X_m\}$ and $\theta : \{1, \ldots, m\} \to \{1, \ldots, m\}$ with $\{1, \ldots, n\} \subseteq \theta(\{1, \ldots, m\})$:*

$$ovl(\exists X_i.Q) = \Pi_{\{1,\ldots,i-1,i+1,\ldots,m\}}(ovl(Q)) \quad ovl(c_\theta Q) = c_\theta\, ovl(Q)$$
$$ovl(\neg Q) = ovl(\mathcal{T}_{\Sigma_1} \times \ldots \times \mathcal{T}_{\Sigma_m}) - ovl(Q) \quad ovl(Q_1 \wedge Q_2) = ovl(Q_1) \cap ovl(Q_2)$$

Proof of Claim 41. The proof is straightforward from the definitions. The next claim relates connectives of sorted FO formulas to operations on tree relations. □

Claim 42. *For all alphabets $\tilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_m)$ and Σ_{m+1}, variables $\tilde{X} = (X_1, \ldots, X_m)$ and X_{m+1} that are pairwise distinct, structures s of tree relations, functions $\theta : \{1, \ldots, m\} \to \{1, \ldots, m\}$ with $\{1, \ldots, n\} \subseteq \theta(\{1, \ldots, m\})$, sorted formulas ϕ, ϕ_1, ϕ_2 in $\text{FO}[\Re]$, and relations symbols $r \in \Re$:*

$$ovl(\mathcal{R}_{\exists X_{m+1}:\Sigma_{m+1}.\phi(\tilde{X}:\tilde{\Sigma})}(s)) = \Pi_{\{1,\ldots,m\}}(ovl(\mathcal{R}_{\phi(\tilde{X}:\tilde{\Sigma},X_{m+1}:\Sigma_{m+1})}(s)))$$
$$ovl(\mathcal{R}_{r(\tilde{X})(X_{\theta(1)}:\Sigma_{\theta(1)},\ldots,X_{\theta(m)}:\Sigma_{\theta(m)})}(s)) = ovl(\mathcal{T}_{\Sigma_{\theta(1)}} \times \ldots \times \mathcal{T}_{\Sigma_{\theta(m)}}) \cap c_\theta\, ovl(r^s)$$
$$ovl(\mathcal{R}_{\phi_1\wedge\phi_2(\tilde{X}:\tilde{\Sigma})}(s)) = ovl(\mathcal{R}_{\phi_1(\tilde{X}:\tilde{\Sigma})}(s)) \cap ovl(\mathcal{R}_{\phi_2(\tilde{X}:\tilde{\Sigma})}(s))$$
$$ovl(\mathcal{R}_{\neg\phi(\tilde{X}:\tilde{\Sigma})}(s)) = ovl(\mathcal{T}_{\Sigma_1} \times \ldots \times \mathcal{T}_{\Sigma_m}) - ovl(\mathcal{R}_{\phi(\tilde{X}:\tilde{\Sigma})}(s))$$

Proof of Claim 42. The proof is straightforward from the definitions and the previous claim. For illustration, we elaborate the case of negation, where the sorting information is needed. Let $L_{\tilde{\Sigma}} = ovl(\mathcal{T}_{\Sigma_1} \times \ldots \times \mathcal{T}_{\Sigma_m})$.

$$\begin{aligned} ovl(\mathcal{R}_{\neg\phi(\tilde{X}:\tilde{\Sigma})}(s)) &= L_{\tilde{\Sigma}} \cap ovl(Q_{\neg\phi(\tilde{X}:\tilde{\Sigma})}(s)) \\ &= L_{\tilde{\Sigma}} \cap (L_{\tilde{\Sigma}} - ovl(Q_{\phi(\tilde{X}:\tilde{\Sigma})}(s))) \quad \text{(previous claim)} \\ &= L_{\tilde{\Sigma}} - ovl(\mathcal{R}_{\phi(\tilde{X}:\tilde{\Sigma})}(s)) \end{aligned}$$

□

Finally, we illustrate the induction proving Proposition 28 for formula $\phi = \neg\phi'$. Since $\phi \in \text{FO}_\exists[\Re]$, formula ϕ' cannot contain existential quantifiers. Furthermore, all automata A_r for relations symbols occurring in ϕ must be deterministic by assumption. By induction hypothesis, there exists a polynomial p' such that for all structures $s = \{A_r\}_{r\in\Re}$ defined automata automata A_r, one can compute in time $p(\sum_{r\in\Re} |A_r|)$ a deterministic automaton A' recognizing the language $ovl(\mathcal{R}_{\phi'(\tilde{X}:\tilde{\Sigma})}(s))$. Recall that $ovl(\mathcal{R}_{\phi(\tilde{X}:\tilde{\Sigma})}(s))$ is equal to $ovl(\mathcal{T}_{\Sigma_1} \times \ldots \times \mathcal{T}_{\Sigma_m}) - ovl(\mathcal{R}_{\phi'(\tilde{X}:\tilde{\Sigma})}(s))$ as shown by the previous claim. We obtain an automaton A recognizing this language by complementing A' and intersecting it with an automaton for $ovl(\mathcal{T}_{\Sigma_1} \times \ldots \times \mathcal{T}_{\Sigma_m})$. This can be done in time $p_1(|A'|)\cdot|\Sigma_1|\cdot\ldots\cdot|\Sigma_m|$ for some polynomial p_1, since A' was deterministic. Furthermore, automaton A can be constructed deterministically from A'. We can thus define polynomial p by $p(\xi) = p_1(p'(\xi))\cdot|\Sigma_1|\cdot\ldots\cdot|\Sigma_m|$.

The only construction, where non-determinism is needed are projections. This is why we require existential quantifiers to appear only in prenex position. Note that the proposition can be extended to general FO formulas, but not in P-time. □