Appeared in **Journal of Logic and Computation**, 9(3):295–318, 1999

Representing and Reasoning on XML Documents: A Description Logic Approach

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{calvanese,degiacomo,lenzerini}@dis.uniroma1.it

Abstract

Recent proposals to improve the quality of interaction with the World Wide Web suggest considering the Web as a huge semistructured database, so that retrieving information can be supported by the task of database querying. Under this view, it is important to represent the form of both the network, and the documents placed in the nodes of the network. However, the current proposals do not pay sufficient attention to represent document structures and reasoning about them. In this paper, we address these problems by providing a framework where Document Type Definitions (DTDs) expressed in the eXtensible Markup Language (XML) are formalized in an expressive Description Logic equipped with sound and complete inference algorithms. We provide methods for verifying conformance of a document to a DTD in polynomial time, and structural equivalence of DTDs in worst case deterministic exponential time, improving known algorithms for this problem which were double exponential. We also deal with parametric versions of conformance and structural equivalence, and investigate other forms of reasoning on DTDs. Finally, we show how to take advantage of the reasoning capabilities of our formalism in order to perform several optimization steps in answering queries posed to a document base.

Keywords: knowledge representation, automated reasoning, description logics, XML, SGML

1 Introduction

The view of the World Wide Web as a large information system constituted by a collection of documents connected by hypertext links, is stimulating many lines of research related to Knowledge Representation and Databases [25, 32, 26, 28]. One of the most interesting aspects addressed in recent papers is the design of suitable mechanisms for querying the World Wide Web information system. While the basic mechanism for retrieving information in this context is browsing and/or searching by keywords, several authors point out that some form of declarative query formulation would greatly improve the effectiveness of the interaction with the Web (see for example [32]).

One possibility of pursuing the goal of querying the World Wide Web is to consider the Web as a huge semi-structured database, so that retrieving information can be supported by the traditional task of database querying. The result of the query could be some representation of the portion of the Web containing the information of interest: from such a portion, further interaction may start, possibly based on browsing and searching. This framework is adopted, for instance, in [26, 28, 32]. One important assumption of these approaches is that the query process operates on the basis of a representation of the structure of the network. For example, the query language WebSQL [32] considers the underlying database as constituted by suitable virtual relations describing the Web in terms of its nodes and its hypertext links.

The current proposals based on the above ideas, however, do not pay sufficient attention to the problem of representing the structure of documents placed in the nodes of the network. Representing such structural aspects, and having the ability to reason about them, would help in several tasks related to query

processing, such as query formulation, optimization and restructuring [15, 35, 21, 32, 23]. The role paid by the information on both document and link structures corresponds to the one paid by the schema and the associated constraints in a traditional database system. In this sense, it is important to study suitable mechanisms for reasoning about the representation of structural aspects. This reasoning facility is the analogue of the schema level reasoning techniques in the traditional database setting (constraint inference in relational database, inheritance and subtyping inference in object-oriented databases, etc.), and enables to improve both the precision of the information retrieved, by providing flexible additional selection criteria, and the efficiency of the retrieval process, by allowing for retrieving just a short description of a large document to decide its relevance, instead of the document itself [31, 29, 30, 15, 35, 32].

In order to address the issue of devising more sophisticated forms of representation and reasoning about document structures, one must take into account that documents in the World Wide Web are described by means of ad-hoc languages. Indeed, the structure of a document is typically made explicit by using special tags to mark its various parts. One of the most prominent formalisms for defining marked-up documents is the *eXtended Markup Language* (XML) [8], which is a specialization of the *Standard Generalized Markup Language* (SGML) [24]. In XML, the structure of marked-up documents is described by means of *Document Type Definitions* (DTDs) which assert the set of "rules" that each document of a given document type must conform to. Such rules can be formalized by means of Extended Context Free Grammars (ECFGs), in such a way that marked-up documents that are instances of a DTD are seen as syntax trees of the corresponding grammar [42]. It is worth noticing that XML DTDs have been used to define wide range of document types, from very general ones, such as generic HTML documents, to very specific ones, e.g. a specific form of email messages.

Several types of reasoning about DTDs are of interest for the purpose of supporting query processing over a document base. Given two DTDs, a natural and fundamental question is whether they are equivalent in some sense [42, 36]. Under the above formalization of DTDs as ECFGs, this question can be reformulated in terms of checking various forms of equivalence between grammars. In particular, checking *strong equivalence* of DTDs, i.e. whether two DTDs define the same sets of documents, can be effectively done by checking whether the two corresponding grammars generate the same sets of syntax trees. Open problems concerning reasoning on DTDs are pointed out in [42], such as:

1. Find algorithms and study the computational properties of structural equivalence, which is a weaker form of equivalence abstracting from tag names in the documents.

2. Determine meaningful variants of structural equivalence, and study their computational complexity.

The goal of this paper is to demonstrate that expressive Description Logics are well suited to represent and reason about the structure of documents. Specifically, we provide the following contributions:

- We present a formalization of XML DTDs in terms of an expressive Description Logic, called $\mathcal{DL}$, equipped with sound, complete, and terminating inference procedures. This logic includes non-first-order constructs, such as reflexive-transitive closure and well-foundedness, which play a crucial role in the formalization. The inference procedures for $\mathcal{DL}$ provide us with a general reasoning mechanism that enables reasoning tasks on DTDs to be effectively carried out. These include the verification of typical forms of equivalences between DTDs [42, 36], such as strong equivalence, structural equivalence, and parametric versions of equivalence. Notably, this general reasoning mechanism allows for verifying structural equivalence in worst case deterministic exponential time, in contrast to the known algorithms which are double exponential.

- We illustrate a method for retrieving a set of documents from a document base, that takes advantage of the reasoning capabilities of $\mathcal{DL}$. Documents are retrieved by means of queries that ask for all documents conforming to a given structure. Reasoning is exploited in order to devise several optimization strategies that improve upon the brute force approach of scanning the entire document base and checking, for every document instance, whether it satisfies the query.

The paper is organized as follows. In Section 2, XML DTDs and documents are introduced, and the basic reasoning tasks on DTDs are defined. In Section 3, the Description Logic $\mathcal{DL}$ is presented. In Section 4, the formalization of DTDs and related reasoning tasks within $\mathcal{DL}$ is developed. In Section 5, we address

the problem of answering queries posed to a document base. In Section 6, we discuss possible extensions of our approach, and compare our proposal with recent work on modeling semi-structured data. Finally, conclusions are drawn in Section 7.

2 The eXtended Markup Language (XML)

The *eXtended Markup Language* is a specialization of the *Standard Generalized Markup Language* whose goal is to facilitate the processing of generic marked-up documents on the World Wide Web in a way that goes beyond what is now possible with HTML documents [8]. We focus only on aspects of XML and marked-up documents that are directly related to the document structure, abstracting with respect to additional features that are related to the physical representation of documents.

2.1 XML DTDs and Documents

XML describes marked-up documents, called *XML documents*, each of which can be considered a pair $(\mathbf{D}, d)$, where $\mathbf{D}$ is a *Document Type Definition (DTD)* and d is the *document instance*. The document instance is made up of units, called *elements*, which denote the logical components of the document and are delimited by *marking tags*. The DTD specifies the logical structures, and hence the markups, that are admissible, in terms of a set of *element type definitions*. In an XML document $(\mathbf{D}, d)$ the document instance d has to conform to the DTD $\mathbf{D}$, according to the definition of conformance provided below.

We start by describing the form of document instances independently of the particular DTD the document instance may conform to. We assume to deal with two alphabets $\mathbf{T}$ of *terminals* and $\mathbf{E}$ of *element types*. To each element type $E \in \mathbf{E}$ we associate a *start tag* <E> and an *end tag* </E>.

Definition 1 The set $docs_{\mathbf{T},\mathbf{E}}$ of all possible document instances that can be built over $\mathbf{T}$ and $\mathbf{E}$ is defined inductively as follows:

- If d is a terminal in $\mathbf{T}$, then $d \in docs_{\mathbf{T},\mathbf{E}}$.
- If d is a sequence of the form <E>$d_1 \cdots d_k$</E>, where $E \in \mathbf{E}$ is an element type and $d_1, \ldots, d_k \in docs_{\mathbf{T},\mathbf{E}}$, then $d \in docs_{\mathbf{T},\mathbf{E}}$.

In the following we assume without loss of generality that the alphabets $\mathbf{T}$ of terminals and $\mathbf{E}$ of element types are fixed, and we denote the set $docs_{\mathbf{T},\mathbf{E}}$ simply by $docs$.

While the terminals in $\mathbf{T}$ are the basic types of XML, such as `#CDATA` and `#PCDATA`, which represent generic (unmarked) strings with no associated structure, the structure of elements corresponding to element types in $\mathbf{E}$ is specified by using DTDs. With the term *symbol*, denoted by the letter S, we mean an element in $\mathbf{T} \cup \mathbf{E}$, i.e., either a generic terminal or an element type.

Definition 2 A *Document Type Definition* (DTD) $\mathbf{D}$ is a pair $(\mathbf{P}, R)$, where $\mathbf{P}$ is a set of *element type definitions*, and $R \in \mathbf{E}$ is the *root element type*, i.e. the element type that specifies the *document type*. Each element type definition has the form $E \rightarrow \alpha$, where E is the defined element type, and α, called *content model*, is an expression over symbols in $\mathbf{T} \cup \mathbf{E}$ constructed according to the following abstract syntax:

$$\alpha ::= S \mid \mathit{empty} \mid \alpha_1|\alpha_2 \mid \alpha_1, \alpha_2 \mid \alpha? \mid \alpha^* \mid \alpha^+$$

i.e., α is a regular expression with *empty* denoting the empty string, "," denoting concatenation, and "|" denoting union, extended with both optional expressions ("?") and transitive closure ("+")[1]. In addition, XML content models may contain the construct *any* that stands for any sequence of elements types defined in the DTD. Formally, *any* is an abbreviation for $(E_1|\cdots|E_n)^*$, where $E_1, \ldots, E_n$ are all element types that appear in $\mathbf{P}$.

Consistently with XML, we assume that for each element type $E \in \mathbf{E}$, $\mathbf{P}$ contains at most one element type definition $E \rightarrow \alpha$ where E appears on the left hand side. We also assume that for each element type E appearing in $\mathbf{P}$, there is an element type definition $E \rightarrow \alpha$ in $\mathbf{P}$ in which E is the symbol on the left hand

```
<!DOCTYPE Mail [
  <!ELEMENT Mail      (From, To, (Subject)?, Body)>
  <!ELEMENT From      (Address)>
  <!ELEMENT To        (Address)+>
  <!ELEMENT Address   (#PCDATA)>
  <!ELEMENT Subject   (#PCDATA)>
  <!ELEMENT Body      (#PCDATA | any)>
]>
```

Figure 1: DTD M for mail documents

```
<Mail>
    <From>
        <Address> Dante@dsn.fi.it </Address>
    </From>
    <To>
        <Address> Beatrice@pitti.fi.it </Address>
        <Address> Virgilio@spqr.rm.it </Address>
    </To>
    <Subject> Appointment </Subject>
    <Body>
        Why don't we meet at disco.inferno at midnight.
        Tell also Caronte.  Cheers,
        - D.A.
    </Body>
</Mail>
```

Figure 2: A document instance conforming to the DTD in Figure 1

side. In fact, if such condition is not satisfied, the DTD can easily be transformed (in polynomial time) into one that generates the same set of document instances, and in which the condition holds.

Example 3 (Mail documents) Figure 1 shows an example of a DTD M for a simple mail document, expressed in XML syntax – it is straightforward to rephrase the element type definitions using the abstract syntax above.

Definition 4 The set $docs(\mathbf{P}, S)$ of document instances generated by a set of element type definitions $\mathbf{P}$ starting from a symbol S is inductively defined as follows:

- If S is a terminal F, then $docs(\mathbf{P}, F) = F$.
- If S is an element type E and $E \to \alpha \in \mathbf{P}$, then $docs(\mathbf{P}, E)$ is the set of sequences <E> $d_1 \cdots d_k$ </E>, where <E> and </E> are the start and end tags associated to E, and $d_1, \ldots, d_k$ are document instances generated by an instance of the content model α. Formally:

$$docs(\mathbf{P}, E) = \{\texttt{<}E\texttt{>}\, d_1 \cdots d_k \,\texttt{</}E\texttt{>} \mid \text{there exists a word } S_1 \cdots S_k \text{ generated by } \alpha \text{ such that } d_i \in docs(\mathbf{P}, S_i), \text{ for } i \in \{1, \ldots, k\}\}$$

The set $docs(\mathbf{D})$ of document instances generated by a DTD $\mathbf{D} = (\mathbf{P}, R)$ is given by $docs(\mathbf{P}, R)$. A document instance d *conforms* to a DTD $\mathbf{D}$ if $d \in docs(\mathbf{D})$.

Example 5 (Mail documents) Figure 2 shows a document instance conforming to the DTD in Figure 1.

[1] Observe that in XML the "&" operator of SGML is not allowed.

From a formal point of view, a DTD can be considered as an *Extended Context Free Grammar* (ECFG) [42], which is used to generate a set of *syntax trees* rather than a language. The set of element types and terminals are the nonterminal and terminal symbols of the ECFG, the root element type is the initial symbol, and the element type definitions are the production rules. Marked-up documents are seen as syntax trees constructed according to the grammar, where the tree structure is determined by the various tags that occur in the document and that constitute the markup.

2.2 Basic Reasoning Tasks on XML DTDs

Besides *conformance* defined above, several other reasoning tasks turn out to be useful when managing XML documents. A fundamental problem is to check various forms of equivalence between DTDs [42, 36]. Additionally, the possibility of checking inclusion, and disjointness between DTDs can be exploited to improve the efficiency of retrieving documents from a document base (see Section 5).

The most basic form of inclusion (equivalence, disjointness) is inclusion (equivalence, disjointness) of the sets of document instances conforming to the two DTDs.

Definition 6 Given two DTDs $\mathbf{D}_1$ and $\mathbf{D}_2$,

- $\mathbf{D}_1$ is *strongly included* in $\mathbf{D}_2$, denoted with $\mathbf{D}_1 \sqsubseteq_s \mathbf{D}_2$, if $docs(\mathbf{D}_1) \subseteq docs(\mathbf{D}_2)$;
- $\mathbf{D}_1$ is *strongly equivalent* to $\mathbf{D}_2$, denoted with $\mathbf{D}_1 \equiv_s \mathbf{D}_2$, if $docs(\mathbf{D}_1) = docs(\mathbf{D}_2)$;
- $\mathbf{D}_1$ is *strongly disjoint* from $\mathbf{D}_2$, denoted with $\mathbf{D}_1 \otimes_s \mathbf{D}_2$, if $docs(\mathbf{D}_1) \cap docs(\mathbf{D}_2) = \emptyset$.

For determining strong inclusion (equivalence, disjointness), the names of the start and end tags that constitute the markup of documents play a fundamental role.

In some cases, however, the actual names of the tags may not be relevant, while the document structure imposed by the tags is of importance. For example, if we simply want to check if two DTDs describe documents with the same level of nesting of tags, the names of the tags are irrelevant. The form of inclusion (equivalence, disjointness) obtained by ignoring the names of tags and considering only their positions is called *structural inclusion (equivalence, disjointness)* [42]. One DTDs is structurally included into another if, when we replace in every document conforming to the DTDs all start and end tags with the unnamed tags `<>` and `</>` respectively, the resulting sets of documents for the two DTDs are one included into the other. Similar definitions hold for structural equivalence and disjointness.

Structural equivalence of two DTDs is decidable, but the known algorithms take time doubly exponential in the size of the two DTDs [42][2].

While the restrictions imposed by strong inclusion (equivalence, disjointness) may be too strict in some cases, structural inclusion, which ignores completely all tag names, may be too weak. A natural generalization of these two concepts is obtained by considering a spectrum of possible inclusions, of which strong and structural inclusion are just the two extremes. The different forms of inclusion are obtained by considering certain tag names as equal, and others as different, when confronting documents. This allows us to parameterize inclusion of DTDs with respect to an equivalence relation on the set of tag names. For example, when checking equivalence of two DTDs, we may want to abstract from the difference between the names `enumerate` and `itemize`. This can be obtained by imposing that the two tags are equivalent, and by checking whether the DTDs enforce the same structure (for example a list of items) on the corresponding parts of the documents.

Formally, we consider an equivalence relation $\mathcal{R}$ on the set $\mathbf{E}$ of element types. For an element type $E \in \mathbf{E}$, we denote by $[E]_{\mathcal{R}}$ the equivalence class of E with respect to $\mathcal{R}$.

Definition 7 The set $docs_{\mathcal{R}}(\mathbf{P}, S)$ of *$\mathcal{R}$-document instances* generated by a set of element type definitions $\mathbf{P}$ starting from a symbol S is inductively defined as follows:

- If S is a terminal F, then $docs_{\mathcal{R}}(\mathbf{P}, F) = F$.

[2]This complexity bound holds if one does not consider the "&" operator of SGML (as in XML), which, if expanded may lead to an additional exponential blowup.

```
<!DOCTYPE Note [
  <!ELEMENT Note      (From, To, Text)>
  <!ELEMENT From      (Address)>
  <!ELEMENT To        (Address)>
  <!ELEMENT Address   (#PCDATA)>
  <!ELEMENT Text      (#PCDATA | any)>
]>
```

Figure 3: DTD N for note documents

- If S is an element type E and $(E \to \alpha) \in \mathbf{P}$, then

$$docs_{\mathcal{R}}(\mathbf{P}, E) = \{\texttt{<}E'\texttt{>}\, d_1 \cdots d_k \,\texttt{</}E'\texttt{>} \mid E' \in [E]_{\mathcal{R}}, \text{ and there exists a word } S_1 \cdots S_k \text{ generated by } \alpha \text{ such that } d_i \in docs_{\mathcal{R}}(\mathbf{P}, S_i), \text{ for } i \in \{1, \ldots, k\}\}$$

The set $docs_{\mathcal{R}}(\mathbf{D})$ of $\mathcal{R}$-document instances generated by a DTD $\mathbf{D} = (\mathbf{P}, R)$ is given by $docs_{\mathcal{R}}(\mathbf{P}, R)$. A document instance d *$\mathcal{R}$-conforms* to a DTD $\mathbf{D}$ if $d \in docs_{\mathcal{R}}(\mathbf{D})$.

Definition 8 A DTD $\mathbf{D}_1$ is *$\mathcal{R}$-included* in a DTD $\mathbf{D}_2$, denoted with $\mathbf{D}_1 \sqsubseteq_{\mathcal{R}} \mathbf{D}_2$, if $docs_{\mathcal{R}}(\mathbf{D}_1) \subseteq docs_{\mathcal{R}}(\mathbf{D}_2)$. *$\mathcal{R}$-equivalence*, denoted with $\equiv_{\mathcal{R}}$, and *$\mathcal{R}$-disjointness*, denoted with $\otimes_{\mathcal{R}}$, of two DTDs are defined in a similar way.

Example 9 (Note documents) The DTD N defining note documents, shown in Figure 3, is strongly disjoint from the DTD M defining mail documents, shown in Figure 1. However, if we abstract from the difference between the names `Mail` and `Note` and the names `Body` and `Text`, then a note is a special kind of mail. Indeed, it is easy to see that for any equivalence relation $\mathcal{R}$ containing the pairs (`Mail`, `Note`) and (`Body`, `Text`), we have that $N \sqsubseteq_{\mathcal{R}} M$.

As already mentioned strong inclusion and structural inclusion are just special cases of $\mathcal{R}$-inclusion (similarly for equivalence and disjointness). In fact, if we choose for $\mathcal{R}$ the equivalence relation in which all equivalence classes are singletons, we obtain strong inclusion (equivalence, disjointness). On the other hand, if $\mathcal{R}$ contains a single equivalence class constituted by the whole set $\mathbf{E}$, we obtain structural inclusion (equivalence, disjointness). Therefore in the following without loss of generality we consider $\mathcal{R}$-inclusion, $\mathcal{R}$-equivalence, and $\mathcal{R}$-disjointness only.

3 The Description Logic for Representing DTDs

We introduce the Description Logic $\mathcal{DL}$ that will be used in Section 4 to represent DTDs. In *Description Logics* (DLs) [33, 19], the domain of interest is modeled by means of *individuals*, *concepts*, and *roles*, denoting objects of the domain, unary predicates, and binary predicates respectively[3]. For the purpose of this paper we do not deal with the possibility of expressing knowledge about individuals, and therefore we conceive a DL as formed by three components:

- A *description language*, which specifies how to construct complex concept and role expressions (also called simply concepts and roles), by starting from a set of atomic symbols and by applying suitable constructs.
- A *knowledge specification mechanism*, which specifies how to construct a DL knowledge base, in which properties of concepts and roles are asserted.
- A set of *basic reasoning services* provided by the DL.

In the rest of the section we describe the specific form that these three components assume in $\mathcal{DL}$.

[3]More general Description Logics make also use of relations, which correspond to n-ary predicates [13].

Concepts C	**Syntax**	**Semantics**
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
universal quantif.	$\forall R.C$	$\{o \mid \forall o' \colon (o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
qual. number restr.	$(\leq n\ P.C)$	$\{o \mid \sharp\{o' \mid (o, o') \in P^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq n\}$
well-founded	$wf(R)$	$\{o_0 \mid \forall o_1, o_2, \ldots$ (ad infinitum) $\exists i \geq 0 \colon (o_i, o_{i+1}) \notin R^{\mathcal{I}}\}$
Roles R	**Syntax**	**Semantics**
atomic role	P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
union	$R_1 \cup R_2$	$R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}}$
concatenation	$R_1 \circ R_2$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}}$
refl. trans. closure	R^*	$(R^{\mathcal{I}})^*$
transitive closure	R^+	$(R^{\mathcal{I}})^+$
identity	$id(C)$	$\{(o, o) \mid o \in C^{\mathcal{I}}\}$

Table 1: Syntax and semantics of $\mathcal{DL}$ concept and role constructs.

3.1 The Description Language of $\mathcal{DL}$

In DLs, starting from a set of *atomic concepts* and *atomic roles*, one can build complex concepts and roles by applying certain *constructs*. It is the set of allowed constructs that characterizes a specific description language. The basic concept constructs encountered in DLs include the *boolean constructs*, denoted "$\sqcap$", "$\sqcup$", and "$\neg$", and interpreted as the corresponding set operations, and *universal* and *existential quantification over roles* [39]. For example, the concept $\texttt{Person} \sqcap \forall \texttt{child.Male} \sqcap \exists \texttt{child.Doctor}$, denotes the set of individuals that are instances of the concept `Person` and are connected through the role `child` only to instances of the concept `Male` and to some instance of the concept `Doctor`. Additionally, more expressive DLs include *number restrictions*, which allow for delimiting the number of times an object is connected to other objects via a certain role, and constructs on roles, such as intersection, or the possibility to construct arbitrary regular expressions over atomic roles [6, 38, 17].

In particular, the Description Logic $\mathcal{DL}$, that we use for formalizing DTDs, is a variant of the very expressive DLs studied in [12, 16, 11, 10, 18]. It includes besides the basic constructs mentioned above, also:

- a very general form of number restrictions, called *qualified number restrictions*, by means of which one can limit for an object o the minimum and maximum number of objects that are instances of a specified concept and that are connected to o via a role;
- the possibility to use roles which are constructed as regular expressions over atomic roles;
- a construct to denote the objects that are the initial point of a sequence of roles which is well-founded.

The full set of constructs of $\mathcal{DL}$ is shown in Table 1, where we denote atomic concepts by A, arbitrary concepts by C, atomic roles by P, and arbitrary roles by R, all possibly with subscripts. We also use the following abbreviations to increase readability: $\bot$ for $\neg\top$, $C_1 \sqcup C_2$ for $\neg(\neg C_1 \sqcap \neg C_2)$, and $\exists R.C$ for $\neg\forall R.\neg C$.

In DLs, the formal semantics is specified through the notion of interpretation. An *interpretation* $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *interpretation domain* and $\cdot^{\mathcal{I}}$ is an *interpretation function* that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, respecting the specific conditions imposed by the structure of the concept or role. $C^{\mathcal{I}}$ and $R^{\mathcal{I}}$ are called the *extension* of C and R respectively.

The semantics of the $\mathcal{DL}$ constructs, shown in Table 1 is quite standard, except for the construct $wf(R)$, called *well-founded*, which is interpreted as those objects that are the initial point of only finite R-chains. Notice that besides reflexive transitive closure of roles ("$*$"), $\mathcal{DL}$ includes also the "$+$" construct for transitive

closure, which turns out to be necessary for a characterization of DTDs in terms of DLs. Obviously one can eliminate "+" from any complex role expression by replacing any occurrence of R^+ with $R \circ R^*$. However, such a replacement may in the worst case lead to an exponential increase in the size of the role expression.

3.2 Knowledge Bases in $\mathcal{DL}$

A $\mathcal{DL}$ *knowledge base* is a set of assertions of the form:

$$C_1 \quad \sqsubseteq \quad C_2$$

where C_1 and C_2 are arbitrary $\mathcal{DL}$ concepts without any restrictions. We use $C_1 \equiv C_2$ as an abbreviation for the pair of assertions $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$.

An interpretation $\mathcal{I}$ *satisfies* the assertion $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. An interpretation is a *model* of a knowledge base $\mathcal{K}$ if it satisfies all assertions in $\mathcal{K}$. Thus, we adopt *descriptive semantics* for cyclic knowledge bases, i.e. knowledge bases in which the concept in the right hand side of an assertion refers (either directly or indirectly via other assertions) to some concept in the left hand side of the assertion [34, 7, 18].

3.3 Reasoning Services in $\mathcal{DL}$

The basic reasoning service in $\mathcal{DL}$ is *satisfiability* of a concept C in a knowledge base $\mathcal{K}$, denoted $\mathcal{K} \models C \not\equiv \bot$. It amounts to check whether $\mathcal{K}$ admits a model in which the extension of C is nonempty. Other reasoning services are *knowledge base satisfiability*, i.e. determining whether a knowledge base admits a model, and *subsumption*. Determining subsumption between two concepts C_1 and C_2 in a knowledge base $\mathcal{K}$, denoted $\mathcal{K} \models C_1 \sqsubseteq C_2$, amounts to check whether $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.

Both knowledge base satisfiability and subsumption can be immediately reduced to concept satisfiability as follows:

- A knowledge base $\mathcal{K}$ is satisfiable if and only if $\top$ is satisfiable in $\mathcal{K}$.
- A concept C_1 is subsumed by a concept C_2 in a knowledge base $\mathcal{K}$ if and only if $C_1 \sqcap \neg C_2$ is not satisfiable in $\mathcal{K}$.

Hence it is sufficient to consider concept satisfiability only.

Theorem 10 Concept satisfiability in $\mathcal{DL}$ is an EXPTIME-complete problem.

Proof Here we give only a sketch of the proof. The complete proof can be found in [11]. The proof exploits a correspondence between DLs and Propositional Dynamic Logics (PDLs) [22] established for the first time in [38] and extended successively to more expressive DLs and PDLs in [17, 12, 11]. In order to decide concept satisfiability in $\mathcal{DL}$ we polynomially reduce it to satisfiability in *repeat*-ADPDL. Repeat Automata Deterministic Propositional Dynamic Logic (*repeat*-ADPDL) [40, 41] is a variant of PDL in which all atomic programs (which correspond to atomic roles) are assumed to be deterministic (i.e., they correspond to globally functional roles), and in which complex programs are represented by means of finite automata over the language of atomic programs, rather than by regular expressions. Additionally, it contains the "repeat" construct over programs (that corresponds to the negation of the well-founded construct over roles).

Qualified number restrictions have no counterpart in PDLs, and therefore, we need to get rid of them in order to exploit the correspondence with PDLs. Following the construction in [17], we define a satisfiability preserving polynomial transformation of a knowledge base that eliminates qualified number restrictions. We introduce for each atomic role P, two new atomic roles F_P and G_P, which are globally functional, and replace every occurrence of $(\leq n\ P.C)$ in the knowledge base with:

$$\forall (F_P \circ G_P^* \circ (id(C) \circ G_P^+)^n).\neg C$$

where R^n denotes the concatenation of R repeated n times.

After this transformation the only roles that appear in the knowledge base are functional, and we are ready to establish a correspondence with *repeat*-ADPDL. In order to deal with the "+" operator for transitive

closure, we exploit the fact that the regular language over atomic roles described by a complex role expression R, can be encoded by means of a finite (non-deterministic) automaton whose size is polynomial in the size of R. Hence, we can encode each concept C of $\mathcal{DL}$ into a formula of *repeat*-ADPDL whose size is polynomial in the size of C.

The EXPTIME upper bound follows from decidability in deterministic exponential time of satisfiability in *repeat*-ADPDL [40, 20].

3.4 Representing Inductive Structures in $\mathcal{DL}$

With the rich language of $\mathcal{DL}$ one can represent and reason on a variety of inductive data structures, such as lists and trees. Generally speaking, the unrestricted form of assertions allows for expressing recursive structures, while the well-founded construct allows for imposing finiteness of paths of a specified form on the structures. The combination of the two can thus be used to define inductive structures in which both infinite and cyclic paths are ruled out. This feature will be important for correctly representing XML documents in $\mathcal{DL}$.

We illustrate the above ideas showing how to represent binary trees. Typically, the class of binary trees is defined inductively as the *smallest set* `BinTree` such that:

- every node with no `left`-successor and no `right`-successor (i.e., every node that is a `Leaf`) is a `BinTree`;
- every node having `left` and `right` successors which are `BinTree`s is a `BinTree`.

This inductive definition is captured by the concept `BinTree` in the following $\mathcal{DL}$ knowledge base:

$$\begin{aligned}
\texttt{Node} &\sqsubseteq (= 1\ \texttt{info}.\top) \\
\texttt{Leaf} &\equiv \texttt{Node} \sqcap \forall(\texttt{left} \cup \texttt{right}).\bot \\
\texttt{BinTree} &\equiv \texttt{Leaf} \sqcup (\texttt{Node} \sqcap \forall(\texttt{left} \cup \texttt{right}).\texttt{BinTree} \sqcap \\
&\qquad (\leq 1\ \texttt{left}.\top) \sqcap (\leq 1\ \texttt{right}.\top) \sqcap \\
&\qquad \mathit{wf}(\texttt{left} \cup \texttt{right}))
\end{aligned}$$

This knowledge base allows us to represent binary trees in the following sense. In every model of the knowledge base, if we consider any instance b of `BinTree`, then the set of objects and links that can be reached from b by following `left` and `right` links form a binary tree. Note that, in this way, a binary tree is actually identified by its root.

The concept `BinTree` is characterized by a recursive equation, in which the term `BinTree` on the left-hand side occurs also in the right-hand side of the equation. Let us remark the difference between a recursive equation of this form and an inductive definition: A recursive equation simply states a certain condition to be satisfied by its solutions, without specifying any selection criteria to choose among all possible solutions. An inductive definition instead, selects the smallest set satisfying the condition, and hence identifies a unique solution. The well-foundedness declaration in the right-hand side of the equation characterizing `BinTree` accomplishes this selection, making the recursive equation of `BinTree` equivalent to an inductive definition.

Once binary trees are represented in the above way they can be easily specialized by selecting for example the kind of information contained in certain nodes, e.g.:

$$\texttt{UrlTree} \sqsubseteq \texttt{BinTree} \sqcap \forall(\texttt{left} \cup \texttt{right})^*.(\neg\texttt{Leaf} \sqcup \exists\texttt{info}.\texttt{URL})$$

or additional structural constraints, such as a specific maximal depth, e.g.:

$$\texttt{DepthTwoTree} \sqsubseteq \texttt{BinTree} \sqcap \forall((\texttt{left} \cup \texttt{right}) \circ (\texttt{left} \cup \texttt{right})).\texttt{Leaf}$$

Obviously, recursively defined structures are taken into account like any other concept definition when reasoning about the knowledge base. Suppose for example that we define `UrlTree2` as the smallest set such that:

- every node that is a `Leaf` and points to an `URL` is an `UrlTree2`;
- every node having `left` and `right` successors which are `UrlTree2`s is an `UrlTree2`.

Such structure is captured by the following additional assertion:

$$\begin{aligned}\texttt{UrlTree2} \quad \equiv \quad & \texttt{Leaf} \sqcap \exists \texttt{info.URL} \sqcup (\texttt{Node} \sqcap \forall(\texttt{left} \cup \texttt{right}).\texttt{UrlTree2} \sqcap \\ & (\leq 1\ \texttt{left.}\top) \sqcap (\leq 1\ \texttt{right.}\top) \sqcap \\ & \mathit{wf}(\texttt{left} \cup \texttt{right}))\end{aligned}$$

One can verify that the knowledge base including all assertions so far correctly implies that `UrlTree2` is logically equivalent to `UrlTree`.

4 Representing and Reasoning over DTDs in DLs

We describe how to construct a $\mathcal{DL}$ knowledge base capable of fully capturing the structural aspects of DTDs. Without loss of generality we refer to a fixed alphabet $\mathbf{E}$ of element types, a fixed alphabet $\mathbf{T}$ of terminals, and a fixed equivalence relation $\mathcal{R}$ on the set $\mathbf{E}$ of element types.

4.1 Representation of DTDs

Given a DTD $\mathbf{D} = (\mathbf{P}, S_0)$, we define a $\mathcal{DL}$ knowledge base $\mathcal{K}$, called *characteristic knowledge base* of $\mathbf{D}$, as follows.

The alphabet of $\mathcal{K}$ includes the following atomic concepts and roles:

- the atomic concepts `Tag` and `Terminal`,
- for each terminal $F \in \mathbf{T}$, one atomic concept F,
- for each element type $E \in \mathbf{E}$, one atomic concept `Start`E and one atomic concept `End`E,
- for each element type definition $(E \rightarrow \alpha) \in \mathbf{P}$, one atomic concept $E_{\mathbf{D}}$,
- the atomic roles `f` and `r`.

The atomic concepts `Start`E and `End`E represent the tags of an element type E and are independent from the specific DTD. The atomic concept $E_{\mathbf{D}}$ associated to an element type definition contains the information about the DTD it belongs to and hence is specific to such a DTD. Indeed, when considering different DTDs $\mathbf{D}_1$ and $\mathbf{D}_2$, both containing an element type definition for the same element type E, the associated knowledge bases contain the distinct atomic concepts $E_{\mathbf{D}_1}$ and $E_{\mathbf{D}_2}$.

The set of assertions of the knowledge base $\mathcal{K}$ is constituted by three parts $\mathcal{K}_{\mathbf{T},\mathbf{E}}$, $\mathcal{K}_{\mathcal{R}}$, $\mathcal{K}_{\mathbf{D}}$, which are defined as specified below. We observe that in defining $\mathcal{K}$, we will not exploit the full power of qualified number restrictions in $\mathcal{DL}$. We refer the reader to Section 6 for a discussion on how qualified number restrictions can be used for capturing additional interesting characteristics of XML documents.

$\mathcal{K}_{\mathbf{T},\mathbf{E}}$: Encoding of general structural properties Our aim is that $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ captures the general structural properties of document instances. In particular, we want to enforce that every model of $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ represents a document instance d by means of a tree. Intuitively, the root of the tree represents the root element of d, and is connected by means of the roles `f` and `r` (standing for "first" and "rest" respectively) to the objects representing the tags associated to the root element and to those representing the components of d. More specifically, for a document instance d with h components, the start tag is represented by the `f`-filler of the root, the first component by the (`r`$\circ$`f`)-filler, the second component by the (`r`$\circ$`r`$\circ$`f`)-filler, the last component by the $\texttt{r}^h \circ \texttt{f}$-filler, for some $h > 0$, and the end tag by the $\texttt{r}^{h+1}$-filler. Document instances are by definition finite, and hence have a finite nesting of components. Since an infinite model of $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ or a model containing cycles would correspond to a document instance with infinite nesting, such models have to be ruled out. Therefore, we need to impose finiteness and acyclicity of all chains of objects connected by $\texttt{f} \cup \texttt{r}$. This can be done by means of the well-foundedness construct.

Figure 4 illustrates the structure of the tree representing a document instance of the form <E><E_1>$\cdots$</E_1>$\cdots$<E_h>$\cdots$</E_h></E>, according to the criteria specified above.

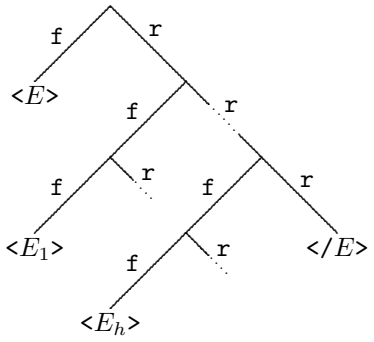

Figure 4: Tree representing the document <E><E1>···</E1>···<Eh>···</Eh></E>

Taking into account the above considerations, we can define $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ to be constituted by the following assertions:

$$
\begin{array}{rcll}
\top & \equiv & (\leq 1\ \mathtt{f}.\top) \sqcap (\leq 1\ \mathtt{r}.\top) \sqcap \mathit{wf}(\mathtt{f} \cup \mathtt{r}) & \\
\mathtt{Tag} & \sqsubseteq & \forall(\mathtt{f} \cup \mathtt{r}).\bot & \\
\mathtt{Terminal} & \sqsubseteq & \forall(\mathtt{f} \cup \mathtt{r}).\bot \sqcap \neg\mathtt{Tag} & \\
F & \sqsubseteq & \mathtt{Terminal} & \text{for each terminal } F \in \mathbf{T} \\
F_1 & \sqsubseteq & \neg F_2 & \text{for each pair of terminals } F_1, F_2 \in \mathbf{T} \text{ such that } F_1 \neq F_2 \\
\mathtt{Start}E & \sqsubseteq & \mathtt{Tag} & \text{for each element type } E \in \mathbf{E} \\
\mathtt{End}E & \sqsubseteq & \mathtt{Tag} & \text{for each element type } E \in \mathbf{E}
\end{array}
$$

Observe that $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ reflects the modeling of binary trees in $\mathcal{DL}$ as illustrated in Section 3.4, with the further requirement that in all models of $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ every object represents a binary tree. Indeed, the first assertion ensures us that in every model of $\mathcal{K}_{\mathbf{T},\mathbf{E}}$, every object is the root a tree in which every node has at most one `f` successor and one `r` successor. The second and the third assertions impose that every instance of `Tag` and `Terminal` are leaves of the tree, and that `Tag` and `Terminal` are disjoint. Finally, the other assertions specify that every instance of F is also an instance of `Terminal`, that two different terminals have disjoint instances, and that `Start`E and `End`E are subsets of `Tag`, for each $E \in \mathbf{E}$.

$\mathcal{K}_{\mathcal{R}}$: Encoding of the equivalence relation $\mathcal{R}$ Our aim is that $\mathcal{K}_{\mathcal{R}}$ fully captures the equivalence relation R. Therefore, $\mathcal{K}_{\mathcal{R}}$ should impose the disjointness of the concepts representing tags of element types belonging to different equivalence classes, and the equivalence of all the concepts representing tags of element types in the same equivalence class. Let $\{\{E_1^1, \ldots, E_{n_1}^1\}, \ldots, \{E_1^m, \ldots, E_{n_m}^m\}\}$ be the set of equivalence classes determined by $\mathcal{R}$. From the above observation it is easy to see that we reach our goal by defining $\mathcal{K}_{\mathcal{R}}$ to be constituted by the assertions:

$$
\begin{array}{rcll}
\mathtt{Start}E_1^i & \sqsubseteq & \neg\mathtt{Start}E_1^j & \\
\mathtt{End}E_1^i & \sqsubseteq & \neg\mathtt{End}E_1^j & \text{for } i, j \in \{1, \ldots, m\} \text{ and } i \neq j \\
\\
\mathtt{Start}E_i^j & \equiv & \mathtt{Start}E_{i+1}^j & \\
\mathtt{End}E_i^j & \equiv & \mathtt{End}E_{i+1}^j & \text{for } i \in \{1, \ldots, n_j - 1\} \text{ and } j \in \{1, \ldots, m\}
\end{array}
$$

In this way, when reasoning, the differences between the various tags associated to equivalent element types are ignored, coherently with the notion of $\mathcal{R}$-inclusion.

$\mathcal{K}_{\mathbf{D}}$: Encoding of the DTD The goal here is to define $\mathcal{K}_{\mathbf{D}}$ in such a way that it encodes the knowledge about the various element type definitions in $\mathbf{D} = (\mathbf{P}, R)$. In order to do so, for every E in $\mathbf{D}$, $\mathcal{K}_{\mathbf{D}}$ must impose suitable conditions on $E_{\mathbf{D}}$, so that in every model of $\mathcal{K}$, the instances of $E_{\mathbf{D}}$ represent the parts of document instances coherent with the definition of E in $\mathbf{D}$.

$$
\begin{array}{rcl}
\texttt{Mail}_M & \equiv & \exists \texttt{f.StartMail} \sqcap \exists(\texttt{r} \circ id(\exists \texttt{f.From}_M) \circ \texttt{r} \circ id(\exists \texttt{f.To}_M) \circ \texttt{r} \\
 & & \circ ((id(\exists \texttt{f.Subject}_M) \circ \texttt{r}) \cup id(\top)) \circ id(\exists \texttt{f.Body}_M) \circ \texttt{r}).\texttt{EndMail} \\
\texttt{From}_M & \equiv & \exists \texttt{f.StartFrom} \sqcap \exists(\texttt{r} \circ id(\exists \texttt{f.Address}_M) \circ \texttt{r}).\texttt{EndFrom} \\
\texttt{To}_M & \equiv & \exists \texttt{f.StartTo} \sqcap \exists(\texttt{r} \circ id(\exists \texttt{f.Address}_M) \circ \texttt{r} \circ (id(\exists \texttt{f.Address}_M) \circ \texttt{r})^*).\texttt{EndTo} \\
\texttt{Subject}_M & \equiv & \exists \texttt{f.StartSubject} \sqcap \exists(\texttt{r} \circ id(\exists \texttt{f.\#PCDATA}) \circ \texttt{r}).\texttt{EndSubject} \\
\texttt{Body}_M & \equiv & \exists \texttt{f.StartBody} \sqcap \exists(\texttt{r} \circ id(\exists \texttt{f.\#PCDATA}) \circ \texttt{r}).\texttt{EndBody} \\
\texttt{Address}_M & \equiv & \exists \texttt{f.StartAddress} \sqcap \exists(\texttt{r} \circ id(\exists \texttt{f.\#PCDATA}) \circ \texttt{r}).\texttt{EndAddress}
\end{array}
$$

Figure 5: Characteristic knowledge base derived from the DTD M (specific part only)

As we said when commenting the definition of $\mathcal{K}_{\mathbf{T},\mathbf{E}}$, the first component (the $\texttt{f}$-filler) of every instance of $E_{\mathbf{D}}$ is its start tag, whereas the last component (the $\texttt{r}^h$-filler) is its end tag. The remaining components (i.e., the $(\texttt{r}^k \circ \texttt{f})$-fillers, with $k < h$) are determined by the form of the element type definition $(E \rightarrow \alpha) \in \mathbf{P}$, and in particular by the form of α. In order to represent α, we use a complex role $\tau(\alpha)$, defined inductively as follows:

$$
\begin{array}{rcl}
\tau(\mathit{empty}) & = & id(\top) \\
\tau(S) & = & id(\exists \texttt{f}.ac(\mathbf{D}, S)) \circ \texttt{r} \\
\tau(\alpha_1 | \alpha_2) & = & \tau(\alpha_1) \cup \tau(\alpha_2) \\
\tau(\alpha_1, \alpha_2) & = & \tau(\alpha_1) \circ \tau(\alpha_2) \\
\tau(\alpha^*) & = & \tau(\alpha)^* \\
\tau(\alpha^+) & = & \tau(\alpha)^+ \\
\tau(\alpha?) & = & \tau(\alpha) \cup id(\top)
\end{array}
$$

where $ac(\cdot, \cdot)$ is a mapping that associates to each pair constituted by the DTD $\mathbf{D}$ and a symbol S an atomic concept as follows:

$$
ac(\mathbf{D}, S) = \begin{cases} E_{\mathbf{D}} & \text{if } S = E \text{ for an element type } E \in \mathbf{E} \\ F & \text{if } S = F \text{ for a terminal } F \in \mathbf{T} \end{cases}
$$

Indeed, $\tau(\alpha)$ reflects the structure imposed by α on the parts of a document instance that are defined by $E \rightarrow \alpha$, and can be explained in terms of an encoding of the tree representing the document instance into a binary tree. It is worth noticing that τ exploits the analogy between the constructs used in expressing content models in XML, and the constructs used to form complex roles in $\mathcal{DL}$. For example, if α has the form α_1, α_2, then the corresponding complex role is $r_1 \circ r_2$, where in turn, r_1 and r_2 are the complex roles corresponding to α_1 and α_2, respectively.

From all the above observations, we can conclude that, for each element type definition $(E \rightarrow \alpha) \in \mathbf{P}$, $\mathcal{K}_{\mathbf{D}}$ contains the assertion:

$$
E_{\mathbf{D}} \quad \equiv \quad \exists \texttt{f.Start}E \sqcap \exists(\texttt{r} \circ \tau(\alpha)).\texttt{End}E
$$

Example 11 (Mail documents) Figure 5 shows the characteristic knowledge base $\mathcal{K}$ for the DTD M described in Figure 1. The general part has been omitted.

The next lemma states a fundamental property of a characteristic knowledge base, which will be used in the following. Let us call "basic" all atomic concepts and roles except for the atomic concepts $E_{\mathbf{D}}$. Then in each model of $\mathcal{K}$, the extension of every atomic concept $E_{\mathbf{D}}$ is completely determined by the extension of basic concepts and roles. Formally:

Lemma 12 Let $\mathbf{D}$ be a DTD, $\mathcal{K}$ be its characteristic knowledge base, and $\mathcal{I}$ and $\mathcal{I}'$ be two models of $\mathcal{K}$ that have the same domain and agree on the interpretation of the basic atomic concepts and roles. Then $\mathcal{I}$ and $\mathcal{I}'$ agree also on the interpretation of $E_{\mathbf{D}}$, for each element type E defined in $\mathbf{D}$.

Proof By contradiction. Let $\mathcal{I}$ and $\mathcal{I}'$ be two models of $\mathcal{K}$ that have the same domain agree on the interpretation of $\texttt{f}$, $\texttt{r}$ and all atomic concepts except for $E_{\mathbf{D}}$. We remind the reader that, for each $E_{\mathbf{D}}$, $\mathcal{K}$ includes the axiom $E_{\mathbf{D}} \equiv \exists \texttt{f.Start}E \sqcap \exists(\texttt{r} \circ \tau(\alpha)).\texttt{End}E$. Let o be an object such that $o \in E_{\mathbf{D}}^{\mathcal{I}}$ but $o \notin E_{\mathbf{D}}^{\mathcal{I}'}$.

We show that we get a contradiction, by induction on the number of $\mathtt{f}$-steps on the longest $\mathtt{r} \circ (\mathtt{f} \cup \mathtt{r})^*$-path from o to a leaf of the tree in $\mathcal{I}$, which by the well-foundedness constraint on $\mathtt{f} \cup \mathtt{r}$ must be finite.

Base case. There are no $\mathtt{f}$-steps in the path. Then since $o \in E_{\mathbf{D}}^{\mathcal{I}} = (\exists \mathtt{f}.\mathtt{Start}E \sqcap \exists(\mathtt{r} \circ \tau(\alpha)).\mathtt{End}E)^{\mathcal{I}}$, and since α generates the empty string (no $\mathtt{f}$-steps are allowed), it follows that $o \in (\exists \mathtt{f}.\mathtt{Start}E \sqcap \exists \mathtt{r}.\mathtt{End}E)^{\mathcal{I}}$. Since $\mathcal{I}$ and $\mathcal{I}'$ agree on $\mathtt{f}$, $\mathtt{r}$, $\mathtt{Start}E$, and $\mathtt{End}E$, it follows that $o \in (\exists \mathtt{f}.\mathtt{Start}E \sqcap \exists \mathtt{r}.\mathtt{End}E)^{\mathcal{I}'}$ and hence $o \in (\exists \mathtt{f}.\mathtt{Start}E \sqcap \exists(\mathtt{r} \circ \tau(\alpha)).\mathtt{End}E)^{\mathcal{I}'}$. Contradiction.

Inductive case. There are $n+1$ $\mathtt{f}$-steps in the path. Let $o_1, \ldots, o_k$ be the objects along the $\mathtt{r} \circ \mathtt{r}^*$-path satisfying $\exists(\mathtt{r} \circ \tau(\alpha)).\mathtt{End}E$ from o, and let o'_i be the $\mathtt{f}$-successor of o_i, for $i \in \{1, \ldots, k\}$. By induction hypothesis, and since $\mathcal{I}$ and $\mathcal{I}'$ agree on the interpretation of each $F \in \mathbf{T}$, we have that $\mathcal{I}$ and $\mathcal{I}'$ agree on the interpretation of $ac(\mathbf{D}, S_j)$ in o_j, for every symbol $S_j \in \mathbf{T} \cup \mathbf{E}$. This implies that they must agree also on the interpretation of $E_{\mathbf{D}}$ in o. Contradiction.

The above lemma ensures us that, given a model $\mathcal{I}$ of $\mathcal{K}$, it is possible to determine whether an object o is an instance of $E_{\mathbf{D}}$ by taking into account only the structure of the $(\mathtt{f} \cup \mathtt{r})^*$ connected component of $\mathcal{I}$ containing o. Observe that, in establishing this property, the well-foundedness construct plays a prominent role, since it ensures that there is no infinite $(\mathtt{f} \cup \mathtt{r})^*$-path starting at o.

4.2 Conformance

The way in which we have defined $\mathcal{K}$ allows us to demonstrate that each $\mathcal{R}$-document instance $d \in \mathit{docs}$ (over $\mathbf{T}$ and $\mathbf{E}$) directly corresponds to a model of $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ and $\mathcal{K}_{\mathcal{R}}$. Indeed, we can define a one-to-one mapping β from $\mathcal{R}$-document instances to models of $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ and $\mathcal{K}_{\mathcal{R}}$.

Given a document instance d, we define $\beta(d)$ by induction on the structure of d as follows:

- If d is a terminal $F \in \mathbf{T}$, then $\beta(d) = (\Delta^{\beta(d)}, \cdot^{\beta(d)})$ is defined as follows: $\Delta^{\beta(d)} = F^{\beta(d)} = \mathtt{Terminal}^{\beta(d)} = \{o\}$, and the extension of the other concepts and of the roles is empty. We say that o is the *root* of $\beta(d)$.
- If d is a sequence of the form $\texttt{<}E\texttt{>}\, d_1 \cdots d_k \,\texttt{</}E\texttt{>}$, where $E \in \mathbf{E}$ is an element type, $\texttt{<}E\texttt{>}$ and $\texttt{</}E\texttt{>}$ are its start and end tags, and $d_1, \ldots, d_k \in \mathit{docs}_{\mathbf{E},\mathbf{T}}$, then $\beta(d) = (\Delta^{\beta(d)}, \cdot^{\beta(d)})$ is obtained as follows[4]:

$$\Delta^{\beta(d)} = \{o, o_b, o_1, \ldots, o_k, o_e\} \uplus \biguplus_{1 \le i \le k} \Delta^{\beta(d_i)}$$
(we say that o is the *root* of $\beta(d)$)

$$\begin{aligned} \mathtt{Start}E'^{\beta(d)} &= \{o_b\} \uplus \biguplus_{1 \le i \le k} \mathtt{Start}E'^{\beta(d_i)} \\ \mathtt{End}E'^{\beta(d)} &= \{o_e\} \uplus \biguplus_{1 \le i \le k} \mathtt{End}E'^{\beta(d_i)} \end{aligned} \quad \text{for each element type } E' \in [E]_{\mathcal{R}}$$

$$\begin{aligned} \mathtt{Start}E'^{\beta(d)} &= \biguplus_{1 \le i \le k} \mathtt{Start}E'^{\beta(d_i)} \\ \mathtt{End}E'^{\beta(d)} &= \biguplus_{1 \le i \le k} \mathtt{End}E'^{\beta(d_i)} \end{aligned} \quad \text{for each element type } E' \in \mathbf{E} \setminus [E]_{\mathcal{R}}$$

$$\begin{aligned} \mathtt{Tag}^{\beta(d)} &= \{o_b, o_e\} \uplus \biguplus_{1 \le i \le k} \mathtt{Tag}^{\beta(d_i)} \\ \mathtt{r}^{\beta(d)} &= \{(o, o_1), (o_1, o_2), \ldots, (o_{k-1}, o_k), (o_k, o_e)\} \uplus \biguplus_{1 \le i \le k} \mathtt{r}^{\beta(d_i)} \\ \mathtt{f}^{\beta(d)} &= \{(o, o_b), (o_1, o'_1), \ldots, (o_k, o'_k)\} \uplus \biguplus_{1 \le i \le k} \mathtt{f}^{\beta(d_i)} \end{aligned}$$
where o'_i is the root of $\beta(d_i)$, for $i \in \{1, \ldots, k\}$

Example 13 (Mail documents) Figure 6 shows the result of applying β to the document instance in Figure 2. For each element type E, the nodes labeled $\texttt{<}E\texttt{>}$ and $\texttt{</}E\texttt{>}$ In the figure, are instances of $\mathtt{Start}E$ and $\mathtt{End}E$ respectively.

Lemma 14 Let $\mathbf{D} = (\mathbf{P}, R)$ be a DTD, d be a document instance, $\beta(d)$ be as specified above, o be the root of $\beta(d)$, and S be a symbol in $\mathbf{T} \cup \mathbf{E}$. Then there is a unique way to extend $\beta(d)$ to a model $\mathcal{I}$ of $\mathcal{K}$, and for each $S \in \mathbf{T} \cup \mathbf{E}$, $d \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$ if and only if $o \in ac(\mathbf{P}, S)^{\mathcal{I}}$.

[4]The symbol $\uplus$ denotes disjoint union.

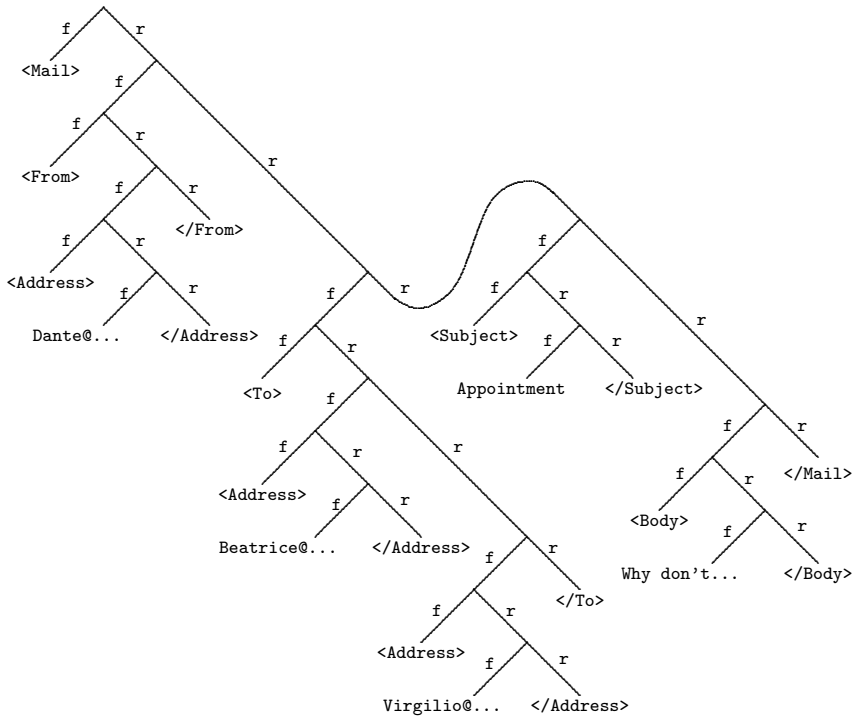

Figure 6: The result of applying β to the document instance in Figure 2

Proof One can easily verify by construction that $\beta(d)$ is a model of $\mathcal{K}_{\mathbf{T},\mathbf{E}} \cup \mathcal{K}_{\mathcal{R}}$. Moreover, Lemma 12 ensures us that, given a DTD $\mathbf{D}$ and a document instance d, there is a unique way to extend $\beta(d)$ to a model $\mathcal{I}$ of $\mathcal{K} = \mathcal{K}_{\mathbf{T},\mathbf{E}} \cup \mathcal{K}_{\mathcal{R}} \cup \mathcal{K}_{\mathbf{D}}$. It remains to show that for each $S \in \mathbf{T} \cup \mathbf{E}$, $d \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$ if and only if $o \in \mathit{ac}(\mathbf{P}, S)^{\mathcal{I}}$. We proceed by induction on the structure of d.

Base case. If d is a terminal $F \in \mathbf{T}$, then the thesis holds trivially.

Inductive case. If d is a sequence of the form $\texttt{<}E\texttt{>}\, d_1 \cdots d_k \,\texttt{</}E\texttt{>}$, then $\beta(d)$ is constructed from $\beta(d_1), \ldots, \beta(d_k)$ as specified above. In particular, let o'_i be the root of $\beta(d_i)$, for $i \in \{1, \ldots, k\}$, and let $(o, o_b) \in \mathtt{f}^{\mathcal{I}}$. If S is a terminal in $\mathbf{T}$ then $d \notin \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$, and since o has o_b as $\mathtt{f}$-successor, $o \notin (\forall \mathtt{f}.\bot)^{\mathcal{I}} \supseteq \mathtt{Terminal}^{\mathcal{I}} \supseteq S^{\mathcal{I}}$, and we are done. If S is an element type in $\mathbf{E}$, let $S \rightarrow \alpha \in \mathbf{P}$ be the corresponding element type definition. Since $o \in (\exists \mathtt{f.Start}E)^{\mathcal{I}}$, then $o \in (\exists \mathtt{f.Start}S)^{\mathcal{I}}$ only if $S \in [E]_{\mathcal{R}}$. Hence $o \in S^{\mathcal{I}}_{\mathbf{D}} = (\exists \mathtt{f.Start}S \sqcap \exists(\mathtt{r} \circ \tau(\alpha))\mathtt{.End}S)^{\mathcal{I}}$ if and only if $S \in [E]_{\mathcal{R}}$ and there exists a string $S_1 \cdots S_k$ generated by α and atomic concepts $\mathit{ac}(\mathbf{D}, S_1), \ldots, \mathit{ac}(\mathbf{D}, S_k)$ such that $o'_i \in \mathit{ac}(\mathbf{D}, S_i)$. By induction hypothesis o'_i is in the extension of $\mathit{ac}(\mathbf{D}, S_i)$ if and only if $d_i \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S_i)$. Hence, by definition of $\mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$, we get that $o \in \mathit{ac}(\mathbf{D}, S)$ if and only if $d \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$.

The following theorem provides the basis for verifying $\mathcal{R}$-conformance by resorting to the translation of DTDs in $\mathcal{DL}$.

Theorem 15 Checking $\mathcal{R}$-conformance of a document instance to a DTD can be polynomially reduced to model checking in $\mathcal{DL}$.

Proof To check whether a document instance d R-conforms to a DTD $\mathbf{D} = (\mathbf{P}, R)$, we construct $\beta(d)$ and extend it to an interpretation $\mathcal{I}_d$ of $\mathcal{K}$ as follows. For each $E \in \mathbf{E}$, we interpret $E_{\mathbf{D}}$ as:

$$E^{\mathcal{I}_d}_{\mathbf{D}} = (\exists \mathtt{f.Start}E)^{\mathcal{I}_d}$$

Observe that if o is the root of $\beta(d)$, then $o \in R^{\mathcal{I}_d}_{\mathbf{D}}$, but that $\mathcal{I}_d$ is not necessarily a model of $\mathcal{K}_{\mathbf{D}}$, and hence of $\mathcal{K}$. The following argument ensures us that the latter is the case exactly if d $\mathcal{R}$-conforms to $\mathbf{D}$.

By Lemma 14, applied to the root element type R of $\mathbf{D}$, we can conclude that if $\mathcal{I}_d$ is a model of $\mathcal{K}$ and $o \in R^{\mathcal{I}_d}$, then $d \in \mathit{docs}_{\mathcal{R}}(\mathbf{D})$. On the other hand, by proceeding by induction on the structure of d, analogously to the proof of Lemma 14, it can be shown that if $d \in \mathit{docs}_{\mathcal{R}}(\mathbf{D})$, then $\mathcal{I}_d$ is a model of $\mathcal{K}$ and $o \in R^{\mathcal{I}_d}$.

The reduction is polynomial since the construction of $\mathcal{K}$ takes polynomial time in the size of $\mathbf{D}$, and the construction of $\mathcal{I}_d$ takes linear time in the size of d.

Corollary 16 Checking $\mathcal{R}$-conformance of a document instance d to a DTD $\mathbf{D}$ can be done in time polynomial in the size of d and $\mathbf{D}$.

Proof By Theorem 15, $\mathcal{R}$-conformance can be polynomially reduced to model checking in $\mathcal{DL}$. Each $\mathcal{DL}$ knowledge base can be translated into a formula of first order logic plus fixpoints (see [2]) which uses at most three variables and two levels of fixpoint nestings [27]. Since $\mathcal{I}_d$ can be considered as a first order structure, the claim follows from the fact that model checking in first order logic plus fixpoint with the above restrictions has polynomial complexity with respect to the size of the structure and the formula.

Example 17 (Mail documents) Let us consider the structure shown in Figure 6. It is easy to see that, if we assign every node that is an $\mathtt{f}$-predecessor of a node labeled $\mathtt{<}E\mathtt{>}$ to the extension of the atomic concept E_M, then we obtain a model of $\mathcal{K}$ where the root is in the extension of $\mathtt{Mail}_M$.

4.3 Inclusion and Disjointness

In order to show how $\mathcal{DL}$ can be used to determine $\mathcal{R}$-inclusion and $\mathcal{R}$-disjointness between DTDs, we first define a mapping γ from models of $\mathcal{K}$ to document instances.

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\mathcal{K}$ and $o \in \Delta^{\mathcal{I}}$. We define $\gamma(o)$ by induction on the number of $\mathtt{f}$-steps on the longest $\mathtt{r} \circ (\mathtt{f} \cup \mathtt{r})^*$-path from o in $\mathcal{I}$. By the well-foundedness constraint on $\mathtt{f} \cup \mathtt{r}$, such path must be finite. $\gamma(o)$ is defined as follows:

- If $o \in F^{\mathcal{I}}$ for some terminal $F \in \mathbf{T}$, then $\gamma(o) = F$.
- If for some element type E, there are some integer $k \geq 0$ and objects o_b, o_e, $o_1, \ldots, o_k$, $o'_1, \ldots, o'_k$, such that $o_b \in \mathtt{Start}E^{\mathcal{I}}$, $o_e \in \mathtt{End}E^{\mathcal{I}}$, $(o, o_1), (o_1, o_2), \ldots, (o_{k-1}, o_k), (o_k, o_e) \in \mathtt{r}^{\mathcal{I}}$, and $(o, o_b), (o_1, o'_1), \ldots, (o_k, o'_k) \in \mathtt{f}^{\mathcal{I}}$, then $\gamma(o) = \mathtt{<}E\mathtt{>}\,\gamma(o'_1) \cdots \gamma(o'_k)\,\mathtt{</}E\mathtt{>}$.
- Otherwise $\gamma(o)$ is undefined.

Lemma 18 Let $\mathbf{D} = (\mathbf{P}, S_0)$ be a DTD, $\mathcal{K}$ be its characteristic knowledge base, $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\mathcal{K}$, $o \in \Delta^{\mathcal{I}}$, and S be a symbol in $\mathbf{T} \cup \mathbf{E}$. Then $o \in \mathit{ac}(\mathbf{P}, S)^{\mathcal{I}}$ if and only if $\gamma(o)$ is defined and $\gamma(o) \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$.

Proof If $\gamma(o)$ is undefined or if S is a terminal, then the thesis holds trivially. So let us assume that $\gamma(o)$ is defined and that S is an element type. Let $S \to \alpha \in \mathbf{P}$ be the corresponding element type definition. We proceed by induction on the number of $\mathtt{f}$-steps on the longest $\mathtt{r} \circ (\mathtt{f} \cup \mathtt{r})^*$-path from o in $\mathcal{I}$.

Base case. There are no $\mathtt{f}$-steps in the path. Then by construction either $\gamma(o) = F$ for some terminal $F \in \mathbf{T}$, or $\gamma(o) = \mathtt{<}E\mathtt{>}\ \mathtt{</}E\mathtt{>}$ for some element type $E \in \mathbf{E}$. The case where $\gamma(o) = F$ is easy. In the second case, $o \in (\exists \mathtt{f}.\mathtt{Start}E \sqcap \exists \mathtt{r}.\mathtt{End}E)^{\mathcal{I}}$. If $S \in [E]_{\mathcal{R}}$ and α generates the empty string, then $o \in \mathit{ac}(\mathbf{P}, S)^{\mathcal{I}}$ and also $\gamma(o) \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$. Otherwise $o \notin \mathit{ac}(\mathbf{P}, S)^{\mathcal{I}}$ and $\gamma(o) \notin \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$.

Inductive case. There are $n + 1$ $\mathtt{f}$-steps in the path. Let $o_1, \ldots, o_k$ be the objects along the $\mathtt{r} \circ \mathtt{r}^*$-path satisfying $\exists(\mathtt{r} \circ \tau(\alpha)).\mathtt{End}E$ from o, and let o'_j be the $\mathtt{f}$-successor of o_j, for $j \in \{1, \ldots, h\}$, where E is the element type such that $\gamma(o) = \mathtt{<}E\mathtt{>}\,\gamma(o'_1) \cdots \gamma(o'_k)\,\mathtt{</}E\mathtt{>}$. If there are symbols $S_1, \ldots, S_k$ in $\mathbf{T} \cup \mathbf{E}$ such that $o'_j \in \mathit{ac}(\mathbf{P}, S_j)^{\mathcal{I}}$, for $j \in \{1, \ldots, k\}$, then by induction hypothesis $\gamma(o'_j) \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S_j)$, and if $S \in [E]_{\mathcal{R}}$ and α generates the string $S_1 \cdots S_k$, then $o \in \mathit{ac}(\mathbf{P}, S)^{\mathcal{I}}$ and also $\gamma(o) \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$. Otherwise $o \notin \mathit{ac}(\mathbf{P}, S)^{\mathcal{I}}$ and $\gamma(o) \notin \mathit{docs}_{\mathcal{R}}(\mathbf{P}, S)$.

We extend the notion of characteristic knowledge base in such a way that it represents a set of DTDs, rather than a single DTD. Given a set $\mathcal{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_n\}$ of DTDs over $\mathbf{E}$, $\mathbf{T}$ and $\mathcal{R}$, we define the characteristic knowledge base of $\mathcal{D}$ simply as $\mathcal{K} = \mathcal{K}_{\mathbf{T},\mathbf{E}} \cup \mathcal{K}_{\mathcal{R}} \cup \mathcal{K}_{\mathbf{D}_1} \cup \cdots \cup \mathcal{K}_{\mathbf{D}_n}$. In other words, we include in $\mathcal{K}$ both $\mathcal{K}_{\mathbf{T},\mathbf{E}}$ and $\mathcal{K}_{\mathcal{R}}$, and we add to $\mathcal{K}$ the encoding of each $\mathbf{D}_i \in \mathcal{D}$. A model of $\mathcal{K}$ represents now a set of document instances. Observe that Lemma 14 and Lemma 18 extend immediately to the case where we consider a set of DTDs, rather than a single DTD.

Exploiting Lemma 14 and Lemma 18 we can provide the following characterization of $\mathcal{R}$-inclusion and $\mathcal{R}$-disjointness between DTDs in terms of satisfiability in $\mathcal{DL}$.

Theorem 19 Let $\mathbf{D}$ and $\mathbf{D}'$ be two DTDs, and $\mathcal{K} = \mathcal{K}_{\mathbf{T},\mathbf{E}} \cup \mathcal{K}_{\mathcal{R}} \cup \mathcal{K}_{\mathbf{D}} \cup \mathcal{K}_{\mathbf{D}'}$ be the knowledge base derived from $\mathbf{T}$, $\mathbf{E}$, $\mathcal{R}$, $\{\mathbf{D}, \mathbf{D}'\}$ as described above. Then

$$\mathbf{D} \sqsubseteq_{\mathcal{R}} \mathbf{D}' \quad \text{if and only if} \quad \mathcal{K} \models R_{\mathbf{D}} \sqsubseteq R'_{\mathbf{D}'} \tag{1}$$

$$\mathbf{D} \otimes_{\mathcal{R}} \mathbf{D}' \quad \text{if and only if} \quad \mathcal{K} \models R_{\mathbf{D}} \sqcap R'_{\mathbf{D}'} \sqsubseteq \bot \tag{2}$$

Proof Let $\mathbf{D} = (\mathbf{P}, R)$ and $\mathbf{D}' = (\mathbf{P}', R')$.

(1) "$\Leftarrow$" By contradiction. Let d be a document instance that conforms to $\mathbf{D}$ but does not conform to $\mathbf{D}'$, let $\mathcal{I}$ be the unique model of $\mathcal{K}$ that extends $\beta(d)$, and let o be the root of $\beta(d)$. Since $d \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, R)$ and $d \notin \mathit{docs}_{\mathcal{R}}(\mathbf{P}', R')$, by Lemma 14, $o \in R^{\mathcal{I}}_{\mathbf{D}}$ and $o \notin {R'}^{\mathcal{I}}_{\mathbf{D}'}$. Contradiction.

"$\Rightarrow$" By contradiction. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\mathcal{K}$ with $o \in \Delta^{\mathcal{I}}$ such that $o \in R^{\mathcal{I}}_{\mathbf{D}}$ and $o \notin {R'}^{\mathcal{I}}_{\mathbf{D}'}$. Then by Lemma 18 $\gamma(o)$ is defined, $\gamma(o) \in \mathit{docs}_{\mathcal{R}}(\mathbf{P}, R)$, and $\gamma(o) \notin \mathit{docs}_{\mathcal{R}}(\mathbf{P}', R')$. Contradiction.

(2) can be proved analogously.

From decidability in deterministic exponential time of logical implication in $\mathcal{DL}$ we obtain as an immediate consequence an EXPTIME upper bound for $\mathcal{R}$-inclusion and $\mathcal{R}$-equivalence between DTDs. This results also in an exponential improvement over previously known algorithms for checking structural equivalence [42].

Corollary 20 $\mathcal{R}$-inclusion, $\mathcal{R}$-equivalence, and $\mathcal{R}$-disjointness between two DTDs can be verified in deterministic exponential time in the size of the DTDs.

5 Retrieving XML Documents from a Document Base

We now describe how to exploit the reasoning techniques presented in the previous section to evaluate queries posed to a database of documents. As before we refer to a fixed alphabet $\mathbf{E}$ of element types, a fixed alphabet $\mathbf{T}$ of terminals, and a fixed equivalence relation $\mathcal{R}$ on the set $\mathbf{E}$ of element types. A document base on $\mathbf{E}$, $\mathbf{T}$, and $\mathcal{R}$ represents a set of XML documents, and is defined as follows.

Definition 21 A *document base* $\mathcal{B}$ is a pair $\mathcal{B} = \langle \mathcal{D}, \mathcal{I} \rangle$, where

- $\mathcal{D}$ is a set of DTDs, with the assumption that for each pair $\mathbf{D}_1, \mathbf{D}_2 \in \mathcal{D}$, it is known whether $\mathbf{D}_1 \sqsubseteq_{\mathcal{R}} \mathbf{D}_2$, and whether $\mathbf{D}_1 \otimes_{\mathcal{R}} \mathbf{D}_2$;
- $\mathcal{I}$ is a set of document instances, with the assumption that for each $d \in \mathcal{I}$ there is at least one $\mathbf{D} \in \mathcal{D}$ such that d conforms to $\mathbf{D}$, and for each pair $d \in \mathcal{I}$, $\mathbf{D} \in \mathcal{D}$, it is known whether d conforms to $\mathbf{D}$.

A query posed to a document base is simply a document type definition, used to retrieve all document instances in the document base that satisfy such definition.

Definition 22 A *query* Q is a DTD, and the evaluation of Q over a document base $\mathcal{B}$ returns as an answer the set $Q(\mathcal{B})$ of all document instances $d \in \mathcal{B}$ such that d $\mathcal{R}$-conforms to Q.

The goal of this section is to present an algorithm for computing the answer $\mathcal{Q}(\mathcal{B})$ to a query Q posed to a document base $\mathcal{B} = \langle \mathcal{D}, \mathcal{I} \rangle$, which exploits the possibility of reasoning over DTDs. The algorithm maintains two sets $\mathcal{S}$ and $\mathcal{J}$ of DTDs and document instances, and computes a set $\mathcal{A}(\mathcal{B}, Q)$ of document instances by proceeding as follows:

1. Let $\mathcal{S}$ be equal to $\mathcal{D}$, and let $\mathcal{J}$ be equal to $\mathcal{I}$.
2. While $\mathcal{S}$ is not empty, repeatedly select a DTD $\mathbf{D}$ from $\mathcal{S}$ such that there is no $\mathbf{D}' \in \mathcal{S}$ with $\mathbf{D} \sqsubseteq_{\mathcal{R}} \mathbf{D}'$, and do the following:
   (a) If $\mathbf{D} \equiv_{\mathcal{R}} Q$, then let $\mathcal{A}(\mathcal{B}, Q)$ be all the document instances d in $\mathcal{I}$ such that d conforms to $\mathbf{D}$, and stop.
   (b) If $\mathbf{D} \sqsubseteq_{\mathcal{R}} Q$, then
      (b.1) move from $\mathcal{J}$ to $\mathcal{A}(\mathcal{B}, Q)$ all document instances that conform to $\mathbf{D}$,
      (b.2) remove from $\mathcal{S}$ every DTD $\mathbf{D}'$ such that $\mathbf{D}' \sqsubseteq_{\mathcal{R}} \mathbf{D}$,
      (b.3) continue with the next iteration of the while-loop.
   (c) If $Q \sqsubseteq_{\mathcal{R}} \mathbf{D}$, then
      (c.1) remove $\mathbf{D}$ from $\mathcal{S}$,
      (c.2) for every DTD $\mathbf{D}'$ in $\mathcal{S}$ such that $\mathbf{D}' \otimes_{\mathcal{R}} \mathbf{D}$, remove $\mathbf{D}'$ from $\mathcal{S}$ and remove from $\mathcal{J}$ every document instance that conforms to $\mathbf{D}'$,
      (c.3) continue with the next iteration of the while-loop.
   (d) If $\mathbf{D} \otimes_{\mathcal{R}} Q$, then
      (d.1) remove from $\mathcal{S}$ every DTD $\mathbf{D}'$ such that $\mathbf{D}' \sqsubseteq_{\mathcal{R}} \mathbf{D}$,
      (d.2) remove from $\mathcal{J}$ every document instance that conforms to $\mathbf{D}'$,
      (d.3) continue with the next iteration of the while-loop.
   (e) Otherwise, remove $\mathbf{D}$ from $\mathcal{S}$, and continue.
3. Add to $\mathcal{A}(\mathcal{B}, Q)$ every document instance d in $\mathcal{J}$ that conforms to Q.

The correctness of the above algorithm is shown in the next theorem.

Theorem 23 Let $\mathcal{B} = \langle \mathcal{D}, \mathcal{I} \rangle$ be a document base, and Q be a query. Then the set $\mathcal{A}(\mathcal{B}, Q)$ computed by the algorithm above is equal to $Q(\mathcal{B})$.

Proof Since Step 3 considers all document instances whose conformance to Q could not be determined by looking only at the DTDs in $\mathcal{D}$, it is sufficient to show that Step 2 of the algorithm does not remove from $\mathcal{A}(\mathcal{B}, Q)$ any document instance that contributes to $Q(\mathcal{B})$.

Step 2.a is obvious: if $\mathbf{D}$ is $\mathcal{R}$-equivalent to Q, then the answer to Q is the set of document instances in $\mathcal{I}$ that conform to $\mathbf{D}$.

Step 2.b deals with the case where $\mathbf{D}$ is $\mathcal{R}$-included in Q. In such a case, the set of document instances conforming to $\mathbf{D}$ takes part to the answer to the query. Moreover, since such a set comprises all documents conforming to the DTDs that are $\mathcal{R}$-included in $\mathbf{D}$, these DTDs need not to be considered anymore and are discarded.

Step 2.c considers the case where Q is $\mathcal{R}$-included in $\mathbf{D}$. Since the document instances satisfying Q are among those that conform to $\mathbf{D}$, the algorithm discards all document instances conforming to some DTD that is $\mathcal{R}$-disjoint from $\mathbf{D}$.

Step 2.d takes care of the case where Q is $\mathcal{R}$-disjoint from $\mathbf{D}$, and therefore, discards all DTDs that are $\mathcal{R}$-included in $\mathbf{D}$, and excludes from the answer all document instances that conform to $\mathbf{D}$.

Observe that the above method, can be seen as an adaptation of the semantic indexing technique developed in DLs [43], where DTDs act as semantic indexes on XML documents in the document base. In this way they help in improving performance of query evaluation with respect to the brute force approach of evaluating document instances one by one. In other words, reasoning on DTDs allows for a more effective query evaluation process. Obviously, since comparing DTDs is costly, the method pays off when the size of DTDs is small (e.g. logarithmic) with respect to the size of the document instances, which is usually the case.

6 Discussion

We have shown how to use the Description Logic $\mathcal{DL}$ to represent and reason on the structural aspects of XML documents. We have also shown how to use $\mathcal{DL}$ reasoning to improve the efficiency of query answering over document bases.

By exploiting the constructs of $\mathcal{DL}$, we are also able to integrate into the structure of documents aspects related to the semantics of the information contained in them. Although for the sake of simplicity we did not include these aspects in the formal development of this paper, we would like to briefly mention how they can be taken into account in our framework.

- DTDs in XML may include attributes describing properties of DTD elements. An attribute for a DTD **D** has a name, a value type, and is associated with an element type of **D**. It is easy to see that such attributes can be modeled in $\mathcal{DL}$ by adding one role for each attribute, and by including in the knowledge base representing the DTD suitable assertions on such roles. In particular, qualified number restrictions can be used to impose constraints on the number of values that a certain element type may have for a given attribute.

- Documents may contain links to other documents. In our framework, links can be easily represented by means of a special concept with suitable roles for the name of the link and the associated anchor. Observe that, since document links can form cycles, documents with links can be considered as graphs, rather than trees. However, the roles used to represent links are different from those used to represent document structures (i.e., `f` and `r`), and therefore the resulting knowledge base can correctly model the situation where finite tree substructures are embedded in arbitrary graphs.

- With respect to the above point, suitable assertions can be used to constrain the anchor to point to a document of a specific DTD, or to limit the number of links pointing to documents of a certain type. To impose the latter type of condition, we again resort to qualified number restrictions.

- If part of a document (corresponding to a terminal symbol F in the DTD) includes a special structure (for example, a list of records, or a table with information about, say, departments and employees), this can be represented by adding suitable properties to the concept corresponding to F.

- The idea of capturing more semantics related to the tags of documents can be pursued in $\mathcal{DL}$ by introducing new concepts and roles and using it for this purpose.

We note that the framework presented in this paper for representing and reasoning on structured documents provides a notable example of handling objects composed of different parts. The *part-whole* relation is seen as having a special importance in several applications [15, 5, 29]. The Description Logic $\mathcal{DL}$, by means of the reflexive-transitive closure and the well-foundedness constructs, is able to capture fundamental aspects of the part-whole relation [29, 4, 37] as shown in [12].

Recently, there has been a strong interest in the Database community on the development of new data models for semi-structured data. The ability to represent data whose structure is less rigid and strict than in conventional databases is indeed considered a crucial aspect in modern approaches to data modeling, and is important in many application areas, such as biological databases, digital libraries, and data integration [1, 9, 15, 32, 35]. Following [1], semi-structured data can be defined as data that is neither raw, nor strictly typed as in conventional database systems. OEM (Object Exchange Model) [3], BDFS (Basic Data model For Semi-structured data) [9], and its extension presented in [14] are recent proposals of models for semi-structured data. They represent data as graphs with labeled edges, where information on both the values and the schema of data are kept. The formalism presented in this paper can be seen as a formalism for representing semi-structured data. Indeed, if we associate to each element E of a DTD **D** one node, and we consider the content model associated to E as the specification of the allowable children of the node, we obtain a method by which a DTD can be seen as a graph G, in such a way that the graphs conforming to G correspond to the document instances conforming to **D**. Notably, the results presented in this paper provide effective procedures for reasoning about semi-structured data schemas, a feature that is missing, for example, in OEM. Further research is needed to compare the expressive power of our formalism with respect to the above mentioned data models.

7 Conclusions

Several recent papers dealing with the problem of retrieving information from a document database such as the World Wide Web argue that the current techniques for representing and reasoning on document structures should be improved. We have provided a view of DTDs as concepts of the expressive Description Logic $\mathcal{DL}$, and we have demonstrated that this approach is indeed very effective for both faithfully representing document structures, and answering some open questions regarding DTD equivalence checking. In particular, we have proposed a method for checking structural equivalence of DTDs in worst case deterministic exponential time, in contrast to the known algorithms which are double exponential. Also, we have shown that conformance in our setting can be done in polynomial time, and query answering can be done efficiently, by taking advantage of the reasoning methods associated to the DL.

Two further research directions are worth pursuing. On the one hand, further aspects of DTDs could be captured in order to represent other properties of documents such as exceptions (as described in [42]). On the other hand, the deductive power of $\mathcal{DL}$ allows one to study new types of reasoning on DTDs, such as further forms of parameterized equivalence (e.g. abstracting from the definition of a specified element) and document classification (infer which is the DTD that best matches a given marked document among a set of candidates).

References

[1] Serge Abiteboul. Querying semi-structured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 1–18, 1997.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[3] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[4] Alessandro Artale, Enrico Franconi, and Nicola Guarino. Open problems with part-whole relations. In *Proc. of the 1996 Description Logic Workshop (DL-96)*, number WS-96-05, pages 70–73. AAAI Press, 1996.

[5] Alessandro Artale, Enrico Franconi, Nicola Guarino, and Luca Pazzi. Part-whole relations in object-centered systems: An overview. *Data and Knowledge Engineering*, 20:347–383, 1996.

[6] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, 1991.

[7] Franz Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18:175–219, 1996.

[8] Tim Bray, Jean Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 – W3C recommendation. Technical report, World Wide Web Consortium, 1998. Available at http://www.w3.org/TR/1998/REC-xml-19980210.

[9] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 336–350, 1997.

[10] Diego Calvanese. Finite model reasoning in description logics. In Luigia C. Aiello, John Doyle, and Stuart C. Shapiro, editors, *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 292–303. Morgan Kaufmann, Los Altos, 1996.

[11] Diego Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1996.

[12] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95)*, number 1013 in Lecture Notes in Computer Science, pages 229–246. Springer-Verlag, 1995.

[13] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-98)*, pages 149–158, 1998.

[14] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. What can knowledge representation do for semi-structured data? In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI-98)*, pages 205–210, 1998.

[15] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In R. T. Snodgrass and M. Winslett, editors, *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 313–324, Minneapolis (Minnesota, USA), 1994.

[16] Giuseppe De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

[17] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 205–212. AAAI Press/The MIT Press, 1994.

[18] Giuseppe De Giacomo and Maurizio Lenzerini. A uniform framework for concept definitions in description logics. *J. of Artificial Intelligence Research*, 6:87–110, 1997.

[19] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, Studies in Logic, Language and Information, pages 193–238. CSLI Publications, 1996.

[20] E. Allen Emerson and Charanjit S. Jutla. On simultaneously determinizing and complementing ω-automata. In *Proc. of the 4th IEEE Symposium on Logic in Computer Science (LICS'89)*, pages 333–342, 1989.

[21] Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.

[22] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

[23] Roy Goldman and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proc. of the 23rd Int. Conf. on Very Large Data Bases (VLDB-97)*, pages 436–445, 1997.

[24] International Organization for Standardization. *ISO-8879: Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*, October 1986.

[25] Craig Knoblock and Alon Y. Levy, editors. *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, number SS-95-08, Menlo Park (U.S.A.), 1995. AAAI Press/The MIT Press.

[26] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proc. of the 21st Int. Conf. on Very Large Data Bases (VLDB-95)*, pages 54–65, 1995.

[27] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[28] L. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the Web. In *Proc. of the 6th Int. Workshop on Reasearch Issues in Data Enginnering: Interoperability of Nontraditional Database Systems*. IEEE Computer Science Press, 1996.

[29] Patrick Lambrix. *Part-Whole Reasoning in Description Logic.* PhD thesis, Dep. of Computer and Information Science, Linköping University, Sweden, 1996.

[30] Patrick Lambrix, Nahid Shahmehri, and Johan Åberg. Towards creating a knowledge base for world-wide web documents. In *Proc. of the IASTED Int. Conf. on Intelligent Information Systems*, pages 507–511, 1997.

[31] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems*, 5:121–143, 1995.

[32] Alberto Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.

[33] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Number 422 in Lecture Notes In Artificial Intelligence. Springer-Verlag, 1990.

[34] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, Los Altos, 1991.

[35] D. Quass, A. Rajaraman, I. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95)*, pages 319–344. Springer-Verlag, 1995.

[36] D. R. Raymond, F. W. Tompa, and D. Wood. From data implementation to data model: Meta-semantic issues in the evolution of SGML. *Computer Standards and Interfaces*, 18:25–36, 1996.

[37] Ulrike Sattler. A concept language for an engineering application with part-whole relations. In Alexander Borgida, Maurizio Lenzerini, Daniele Nardi, and Bernhard Nebel, editors, *Working Notes of the 1995 Description Logics Workshop*, Technical Report, RAP 07.95, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", pages 119–123, Rome (Italy), 1995.

[38] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 466–471, Sydney, Australia, 1991.

[39] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[40] Moshe Y. Vardi. The taming of converse: Reasoning about two-way computations. In R. Parikh, editor, *Proc. of the 4th Workshop on Logics of Programs*, number 193 in Lecture Notes in Computer Science, pages 413–424. Springer-Verlag, 1985.

[41] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986. A preliminary version appeared in *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC-84).*

[42] Derick Wood. Standard Generalized Markup Language: Mathematical and philosophical issues. In Jan van Leeuwen, editor, *Computer Science Today, Recent Trends and Developments*, number 1000 in Lecture Notes in Computer Science, pages 344–365. Springer-Verlag, 1995.

[43] William A. Woods. Understanding subsumption and taxomony: A framework for progress. In J. F. Sowa, editor, *Principles of Semantic Networks*, pages 45–94. Morgan Kaufmann, Los Altos, 1991.