Distributed XML Design

Serge Abiteboul
INRIA Saclay – Île-de-France
& University Paris Sud, FR
serge.<lastname>@inria.fr

Georg Gottlob*
University of Oxford, UK
georg.gottlob@
comlab.ox.ac.uk

Marco Manna
Department of Mathematics,
University of Calabria, IT
<lastname>@mat.unical.it

ABSTRACT

A *distributed XML document* is an XML document that spans several machines or Web repositories. We assume that a distribution design of the document tree is given, providing an XML tree some of whose leaves are "docking points", to which XML subtrees can be attached. These subtrees may be provided and controlled by peers at remote locations, or may correspond to the result of function calls, e.g., Web services. If a global type τ, e.g. a DTD, is specified for a distributed document T, it would be most desirable to be able to break this type into a collection of local types, called a local typing, such that the document satisfies τ if and only if each peer (or function) satisfies its local type. In this paper we lay out the fundamentals of a theory of local typing and provide formal definitions of three main variants of locality: local typing, maximal local typing, and perfect typing, the latter being the most desirable. We study the following relevant decision problems: (i) given a typing for a design, determine whether it is local, maximal local, or perfect; (ii) given a design, establish whether a (maximal) local, or perfect typing does exist. For some of these problems we provide tight complexity bounds (polynomial space), while for the others we show exponential upper bounds. A main contribution is a polynomial-space algorithm for computing a perfect typing in this context, if it exists.

*Computing Laboratory and Oxford-Man Institute of Quantitative Finance, University of Oxford.

S. Abiteboul was supported by the ERC Advanced Grant Webdam and by the French ANR Grant Docflow. G. Gottlob was supported by EPSRC grant EP/E010865/1. The authors acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599 (See Section 8 for more detailed acknowledgments.)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.
Copyright 2009 ACM 978-1-60558-553-6 /09/06 ...$5.00.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed databases; H.2.1 [**Logical Design**]: Data models; Schema and subschema

General Terms

Design, Languages, Theory

1. INTRODUCTION

With the Web, information tends to be more and more distributed. In particular, the distribution of XML data is essential in many areas such as e-commerce (shared product catalog), collaborating editing (e.g., based on WebDAV [13]), or network directories [16]. (See also the W3C XML Fragment Interchange Working group [11].) It becomes often cumbersome to verify the validity, e.g., the type, of such a hierarchical structure spanning several machines. In this paper, we consider typing issues raised by the distribution of XML documents. We introduce nice properties that the distribution should obey to facilitate type verification based on locality conditions. We propose an automata-based study of the problem. Our theoretical investigation provides a starting point for the distributed validation of tree documents (verification) and for selecting a distribution for Web data (design). In general, it provides new insights in the typing of XML documents.

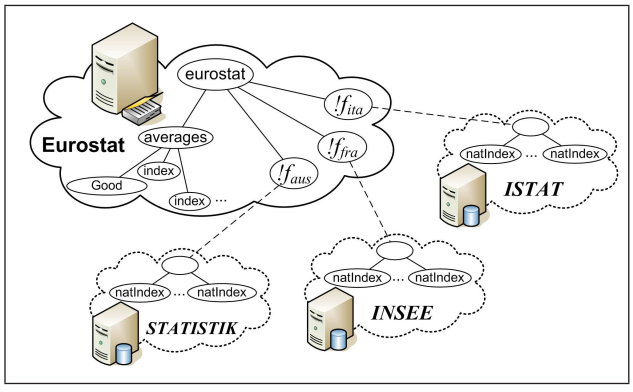


Figure 1: Distributed XML Document T_0

A *distributed document* is given by an XML "root" document T, that is stored locally at some site, some of which leaves refer to external resources, say $f_1, \ldots, f_n$, that each provides additional XML data to be attached to T. The *extension* $\mathsf{ext}(T)$ of T is the document obtained by replacing each node labeled f_i with the actual XML tree (or forest)

provided by resource f_i. Intuitively, $\mathsf{ext}(T)$ is the effective entire document one is interested in, where, however, different parts of this document are stored at different locations, maintained by different peers, and/or provided by programs or Web service calls. Figure 1 shows a (drastically simplified) possible distributed XML document for the *National Consumer Price Index (NCPI)* [1] maintained by the *Eurostat* [2]. The NCPI is a document containing consumer price data for each EC country. We assume that the national data are maintained in local XML repositories by each country's national statistics bureau (INSEE for France, Statistik Austria, Istat for Italy, UK Statistics Authority, and so on). Each national data set is under the strict control of its respective statistics bureau. The root document T_0 is maintained by Eurostat in Luxembourg and has a function f_i for each different country. In addition, T_0 contains average data for the entire EU zone. Figure 2 shows a possible extension $\mathsf{ext}(T_0)$ of T_0, where the actual data values are omitted.

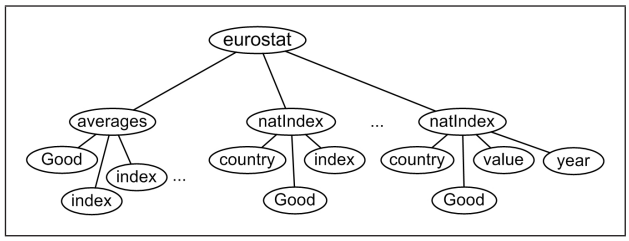


Figure 2: An extension $\mathsf{ext}(T_0)$ of document T_0

Typically, a global designer specifies a target type τ, together with a root document T. Such a type τ could be given, for example, in form of a DTD, an XML Schema, a regular tree language, or a tree automaton. The specification of τ means that the extension $\mathsf{ext}(T)$ is required to satisfy τ, formally, $\mathsf{ext}(T) \models \tau$. While this is done just in a similar way and spirit as when one specifies a type for a normal XML document, we now have a problem: *How do we enforce the validity w.r.t. this global type?* and *what should be the role of each peer in this enforcement of the type?*. In particular, can the global type τ be broken down into local types $\tau_1, \ldots, \tau_n$, one for each function f_i occurring in T, so that, to obtain a valid document $\mathsf{ext}(T)$, it suffices to make sure that each resource f_i provides local data satisfying τ_i.

A main issue here is to guarantee the consistency of the information. It can be the case that the validity of an update at resource f_i depends on the data in another resource. We would like to avoid such situation if possible. More precisely, we would like to provide each f_i with a typing τ_i that guarantees that (i) if each f_i verifies its type, then the global type is verified (soundness), and (ii) we do not introduce more restrictions than this global type (completeness). We call such a typing *local typing*. We study (maximal) local typings. We are also interested in *perfect typings*, namely when the typing is a unique maximum for all sound typings.

From a formal viewpoint, we use Active XML terminology and notation for describing distributed documents [1]. For types, we consider abstract versions of DTDs and XML Schemas [19]. We lay out the fundamentals of a theory of local typing and provide formal definitions of three main variants of locality: local typing, maximal local typing, and perfect typing, the latter being the most desirable. We study the following relevant verification problems: given a typing for a design, determine whether it is local, maximal local, or perfect (we call these problems DTD-LOC, DTD-ML and DTD-PERF, respectively). We also study the corresponding design problems: given a particular design, establish whether a local, maximal local, or perfect typing does exist (call these problems $\exists$DTD-LOC, $\exists$DTD-ML, $\exists$DTD-PERF, respectively) and, of course, find them.

The analysis carried out in this paper provides tight complexity bounds for some of these problems. In particular, problems DTD-LOC, DTD-ML, DTD-PERF and $\exists$DTD-PERF are **PSPACE**-complete. For the other problems (proved to be **PSPACE**-hard), we show exponential upper bounds, but a precise characterization of their computational complexity is still an open problem. To study this problem, we show that the problem for trees can be reduced to a problem on words. The problems on words are solved using automata techniques. In particular, a main contribution is a polynomial-space construction of a perfect typing, if it exists. Section 6 also considers more powerful typings, namely nondeterministic bottom-up tree automata (typically the largest class that is considered). In this setting, the direct reduction to words cannot be used. However, we show how the problem can be solved using tree automata techniques.

Before mentioning some related works and concluding this section, we further illustrate these concepts by detailing our *Eurostat* example. We first assume that Eurostat specifies the global type τ_0 for the distributed NCPI document, where τ_0 is given by the DTD shown in Figure 3. Briefly, DTD τ_0 requires that $\mathsf{ext}(T_0)$ consists of a subtree containing average data for *Goods* (such as food, energy, education, and so on). Each *Good* item is evaluated in different years by means of an *index*. Moreover, $\mathsf{ext}(T_0)$ may contain a forest of *nationalIndex*, namely indexes associated to goods in precise countries. To comply with different national databases, two different formats are allowed: *(country, Good, index)* or *(country, Good, value, year)*. It is easy to see that the pair $\langle \tau_0, T_0 \rangle$ allows a local typing (see Figure 4) that is even perfect (so, can be found out by the algorithm shown in Section 5), as we will clarify in the next section.

```
<!ELEMENT eurostat (averages, nationalIndex*)>
  <!ELEMENT averages (Good, index+)+>
  <!ELEMENT nationalIndex (country, Good,
                           (index | value, year))>
     <!ELEMENT index (value, year)>
<!ELEMENT Good (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

Figure 3: DTD τ_0

Suppose now that a designer defined instead the DTD τ_1 shown in Figure 5 as global type (the elements defined as in τ_0 are omitted.) The pair $\langle \tau_1, T_0 \rangle$ would be a bad design since τ_1 imposes to all countries to adopt the same format for their indexes (*natIndA* or *natIndB*). But this represents a constraint that cannot be controlled locally. Indeed, this new design does not admit any local typing. The nice locality properties of designs are obvious in such simplistic examples. However, when dealing with a large number of peers with very different desires and complex documents,

[1] See `http://epp.eurostat.ec.europa.eu`
[2] See `http://ec.europa.eu/eurostat`

```
!f_aus: <!ELEMENT root (nationalIndex*)>
          <!ELEMENT nationalIndex (country, Good,
                                   (index | value, year))>
            <!ELEMENT index (value, year)>
!f_fra: <!ELEMENT root (nationalIndex*)>
        ...
!f_ita: <!ELEMENT root (nationalIndex*)>
        ...
```

Figure 4: Perfect Typing for $\langle \tau_0, T_0 \rangle$

the problem rapidly starts defeating human expertise. The techniques developed in this paper are meant to support experts in designing such distributed document schemas.

```
<!ELEMENT eurostat (averages, (natIndA* | natIndB*))>
  <!ELEMENT averages (Good, index+)+>
  <!ELEMENT natIndA (country, Good, index)>
  <!ELEMENT natIndB (country, Good, value, year)>
```

Figure 5: DTD τ_1

Distributed data design has already been quite studied in particular for relational databases; see [7, 18]. Some previous works have considered the design of Web applications [6]. They lead to the design of Web sites. The design there is guided by an underlying process. It leads to a more dynamic notion of typing, where part of the content evolves in time, e.g., creating a cart for a customer. For obvious reasons, distributed XML has raised a lot of attention recently. Most works focused on query optimization, e.g., [3]. The few that consider design typically assume no ordering or only limited one [5]. This last work would usefully complement the techniques presented here. Also, works on relational database and LDAP [3] design focus on unordered collections. Even the W3C goes in this direction with a working group on *XML Fragment Interchange* [11]. The goal is to be able to process (e.g., edit) document fragments independently. Quoting the W3C Candidate Recommendation: "It may be desirable to view or edit one or more [fragments] while having no interest, need, or ability to view or edit the entire document." This is clearly related to the problem we study here.

The paper is organized as follows. Section 2 formally introduces our notions of (distributed) XML document, type, and defines the decisional problems studied. It also provides an overview of the results. Section 3 presents basic results. Sections 4 and 5 present the main results. Section 6 discusses extension to more complex types. Section 7 concludes and mentions possible areas for further research. Finally, in Section 8 we make a number of acknowledgments.

2. THE TYPING PROBLEMS

Distributed Documents (or Distributed Trees), like AXML documents [1, 2], are XML documents that may contain embedded function calls. The result of a *function call* $!f$ is still an XML document. When $!f$ is invoked, its result is used to enrich the original document. This process is termed materialization. In a distributed architecture (for instance a P2P architecture), a node x of a distributed tree T containing a function $!f$ is called a *docking point* of T. The docking point x connects the peers that invoke the function $!f$ and the peers that provide the corresponding XML document.

An interesting task is to associate a type τ_i (a DTD) to each call $!f_i$ in such a way that the XML document returned as answer is *valid* w.r.t. this type and any materialization process always produces a valid document w.t.r. a given global type τ that is also specified by a DTD. A global type and a distributed document represent the *design* of a given distributed architecture. A collection of types associated to the function calls in such a design is called a *typing*. Given a distributed design, we want to know whether either a precise typing has some properties or a typing with some properties does exist. (More formal definitions follow.)

XML. In this paper, we use a widespread abstraction of XML documents and DTDs (or XML Schemas) focusing on document structure [19]. An XML document is a *finite ordered, unranked tree* (hereafter just a tree) t with nodes labeled over an alphabet Σ. Given a node x of t, we denote by $\mathsf{lab}(x)$ its label, by $\mathsf{labcdrn}(x) \in \Sigma^*$ the string containing the labels of the children of x in left-to-right order, and by $\mathsf{labpar}(x)$ the label of the parent of x. Clearly, if $\mathsf{labcdrn}(x) = \varepsilon$, then x is a leaf node, while if $\mathsf{labpar}(x) = \varepsilon$, then x is the root of t. The size of t, denoted by $\|t\|$, is the number of its nodes.

Types. A DTD is formalized as a triple $\tau = \langle \Sigma, \pi, s \rangle$ where Σ is an alphabet (the *element names*), π is a function that maps each symbol in Σ to a nondeterministic finite state automaton (NFA for short) still over Σ, and $s \in \Sigma$ is the *start symbol*. The language $[\tau]$ of τ is a set of trees such that for each tree $t \in [\tau]$, $\mathsf{lab}(\mathsf{root}(t)) = s$ and for every node x, with $\mathsf{lab}(x) = a$, $\mathsf{labcdrn}(x) \in [\pi(a)]$. The NFA associated to an element name is also called its *content model*. In this paper, we often specify π as a function that maps Σ-symbols to Σ-regexes since any regular expression of size n can be transformed into an equivalent ε-free NFA with $\mathcal{O}(n \log^2 n)$ transitions in time $\mathcal{O}(n \log^2 n)$ [12, 15]. An example of DTD is $\tau_1 = \langle \{s_1, c\}, \pi_1, s_1 \rangle$ with $\pi_1(s_1) = c^*$ and $\pi_1(c) = \varepsilon$. In the remaining of the paper, we omit to specify regexs such as $\pi_1(c) = \varepsilon$; i.e., if no regex is given for a label, nodes with this label are assumed to be (solely) leaves. We consider only *reduced* DTDs where there is no unreachable symbol and the language related to each nonterminal symbol is nonempty.

The expressive power of DTDs can be extended by specializing element names, as, e.g., in XML Schema [24] and Relax NG [8]. Then, an *extended* DTD (EDTD) is a quintuple $\tau = \langle \Sigma, \Delta, \pi, s, \mu \rangle$ where $\langle \Delta, \pi, s \rangle$ is a DTD on the *specialized element names* Δ, and μ is a mapping from Δ to Σ. In this case, for a labeled tree t, $t \in [\tau]$ if there exists a tree t' in $[\langle \Delta, \pi, s \rangle]$ such that $t = \mu(t')$ (where μ is extended to trees). Tree t' can be seen as a witness for t. In particular, an XML Schema can be seen as a EDTD with an extra constraint (the Element Declarations Consistent). It essentially prohibits that, in any content model, an element has two different specializations.

Distributed documents. As previously mentioned, we use Active XML terminology and notation for describing distributed documents. Let Σ_L and Σ_F be two alphabets, respectively, of *labels* (such as a, b, etc.) and *function symbols* (such as $!f$, $!g$, etc.). A *distributed document* or *distributed tree* T is a tree over $\Sigma_L \cup \Sigma_F$ where any function-node is a

[3]Lightweight Directory Access Protocol (LDAP) is a set of open protocols used to access centrally stored information over a network.

leaf node and where $\mathsf{root}(T) \in \Sigma_L$. A *forest* is a sequence of distributed trees. The *extension* (or *semantics*) of a function $!f$ is a forest and denoted by $\mathsf{ext}(!f)$. The extension $\mathsf{ext}(T)$ of a tree T is obtained by replacing each $!f$ by $\mathsf{ext}(!f)$. A *type* τ for a distributed tree T is a DTD (or an XML Schema). Tree T satisfies type τ if $\mathsf{ext}(T)$ does. Since we do not want to cope with nonregularity we assume that the same function does not occur twice in a tree. For instance, in the tree $T = a(!f!f)$ the children of a in any $\mathsf{ext}(T)$ are of the form ww for some word w. But since this is not a regular language, the type of T cannot be defined by a DTD. Clearly, the same type can be associated to different functions.

Distributed document typing. In this section, we first define the concept of typing for a distributed tree. We then introduce the notion of design and interesting properties of typings for designs.

Let $T(!f_n)$ be a distributed tree. We want to type its function calls. The answer of some $!f_i$ may be a forest, which is not an XML document any more. So, for the type τ_i associated to $!f_i$, we actually use a DTD having a root with a label s_i that does not appear in any other part of τ_i. The answer of $!f_i$ is the forest of children of the s_i-root of a document of this type. Now, let $(!f_n)$ be a finite sequence[4] of distinct function calls. A *typing* for a distributed tree $T(!f_n)$ is a positional mapping from the functions in $(!f_n)$ to a sequence (τ_n) of types (DTDs). Informally, we denote by $T(\tau_n)$ the new type defined from T by replacing each f_i with τ_i. This raises consistency issues. See further for a formal definition.

Example 1. Consider the distributed tree $T = a(b\ !f_1\ d\ !f_2)$. The pair $\tau_1 = \langle\{s_1, c\}, \pi_1, s_1\rangle$ and $\tau_2 = \langle\{s_2, e\}, \pi_2, s_2\rangle$ of DTDs, with $\pi_1(s_1) = c^*$ and $\pi_2(s_2) = e^*$, is a typing for T. The activation of both f_1 and f_2 may return trees $s_1(cc)$ and $s_2(e)$, respectively. These trees can be plugged into T producing the extension $a(bccde)$. Finally, $T(\tau_1, \tau_2)$ is the DTD $\langle\{a, b, c, d, e\}, \pi, a\rangle$ where $\pi(a) = bc^*de^*$. □

Formally, a typing for T is a sequence (τ_n) such that $T(\tau_n) = \langle\Sigma_L, \pi, s\rangle$ obtained as follows is a well-defined DTD:

1. Σ_L is the union of the element names in $T(!f_n)$ and all the labels in each type τ_i;
2. $s = \mathsf{lab}(r)$, where r is the root of T;
3. for each node x of T with label $a \in \Sigma_L$ and such that $\mathsf{labcdrn}(x) = w_1 \dots w_k$, $\pi(a)$ is the regex $r_1 \dots r_k$, where $r_j = w_j$ if $w_j \in \Sigma_L$, while $r_j = \pi_i(s_i)$ if w_j is the function f_i associated to the type $\tau_i = \langle\Sigma_{Li}, \pi_i, s_i\rangle$. (For NFAs, $\pi(a)$ is defined in a similar way.)
4. for each $\tau_i = \langle\Sigma_{Li}, \pi_i, s_i\rangle$ and for each $a \in \Sigma_{Li} \setminus \{s_i\}$, $\pi(a) = \pi_i(a)$.

In this case, we say that (τ_n) is *consistent for* $T(!f_n)$.

Note that (τ_n) may be inconsistent for $T(!f_n)$ because the same element name is assigned two different content models either by τ_i and τ_j, $i \neq j$, by T and some τ_i, or by two occurrences of that label in T. Observe also that T plays here the role of a type. Indeed, one could consider design problems where we start from a distributed type. Our results can be extended to such a setting.

[4]We denote a finite sequence of objects $(x_1, \dots, x_n)$ over an index set $I = \{1, \dots, n\}$ by (x_n) and we often omit the specification of the index set I.

All these definitions about trees can be adapted to strings in a straightforward way. (We will see the importance of problems on strings in Section 3.) A *distributed string*

$$w(!f_n) = w_0!f_1w_1 \dots !f_nw_n$$

is a string on alphabet $\Sigma_L \cup \Sigma_F$ where: $n \geq 1$; $w_i \in \Sigma_L^*$ for each $i \in \{0, \dots, n\}$; $!f_i \in \Sigma_F$ for each $i \in \{1, \dots, n\}$; and $!f_i \neq !f_j$ for each $i \neq j$. A *type* is overloaded by an NFA. A *typing* for $w(!f_n)$ is still a positional mapping from the functions in $(!f_n)$ to a sequence (τ_n) of NFAs. With $w(\tau_n)$ we denote a new type (NFA) defined from w by replacing each f_i with τ_i.

Let τ and τ' be two types (DTDs or NFAs). We say that:

- $\tau < \tau'$ (smaller) iff $[\tau] \subset [\tau']$
- $\tau \leq \tau'$ (smaller or equivalent) iff $[\tau] \subseteq [\tau']$
- $\tau \equiv \tau'$ (equivalent) iff $[\tau] = [\tau']$
- $\tau \shuffle \tau'$ (independent) iff $[\tau] \cap [\tau'] = \emptyset$
- $\tau \nsim \tau'$ (incomparable) iff $[\tau] \not\subseteq [\tau']$ and $[\tau] \not\supseteq [\tau']$ and $[\tau] \cap [\tau'] \neq \emptyset$
- $\tau \not\equiv \tau'$ (different) iff either $\tau \nsim \tau'$ or $\tau \shuffle \tau'$

PROPOSITION 1. *Two DTDs τ_1 and τ_2 are equivalent if (i) they have the same root; (ii) they use the same element names; and (iii) for each element name, its content models in the two, are equivalent.*

Given two typings (τ_n) and (τ'_n), we say that:

- $(\tau_n) < (\tau'_n)$ iff $(\tau_n) \leq (\tau'_n)$ and $\tau_i < \tau'_i$ for some i
- $(\tau_n) \leq (\tau'_n)$ iff $\tau_i \leq \tau'_i$ for each i
- $(\tau_n) \equiv (\tau'_n)$ iff $\tau_i \equiv \tau'_i$ for each i
- $(\tau_n) \nsim (\tau'_n)$ iff $\tau_i \nsim \tau'_i$ for some i
- $(\tau_n) \shuffle (\tau'_n)$ iff $\tau_i \shuffle \tau'_i$ for each i
- $(\tau_n) \not\equiv (\tau'_n)$ iff $\tau_i \not\equiv \tau'_i$ for some i

We call *design* a pair $\langle\tau, T(!f_n)\rangle$, i.e., a design consists in a distributed document and a (target) type for that document. The problems we consider concern typing such designs.

Definition 1. For a design $\langle\tau, T(!f_n)\rangle$, a typing (τ_n) is:

- *sound* if $T(\tau_n) \leq \tau$;
- *maximal* if it is sound and if there exists no other sound typing (τ'_n) for $T(!f_n)$ and τ such that $(\tau_n) < (\tau'_n)$;
- *complete* if $\tau \leq T(\tau_n)$;
- *local* if $T(\tau_n) \equiv \tau$, namely if it is sound and complete;
- *perfect* if it is local, and if for each other sound typing (τ'_n), we have $(\tau'_n) \leq (\tau_n)$. □

Clearly, local typings present the advantage of allowing a local verification of document consistency (soundness and completeness by definition). Also, no consistent document is ruled out (completeness). Maximal locality guarantees that in some sense, no unnecessary constraints are imposed to the participants. Finally perfect typings are somewhat the

ultimate one can expect in terms of not imposing constraints to the participants. Many designs will not accept a perfect typing. However, there are maximal sound typings which are not local. This is not surprising as there are designs that have at least a sound typing but do not allow any local at all, and clearly, if there is a sound typing, then there must also exist a maximal sound one. We will see examples that separate these different classes further. But before, we formally state the problems studied in this paper.

Definition 2. DTD-LOC, DTD-ML, DTD-PERF are the following decision problems. Given a DTD design $\langle \tau, T(!f_n)\rangle$ and a typing (τ_n), is (τ_n) a local, or maximal local, or perfect typing for $\langle \tau, T(!f_n)\rangle$, respectively?

Definition 3. ∃DTD-LOC, ∃DTD-ML, ∃DTD-PERF are the following decision problems. Given a DTD design $\langle \tau, T(!f_n)\rangle$, does there exist a local, or maximal local, or perfect typing for this design, respectively?

We similarly define the corresponding word problems, that is (∃)NFA-LOC, (∃)NFA-ML, (∃)NFA-PERF. Table 1 gives an overview of the complexity results for the typings problems previously defined. We will see in Section 3 that each problem on trees is logspace-reducible to a set of problems on strings. Thus, it suffices to prove the results in Table 1 for words. All the problems are **PSPACE**-hard, and in particular, we prove that NFA-LOC, NFA-ML, NFA-PERF and ∃NFA-PERF are **PSPACE**-complete, whereas we show an upper bound for the problems ∃NFA-LOC and ∃NFA-ML in 2-**EXPSPACE**.

We pay more attention to maximal locality rather than maximality alone as in the latter case the existential problem is trivial (**NL**-complete) while the checking problem has the same complexity in both cases (**PSPACE**-complete).

Table 1: Complexity Results

DTD-/NFA-	∃DTD-/∃NFA-	
PSPACE-complete	2-**EXPSPACE**	-LOC
PSPACE-complete	2-**EXPSPACE**	-ML
PSPACE-complete	**PSPACE**-complete	-PERF

3. BASIC RESULTS

In this section, we first present examples that separate the different design properties of typings. We then show the reduction to word problems.

Example 2. Let $\tau = \langle\{s, a, b, c\}, \pi, s\rangle$ be a type where $\pi(s) = a^*bc^*$ and $T = s(!f_1!f_2)$ be a distributed tree. Then, (a^*bc^*, c^*) and (a^*, a^*bc^*) are two *maximal local typings*, so there is no perfect typing for this design. Observe that $(a?, a^*bc^*)$ is a *local typing* that is not maximal because it imposes unnecessary constraints to the local sites. If desired, one could leave them more freedom, e.g., type the first function with a^*. □

Example 3. Let $\tau = \langle\{s, a, b, c\}, \pi, s\rangle$ be a type where $\pi(s) = a^*bc^*$ and $T = s(!f_1b!f_2)$ be a distributed tree. The typing (a^*, c^*) is *perfect*. This has to be an excellent typing since there is no alternative maximal local typing. □

Example 4. Let $\tau = \langle\{s, a, b\}, \pi, s\rangle$ be a type where $\pi(s) = (ab)^*$ and $T = s(!f_1!f_2)$ be a distributed tree. The typing $((ab)^*, (ab)^*)$ is a unique maximal local but it is not perfect. Consider, in fact, typing (a, b). It is sound but $(a, b) \leq ((ab)^*, (ab)^*)$ does not hold. Clearly, a perfect typing cannot exist. □

Example 5. Let $\tau = \langle\{s, a, b\}, \pi, s\rangle$ be a type where $\pi(s) = (ab)^+$ and $T = s(!f_1!f_2)$ be a distributed tree. There are three maximal local typings: $((ab)^*, (ab)^+)$, $((ab)^*a, b(ab)^*)$, and $((ab)^+, (ab)^*)$ depending on whether either ε, a, or none of them are in $\mathsf{ext}(!f_1)$, respectively. □

To complete our separation among the classes characterizing different typings we point out that every perfect typing is necessarily unique maximal local. The converse is not true as proved by Example 4. We now show how to reduce a typing problem on trees to a set of typing problems on strings. (The first proof is omitted for space limitation.)

THEOREM 2. *There exists a local DTD-typing for the design $\langle \tau, T(!f_n)\rangle$ with some DTD $\tau = \langle \Sigma_L, \pi, s\rangle$, if and only if for each node x in T, such that $\mathsf{lab}(x) \in \Sigma_L$, there is a local typing for the design $\langle \pi(\mathsf{lab}(x)), \mathsf{labcdrn}(x)\rangle$.*

The previous theorem considers typing with DTDs. With minor modifications, one could extend this result (and Corollary 3) to XML schemas. Returning to DTDs, we have:

COROLLARY 3. *Problems* DTD-LOC, DTD-ML, DTD-PERF, ∃DTD-LOC, ∃DTD-ML, ∃DTD-PERF *are logspace Turing reducible to* NFA-LOC, NFA-ML, NFA-PERF, ∃NFA-LOC, ∃NFA-ML, ∃NFA-PERF, *respectively.*

PROOF. (Sketch.) Let $\tau = \langle \Sigma_L, \pi, s\rangle$ be a type and $T(!f_n)$ be a distributed tree. Consider, for instance, the DTD-LOC problem. Scan the tree in document order, which is well known to be feasible in logarithmic space [9]. For each node x in T such that $\mathsf{lab}(x) \in \Sigma_L$ and $\mathsf{labcdrn}(x)$ contains at least a function, solve the problem NFA-LOC for the pair $\langle \pi(\mathsf{lab}(x)), \mathsf{labcdrn}(x)\rangle$. □

Remark. More tractable results may be obtained by considering restricted classes of regular expressions [10, 17].

4. THE TYPING PROBLEMS FOR WORDS

We study in this section the typing problems for words. (Recall that the problem for trees has been reduced to problems for words.) We present a number of complexity results. We leave for the next section, two issues, namely NFA-PERF and ∃NFA-PERF, for which we will need a rather complicated automata construction. We start by recalling a definition and a result that we will use further.

Definition 4. NFA-EQUIV problem. Given two NFAs, are they equivalent?

THEOREM 4. *[22]* NFA-EQUIV *is* ***PSPACE****-complete.*

The hardness of the NFA-EQUIV problem is used to show some hardness results of our problems.

THEOREM 5. *Problems* NFA-LOC, NFA-ML, NFA-PERF *are* ***PSPACE****-hard.*

PROOF. We define a logspace transformation φ, in such a way that NFA-EQUIV $\leq_m^{\mathbf{L}}$ NFA-LOC. Afterwards, we show that the statement also holds for the other two problems. Let $\mathcal{A}$, $\mathcal{A}_1$ be two arbitrary NFAs. The application of φ to the pair $\mathcal{A}$, $\mathcal{A}_1$ produces the design $\langle \tau, w\rangle$ and the typing τ_1, where $\tau = \mathcal{A}$, $w = !f_1$ and $\tau_1 = \mathcal{A}_1$. Since $w(\tau_1) = \mathcal{A}_1$, it is clear that $\tau \equiv w(\tau_1)$ iff $\mathcal{A} \equiv \mathcal{A}_1$. Finally, we just notice that $\mathcal{A} \equiv \mathcal{A}_1$ iff τ_1 is both perfect and maximal local as w consists of just a function. □

We now consider upperbounds. Section 5 will show that NFA-PERF is in **PSPACE**. We next show that NFA-LOC is.

THEOREM 6. NFA-LOC *is in* ***PSPACE*** *(so the problem is* ***PSPACE****-complete).*

PROOF. Let $w(!f_n)$ be a distributed string, τ be an NFA, and (τ_n) be a typing. Since the new automaton $w(\tau_n)$ has size $\mathcal{O}(\|w\| + |(\tau_n)|)$, we can check in polynomial space if $w(\tau_n) \equiv \tau$. □

Now consider NFA-ML.

THEOREM 7. NFA-ML *is in* ***PSPACE*** *(so the problem is* ***PSPACE****-complete).*

PROOF. Let $w(!f_n)$ be a distributed string, τ be an NFA, and (τ_n) be a typing. First of all, we check if (τ_n) is local. We proved that this problem is doable in **PSPACE**. Subsequently, we check if (τ_n) is *not* maximal. In particular, (τ_n) is not maximal if there exists (at least) an $i \in \{1, \ldots, n\}$ such that the NFA $w(\tau'_n) \cap \tau$, say $\mathcal{A}$, defines a nonempty language, where $(\tau'_n) = \tau_1, \ldots \bar{\tau}_i, \ldots \tau_n$. Since τ_i is nondeterministic, the NFA $\bar{\tau}_i$ accepting the complement of language $[\tau_i]$ may even require $\mathcal{O}(2^k)$ states, where k is the number of states of τ_i [14], and consequently the size of $\mathcal{A}$ would be exponential. For finite automata (deterministic or not) the nonemptiness problem is basically the same as the graph reachability problem and it is thus **NL**-complete [20]. We can avoid the materialization of $\mathcal{A}$ with an "on-the-fly" construction of the cross product of $w(\tau'_n)$ and τ. Hence, an **NL** algorithm on a non-materialized (single) exponential automaton leads to **PSPACE**. □

Let us turn to the hardness of the ∃-versions of the problems.

THEOREM 8. *Problems* ∃NFA-LOC, ∃NFA-ML, *and* ∃NFA-PERF *are* ***PSPACE****-hard.*

PROOF. We define a logspace transformation φ, in such a way that the following relations hold: (1) NFA-EQUIV $\leq_m^{\mathbf{P}}$ ∃NFA-LOC; (2) NFA-EQUIV $\leq_m^{\mathbf{P}}$ ∃NFA-ML; (3) NFA-EQUIV $\leq_m^{\mathbf{P}}$ ∃NFA-PERF. Let $\mathcal{A}_1 = \langle K_1, \Sigma_1, \Delta_1, s_1, F_1\rangle$, $\mathcal{A}_2 = \langle K_2, \Sigma_2, \Delta_2, s_2, F_2\rangle$ be two arbitrary NFAs. The application of φ to the pair $(\mathcal{A}_1, \mathcal{A}_2)$ produces the design $\langle \mathcal{A}, w\rangle$ where w is the distributed string $!f_1\ c\ !f_2$, c is a terminal symbol which does not belong to $(\Sigma_1 \cup \Sigma_2)$, while automaton $\mathcal{A} = \langle K, \Sigma, \Delta, s, F\rangle$ is defined as follows: (i) $K = K_1 \cup K_2 \cup \{s, p_c, q_c\}$; (ii) $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{a, b, c\}$; (iii) $\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, a, p_c), (s, b, p_c), (p_c, c, q_c), (q_c, \varepsilon, s_1), (q_c, \varepsilon, s_2)\}$; (iv) $F = F_1 \cup F_2$. Intuitively, if we consider $\mathcal{A}_1$ and $\mathcal{A}_2$ as regexes, then $\mathcal{A}$ is $(ac\mathcal{A}_1 + bc\mathcal{A}_2)$. We claim that there is a local typing (similarly, ML, or perfect) for $\langle \mathcal{A}, w\rangle$ iff $\mathcal{A}_1 \equiv \mathcal{A}_2$. First of all, we observe that transformation φ is extremely simple and it is clearly in logspace. In fact, string w is a constant, while the choice of a terminal symbol which does not appear in $\mathcal{A}_1$ nor in $\mathcal{A}_2$ can be done in logspace, and also $\mathcal{A}$ can be obtained by merging $\mathcal{A}_1$ and $\mathcal{A}_2$ with a constant number of transitions. We prove the statement for (1) and we just notice that whenever there is a local typing for $\langle \mathcal{A}, w\rangle$, then the typing $((a + b), \mathcal{A}_1)$ is perfect (thus, also maximal).

($\Rightarrow$) *If there is a local typing for* $\langle \mathcal{A}, w\rangle$ *then* $\mathcal{A}_1 \equiv \mathcal{A}_2$. Since $\mathcal{A} = (ac\mathcal{A}_1 + bc\mathcal{A}_2)$, then $[ac\mathcal{A}_1]$ and $[bc\mathcal{A}_2]$ form a partition of $[\mathcal{A}]$. In this case, any local typing must have the following form $((aX_1 + bX_2), Y)$ where X_1, X_2, Y are NFAs. Clearly, all the strings accepted by w are obtained by aX_1cY and bX_2cY. Then $c\mathcal{A}_1 \equiv X_1cY$ and $c\mathcal{A}_2 \equiv X_2cY$ must hold. But since any string in $[\mathcal{A}_1]$ or $[\mathcal{A}_2]$ does not start with c, then necessarily $[X_1] = [X_2] = \varepsilon$. This way, $\mathcal{A}_1 \equiv Y$ and $\mathcal{A}_2 \equiv Y$ and then $\mathcal{A}_1 \equiv \mathcal{A}_2$.

($\Leftarrow$) *If* $\mathcal{A}_1 \equiv \mathcal{A}_2$, *there is a local typing for* $\langle \mathcal{A}, w\rangle$. This part of the proof is trivial because $((a + b), \mathcal{A}_1)$ always represents a local typing for $\langle \mathcal{A}, w\rangle$. □

We now have lower bounds for all these problems and some upper bounds. We will derive missing upper bounds using the construction of automata that we call "perfect" for given design problems.

5. PERFECT AUTOMATON FOR WORDS

We next present the construction of the *perfect automaton* for a design word problem. The perfect automaton has the property that if a perfect typing exists for this problem, it is "highlighted" by the automaton. This will provide a PSPACE procedure for finding this perfect typing if it exists.

Given a DTD τ and the reduction between trees and strings, we consider, the NFA $\mathcal{A}$ for each content model in τ. We start from $\mathcal{A}$ to build the so called perfect automaton Ω which exhibits many interesting properties. Let $\mathcal{A} = \langle K, \Sigma, \Delta, s, F\rangle$ be an NFA. We can assume w.l.o.g. that it has no ε-transition. The *extended transition relation* $\Delta^* \subseteq K \times \Sigma^* \times K$ is the reflexive-transitive closure of Δ and it is defined as follows: (i) for each $q \in K$, $(q, \varepsilon, q) \in \Delta^*$; (ii) for each string $a_1 ... a_n \in \Sigma^+$, $(q_0, a_1 ... a_n, q_n) \in \Delta^*$ iff there is a sequence of transitions of the form $(q_0, a_1, q_1), \ldots, (q_{n-1}, a_n, q_n)$ in Δ. (Observe that the language $[\mathcal{A}]$ may be defined as $\{w \in \Sigma^* : (s, w, q_f) \in \Delta^*,\ q_f \in F\}$.)

Given two states q_i, q_f in K, a string w in Σ^* is said to be *delimited* in $\mathcal{A}$ by q_i and q_f if $(q_i, w, q_f) \in \Delta^*$. By exploiting this notion, the sets of all the states delimiting w in $\mathcal{A}$ are defined as follows:

$$Ini(\mathcal{A}, w) = \{q_i \in K :\ \exists q_f \in K \text{ s.t. } (q_i, w, q_f) \in \Delta^*\}$$

$$Fin(\mathcal{A}, w) = \{q_f \in K :\ \exists q_i \in K \text{ s.t. } (q_i, w, q_f) \in \Delta^*\}$$

In particular, if $w = \varepsilon$, these two sets are $Ini(\mathcal{A}, \varepsilon) = Fin(\mathcal{A}, \varepsilon) = K$. $Ini(\mathcal{A}, w)$ is called the set of *initial states* while $Fin(\mathcal{A}, w)$ is the set of *final states* for the word w. Given two states q_i, q_f in K, the *local automaton* $\mathcal{A}(q_i, q_f) = \langle K' \subseteq K, \Sigma, \Delta', q_i, \{q_f\}\rangle$ induced from $\mathcal{A}$ by q_i, q_f is a portion of $\mathcal{A}$ containing all those transitions of $\mathcal{A}$ leading from q_i to q_f. More precisely, for each pair of states q, q' in K and for each symbol a in Σ, $(q, a, q') \in \Delta'$ iff there are two strings u, v in Σ^* such that: $(q_i, u, q) \in \Delta^*$, $(q, a, q') \in \Delta$, and $(q', v, q_f) \in \Delta^*$. Finally, given two strings w_1, w_2 in Σ^+, then $\mathcal{A}(w_1, w_2)$ is the *set of all local automata* induced by w_1 and w_2. It is formally defined as

$$\mathcal{A}(w_1, w_2) = \{\mathcal{A}(q_i, q_f) : q_i \in Fin(\mathcal{A}, w_1),\ q_f \in Ini(\mathcal{A}, w_2)\}.$$

In particular, if $w_i = \varepsilon$ for some $i \in \{1, \ldots n\}$, the distributed string contains consecutive functions. In particular for the previous definitions we have:

$$\mathcal{A}(w_1, \varepsilon) = \{\mathcal{A}(q_i, q_f) : q_i \in Fin(\mathcal{A}, w_1) \text{ and } q_f \in K\}$$

$$\mathcal{A}(\varepsilon, w_2) = \{\mathcal{A}(q_i, q_f) : q_i \in K \text{ and } q_f \in Ini(\mathcal{A}, w_2)\}$$

$$\mathcal{A}(\varepsilon, \varepsilon) = \{\mathcal{A}(q_i, q_f) : q_i, q_f \in K\}.$$

Similarly, given a string w in Σ^*, $\mathcal{A}(w)$ is the *set of all local automata* induced by w. It is defined as $\mathcal{A}(w) = \{\mathcal{A}(q_i, q_f) : (q_i, w, q_f) \in \Delta^*\}$ and in particular $\mathcal{A}(\varepsilon) = \{\mathcal{A}(q, q) : q \in K\}$ is a set of $|K|$ automata, one for each state in K.

Let $w(!f_n)$ be a distributed string and $\mathcal{A}$ be an NFA. The *perfect automaton* w.r.t. $\mathcal{A}$ and w consists of several local automata suitably joined together by ε-transitions. It is denoted by both $\Omega(\mathcal{A}, w)$ and just Ω whenever it is clear from the context who are $\mathcal{A}$ and w. Algorithm 1 describes how to build the perfect automaton (assume that any pair of local automata have disjoint sets of states labeled as in $\mathcal{A}$), while Figure 6 shows the perfect automaton obtained by a given finite state machine and a distributed string. We say that $\mathcal{A}$ is *compatible* with w if the set of all (legal) local automata in Ω is not empty after correction steps, or equivalently, if there exists at least a sound typing. Moreover,

- $Seq(\Omega)$ denotes the set of all the sequences W_0, X_1, W_1, …, X_n, W_n of connected automata in Ω such that: W_0 is an automaton in $\mathcal{A}(w_0)$, while W_i and X_i are, respectively, in $\mathcal{A}(w_i)$ and $\mathcal{A}(w_{i-1}, w_i)$ for any $i \in \{1, \ldots, n\}$;
- $Typ(\Omega) = \{(X_i) : W_0, X_1, W_1, \ldots, X_n, W_n \in Seq(\Omega)\}$ is the set containing all different typings $(X_1, \ldots, X_n)$ from any sequence in $Seq(\Omega)$;
- $Aut(\Omega_i) = \{X_i : (X_1, \ldots, X_n) \in Typ(\Omega)\}$ is the set of all legal automata in $\mathcal{A}(w_{i-1}, w_i)$;
- $\Omega_i = \cup Aut(\Omega_i)$ is the type obtained by the union of all automata $Aut(\Omega_i)$;
- (Ω_i) is *the typing* for w and $\mathcal{A}$ obtained from Ω.

Let $(\mathcal{A}_n)$ be a sequence of automata. We define the *direct extension* of $(\mathcal{A}_n)$ as the set of string defined as $[(\mathcal{A}_n)] = \{u_1 \ldots u_n \mid \text{for each } i \;\; u_i \in [\mathcal{A}_i]\}$.

LEMMA 9. *For any NFA $\mathcal{A}$, then $\Omega \leq \mathcal{A}$ holds. The contrary ($\mathcal{A} \leq \Omega$) does not hold.*

PROOF. Given a string u in $[\Omega]$, then there exists a sequence (τ_{2n+1}) of automata in $Seq(\Omega)$ accepting u and expressible as $\mathcal{A}(s, q_0)$, $\mathcal{A}(q_0, s_1)$, $\mathcal{A}(s_1, q_1)$, …, $\mathcal{A}(q_{n-1}, s_n)$, $\mathcal{A}(s_n, q_n)$ for some states $q_0, s_1, q_1 \ldots, s_n, q_n$. Moreover, by definition of direct extension, for each string $u_0\sigma_1u_1 \ldots \sigma_nu_n$ in $[(\tau_{2n+1})]$ we have that $u_0 \in [\mathcal{A}(s, q_0)]$, $\sigma_i \in [\mathcal{A}(q_{i-1}, s_i)]$ and $u_i \in [\mathcal{A}(s_i, q_i)]$, for each $i \in \{1, \ldots, n\}$. But, by definition of local automata, the following sequence of transitions:

$$(s, w_0, q_0) \in \Delta^*,\ (q_0, \sigma_1, s_1) \in \Delta^*,\ (s_1, w_1, q_1) \in \Delta^*,\ \ldots$$
$$\ldots,\ (q_{n-1}, \sigma_n, s_n) \in \Delta^*,\ (s_n, w_n, q_n) \in \Delta^* \in F$$

is also derivable by $\mathcal{A}$.

For the second part of the proof consider the string $w = a!fc$ and the regex $abc + d$. □

LEMMA 10. *Let $w(!f_n)$ be a string compatible with an NFA $\mathcal{A}$. Any typing in $Typ(\Omega)$ is sound for w and $\mathcal{A}$.*

PROOF. Given any typing (X_n) in $Typ(\Omega)$, by definition, there is a sequence (τ_{2n+1}) of automata such that $X_i = \tau_{2i}$ for each $i \in \{1, \ldots, n\}$. By Lemma 9 $(\tau_{2n+1}) \leq \mathcal{A}$ holds. Moreover as, by definition, the extension of $w(X_n)$ is $[w(X_n)] = \{w_0\sigma_1w_1 \ldots \sigma_nw_n : \sigma_i \in [X_i],\ 1 \leq i \leq n\}$. Then $w(X_n) \leq (\tau_{2n+1})$ as well since all strings $w_0, \ldots, w_n$ are accepted by $\tau_1, \tau_3 \ldots, \tau_{2n+1}$, respectively, by definition of local automata induced by a single string. Therefore, $w(X_n) \leq \mathcal{A}$. □

Algorithm 1 PERFECTAUTOMATON$(w, \mathcal{A})$

1. **Input**: $w(!f_n) = w_0!f_1w_1 \ldots !f_nw_n$, $\mathcal{A} = \langle K, \Sigma, \Delta, s, F\rangle$
2. **Output**: $\Omega(\mathcal{A}, w) := \emptyset$
3. **for each** automaton $W \in \mathcal{A}(w_0)$ **do**
   ▷ add W to Ω
4. **for each** $i \in \{1, \ldots, n\}$ **do**
   ▷ **for each** automaton $X \in \mathcal{A}(w_{i-1}, w_i)$ **do**
   a. add X to Ω
   b. **for each** automaton $W \in \mathcal{A}(w_{i-1})$ **do**
   – **if** label$(q_{fin}(W)) =$ label$(q_{ini}(X))$
   · add the transition $(q_{fin}(W), \varepsilon, q_{ini}(X))$ to Ω
   c. **for each** automaton $W \in \mathcal{A}(w_i)$ **do**
   – add W to Ω
   – **if** label$(q_{fin}(X)) =$ label$(q_{ini}(W))$
   · add the transition $(q_{fin}(X), \varepsilon, q_{ini}(W))$ to Ω

`//Correction steps:`

5. **for each** automaton $W \in \mathcal{A}(w_0)$ **do**
   – **if** label$(q_{ini}(W)) \neq s$ `//if` $w_0 = \varepsilon$
   · remove W from Ω `//it is illegal`
6. merge all automata in Ω being in $\mathcal{A}(w_0)$ according to their labels and use the (unique) initial state as initial state for Ω
7. **for each** automaton $W \in \mathcal{A}(w_n)$ **do**
   – **if** label$(q_{fin}(W)) \in F$
   · $F(\Omega) = F(\Omega) \cup \{q_{fin}(W)\}$
   else `//if` $w_n = \varepsilon$
   · remove W from Ω `//it is illegal`
8. **for each** automaton $A \in \Omega$ **do**
   – **if** (there is no path from $q_{ini}(\Omega)$ to A **or**
   there is no path from A to any final state of Ω)
   · remove A from Ω `//it is illegal`

THEOREM 11. *Let $w(!f_n)$ be a distributed string compatible with a given NFA $\mathcal{A}$, and (τ_n) be a sound typing for them. Then, both $w(\tau_n) \leq \Omega$ and $(\tau_n) \leq (\Omega_n)$ hold.*

PROOF. Since (τ_n) is sound for w and $\mathcal{A}$, then $w(\tau_n) \leq \mathcal{A}$ holds. In particular, for each string $\chi = w_0\sigma_1w_1 \ldots \sigma_nw_n$ in $[w(\tau_n)]$, where each $\sigma_i \in [\tau_i]$, there is a sequence of states $q_0, s_1, q_1 \ldots, s_n, q_n$ proving the membership of χ in $[\mathcal{A}]$ by the following sequence of transitions

$$(s, w_0, q_0) \in \Delta^*,\ (q_0, \sigma_1, s_1) \in \Delta^*,\ (s_1, w_1, q_1) \in \Delta^*,\ \ldots$$
$$\ldots,\ (q_{n-1}, \sigma_n, s_n) \in \Delta^*,\ (s_n, w_n, q_n) \in \Delta^*$$

where $q_n \in F$ holds as well. But, this means that the sequence $\mathcal{A}(s, q_0)$, $\mathcal{A}(q_0, s_1)$, $\mathcal{A}(s_1, q_1)$, …, $\mathcal{A}(q_{n-1}, s_n)$, $\mathcal{A}(s_n, q_n)$ of automata belongs to $Seq(\Omega)$, so $w(\tau_n) \leq \Omega$ holds. Moreover, since each $\mathcal{A}(q_{i-1}, s_i) \in Aut(\Omega_i)$, it follows that $\tau_i \leq \Omega_i$ for each i, that is $(\tau_n) \leq (\Omega_n)$. □

COROLLARY 12. *Let $w(!f_n)$ be a distributed string compatible with a given NFA $\mathcal{A}$, and (τ_n) be a local typing for them. Then, $w(\tau_n) \equiv \Omega \equiv \mathcal{A}$ holds.*

PROOF. By Lemma 9 and Theorem 11. □

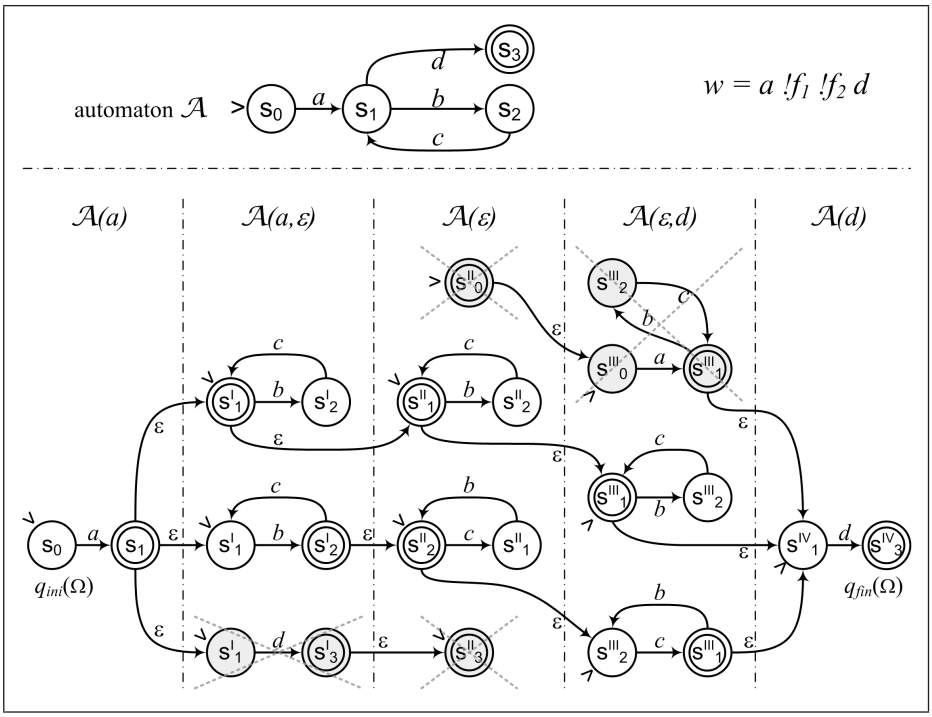


Figure 6: A perfect automaton (construction)

THEOREM 13. *Let $w(!f_n)$ be a distributed string and $\mathcal{A}$ be an NFA compatible with w. There is a perfect typing for w and $\mathcal{A}$ if and only if $w(\Omega_n) \equiv \mathcal{A}$. If so, the perfect typing is exactly (Ω_n).*

PROOF. ($\Rightarrow$) *if there is a perfect typing for w and $\mathcal{A}$ then* $w(\Omega_n) \equiv \mathcal{A}$. If w and $\mathcal{A}$ admit a perfect typing, say (τ_n), then (as it is also sound), by Theorem 11, $(\tau_n) \leq (\Omega_n)$. Suppose that $(\tau_n) < (\Omega_n)$ held. There would be (at least) an $i \in \{1, \ldots, n\}$ such that $\tau_i < \Omega_i$. In other words, there would be an automaton $\tau_i' \in Aut(\Omega_i)$ accepting some strings rejected by τ_i. Consider the typing $(\tau_n') \in Typ(\Omega)$ containing τ_i' in position i. By Lemma 10, (τ_n') is sound and then $\tau_i' \leq \tau_i$, by definition. But this is a contradiction. Therefore $(\tau_n) \equiv (\Omega_n)$ and then $w(\Omega_n) \equiv \mathcal{A}$, as (τ_n) is also local.

($\Leftarrow$)*if $w(\Omega_n) \equiv \mathcal{A}$ then there is a perfect typing for w and $\mathcal{A}$.* This is true since (Ω_n) is local and because, by Theorem 11, $(\tau_n) \leq (\Omega_n)$ for any sound typing (τ_n). □

The following two examples show that if there exists a local typing (τ_n) for w and $\mathcal{A}$, then $(\tau_n) < (\Omega_n)$ might hold. This can happen even if (τ_n) is a unique maximal local.

Example 6. Consider the string $w = a\ !f_1\ c\ !f_2\ e$ and the regular expression $\tau = abccde$ compatible with w. Clearly, the typing (b, cd) is local (sound and complete) for w and τ because $w(b, cd) \equiv \tau$. Nevertheless, $(\Omega_2) = (bc?,\ c?d)$ is (strictly) greater then (b, cd) since $[bc?] = \{b,\ bc\} \supset \{b\}$ and $[c?d] = \{d,\ cd\} \supset \{cd\}$. □

Example 7. Let $w = a\ !f_1\ !f_2\ d$ be a distributed string and τ be the regular expression $a(bc)^*d$. Clearly, the typing $((bc)^*, (bc)^*)$ is *local* (also unique maximal local but not *perfect*). But, as consequence of construction of perfect automaton, we have: $Aut(\Omega_1) = \{(bc)^*, (bc)^*b\}$ and $Aut(\Omega_2) = \{(bc)^*, c(bc)^*\}$. Consequently, $\Omega_1 \equiv ((bc)^*b?)$ and $\Omega_2 \equiv (c?(bc)^*)$ do not represent a sound (and hence local) typing since they allow strings such as *abccbcd* or *abcbbcd* that are not accepted by τ. □

The following example shows that even if there is no local typing for w and τ, then $\Omega \equiv \tau$ may hold.

Example 8. Let a τ be the regular expression $ab + ba$ and $w = !f_1 !f_2$. There are two *sound typings*: (a, b) and (b, a), but there is no local typing. However, $\Omega \equiv \tau$. □

We can now use the perfect automata construction to characterize the complexity of NFA-PERF. We use the next lemma:

LEMMA 14. *Let $w(!f_n)$ be a distributed string and $\mathcal{A}$ be a k-state NFA. The algorithm for building the perfect automaton $\Omega(\mathcal{A}, w)$ works in polynomial time.*

PROOF. Any set $\mathcal{A}(w_i)$ or $\mathcal{A}(w_{i-1}, w_i)$ contains at most k^2 automata each of which having size $\mathcal{O}(k)$. Therefore, the number of macro-iterations of the algorithm are $\mathcal{O}(nk^2)$, while the size of Ω is $\mathcal{O}(nk^3)$. For each w_i, the sets $Ini(\mathcal{A}, w_i)$ and $Fin(\mathcal{A}, w_i)$ can be obtained in nondeterministic logarithmic space (thus in polynomial time) because for any pair of states q_1, q_2 in $\mathcal{A}$, we check if the string w_i is in the language $[\mathcal{A}(q_1, q_2)]$. Finally, all the automata in $\mathcal{A}(w_i)$ and $\mathcal{A}(w_{i-1}, w_i)$ are nothing else but different copies of $\mathcal{A}$ having different initial and finial states. □

Now, we have:

THEOREM 15. NFA-PERF *is in* ***PSPACE****. So it is also* ***PSPACE****-complete by Theorem 5.*

PROOF. Let $w(!f_n)$ be a distributed string, τ be an NFA, and (τ_n) be a typing. Construct the perfect automaton $\Omega(\tau, w)$. By Lemma 14, Ω can be built in polynomial time w.r.t. $|\tau| + \|w\|$. Then, check in polynomial space if $w(\Omega_n) \equiv \tau \equiv w(\tau_n)$. □

And w.r.t. finding a perfect typing (if it exists), we have:

THEOREM 16. ∃NFA-PERF *is in* ***PSPACE****. So it is also* ***PSPACE****-complete by Theorem 8.*

PROOF. Let $\langle \tau, w(!f_n)\rangle$ be a (string) design. Construct the perfect automaton $\Omega(\tau, w)$. By Lemma 14, Ω can be built in polynomial time w.r.t. $|\tau| + \|w\|$. Then, check if $w(\Omega_n) \equiv \tau$, which is feasible in polynomial space. □

Additional properties. We now shows how to exploit perfect automaton properties to find (maximal) sound typings when a design does not allow any perfect. Clearly, this technique can be used for seeking (maximal) local typings as well (it leads to our 2-**EXPSPACE** upper bound.) Let $w(!f_i) = w_0!f_1w_1 \ldots !f_nw_n$ be a distributed string and $\mathcal{A}$ be a type defined by an NFA compatible with w. All the automata belonging to $Aut(\Omega_i)$ can be decomposed in at most $2^{|Aut(\Omega_i)|} - 1$ different automata such that there are no two of them accepting the same string. In particular, this new set is denoted by $Dec(\Omega_i)$ and defined as follows:

$$\{\cap\mathbb{A}_1 \mid \cup\mathbb{A}_2 \ : \ \emptyset \neq \mathbb{A}_1 \subseteq Aut(\Omega_i),\ \mathbb{A}_2 = Aut(\Omega_i) \mid \mathbb{A}_1\}$$

An example for three automata is given in Figure 7. Finally,

$$Dec(\Omega) = \{(D_1, \ldots D_n) : D_i \in Dec(\Omega_i)\}$$

is the set of all different typings from $Dec(\Omega_1)\times\ldots\times Dec(\Omega_n)$. Given a typing (τ_n), we say that $(\tau_n) \in Dec(\Omega)$ if there exists a sequence $(D_n) \in Dec(\Omega)$ such that $\tau_i \equiv D_i$, for each i.

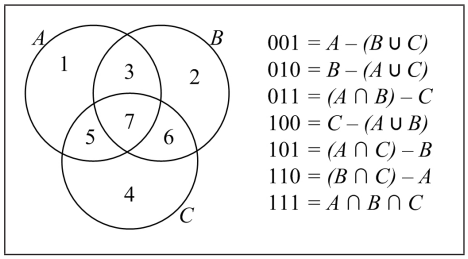


Figure 7: Partitioning of (three) sets and enumeration of the parts

Given a type $\tau \leq \Omega_i$ for some $i \in \{1, \ldots, n\}$, $Dec(\tau, i) = \{\tau \cap \tau' : \tau' \in Dec(\Omega_i)\}$ denotes the partition of τ, namely $\cup Dec(\tau, i) \equiv \tau$, obtained by its projection on $Dec(\Omega_i)$. Let (τ_n) be any typing for a distributed string $w(!f_n)$. Given a string $u \in \Sigma_L^*$ and an $i \in \{1, \ldots, n\}$, then $(\tau_n)_{[\tau_i|u]}$ denotes the new typing obtained from (τ_n) by replacing τ_i with the minimum NFA accepting only string u. In particular $[w(\tau_n)_{[\tau_i|u]}]$ is defined as $\{w_0\sigma_1w_1 \ldots \sigma_nw_n \ : \ \sigma_i = u, \sigma_j \in [\tau_j]\ \forall j \neq i\}$ and clearly,

$$w(\tau_n) \quad \equiv \bigcup_{u\in[\tau_i]} w(\tau_n)_{[\tau_i|u]}$$

We define now, *extension*, of (τ_n) a new typing obtained from (τ_n) by replacing τ_i with new type $(\tau_i \cup \tau)$, and denoted by $(\tau_n)_{[\tau_i\cup\tau]}$. In particular,

$$w(\tau_n)_{[\tau_i\cup\tau]} \quad \equiv \bigcup_{u\in[\tau_i\cup\tau]} w(\tau_n)_{[\tau_i|u]}$$

Clearly, if $\tau \leq \tau_i$, then $(\tau_n) \equiv (\tau_n)_{[\tau_i\cup\tau]}$. Otherwise $(\tau_n) < (\tau_n)_{[\tau_i\cup\tau]}$.

LEMMA 17. *Let (τ_n) be a sound typing for a string $w(!f_n)$ and an NFA $\mathcal{A}$ compatible with w. For each $i \in \{1, \ldots, n\}$, then $(\tau_n)_{[\tau_i\cup\tau]}$ is still sound if τ is an element of $Dec(\Omega_i)$ that is incomparable with τ_i.*

PROOF. In order to prove the statement, we show that $w(\tau_n)_{[\tau_i|u]} \leq \Omega$ holds for each $u \in [\tau] \setminus [\tau_i]$ (recall that, by Lemma 9, $\Omega \leq \mathcal{A}$). By definition, $\tau \nsim \tau_i$ entails that there is (at least) a string accepted by both τ_i and τ. Let u' be any of these strings. Since, by Theorem 11, $\tau_i \leq \Omega_i$, then there is a nonempty set $\mathbb{A} \subseteq Aut(\Omega_i)$ containing all-and-only the automata accepting u'. Clearly, since $\tau \in Dec(\Omega_i)$ and $u' \in [\tau]$, then τ is also in $Dec(\tau', i)$ for each $\tau' \in \mathbb{A}$. This means that each string $u \in [\tau] \setminus [\tau_i]$ is accepted by all-and-only the automata in $\mathbb{A}$ as well. By Theorem 11, $w(\tau_n) \leq \Omega$, and in particular $w(\tau_n)_{[\tau_i|u']} \leq \Omega$, as $u' \in [\tau_i \cap \tau]$. In other words, any string in $[w(\tau_n)_{[\tau_i|u']}]$ is accepted by (at least) a sequence of automata in $Seq(\Omega)$. Finally, as both u and u' are recognized by all-and-only the automata in $\mathbb{A}$, then each string $w_0\sigma_1w_1 \ldots \sigma_nw_n$ in $[w(\tau_n)_{[\tau_i|u]}]$ (with $\sigma_i = u$) has a twin in $[w(\tau_n)_{[\tau_i|u']}]$ (with $\sigma_i = u'$) and both of them are accepted by exactly the same sequences in $Seq(\Omega)$. □

THEOREM 18. *Let (τ_n) be a maximal typing for a distributed string $w(!f_n)$ and an NFA $\mathcal{A}$ compatible with w. Then for each i, $Dec(\tau_i, i) \subseteq Dec(\Omega_i)$.*

PROOF. Let i be an index arbitrarily fixed in $\{1, \ldots, n\}$. As (τ_n) is maximal then, by definition, it is sound and, by Theorem 11, $\tau_i \leq \Omega_i$. Let D_i be a copy of $Dec(\Omega_i)$. Then $\tau_i \leq \cup D_i$. Remove, now, all automata in D_i independent from τ_i (if there was someone). Still, $\tau_i \leq \cup D_i$ holds. Hence, consider the two possible (and alternative) cases:

$$(1)\ \ \tau_i \equiv \cup D_i \quad \textit{or} \quad (2)\ \ \tau_i < \cup D_i$$

In the first case the theorem is already proven. While, in the latter case, there is (at least) an automaton $\tau \in D_i$ incomparable with τ_i entailing relation $(\tau_n) < (\tau_n)_{[\tau_i\cup\tau]}$. But since $(\tau_n)_{[\tau_i\cup\tau]}$ is still sound (see Lemma 17), then there is a contradiction because (τ_n) is assumed to be maximal. □

6. TREE AUTOMATA TYPING

We consider in this section some more powerful typings based on nondeterministic bottom-up tree automata. This is typically the largest class considered for XML documents. In this setting, the direct reduction to words cannot be used. However, we show how to solve the problem using tree automata techniques.

nUTAs. A *nondeterministic Unranked (bottom-up) Tree Automaton* (nUTA) over alphabet Σ is a quadruple $\mathcal{A} = \langle K, \Sigma, \Delta, F\rangle$, where K is a finite set of *states*, $F \subseteq K$ is the set of *final states*, and Δ, the *transition relation*, is a finite set of rules of the form $a(M) \to q$ where M is an NFA over alphabet K, namely $[M] \subseteq K^*$. Without loss of generality, we consider *normalized* nUTAs where for each $a \in \Sigma$ and each $q \in K$, there is at most one transition rule of the form $a(M) \to q$. A tree t belongs to $[\mathcal{A}]$ if and only if there is a mapping μ from the nodes of t to K such that: (i) $\mu(r) \in F$ if r is the root of t; (ii) for each node x of t with label a and with children $y_1 \ldots y_k$, there is a rule $a(M) \to \mu(x)$ in Δ and the string $\mu(y_1) \ldots \mu(y_k)$ belongs to $[M]$.

nUTA-based distributed design. Given a distributed document $T(!f_n)$ and a type τ defined by an nUTA. A typing for T and τ is still a positional mapping from the functions in $(!f_n)$ to a sequence (τ_n) of types (here nUTAs). Recall that the trees of type τ_i have their roots labeled by a unique label s_i; see the definition of distributed document typing in Section 2. We furthermore assume w.l.o.g. that each of these nUTA τ_i has a single accepting state, say q_i, so by construction includes a single rule of the form $s_i(M_i) \to q_i$. The new type $T(\tau_n) = \langle K, \Sigma_L, \Delta, F\rangle$ is obtained as follows:

1. Σ_L is the union of the element names in $T(!f_n)$ and of the labels in each type τ_i;
2. K contains a state q^x for each node x in T labeled in Σ_L and all the states in each type τ_i;
3. $F = \{q^r\}$ where r is the root of T;
4. For each τ_i, Δ contains all the rules of τ_i except for its rule of the form $s_i(M_i) \to q_i$;
5. For each node x of T with label $a \in \Sigma_L$, Δ contains a rule of the form $a(M^x) \to q^x$ where M^x is an NFA accepting:
   - only the empty string if x is a leaf node;
   - the language $L_1 \dots L_k$ if x is a non-leaf node with children $y_1 \dots, y_k$ where: (i) each $L_j = \{q^{y_j}\}$ if $\mathsf{lab}(y_j)$ is in Σ_L; (ii) each $L_j = [M_i]$ if $\mathsf{lab}(y_j) = f_i$ and $s_i(M_i) \to q_i$ is the (unique) rule in τ_i having q_i as final state.

THEOREM 19. *[21]+[23] n*UTA-EQUIV *is **EXP**-complete.*

THEOREM 20. *Problems n*UTA-LOC*, n*UTA-ML*, n*UTA-PERF *are **EXP**-hard.*

PROOF. It follows by slightly modifying the proof of Theorem 5 where $\mathcal{A}$ and $\mathcal{A}_1$ are here replaced by nUTAs. The logspace transformation φ does not change, namely it works in such a way that nUTA-EQUIV $\leq_m^{\mathbf{L}}$ nUTA-LOC. □

THEOREM 21. *n*UTA-LOC *is **EXP**-complete.*

PROOF. **(Membership)** Let $T(!f_n)$ be a distributed string, τ be an nUTA, and (τ_n) be a typing. Build $T(\tau_n)$ (in polynomial time) and check in exponential time if $T(\tau_n) \equiv \tau$.
(Hardness) By Theorem 20. □

THEOREM 22. *n*UTA-ML *is **EXP**-complete.*

PROOF. It follows by slightly modifying the proof of Theorem 7 where τ is an nUTA and w is a tree T. We have proved that locality is feasible in **EXP**. Moreover, as τ_i is nondeterministic, nUTA $\bar{\tau}_i$ accepting the complement of language $[\tau_i]$ may be exponentially larger in size. For finite tree automata the nonemptiness problem is **P**-complete [25]. We can avoid the materialization of $T(\tau'_n) \cap \tau$ with an "on-the-fly" construction of the *direct product* of $T(\tau'_n)$ and τ. Hence, a polynomial time algorithm on a non-materialized (single) exponential automaton leads to **EXP**. □

Perfect tree automaton. We next extend the construction of the perfect automaton for words to nUTAs. Perfect typings may be obtained using this construction.

Let $\mathcal{A} = \langle K, \Sigma, \Delta, F\rangle$ be an nUTA and $T(!f_n)$ be a distributed document on alphabet $\Sigma_L \cup \Sigma_F$. Since we consider normalized nUTAs, we denote by $M(a, q)$ the NFA in the rule $a(M) \to q$ for each $a \in \Sigma$ and $q \in K$.

Step 1. **For each** node x of T having label $a \in \Sigma_L$ **do**:
Step 1.1. Consider the set of all rules in Δ of the form

$$a(M_1) \to q_1, \dots, a(M_k) \to q_k$$

Step 1.2. **Guess** a subset of states $S^x \subseteq \{q_1, \dots, q_k\}$.
Step 1.3. Build for node x the NFA

$$\mathcal{M}^x = \bigcup_{q \in S^x} M(a, q)$$

Step 2. **For each** node x of T having label $a \in \Sigma_L$ **do**:
Step 2.1. Consider the labels of the children of x

$$w^x = \mathsf{labcdrn}(x) = w_0 !f_1 w_1 \dots !f_h w_h \quad (h \geq 0)$$

Step 2.2. **For each** $w_i = w_i[1], \dots, w_i[\ell]$ build the set $\mathcal{M}^x(w_i)$ of automata as follows. An automaton $\mathcal{M}^x(p_0, p_\ell)$ is in $\mathcal{M}^x(w_i)$ if $\mathcal{M}^x$ allows a sequence of transitions

$$(p_0, q_1, p_1), \dots, (p_{\ell-1}, q_\ell, p_\ell)$$

such that $q_j \in S^{y_j}$ where y_j is the node of T associated to the label $w_i[j]$, for each $j \in \{1, \dots, \ell\}$.

Step 2.3. Build the perfect automaton Ω^x for $\mathcal{M}^x$ and w^x as shown for words, by using the new definition of $\mathcal{M}^x(w_i)$.

Step 2.4. Construct the type $w^r(\Omega_h^x)$ accepting the language $L_0\ [\Omega_1^x]\ L_1\ \dots\ [\Omega_h^x]\ L_h$ such that L_i is the language $S^{y_1}\ \dots\ S^{y_\ell}$ and where any y_j is the node of T associated to the label $w_i[j]$.

7. CONCLUSION

As explained in the introduction, this work can serve as a basis for designing the distribution of a document. It would be interesting to extend to richer Web data. First, this would involve graph data and not just tree data. Then one should consider unordered collections and functional dependencies as in the relational model. Other dependencies and in particular inclusion dependencies would also clearly make sense in this setting.

Database design has a long history, see most database text book. Distributed database design has also been studied since the early days of databases, but much less, because distributed data management was limited by the difficulty to deploy distributed databases. The techniques that were developed, e.g., vertical and horizontal partitioning, are very different from the ones presented here because we focus on ordered trees and collections are not ordered in relational databases. We believe that traditional database studies even on mainly theoretical topics such as normal forms are also relevant in a Web setting. An interesting direction of research is to introduce some of these techniques in our setting.

In the paper, the focus was on local typing that forces verification to be purely local. More generally, it would also be interesting to consider typings of the resources that would minimize the communications needed for type checking (and not completely avoid them.) Also, the tree automata that are considered in Section 6 are nondeterministic. For performance reasons, typing is typically performed by much more restricted automata that are not only deterministic but furthermore unambiguous [4]. Finally, it would be interesting to study the effect of such restrictions on checking properties of distributed typings, consider cases where a distributed document T may change from time to time by adhering to some global DTD which uses function symbols in the DTD itself, as well as to study the impact of distributed typing (as studied here) on query optimization.

8. ACKNOWLEDGMENTS

Serge Abiteboul's work was supported by the European Research Council Advanced Grant Webdam and by the French ANR Grant Docflow. Georg Gottlob's work was supported by EPSRC grant EP/E010865/1 "Schema Mappings and Automated Services for Data Integration and Exchange". Gottlob also gratefully acknowledges a Royal Society Wolfson Research Merit Award. Marco Manna acknowledges the support and hospitality of the Oxford-Man Institute of Quantitative Finance, where he worked as an Academic Visitor. Finally, the authors also want to thank the reviewers for very useful comments and suggestions that helped them to noticeably improve the quality of this paper.

9. REFERENCES

[1] S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *The VLDB Journal*, 17(5):1019–1040, 2008.

[2] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD int. conference on Management of data*, pages 527–538, 2003.

[3] S. Abiteboul, I. Manolescu, and E. Taropa. A framework for distributed XML data management. In Y. E. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Böhm, A. Kemper, T. Grust, and C. Böhm, editors, *EDBT*, volume 3896 of *Lecture Notes in Computer Science*, pages 1049–1058. Springer, 2006.

[4] S. Abiteboul, T. Milo, and O. Benjelloun. Regular rewriting of Active XML and unambiguity. In *Symposium on Principles of database systems*, 2005.

[5] J.-M. Bremer and M. Gertz. On distributing XML repositories. In V. Christophides and J. Freire, editors, *WebDB*, pages 73–78, 2003.

[6] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. *Comput. Netw.*, 33(1-6):137–157, 2000.

[7] S. Ceri, B. Pernici, and G. Wiederhold. An overview of research in the design of distributed databases. *IEEE Database Eng. Bull.*, 7(4):46–51, 1984.

[8] J. Clark and M. Murata. *RELAX NG Specification.* OASIS, 1 edition, December 2001.

[9] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *J. Algorithms*, 8(5):385–394, 1987.

[10] G. Ghelli, D. Colazzo, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. In M. Arenas and M. I. Schwartzbach, editors, *DBPL*, volume 4797 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2007.

[11] P. Grosso and D. Veillard. XML fragment interchange. Internet Publication, Feb 2001. W3C Candidate Recommendation 12 February 2001.

[12] C. Hagenah and A. Muscholl. Computing epsilon-free NFA from regular expressions in o(n $\log^2$(n)) time. In *MFCS '98: Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science*, pages 277–285, London, UK, 1998. Springer-Verlag.

[13] L. O. Hernández and M. Pegah. WebDAV: what it is, what it does, why you need it. In *SIGUCCS '03: Proceedings of the 31st annual ACM SIGUCCS conference on User services*, pages 249–254, New York, NY, USA, 2003. ACM.

[14] M. Holzer and M. Kutrib. State complexity of basic operations on nondeterministic finite automata. In J.-M. Champarnaud and D. Maurel, editors, *CIAA*, volume 2608 of *Lecture Notes in Computer Science*, pages 148–157. Springer, 2002.

[15] J. Hromkovic, S. Seibert, and T. Wilke. Translating regular expressions into small epsilon-free nondeterministic finite automata. In *STACS '97: Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science*, pages 55–66, London, UK, 1997. Springer-Verlag.

[16] H. V. Jagadish, L. V. S. Lakshmanan, T. Milo, D. Srivastava, and D. Vista. Querying network directories. *SIGMOD Rec.*, 28(2):133–144, 1999.

[17] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In J. Fiala, V. Koubek, and J. Kratochvíl, editors, *MFCS*, volume 3153 of *Lecture Notes in Computer Science*, pages 889–900. Springer, 2004.

[18] M. T. Özsu and P. Valduriez. Distributed database systems: Where are we now? *Computer*, 24(8):68–78, 1991.

[19] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS '00: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 35–46, New York, NY, USA, 2000. ACM.

[20] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[21] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990.

[22] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9, New York, NY, USA, 1973. ACM.

[23] D. Suciu. Typechecking for semistructured data. In *DBPL '01: Revised Papers from the 8th International Workshop on Database Programming Languages*, pages 1–20, London, UK, 2002. Springer-Verlag.

[24] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures second edition. Internet Publication, Oct 2004. Recommendation, World Wide Web Consortium, Boston, Tokyo, Sophia Antipolis.

[25] M. Veanes. On computational complexity of basic decision problems of finite tree automata. Technical Report 133, Uppsala Programming Methodology and Artificial Intelligence Laboratory, Sweden, Jan. 1997.