On Reasoning about Structural Equality in XML: A Description Logic Approach

David Toman and Grant Weddell

School of Computer Science, University of Waterloo

Abstract

We define a boolean complete dialect of description logic called $\mathcal{DLFD}_{\text{reg}}$ that can be used to reason about structural equality in semistructured ordered data in the presence of document type definitions. This application depends on the novel ability of $\mathcal{DLFD}_{\text{reg}}$ to express functional dependencies over possibly infinite sets of feature paths defined by regular languages. We also present a decision procedure for the associated logical implication problem. The procedure underlies a mapping of such problems to satisfiability problems of $\text{Datalog}^{\neg}_{nS}$.

Key words: XML Schema, Functional Dependencies, Description Logics.

1 Introduction

Equality is a fundamental notion in any interaction with data, and the need to reason about equality during query optimization and evaluation is unavoidable in any practical data model and query language. Although the problem of reasoning about equality has been studied extensively for relational and object oriented data models, this is not the case for the more recent semistructured ordered models and query languages such as XML and XQuery. With XML in particular, there are three notions of equality that have surfaced in the literature. *Label* equality, based on equality of strings denoting *element tags*, and *node* equality, based on node identity, are two of these notions that have simple and efficient implementations. *Structural equality* between arbitrary ordered forests representing XML documents is a third and much more costly variant of equality. Structural equality, however, is the basis for comparing XML values in the XQuery language [1]. In particular, structural equality is

Email address: `{david,gweddell}@uwaterloo.ca` (David Toman and Grant Weddell).

heavily used by `where` clauses in FLWR expressions, by *duplicate elimination* operators, etc.

Example 1 Consider the following XQuery expression that constructs the names of employees who have received mail:

```
for x in doc(personnel)//emp,
    y in doc(shipping)//received/emp
where x=y
return x/name
```

The detection of matching *employee* subdocuments in the `where` clause requires, according to the XQuery specification, a potentially expensive structural equality comparison [1]. However, knowing that employee documents are structurally equal if they have the same employee number, a fixed and presumably small part of the *employee* subdocument, would enable this test to be replaced by an efficient test for node equality on respective employee number subdocuments.

This optimization depends on our ability to specify and reason about equality, in particular about an extended notion of *functional dependencies* that hold in XML documents. Note that detection of duplicate employee subdocuments and several other XQuery operations based on structural equality can similarly benefit from these additional reasoning capabilities.

In this paper, we propose a new approach to reasoning about structural equality in XML documents. The proposed approach is indirect; we begin by defining a boolean complete dialect of a *description logic*, $\mathcal{DLFD}_{\mathrm{reg}}$. This dialect has the crucial ability to reason about implied structural equality using *regular path functional dependencies*. The contributions of this paper are as follows:

- We introduce the notions of *regular attribute restrictions* and of *regular path functional dependencies*, based on regular sets of feature path descriptions, in the context of a boolean complete description logic;
- We provide a sound and complete reasoning procedure for $\mathcal{DLFD}_{\mathrm{reg}}$, including tight complexity bounds for the associated implication problem; and
- We show a link between reasoning in this logic and reasoning about structural equality in XML documents.

The ability to formulate implication problems using path functional dependencies has a number of important applications in query optimization. In particular, the correctness of query rewrite rules that relate to duplicate elimination [2], to sort operator elimination [3], and to a variety of other optimizations [4,5] can be usefully abstracted as logical implication problems. Similar results can be obtained in the XML/XQuery setting using the regular path functional dependencies proposed in this paper.

1.1 Related Work and Outline

There is a growing body of work on integrity constraints that have been proposed in the literature to restrict the structure of XML documents. In particular, constraints that resemble a form of keys and functional dependencies have been considered in [6–9]. However, there still remains the problem of reasoning about arbitrary structural equality between (sub)documents with unstructured components.

As stated above, this issue of structural equality is the focus of this paper and is considered in the context of the description logic $\mathcal{DLFD}_{\mathrm{reg}}$. Of particular significance is that our results complement earlier work by Calvanese et al. [10] that proposed the use of a DL to reason about *document type definitions* (or DTDs). In particular, they considered the problem of comparing DTDs to determine various inclusion relationships.

An "fd" concept constructor for functional constraints over features was first considered in the context of a very simple DL dialect with a PTIME decision procedure in [11]. A more general form of this constructor with feature paths, considered subsequently in [2], made it possible to capture constraints that resembled path functional dependencies [5]. This later work showed that the complexity of reasoning remained PTIME if all occurrences of the constructor in a given DL terminology satisfied a prefix condition. The complexity in the general case for this dialect was resolved in [12] to be complete for DEXPTIME. The result was obtained by comparing the problem to the recognition problem for $\mathrm{Datalog}_{nS}$ and is reproduced in our discussion of lower bounds for $\mathcal{DLFD}_{\mathrm{reg}}$. [12] also showed that the bound continued to hold for boolean complete dialects that allow for defined concepts in terminologies.

The remainder of the paper is organized as follows. Section 2 defines the syntax and semantics for $\mathcal{DLFD}_{\mathrm{reg}}$, and introduces a *logic programming language* called $\mathrm{Datalog}_{nS}^{\neg}$. In Sections 3 and 4, we study the complexity associated with reasoning in $\mathcal{DLFD}_{\mathrm{reg}}$. Section 5 outlines how $\mathcal{DLFD}_{\mathrm{reg}}$ can then be used to reason about structural equality in XML in the presence of document type definitions. We conclude with a summary and a list of conjectures and open questions in Section 6.

2 Definitions

In this section, we focus on definitions that are needed for the main technical discussions. A definition of "XML in a nutshell" and of an embedding of XML in $\mathcal{DLFD}_{\text{reg}}$ interpretations is given at the start of Section 5.

2.1 Description Logic

Apart from *concepts*, which are descriptions of sets, description logics that derive from $\mathcal{ALC}$ [13], such as $\mathcal{SHIQ}$ [14] and $\mathcal{DLR}$ [15], start with descriptions of relations as basic building blocks. Binary relations are a particular focus and are called *roles*. The description logic $\mathcal{DLFD}_{\text{reg}}$ introduced in this section takes an alternative approach that starts by augmenting concepts with the notion of *attributes* (also called *features* in the literature). Attributes correspond to unary functions on an underlying object domain.

This approach has several advantages. First, the decidability and complexity of logical implication for $\mathcal{DLFD}_{\text{reg}}$ is closely related to reasoning in a logic programming language of monadic predicates and functions called $\text{Datalog}^{\neg}_{nS}$ [16,17] that will be introduced later in this section. Second, the use of this language provides a uniform framework in which we can study various extensions of $\mathcal{DLFD}_{\text{reg}}$, how roles can be simulated by attributes for example. And third, the connection to logic programming creates an opportunity to inherit a rich set of techniques for efficiently implementing logic programs [18,19].

The syntax and semantics of $\mathcal{DLF}_{\text{reg}}$ and $\mathcal{DLFD}_{\text{reg}}$ is given by the following.

Definition 2 (Regular Path Expression Sets) *Let F be a set of attributes. A* regular path expression *is defined by the grammar*

$$L ::= f \mid \mathsf{Id} \mid L_1.L_2 \mid L_1, L_2 \mid (L)*\,,$$

where $f \in F$ and where Id *denotes the empty string, '.' the concatenation, ',' the union, and '*' the Kleene closure. A string generated by a regular path expression L is called a* path expression, *and is assumed to conform to the grammar "$\mathsf{Pf} ::= f.\,\mathsf{Pf} \mid \mathsf{Id}$". The set of path expressions generated by L is denoted $\mathcal{L}(L)$.*

Definition 3 (Syntax and Semantics of $\mathcal{DLF}_{\text{reg}}$ and $\mathcal{DLFD}_{\text{reg}}$) *Let C be a set of primitive concepts disjoint from F. The derived* concept descriptions *for $\mathcal{DLFD}_{\text{reg}}$ are defined by the grammar in Figure 1. The sequence of productions for this grammar define a collection of* concept constructors. *The constructor $\sqcap$ is referred to as* concept conjunction, *$\neg$ as* concept negation,

SYNTAX		SEMANTICS: DEFN OF "$(\cdot)^{\mathcal{I}}$"
D ::=	C	$(C)^{\mathcal{I}} \subseteq \Delta$
|	$\top$	Δ
|	$D_1 \sqcap D_2$	$(D_1)^{\mathcal{I}} \cap (D_2)^{\mathcal{I}}$
|	$\neg D$	$\Delta \setminus (D)^{\mathcal{I}}$
|	$\forall L.D$	$\{x \in \Delta : (\mathsf{Pf})^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}$ for all $\mathsf{Pf} \in \mathcal{L}(L)\}$
|	$\exists L.D$	$\{x \in \Delta : (\mathsf{Pf})^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}$ for some $\mathsf{Pf} \in \mathcal{L}(L)\}$
E ::=	D	
|	$D : L \to L'$	$\Big\{x \in \Delta : \forall y \in (D)^{\mathcal{I}}. \Big(\bigwedge_{\mathsf{Pf} \in \mathcal{L}(L)} (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\Big) \Rightarrow \Big(\bigwedge_{\mathsf{Pf} \in \mathcal{L}(L')} (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\Big)\Big\}$

Fig. 1. SYNTAX AND SEMANTICS OF $\mathcal{DLF}_{\mathrm{reg}}$ AND $\mathcal{DLFD}_{\mathrm{reg}}$.

and $\forall L.$ *and* $\exists L.$ *as* regular attribute restriction. *A concept formed by an application of the final production in the grammar is called* equality generating. *The derived concept descriptions for $\mathcal{DLF}_{\mathrm{reg}}$ are those in $\mathcal{DLFD}_{\mathrm{reg}}$ that are not equality generating.*[1]

An inclusion dependency $\mathcal{C}$ *is an expression of the form* $D \sqsubseteq E$. *If* E *is an equality generating concept, then the dependency has the form*

$$D_1 \sqsubseteq D_2 : L_1 \to L_2,$$

and is called a regular path functional dependency *(regular PFD).*

The semantics *of expressions is given with respect to a structure* $(\Delta, \cdot^{\mathcal{I}})$, *where* Δ *is a domain of "objects", and* $(.)^{\mathcal{I}}$ *an interpretation function that fixes the interpretations of primitive concepts to be subsets of* Δ *and attributes to be total functions over* Δ. *The interpretation is extended to path expressions,* $(\mathsf{Id})^{\mathcal{I}} = \lambda x.x$ *and* $(f.\,\mathsf{Pf})^{\mathcal{I}} = (\mathsf{Pf})^{\mathcal{I}} \circ (f)^{\mathcal{I}}$, *and to derived concept descriptions, cf. Figure 1.*

An interpretation satisfies an inclusion dependency $D \sqsubseteq E$ *if* $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$.

A terminology $\mathcal{T}$ *consists of a finite set of inclusion dependencies* $\{\mathcal{C}_1, ..., \mathcal{C}_m\}$. *The* logical implication problem *asks if* $\mathcal{T} \models \mathcal{C}$ *holds; that is, if all interpretations that satisfy each inclusion dependency in* $\mathcal{T}$ *must also satisfy the* inclusion dependency $\mathcal{C}$ *(often called the* posed question*).*

[1] Some of the constructors have been introduced to simplify our presentation and do not contribute to the overall expressiveness of our DL dialects. In particular, $\top$ corresponds to $\neg(C \sqcap \neg C)$, for an arbitrary primitive concept C, and $\exists L.D$ corresponds to $\neg(\forall L.\neg D)$.

Notation 4 We simplify the notation for path expressions in the rest of the paper by omitting the trailing Id, and also write $\mathsf{Pf}_1 \,.\, \mathsf{Pf}_2$ to denote a syntactic composition of Pf_2 with Pf_1.

2.2 Datalog with limited use of Successor Functions

There has been a great deal of work by database theoreticians on Datalog, a "function-free" class of logic programs [20,21]. More recently, Chomicki and Imeliński [17] have shown how Datalog extended with unary "successor" functions in the manner characterized by the following definitions retains decidability of an associated satisfiability problem.

Definition 5 (Datalog$_{nS}$ [17]) *Let* 0 *be a constant symbol,* x *a variable, and* f_i *unary function symbols. Datalog*$_{nS}$ terms *are of the form* 0, x, *or* $f_i(t)$ *given another Datalog*$_{nS}$ *term* t. *A term is* ground *if it does not contain the variable* x. *An* atom *is a monadic predicate applied on a term,* $P(t)$; *a* ground atom *is a monadic predicate applied to a ground term. A Datalog*$_{nS}$ *program is a finite set of* definite Horn *clauses of the form*

$$P \leftarrow Q_1 \wedge \ldots \wedge Q_n$$

constructed from atoms P *and* Q_i. *A* recognition problem *for a Datalog*$_{nS}$ *program* Π *and a ground atom* Q, *written* $\Pi \models Q$, *is to determine if* Q *is true in all models of* Π.

A recognition problem $\Pi \models Q$ is equivalent to determining if $\Pi \cup \{\neg Q\}$ is not satisfiable. This is in turn equivalent to determining if $Q \in T_\Pi^\omega(\emptyset)$, where T_Π is the immediate consequence operator associated with the program Π [22].

Although Chomicki and Imeliński consider a more general variety of Datalog$_{nS}$ programs [17], the formulation above has the same complexity and will simplify the presentation of our lower bound results. Our results relating to upper bounds will use Datalog$^{\neg}_{nS}$ which is an extension of Datalog$_{nS}$ that allows the use of negated literals in bodies of clauses in Datalog programs.

Definition 6 (Datalog$^{\neg}_{nS}$) *A* Datalog$^{\neg}_{nS}$ program Π *is a finite set of clauses of the form*

$$P \leftarrow Q_1 \wedge \ldots \wedge Q_n \wedge \neg R_1 \wedge \ldots \wedge \neg R_k$$

and integrity constraints

$$\leftarrow Q_1 \wedge \ldots \wedge Q_n \wedge \neg R_1 \wedge \ldots \wedge \neg R_k$$

constructed from atoms P, Q_i, *and* R_j.

Unfortunately, $\text{Datalog}_{nS}^{\neg}$ programs are not able to adopt the simple *minimal model* semantics of Datalog_{nS}. However, Gelfond and Lifschitz have extended the minimal model semantics approach and defined *stable models* of $\text{Datalog}_{nS}^{\neg}$ programs, models that share many of the properties of minimal models [23]. Indeed, these models are defined as minimal models of a transformed $\text{Datalog}_{nS}^{\neg}$ program using the Gelfond-Lifschitz (GL) transformation.

Definition 7 ($\text{Datalog}_{nS}^{\neg}$ semantics) *A* satisfiability problem *for a $\text{Datalog}_{nS}^{\neg}$ program Π is to determine if Π has a stable model compatible with all integrity constraints in Π.*

We assume that the unary function symbols are from the same alphabet F used for primitive attributes. Thus, for example, path expressions in $\mathcal{DLFD}_{\text{reg}}$ and terms in $\text{Datalog}_{nS}^{\neg}$ can naturally correspond to each other.

Notation 8 For each path expression $\mathsf{Pf} = f_1. \cdots .f_k. \mathsf{Id}$, we define the pair of $\text{Datalog}_{nS}^{\neg}$ functional terms,

$$\overline{\mathsf{Pf}}(0) = f_k(\cdots f_1(0) \cdots) \text{ and}$$
$$\overline{\mathsf{Pf}}(x) = f_k(\cdots f_1(x) \cdots),$$

where 0 is a distinguished constant and x is a variable. Similarly, for every $\text{Datalog}_{nS}^{\neg}$ term $t(x) = f_1(\cdots f_k(x) \cdots)$ there is a path expression $\mathsf{Pf} = f_k. \cdots .f_1. \mathsf{Id}$ such that $t(x) = \overline{\mathsf{Pf}}(x)$.

Similarly, we can establish a relationship between monadic predicate symbols in $\text{Datalog}_{nS}^{\neg}$ and concept descriptions in $\mathcal{DLF}_{\text{reg}}$(since their interpretations correspond to sets of objects).

It is known that a $\text{Datalog}_{nS}^{\neg}$ program has a model if and only if it has a Herbrand model [16,17,22]. As a consequence, syntactic techniques can be used for model construction. This allows us to establish the complexity bounds for $\text{Datalog}_{nS}^{\neg}$ problems as follows:

Proposition 9 *The satisfiability problem for $\text{Datalog}_{nS}^{\neg}$ is DEXPTIME-complete. The lower bound holds for Datalog_{nS} programs.*

<u>**Proof:**</u> (sketch) Fernández, *et al* [24] define an *evidential transformation* that, given a $\text{Datalog}_{nS}^{\neg}$ program, constructs a disjunctive Datalog_{nS} program and a set of integrity constraints. The minimal models of such a program that satisfy the constraints coincide with the stable models of the original $\text{Datalog}_{nS}^{\neg}$ program. Note that such minimal models can in turn be constructed by iterating an appropriate consequence operator [25–28]. Furthermore, Chomicki and Imieliński [17] have defined a finite graph representation for minimal models.

Their technique immediately generalizes to finite sets of minimal models for disjunctive Datalog$_{nS}$, which in turn yields a decidability result and a DEXPTIME upper bound on the satisfiability problem for Datalog$^{\neg}_{nS}$. (Note that checking the integrity constraints can be accomplished in time proportional to the size of the finite graphs.)

The lower bound is presented in [17] by building on results for satisfiability of Ackerman class formulae [29,30]. $\square$

In particular, the extension of Datalog$_{nS}$ to Datalog$^{\neg}_{nS}$ maintains a DEXPTIME complexity upper bound for its satisfiability problem. Note that another way of establishing this result would be to follow the automata connection for μ-calculus [31,32] by translating Datalog$^{\neg}_{nS}$ programs to formulas with explicit fixpoints. In this case, successor functions would correspond to modalities in μ-calculus formulas.

3 Lower Bounds

In this section, we prove that very simple formulations of regular attribute restriction of the form $\forall f.D$ in $\mathcal{DLF}_{\mathrm{reg}}$ terminologies, and of regular PFDs of the form

$$D : \{\mathsf{Pf}_1, \ldots, \mathsf{Pf}_k\} \to \mathsf{Pf}$$

in $\mathcal{DLFD}_{\mathrm{reg}}$ terminologies can separately lead to DEXPTIME hardness for their respective logical implication problems.

3.1 Lower Bound due to Regular Attribute Restrictions

We show that every Datalog$_{nS}$ recognition problem can be simulated by a $\mathcal{DLF}_{\mathrm{reg}}$ implication problem. For this reduction, each monadic predicate symbol is assumed to also qualify as a primitive concept name in $\mathcal{DLF}_{\mathrm{reg}}$.

Definition 10 *Let Π be a Datalog$_{nS}$ program and $P(\overline{\mathsf{Pf}}(0))$ a ground atom. We construct an implication problem for $\mathcal{DLF}_{\mathrm{reg}}$ as follows: a terminology from clauses in Π,*

$$\begin{aligned}\mathcal{T}_\Pi = \{\ \forall\, \mathsf{Pf}_1 .Q_1 \sqcap \ldots \sqcap \forall\, \mathsf{Pf}_k .Q_k \sqsubseteq \forall\, \mathsf{Pf}\, .P :&\\ P(\overline{\mathsf{Pf}}(x)) \leftarrow Q_1(\overline{\mathsf{Pf}}_1(x)), \ldots, Q_k(\overline{\mathsf{Pf}}_k(x)) \in \Pi\ \}&,\end{aligned}$$

and the posed question from ground facts $Q_i(\overline{\mathsf{Pf}}_i(0)) \in \Pi$ and the ground atom

$G = P(\overline{\mathsf{Pf}}(0))$,

$$\mathcal{C}_{\Pi,G} = \forall\,\mathsf{Pf}_1\,.Q_1 \sqcap \cdots \sqcap \forall\,\mathsf{Pf}_k\,.Q_k \sqsubseteq \forall\,\mathsf{Pf}\,.P,$$

where $\forall\,\mathsf{Pf}\,.D$ *denotes the description* $\forall L.D$ *where* $\mathcal{L}(L) = \{\mathsf{Pf}\}$.

Theorem 11 *Let* Π *be a Datalog*$_{nS}$ *program and* G *a ground atom. Then*

$$\Pi \models G \iff \mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}.$$

Proof: $\Rightarrow$: We show that $P(\overline{\mathsf{Pf}}(0)) \in T_\Pi^\omega(\emptyset)$ implies $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.

If $P(\overline{\mathsf{Pf}}(0)) \in T_\Pi^\omega(\emptyset)$ then there must be $m > 0$ such that $p(\overline{\mathsf{Pf}}(0)) \in T_\Pi^m(\emptyset)$. Then, by induction on m, we have:

$m = 1$: immediate as it must be the case that $p(\overline{\mathsf{Pf}}(0))$ must be one of the facts in P, e.g., $Q_i(\overline{\mathsf{Pf}}_i(0))$. Then, however, the inclusion dependency $\mathcal{C}_{\Pi,G}$ is of the form

$$\mathcal{C}_{\Pi,G} = \forall\,\mathsf{Pf}_1\,.Q_1 \sqcap \ldots \sqcap \forall\,\mathsf{Pf}_k\,.Q_k \sqsubseteq \forall\,\mathsf{Pf}_i\,.Q_i,$$

is trivial and is satisfied by all interpretations. Therefore, $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.

$m > 1$: if $P(\overline{\mathsf{Pf}}(0)) \in T_\Pi^m(\emptyset)$ then there is a term $t(0)$ (to be substituted for x) and a clause $P(\overline{\mathsf{Pf}}'(x)) \leftarrow Q'_1(\overline{\mathsf{Pf}}'_1(x)), \ldots, Q'_l(\overline{\mathsf{Pf}}'_l(x))$ in Π such that $\overline{\mathsf{Pf}}(0) = \overline{\mathsf{Pf}}'(t(0))$ and $Q'_i(\overline{\mathsf{Pf}}'_i(t(0))) \in T_\Pi^{m-1}(\emptyset)$. By the inductive hypothesis we have $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,Q'_i(\overline{\mathsf{Pf}}'_i(t(0)))}$ and therefore, by composition with the inclusion dependency

$$\forall\,\mathsf{Pf}_1\,.Q'_1 \sqcap \ldots \sqcap \forall\,\mathsf{Pf}_k\,.Q'_l \sqsubseteq \forall\,\mathsf{Pf}\,.P,$$

that must exist by virtue of the definition of $\mathcal{T}_\Pi$, we have $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.

$\Leftarrow$: Assume $P(\overline{\mathsf{Pf}}(0)) \notin T_\Pi^\omega(\emptyset)$. We construct an interpretation $\mathcal{I} = (\Delta, (.)^\mathcal{I})$ that satisfies $\mathcal{T}_\Pi$ but violates $\mathcal{C}_{\Pi,G}$ as follows: Let Δ be the set of all Datalog$_{nS}^\neg$ terms. We define

$$(f_i)^\mathcal{I}(t) = f(t)$$
$$t \in (P)^\mathcal{I} \iff P(t) \in T_\Pi^\omega(\emptyset).$$

We show that $\mathcal{I} \models \mathcal{T}_\Pi$. Consider an inclusion dependency

$$\forall\,\mathsf{Pf}_1\,.Q_1 \sqcap \ldots \sqcap \forall\,\mathsf{Pf}_k\,.Q_k \sqsubseteq \forall\,\mathsf{Pf}\,.P \in \mathcal{T}_\Pi.$$

This dependency could only be violated if there was an object $t \in \Delta$ such that $(\mathsf{Pf}_i)^\mathcal{I}(t) \in (Q_i)^\mathcal{I}$ for $0 < i \leq k$ and $(\mathsf{Pf})^\mathcal{I}(t) \notin (P)^\mathcal{I}$. This, in turn implies $Q_i(\overline{\mathsf{Pf}}_i(t)) \in T_\Pi^\omega(\emptyset)$ while $P(\overline{\mathsf{Pf}}(t)) \notin T_\Pi^\omega(\emptyset)$. However, this contradicts the fact that $T_\Pi^\omega(\emptyset)$ is a model of Π, since

$$P(\overline{\mathsf{Pf}}(x)) \leftarrow Q_1(\overline{\mathsf{Pf}}_1(x)), \ldots, Q_k(\overline{\mathsf{Pf}}_k(x)) \in \Pi.$$

Thus $\mathcal{I} \models \mathcal{T}_\Pi$.

Conversely, $\mathcal{I} \not\models \mathcal{C}_{\Pi,G}$: $0 \in (\forall\, \mathsf{Pf}_i\,.Q_i)^{\mathcal{I}}$ since $Q_i(\overline{\mathsf{Pf}}_i(0)) \in T_{\Pi}^{\omega}(\emptyset)$, but $0 \notin (\forall\, \mathsf{Pf}\,.P)^{\mathcal{I}}$ since we have assumed $P(\overline{\mathsf{Pf}}(0)) \notin T_{\Pi}^{\omega}(\emptyset)$. □

Observe that the constructed implication problem satisfies $|\mathcal{T}_{\Pi}| + |\mathcal{C}_{\Pi,G}| \in \mathcal{O}(|\Pi| + |G|)$. The lower complexity bound thus follows from [17,30].

Corollary 12 *The implication problem for $\mathcal{DLF}_{\mathrm{reg}}$ is DEXPTIME hard.*

Since concept negation is not used in our reduction and since all regular attribute dependencies $\forall\, \mathsf{Pf}\,.D$ generated by the reduction can be rewritten $\forall f_1. \cdots .\forall f_k.D$, this remains true for the fragment of $\mathcal{DLF}_{\mathrm{reg}}$ in which all concept constructors are conjunctions and attribute restrictions of the form "$\forall f.D$".

3.2 Lower Bound due to Regular PFDs

We study the problem for a subset of $\mathcal{DLFD}_{\mathrm{reg}}$ in which all subsumption constraints are of the form

$$\top \sqsubseteq \top : \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \rightarrow \mathsf{Pf}\,.$$

An implication problem in this subset is called a *PFD membership problem.*

To reduce notational overhead in the remainder of this section, we use the shorthand $\mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \rightarrow \mathsf{Pf}$ for such constraints. It will also simplify matters to assume that each monadic predicate symbol P in Datalog$_{nS}$ maps to a distinct attribute p in $\mathcal{DLFD}_{\mathrm{reg}}$, and that each such p differs from the attributes corresponding to unary function symbols in Datalog$_{nS}$.

Note that the above form is a special case of regular PFDs in which the associated regular path function sets are finite.

Definition 13 *Let Π be an arbitrary Datalog$_{nS}$ program and $G = P(\overline{\mathsf{Pf}}(0))$ a ground atom. We construct an implication problem for $\mathcal{DLFD}_{\mathrm{reg}}$ as follows: a terminology from clauses in Π,*

$$\begin{aligned}\mathcal{T}_{\Pi} = \{\ \{&\mathsf{Pf}'_1\,.p'_1, \ldots, \mathsf{Pf}'_k\,.p'_k \rightarrow \mathsf{Pf}'\,.p' :\\ &P'(\overline{\mathsf{Pf}}'(x)) \leftarrow P'_1(\overline{\mathsf{Pf}}'_1(x)), \ldots, P'_k(\overline{\mathsf{Pf}}'_k(x)) \in \Pi\ \},\end{aligned}$$

and the posed question,

$$\mathcal{C}_{\Pi,G} = \mathsf{Pf}_1\,.p_1, \ldots, \mathsf{Pf}_k\,.p_k \rightarrow \mathsf{Pf}\,.p,$$

where $P_1(\overline{\mathsf{Pf}}_1(0)), \ldots, P_k(\overline{\mathsf{Pf}}_k(0))$ are the ground facts in Π.

Theorem 14 *Let Π be an arbitrary Datalog$_{nS}$ program and $G = P(\overline{\mathsf{Pf}}(0))$ a ground atom. Then $\Pi \models G \iff \mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.*

<u>**Proof:**</u> $\Rightarrow$: We show that $P(\overline{\mathsf{Pf}}(0)) \in T^\omega_\Pi(\emptyset)$ implies $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.

If $P(\overline{\mathsf{Pf}}(0)) \in T^\omega_\Pi(\emptyset)$ then there must be $m > 0$ such that $P(\overline{\mathsf{Pf}}(0)) \in T^m_\Pi(\emptyset)$. Then, by induction on m, we have:

$m = 1$: immediate as it must be the case that $P(\overline{\mathsf{Pf}}(0))$ must be one of the facts in Π, e.g., $P_i(\overline{\mathsf{Pf}}_i(0))$ and therefore also $\mathsf{Pf}\,.p = \mathsf{Pf}_i\,.p_i$. Consequently, $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$ as $\mathcal{C}_{\Pi,G}$ is a trivial path-functional dependency.

$m > 1$: if $P(\overline{\mathsf{Pf}}(0)) \in T^m_\Pi(\emptyset)$ then there is a term $t(0)$ (to be substituted for x) and a clause $P(\overline{\mathsf{Pf}}'(x)) \leftarrow P_1(\overline{\mathsf{Pf}}'_1(x)), \ldots, P_l(\overline{\mathsf{Pf}}'_l(x))$ in Π: such that $\overline{\mathsf{Pf}}(0) = \overline{\mathsf{Pf}}'(t(0))$ and $P_i(\overline{\mathsf{Pf}}'_i(t(0))) \in T^{m-1}_\Pi(\emptyset)$. By the inductive hypothesis we have $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,P_i(\overline{\mathsf{Pf}}'_i(t(0)))}$ and therefore, by composition with the path-functional dependency $\mathsf{Pf}'_1\,.p_1, \ldots, \mathsf{Pf}'_l\,.p_l \to \mathsf{Pf}\,.p \in \mathcal{T}_\Pi$ we have $\mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.

$\Leftarrow$: Assume $P(\overline{\mathsf{Pf}}(0)) \notin T^\omega_\Pi(\emptyset)$. We construct an counterexample interpretation as follows: let $o_1, o_2 \in \Delta$ be two distinct objects and T_1, T_2 two complete infinite trees rooted by these two objects with edges labeled by primitive attributes f_i and p_i. Moreover, if $P'(\overline{\mathsf{Pf}}'(0)) \in T^\omega_\Pi(\emptyset)$ we merge the two subtrees identified by the path $\mathsf{Pf}'\,.p'$ starting from the respective roots of the trees. The resulting (directed acyclic) graph provides an interpretation in which the nodes of the graph represent elements of Δ and edges give interpretation to primitive attributes such that:

(1) $(\mathsf{Pf}_i\,.p_i)^\mathcal{I}(o_1) = (\mathsf{Pf}_i\,.p_i)^\mathcal{I}(o_2)$ for all $P_i(\overline{\mathsf{Pf}}_i(0)) \in \Pi$, and

(2) every constraint in $\mathcal{T}_\Pi$ is satisfied by the constructed interpretation. Assume to the contrary, that the interpretation violated a constraint

$$\mathsf{Pf}'_1\,.p'_1, \ldots, \mathsf{Pf}'_l\,.p'_l \to \mathsf{Pf}'\,.p' \in \mathcal{T}_\Pi.$$

Then there would have to be two distinct elements x_1, x_2 such that $(\mathsf{Pf}'_i\,.p'_i)^\mathcal{I}(x_1) = (\mathsf{Pf}'_i\,.p'_i)^\mathcal{I}(x_2)$ but $(\mathsf{Pf}'\,.p')^\mathcal{I}(x_1) \neq (\mathsf{Pf}'\,.p')^\mathcal{I}(x_2)$. From the construction of the interpretation and the fact that the sets of predicate and function symbols are disjoint we know that $P'_i(\overline{\mathsf{Pf}}'_i(t(0))) \in T^\omega_\Pi(\emptyset)$ where t is a term corresponding to the paths from o_1 and o_2 to x_1 and x_2, respectively (note that all the paths that end in a particular common node in the constructed interpretation are symmetric). However, then $P'(\overline{\mathsf{Pf}}'(t(0))) \in T^\omega_\Pi(\emptyset)$ using the clause in Π associated with the violated constraint in $\mathcal{T}_\Pi$, and thus $(\mathsf{Pf}'\,.p')^\mathcal{I}(x_1) = (\mathsf{Pf}'\,.p')^\mathcal{I}(x_2)$, a contradiction.

But then $(\mathsf{Pf}\,.p)^\mathcal{I}(o_1) \neq (\mathsf{Pf}\,.p)^\mathcal{I}(o_2)$ since $P(\overline{\mathsf{Pf}}(0)) \notin T^\omega_\Pi(\emptyset)$, a contradiction.
$\square$

Again, for the constructed implication problem we have $|\mathcal{T}_\Pi| + |\mathcal{C}_{\Pi,G}| \in \mathcal{O}(|\Pi| + |G|)$. Thus:

Corollary 15 *The PFD logical implication problem is DEXPTIME-hard.*

4 Decision Procedures

We now consider the other direction: can $\mathrm{Datalog}_{nS}^{\neg}$ simulate implication problems in $\mathcal{DLF}_{\mathrm{reg}}$ and various extensions, including $\mathcal{DLFD}_{\mathrm{reg}}$? We show how this can done in this section. We begin by exhibiting a reduction to $\mathrm{Datalog}_{nS}^{\neg}$ satisfiability problems for implication problems in $\mathcal{DLF}_{\mathrm{reg}}$. The case for implication problems in $\mathcal{DLFD}_{\mathrm{reg}}$, which extends $\mathcal{DLF}_{\mathrm{reg}}$ with regular PFD constraints, is considered next. And finally, we consider reductions for $\mathcal{DLF}_{\mathrm{reg}}$ extended with other constructs relating to *roles* that are commonly available in DL dialects. In our various simulations, $\mathcal{DLF}_{\mathrm{reg}}$'s concept descriptions D are modeled by monadic predicates $P_D(x)$.

4.1 A Decision Procedure for $\mathcal{DLF}_{\mathrm{reg}}$

Our construction of a corresponding $\mathrm{Datalog}_{nS}^{\neg}$ satisfiability problem for a given $\mathcal{DLF}_{\mathrm{reg}}$ implication problem proceeds in three steps. The first encodes the structural properties of regular expression sets as $\mathrm{Datalog}_{nS}^{\neg}$ assertions.

Definition 16 (Constraints for Regular Path Expressions)
Let P be a monadic predicate and let G_L denote a right-linear grammar for a given regular path expression L.[2] *We define a $\mathrm{Datalog}_{nS}^{\neg}$ program*

$$\Pi_{P,L} = \{N_{P,L,A}(x) \leftarrow \mathcal{M}(P,\alpha) : A \rightarrow \alpha \in G_L\}$$

where

$$\mathcal{M}(P,\alpha) = \begin{cases} P(x) & \text{if } \alpha = \mathsf{Id}; \\ P(f(x)) & \text{if } \alpha = f; \\ N_{P,L,B}(f(x)) & \text{if } \alpha = fB. \end{cases}$$

For any pair of regular path expressions L_1 and L_2, we assume the nonterminal symbols in G_{L_1} and G_{L_2} are unique, and define $\Pi_{\mathcal{L}}$ as the union of all $\Pi_{P,L}$.

Intuitively, the atomic formula $N_{P,L,A}(t)$ is true for a term t if and only if there is $\mathsf{Pf} \in \mathcal{L}(A, G_L)$ such that $P(\overline{\mathsf{Pf}}(t))$ is true, where $\mathcal{L}(A, G_L)$ is the language

[2] A grammar is right regular if and only if each production is of the form $A \rightarrow \mathsf{Id}$, $A \rightarrow a$ or $A \rightarrow bB$, where A and B are nonterminal symbols and a and b are terminal symbols. It is well known that such a grammar exists for any regular expression [33].

generated by G_L from the nonterminal A. Note that it is essential to use the *minimal model semantics* to define $N_{P,L,A}(x)$. However, the rules that define the atoms associated with G_L's nonterminal symbols are Datalog$_{nS}$ rules and thus a unique minimal model exists and can be equivalently defined by an explicit *least fixpoint formula.*

Lemma 17 *Let Π be a Datalog$^{\neg}_{nS}$ program, such that $\Pi_{P,L} \subseteq \Pi$ and in which the literals $N_{P,L,A}$ can appear on the left hand side of a rule only in $\Pi_{P,L}$. Then, in any stable model M of Π we have $N_{P,L,A}(t) \in M \iff P(\overline{\mathsf{Pf}}(t)) \in M$ for some $\mathsf{Pf} \in \mathcal{L}(A, G_L)$.*

Proof: The if direction can be easily shown by induction on the length of a derivation of Pf (from A) in G_L; the only-if direction follows from minimality of M, since only clauses in $\Pi_{P,L}$ can assert $N_{P,L,A}$ in M. □

The second step encodes the structural properties of $\mathcal{DLF}_{\text{reg}}$ as Datalog$^{\neg}_{nS}$ assertions.

Definition 18 ($\mathcal{DLF}_{\text{reg}}$ Concept Formation Constraints) *Let D, D_1, and D_2 range over concept descriptions and L over regular path expressions. We define*

$$\Pi_{\mathcal{DLF}_{\text{reg}}} = \Pi_{\mathcal{L}} \cup \left\{ \begin{array}{rl} P_D(x) & \leftarrow \neg P_{\neg D}(x), \\ P_{\neg D}(x) & \leftarrow \neg P_D(x), \\ & \leftarrow P_D(x) \wedge P_{\neg D}(x), \\ P_{D_1 \sqcap D_2}(x) & \leftarrow P_{D_1}(x) \wedge P_{D_2}(x), \\ P_{D_1}(x) & \leftarrow P_{D_1 \sqcap D_2}(x), \\ P_{D_2}(x) & \leftarrow P_{D_1 \sqcap D_2}(x), \\ P_{\exists L.D}(x) & \leftarrow N_{P_D,L,S_L}(x), \\ & \leftarrow P_{\exists L.D}(x) \wedge \neg N_{P_D,L,S_L}(x), \end{array} \right\},$$

where S_L are the start symbols in the grammars G_L for regular path expressions L.

To simplify the development in the rest of the paper we assume that the $\forall L.D$ concept constructor is a shorthand for $\neg\exists L.\neg D$.

The set $\Pi_{\mathcal{DLF}_{\text{reg}}}$ captures the structural relationships between $\mathcal{DLF}_{\text{reg}}$ concepts. Although the set is infinite in general, the set of concepts and regular path expressions appearing in a particular implication problem, $\mathcal{T} \models \mathcal{C}$, is finite. Hence, one can restrict the set of assertions in $\Pi_{\mathcal{DLF}_{\text{reg}}}$ to a finite subset

$\Pi^{\mathcal{T},\mathcal{C}}_{\mathcal{DLF}_{\text{reg}}}$ that contains only predicates that define concepts and regular grammars in $\mathcal{T} \cup \{\mathcal{C}\}$. In the following, we omit the superscripts whenever clear from the context.

The translation of the inclusion dependencies is the final third step in the overall translation of a $\mathcal{DLF}_{\text{reg}}$ implication problem.

Definition 19 *Let $\mathcal{T}$ and $\mathcal{C} \equiv D \sqsubseteq E$ be a $\mathcal{DLF}_{\text{reg}}$ terminology and an inclusion dependency, respectively. We define*

$$\Pi_{\mathcal{T}} = \{P_E(x) \leftarrow P_D(x) : D \sqsubseteq E \in \mathcal{T}\}$$

and

$$\Pi_{\mathcal{C}} = \{P_D(0) \leftarrow, \leftarrow P_E(0)\}.$$

The two clauses $\Pi_{\mathcal{C}}$ represent the skolemized version of $\neg\forall x.P_D(x) \rightarrow P_E(x)$; 0 is the Skolem constant for x. As usual, a model "containing" $\Pi_{\mathcal{C}}$ is a counterexample for $\mathcal{C}$.

Lemma 20 *Let $\Pi = \Pi_{\mathcal{DLF}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ be a Datalog$^{\neg}_{nS}$ program. Then in any stable model M of Π we have:*

(1) $P_D(t) \in M \iff P_{\neg D}(t) \notin M$,
(2) $P_{D_1 \sqcap D_2}(t) \in M \iff P_{D_1}(t) \in M \wedge P_{D_2}(t) \in M$, and
(3) $P_{\exists L.D}(t) \in M \iff P_D(\overline{\mathsf{Pf}}(t)) \in M$ for some $\mathsf{Pf} \in \mathcal{L}(L)$

for all ground terms t and descriptions D that appear in $\mathcal{T} \cup \{\mathcal{C}\}$.

Proof:

(1) By case analysis: $P_D(t), P_{\neg D}(t) \in M$ violates the $\leftarrow P_D(x) \wedge P_{\neg D}(x) \in \Pi$ constraint; if $P_D(t), P_{\neg D}(t) \notin M$ then the clauses $P_D(t) \leftarrow$ and $P_{\neg D}(t) \leftarrow$ are elements of the Gelfond-Lifschitz transformation of Π and thus M is not a stable model of Π. The remaining cases satisfy (1).

(2) Immediate from definition of $\Pi_{\mathcal{DLF}_{\text{reg}}}$.

(3) Follows from Lemma 17 and the clauses constraining $P_{\exists L.D}$ in $\Pi_{\mathcal{DLF}_{\text{reg}}}$. $\Box$

Theorem 21 *Let $\mathcal{T}$ and $\mathcal{C}$ be a $\mathcal{DLF}_{\text{reg}}$ terminology and inclusion dependency, respectively. Then $\mathcal{T} \models \mathcal{C} \iff \Pi_{\mathcal{DLF}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ is not satisfiable.*[3]

Proof: Let M be a stable model of $\Pi_{\mathcal{DLF}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$. We construct an interpretation $\mathcal{I}_M = (\Delta, (.)^{\mathcal{I}})$ that satisfies $\mathcal{T}$ but violates $\mathcal{C}$ as follows: Let Δ

[3] Recall our assumption that satisfiability for Datalog$^{\neg}_{nS}$ is defined with respect to stable models.

be the set of all $\text{Datalog}_{nS}^{\neg}$ terms. We define

$$(f_i)^{\mathcal{I}}(t) = f(t)$$
$$t \in (C_j)^{\mathcal{I}} \iff P_{C_j}(t) \in M$$

for primitive attributes f_i and primitive concept descriptions C_j. We first show, by induction on the structure of D, that $t \in (D)^{\mathcal{I}} \iff P_D(t) \in M$ holds for every concept description D. The base case is given above, the inductive cases follow from Lemma 20.

Now it is easy to show that $\mathcal{I}_M \models \mathcal{T}$ since for every inclusion dependency $D \sqsubseteq E \in \mathcal{T}$ we have a clause $P_E(x) \leftarrow P_D(x) \in \Pi_{\mathcal{T}}$. However, $\mathcal{I}_M \not\models \mathcal{C}$ since $P_D(0) \in M$ but $P_E(0) \notin M$ as M is a model for $\Pi_{\mathcal{C}}$.

On the other hand, let $\mathcal{I} = (\Delta, (.)^{\mathcal{I}})$ be an interpretation such that $\mathcal{I} \models \mathcal{T}$ but $\mathcal{I} \not\models \mathcal{C}$. Then there must be an object $o \in \Delta$ such that $o \in (D)^{\mathcal{I}} - (E)^{\mathcal{I}}$. We construct a Herbrand interpretation M for $\Pi_{\mathcal{DLF}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ as follows

$$\begin{aligned} M_{\mathcal{I}} = \{&P_D(\overline{\mathsf{Pf}}(0)) : o.(\mathsf{Pf})^{\mathcal{I}} \in (D)^{\mathcal{I}}\} \\ \cup\ \{&N_{P_D,L,A}(\overline{\mathsf{Pf}}(0)) : o.(\mathsf{Pf})^{\mathcal{I}}.(\mathsf{Pf}')^{\mathcal{I}} \in (D)^{\mathcal{I}}, \mathsf{Pf}' \in \mathcal{L}(A, G_L)\} \end{aligned}$$

where G_L are the right-regular grammars associated with regular path expressions in $\exists L.D$ that appear in $\mathcal{T} \cup \{\mathcal{C}\}$. The interpretation $M_{\mathcal{I}}$ is a stable model of $\Pi_{\mathcal{DLF}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$, since it is the minimal model of the GL-transformation of $\Pi_{\mathcal{DLF}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ with respect to $M_{\mathcal{I}}$ (the only clauses affected by the transformation are the structural rules $\Pi_{\mathcal{DLF}_{\text{reg}}}$). The remainder of the proof is a straightforward case analysis; minimality follows from Lemma 20 (since in any interpretation either $o \in (D)^{\mathcal{I}}$ or $o \in (\neg D)^{\mathcal{I}}$ for all $o \in \Delta$) and Lemma 17 (since whenever $o \in (\exists L.D)^{\mathcal{I}}$ then there must be a $\mathsf{Pf} \in \mathcal{L}(L)$ such that $o.(\mathsf{Pf})^{\mathcal{I}} \in (D)^{\mathcal{I}}$). □

This result shows that $\mathcal{DLF}_{\text{reg}}$ is essentially an alternative *variable-free* syntax for $\text{Datalog}_{nS}^{\neg}$. Also, as a consequence we have:

Corollary 22 *The implication problem for $\mathcal{DLF}_{\text{reg}}$ is DEXPTIME-complete.*

4.2 Adding Regular PFDs: $\mathcal{DLFD}_{\text{reg}}$

We now consider $\mathcal{DLFD}_{\text{reg}}$ which adds equality generating concepts to $\mathcal{DLF}_{\text{reg}}$. In particular, we consider logical implication problems with inclusion dependencies that correspond to regular PFDs. Recall that such dependencies have the form $D_1 \sqsubseteq D_2 : L_1 \to L_2$.

There are two cases to consider that, for a $\mathcal{DLFD}_{\text{reg}}$ implication problem $\mathcal{T} \models \mathcal{C}$, depend on the structure of $\mathcal{C}$.

Case 1: $\mathcal{C}$ is not a regular PFD. In this case, it is straightforward to show that any regular PFDs occurring in $\mathcal{T}$ will not interfere with the original decision procedure from Section 4.1.

Lemma 23 *Let $\mathcal{T}$ be a $\mathcal{DLFD}_{\text{reg}}$ terminology, $\mathcal{T}'$ the set of inclusion dependencies in $\mathcal{T}$ that are not regular PFD, and $\mathcal{C}$ a $\mathcal{DLFD}_{\text{reg}}$ concept that is not a regular PFD. Then if $\mathcal{T}' \models \mathcal{C}$ then $\mathcal{T} \models \mathcal{C}$.*

Proof: An interpretation $\mathcal{I}$ that satisfies $\mathcal{T}$ also satisfies $\mathcal{T}'$ and thus, since $\mathcal{C}$ is violated by $\mathcal{I}$, it cannot be a consequence of $\mathcal{T}'$.

On the other hand, given an interpretation $\mathcal{I}$ of $\mathcal{T}'$ that violates $\mathcal{C}$, we can construct an interpretation $\mathcal{I}_{M_\mathcal{I}}$ using the construction used in the proof of Theorem 21. This new interpretation, $\mathcal{I}_{M_\mathcal{I}}$, satisfies $\mathcal{T}$, since $\mathcal{I}_{M_\mathcal{I}}$ is a tree model of $\mathcal{T}'$ and thus all regular PFDs in $\mathcal{T}$ are (trivially) satisfied. Thus $\mathcal{I}_{M_\mathcal{I}} \models \mathcal{T}$ and violates $\mathcal{C}$. $\square$

Thus, the implication problem reduces to the problem in Section 4.1 since, by the above Lemma, the regular PFDs in $\mathcal{T}$ do not interfere with the decision procedure.

Case 2: $\mathcal{C}$ is a regular PFD $D_1 \sqsubseteq D_2 : L_1 \to L_2$. To falsify $\mathcal{C}$, it must be possible to have *two* objects, one in D_1 and another in D_2, that satisfy the preconditions of the dependency but that fail to satisfy the conclusion. We therefore construct two copies of the interpretation for the pure $\mathcal{DLF}_{\text{reg}}$ constraints in $\mathcal{T}$ similarly to [34,5]. However, as Herbrand terms are essentially the same in the two copies, it is sufficient to distinguish them by renaming the predicate symbols [12]. In addition, we need to model the "rules" of equality and their interaction with concept descriptions. The structural rules for $\mathcal{DLFD}_{\text{reg}}$ are thus defined as follows:

$$\Pi_{\mathcal{DLFD}_{\text{reg}}} = \Pi^G_{\mathcal{DLF}_{\text{reg}}} \cup \Pi^B_{\mathcal{DLF}_{\text{reg}}} \cup \left\{ \begin{array}{r} \mathsf{Eq}(x) \leftarrow \neg\mathsf{NEq}(x) \\ \mathsf{NEq}(x) \leftarrow \neg\mathsf{Eq}(x) \\ \leftarrow \mathsf{NEq}(x) \wedge \mathsf{Eq}(x) \\ \mathsf{Eq}(f_i(x)) \leftarrow \mathsf{Eq}(x) \\ P^G_D(x) \leftarrow P^B_D(x) \wedge \mathsf{Eq}(x) \\ P^B_D(x) \leftarrow P^G_D(x) \wedge \mathsf{Eq}(x) \end{array} \right\},$$

where Π^G and Π^B are sets of assertions Π in which every predicate symbol is renamed to a "green" version and a "blue" version.

Definition 24 *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFD}_{\text{reg}}$ implication problem for which $\mathcal{C}$ is the regular PFD*

$$D_1 \sqsubseteq D_2 : L_1 \to L_2,$$

let $\mathcal{T}''$ be all regular PFDs in $\mathcal{T}$, and let $\mathcal{T}' = \mathcal{T} \setminus \mathcal{T}''$. We define

$$\Pi_{\mathcal{C}} = \{P^G_{D_1}(0) \leftarrow, P^B_{D_2}(0) \leftarrow, \leftarrow \neg N_{\mathsf{NEq},L_2,S_2}(0), \leftarrow N_{\mathsf{NEq},L_1,S_1}(0), \}$$

and

$$\Pi_{\mathcal{T}} = \Pi^G_{\mathcal{T}'} \cup \Pi^B_{\mathcal{T}'} \cup \left\{ \begin{array}{r} \leftarrow P^G_{D_i}(x) \wedge P^B_{D_j}(x) \wedge \neg N_{\mathsf{NEq},L_i,S_i}(x) \wedge N_{\mathsf{NEq},L_j,S_j}(x) \\ \leftarrow P^B_{D_i}(x) \wedge P^G_{D_j}(x) \wedge \neg N_{\mathsf{NEq},L_i,S_i}(x) \wedge N_{\mathsf{NEq},L_j,S_j}(x) \\ \textit{for } (D_i \sqsubseteq D_j : L_i \to L_j) \in \mathcal{T}'' \end{array} \right\}$$

where S_i and S_j are the start symbols in the grammars for L_i and L_j, respectively.

Theorem 25 *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFD}_{\text{reg}}$ implication problem in which $\mathcal{C}$ is of the form $D_1 \sqsubseteq D_2 : L \to L'$. Then $\mathcal{T} \models \mathcal{C}$ if and only if $\Pi_{\mathcal{DLFD}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ is not satisfiable.*

Proof: Let M be a stable model of $\Pi_{\mathcal{DLFD}_{\text{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$. We construct an interpretation $\mathcal{I}_M = (\Delta, (.)^{\mathcal{I}})$ as follows:

The domain Δ is defined as

$$\Delta = \{t_U : \mathsf{Eq}(t) \in M\} \cup \{t_G, t_B : \mathsf{NEq}(t) \in M\}$$

for t a $\text{Datalog}^{\neg}_{nS}$ ground term. The t_U, t_G, and t_B are three distinct objects associated with each term t.

The interpretations of primitive attributes and concept descriptions are defined as follows

$$(f_i)^{\mathcal{I}}(t) = \begin{cases} f(t)_U \text{ if } f(t)_U \in \Delta \\ f(t)_G \text{ if } f(t)_G \in \Delta, t = t_G \\ f(t)_B \text{ if } f(t)_B \in \Delta, t = t_B \end{cases} \quad \begin{array}{l} t_U \in (D)^{\mathcal{I}} \text{ if } P^G_D(t) \in M, P^B_D(t) \in M \\ t_G \in (D)^{\mathcal{I}} \text{ if } P^G_D(t) \in M \\ t_B \in (D)^{\mathcal{I}} \text{ if } P^B_D(t) \in M \end{array}$$

Similarly to the proof of Theorem 21, we can verify that $\mathcal{I}_M \models \mathcal{T}$ but $\mathcal{I}_M \not\models \mathcal{C}$.

On the other hand, given an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \not\models \mathcal{C}$ it must be the case that there are two objects o_1 and o_2 in Δ such that $o_1 \in (D_1)^{\mathcal{I}}$, $o_2 \in (D_2)^{\mathcal{I}}$, $o_1.(\mathsf{Pf})^{\mathcal{I}} = o_2.(\mathsf{Pf})^{\mathcal{I}}$ for all $\mathsf{Pf} \in \mathcal{L}(L)$, and $o_1.(\mathsf{Pf}')^{\mathcal{I}} \neq o_2.(\mathsf{Pf}')^{\mathcal{I}}$ for some $\mathsf{Pf}' \in \mathcal{L}(L')$. We construct a Herbrand interpretation $M_{\mathcal{I}}$ as follows:

$$
\begin{aligned}
M_{\mathcal{I}} = \{ & P^G_D(\overline{\mathsf{Pf}}(0)) : o_1.(\mathsf{Pf})^{\mathcal{I}} \in (D)^{\mathcal{I}}\} \\
& \cup \{P^B_D(\overline{\mathsf{Pf}}(0)) : o_2.(\mathsf{Pf})^{\mathcal{I}} \in (D)^{\mathcal{I}}\} \\
& \cup \{\mathsf{Eq}(\overline{\mathsf{Pf}}(0)) : o_1.(\mathsf{Pf})^{\mathcal{I}} = o_2.(\mathsf{Pf})^{\mathcal{I}}\} \\
& \cup \{\mathsf{NEq}(\overline{\mathsf{Pf}}(0)) : o_1.(\mathsf{Pf})^{\mathcal{I}} \neq o_2.(\mathsf{Pf})^{\mathcal{I}}\} \\
& \cup \{N^G_{P_D,L,A}(\overline{\mathsf{Pf}}(0)) : \exists\, \mathsf{Pf} \in \mathcal{L}(A, G_L).\ o_1.(\mathsf{Pf})^{\mathcal{I}}.(\mathsf{Pf}')^{\mathcal{I}} \in (D)^{\mathcal{I}}\} \\
& \cup \{N^B_{P_D,L,A}(\overline{\mathsf{Pf}}(0)) : \exists\, \mathsf{Pf} \in \mathcal{L}(A, G_L).\ o_2.(\mathsf{Pf})^{\mathcal{I}}.(\mathsf{Pf}')^{\mathcal{I}} \in (D)^{\mathcal{I}}\} \\
& \cup \{N_{\mathsf{NEq},L,A}(\overline{\mathsf{Pf}}(0)) : \exists\, \mathsf{Pf} \in \mathcal{L}(A, G_L).\ o_1.(\mathsf{Pf})^{\mathcal{I}}.(\mathsf{Pf}')^{\mathcal{I}} \neq o_2.(\mathsf{Pf})^{\mathcal{I}}.(\mathsf{Pf}')^{\mathcal{I}} \in (D)^{\mathcal{I}}\},
\end{aligned}
$$

where descriptions D and regular path expressions L range over those present in $\mathcal{T} \cup \{\mathcal{C}\}$. The remainder of the proof is a simple verification that $M_{\mathcal{I}}$ is indeed a stable model of $\Pi_{\mathcal{DLFD}_{\mathrm{reg}}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$ and follows the same line of argument as in the proof of Theorem 21. □

Since all $\mathcal{DLF}_{\mathrm{reg}}$ implication problems are also $\mathcal{DLFD}_{\mathrm{reg}}$ implication problems, we have:

Corollary 26 *The implication problem for $\mathcal{DLFD}_{\mathrm{reg}}$ is DEXPTIME-complete.*

An interesting observation at this point is that, since $\mathcal{DLFD}_{\mathrm{reg}}$ can be *coded* in $\mathrm{Datalog}^{\neg}_{nS}$ which can in turn be *coded* in extremely simple fragments of $\mathcal{DLF}_{\mathrm{reg}}$ (cf. Section 3), all reasoning in very powerful and expressive Description Logics can ultimately be reduced to reasoning about simple attribute value restrictions in a boolean complete description logic.

4.3 Other DL Constructors

In this section, we briefly discuss other constructs commonly available in description logics and their relation to the development in the preceding section. Essentially, we argue that many DL features can be also coded in $\mathrm{Datalog}^{\neg}_{nS}$. The formal proofs of these claims are similar to those of Theorems 21 and 25 and are omitted. Also we simplify the notation, allowing ourselves to use a general first-order syntax instead of the strict clausal syntax of $\mathrm{Datalog}^{\neg}_{nS}$.

4.3.1 Adding Roles to $\mathcal{DLF}_{\mathrm{reg}}$: $\mathcal{DLFR}$

It is very common for description logics to include *roles* that correspond to binary relations between concepts. At the least, roles are manifest by a pair of concept constructors that capture *existential quantification* and *role value restriction* with respective syntax and semantics defined as follows (R denotes a role name is this syntax; we refer to $\mathcal{DLF}_{\text{reg}}$ extended with these constructors as $\mathcal{DLFR}$).

Additional Syntax	Semantics: Defn of "$(\cdot)^{\mathcal{I}}$"
$D ::= \exists R.D$	$\{x \in \Delta : \exists y \in (D)^{\mathcal{I}} : (x,y) \in (R)^{\mathcal{I}}\}$
$\mid \ \forall R.D$	$\{x \in \Delta : \forall (x,y) \in (R)^{\mathcal{I}} : y \in (D)^{\mathcal{I}}\}$

However, while general Ackermann formulae allow arbitrary arity relations in their matrix, they still require the use of a *single* universal ($\forall$) quantifier in their prefix. This prevents a direct formulation of the $\forall R.D$ concept, for example, since two $\forall$'s are needed (then adding unary function symbols leads to undecidable theories [35]).

We therefore use a less direct formulation by modeling roles in $\mathcal{DLFR}$ via attributes. The essential problem is that $\exists R.D_i$ concepts can force a single object to be related via a role R to multiple objects satisfying different constraints D_i, that, in general, may be disjoint. However, as all concept descriptions are essentially monadic predicates, one can syntactically determine the maximal number of such objects needed for a given implication problem.

Definition 27 ($\exists$-rank) *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFR}$ implication problem. The number of distinct occurrences of $\exists R.D$ in $\mathcal{T} \cup \{\mathcal{C}\}$ is denoted* $\text{Rank}(\mathcal{T},\mathcal{C})$.

Now let $\mathcal{T}$ and $\mathcal{C}$ be fixed, and ρ and $\delta_0,\ldots,\delta_l$, where $l = \text{Rank}(\mathcal{T},\mathcal{C})$, be function symbols neither in $\mathcal{T}$ nor in $\mathcal{C}$. We model a role R by a monadic predicate P_R. The fact that $(o,o') \in (R)^{\mathcal{I}}$, for $o, o' \in \Delta$, is captured by asserting $P_R(\delta_i(o))$ and $o' = \rho(\delta_i(o))$ for some $0 \le i \le l$. The new predicates P_R (and function symbols δ_i and ρ) are constrained to simulate the behavior of the $\forall R.D$ and $\exists R.D$ concepts by the following assertions:

$$\Pi_{\mathcal{DLFR}} = \Pi_{\mathcal{DLF}_{\text{reg}}} \cup \left\{ \begin{array}{l} \forall x.P_{\exists R.D}(x) \leftrightarrow \bigvee_{j=0}^{l} \left[P_R(\delta_j(x)) \wedge P_D(\rho(\delta_j(x)))\right] \\ \forall x.P_{\forall R.D}(x) \leftrightarrow \bigwedge_{j=0}^{l} \left[P_R(\delta_j(x)) \rightarrow P_D(\rho(\delta_j(x)))\right] \end{array} \right\}.$$

For any fixed implication problem of size n the size of the assertions is $O(n^2)$.

Theorem 28 *Let $\mathcal{T}$ and $\mathcal{C}$ be a $\mathcal{DLFR}$ terminology and inclusion dependency. Then $\mathcal{T} \models \mathcal{C} \iff \Pi_{\mathcal{DLFR}} \cup \Pi_{\mathcal{T}} \cup \Pi_{\mathcal{C}}$* is not satisfiable. *The latter can be decided in DEXPTIME.*

Note the similarity with the *tree* models for $\mathcal{DLR}$: essentially, one can convert a dag induced by roles into a tree by replacing $(o_1,o_2) \in (R)^{\mathcal{I}}$ by (o_1,o_2') and

(o_1', o_2) where o_1 and o_1' (o_2 and o_2', respectively) belong to the same concepts. Such an interpretation is still a model.

4.3.2 Role Inverses and Other Role Constructs

To model an inverse role R^{-1}, we need to modify the definitions for $P_{\forall R.D}$ and $P_{\exists R.D}$ to take account of the fact that if a parent of an object (i.e., the argument of the function simulating the role) is related to a $\forall R.D$ object via an inverse R^{-1}, then this object must be related to the parent via the original role R. Thus the parent must satisfy D (the $\exists R.D$ argument is similar). These observations are captured by the following assertions (the case for $P_{\forall R^{-1}.D}$ and $P_{\exists R^{-1}.D}$ is similar).

$$
\begin{array}{lll}
\forall x.P_{\exists R.D}(x) & \leftrightarrow \bigvee_{j=0}^{l} [P_R(\delta_j(x)) \wedge P_D(\rho(\delta_j(x)))] & \text{for } x \neq \rho(\delta_i(y)) \\
\forall x.P_{\exists R.D}(\rho(\delta_i(x))) & \leftrightarrow \bigvee_{j=0}^{l} [P_R(\delta_j(\rho(\delta_i(x)))) \wedge P_D(\rho(\delta_j(\rho(\delta_i(x)))))] & \\
 & \qquad\qquad \vee\ [P_{R^{-1}}(\delta_i(x)) \wedge P_D(x)] & \text{otherwise} \\
\forall x.P_{\forall R.D}(x) & \leftrightarrow \bigwedge_{j=0}^{l} [P_R(\delta_j(x)) \rightarrow P_D(\rho(\delta_j(x)))] & \text{for } x \neq \rho(\delta_i(y)) \\
\forall x.P_{\forall R.D}(\rho(\delta_i(x))) & \leftrightarrow \bigwedge_{j=0}^{l} [P_R(\delta_j(\rho(\delta_i(x)))) \rightarrow P_D(\rho(\delta_j(\rho(\delta_i(x)))))] & \\
 & \qquad\qquad \wedge\ [P_{R^{-1}}(\delta_i(x)) \rightarrow P_D(x)] & \text{otherwise}
\end{array}
$$

Note that the $x \neq \rho(\delta_i(y))$ condition can be eliminated by enumerating all terms that do not have the above form (i.e., do not start with $\rho\delta_i$). Such an enumeration is finite (still $O(n^2)$). Theorem 28 and its proof naturally extend to this setting. Other constructs that involve roles, such as *numerical restrictions*, *role hierarchies*, *role constructors*, *roles with arity* ≥ 2, and their combinations, can be similarly captured by appropriate assertions over the monadic predicates that model roles as long as the DL dialect itself has a tree model property [15,14].

5 Structural Equality in XML

We now show how to map XML documents to $\mathcal{DLFD}_{\text{reg}}$ interpretations in a way that enables useful reasoning about the structural equality of arbitrary subdocuments. To begin, it will be useful to have a concise definition of an XML document. Our formulation is based on the common practice of interpreting an XML document or a document collection as an ordered forest of rooted node-labeled ordered trees.

Definition 29 (XML Forests and Trees) *Let* PCDATA *be a set of strings.*

We define the set XF *of XML forests inductively by*

$$\mathsf{XF} \quad = \quad [\,] \;\mid\; [\mathsf{XNode}(s,x)] \;\mid\; x@y,$$

where s *is in* PCDATA*,* x *and* y *are in* XF*,* $[\,]$ *denotes an empty forest*[4]*,* $[\mathsf{XNode}(s,x)]$ *denotes a forest containing a single tree with a root labeled* s *(the string* s *represents a tag or* `PCDATA`*) and an ordered forest* x*, and* $x@y$ *denotes a concatenation of two ordered forests.*

This formulation of XML is very simple with no explicit accounting of node identity, of *element*, *attribute* or *text* node types, or of "don't care" child node order. However, such features can easily be added by additional encoding conventions that relate either to node labeling or to subtree patterns. A text leaf node with CDATA "`abc`", for example, might be encoded using the label "`text:abc`". A similar approach can be taken to represent attributes, etc.

Reasoning about XML documents in $\mathcal{DLFD}_{\mathrm{reg}}$ is achieved by mapping ordered forests corresponding to XML documents to $\mathcal{DLFD}_{\mathrm{reg}}$ interpretations. As in [10], we encode arbitrary XML forests by *binary trees* in which the *first* edge connects parent nodes with their first child and the *next* edge with their right sibling [36]. However, to be able to reason about structural equality, we add a third *label* edge connecting a node with its string label. An *infinite completion* of such a tree yields an $\mathcal{DLFD}_{\mathrm{reg}}$ interpretation. Formally:

Definition 30 *Let* $\mathcal{F} \in \mathsf{XF}$ *be a XML forest. We define an* $\mathcal{DLFD}_{\mathrm{reg}}$ *interpretation that represents this forest in two steps.*

(1) Let $\mathsf{PCDATA} \subset \Delta$ *be the set of all strings. For every document tag* `<a>` *we define a primitive concept* C_{a} *interpreted by* $(C_{\mathsf{a}})^{\mathcal{I}} = \{\texttt{<a>}\} \subset \mathsf{PCDATA}$*. In particular,* $(C_\epsilon)^{\mathcal{I}}$ *is interpreted as the singleton set consisting of the empty string.*

(2) The tree structure of the XML document is then captured by defining the interpretation for an additional primitive concept C_{XML}*, satisfying* $(C_{\mathrm{XML}})^{\mathcal{I}} \cap \mathsf{PCDATA} = \emptyset$*, and the interpretation of the primitive features* f*,* n*, and* l*. The interpretation of the primitive concept* C_{XML} *is defined by simultaneous induction on the structure of* $\mathcal{F}$ *utilizing partial interpretations* $(C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}}$ *and an auxiliary* $r_{\mathcal{F}}$ *constant (denoting the root of the encoded document* $\mathcal{F}$*).*

- $\mathcal{F}$ *is an empty forest. Then* $(C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}} = \emptyset$ *and* $r_{\mathcal{F}}$ *is an arbitrary element* $d \in \Delta - \mathsf{PCDATA}$ *for which* $(l)^{\mathcal{I}}(d) = \epsilon$*.*
- $\mathcal{F}$ *is a tree* $\mathsf{XNode}(s, \mathcal{F}')$*. Then*

$$(C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}} := (C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}'} \cup \{d\} \;\;\text{for}\;\; d \in \Delta - \mathsf{PCDATA} \cup (C_{\mathrm{XML}})^{\mathcal{I}}_{\mathcal{F}'}.$$

[4] We employ common list notation to represent ordered forests.

```
<emp>
  <eid>1234</eid>
  <name>Mary</name>
  <children>
     <name>Mary</name>
     <name>Bob</name>
  </children>
  ...
</emp>
```

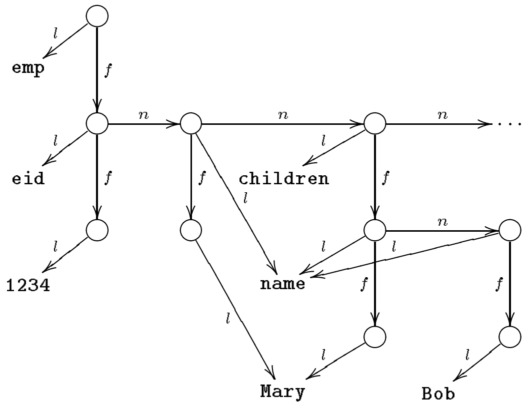

Fig. 2. An XML Document and $\mathcal{DLFD}_{\text{reg}}$ interpretation.

In addition, we modify the interpretation of the primitive features asserting that $(l)^{\mathcal{I}}(d) = s \in \mathsf{PCDATA}$*, and* $(f)^{\mathcal{I}}(d) = r_{\mathcal{F}'}$ *and set* $r_{\mathcal{F}} = d$*.*

- $\mathcal{F}$ *is a forest of trees* $\mathcal{T}_1\mathcal{T}_2\ldots\mathcal{T}_k$ *such that* $(C_{\text{XML}})^{\mathcal{I}}_{\mathcal{T}_i} \cap (C_{\text{XML}})^{\mathcal{I}}_{\mathcal{T}_j} = \emptyset$*.* [5] *Then*

$$(C_{\text{XML}})^{\mathcal{I}}_{\mathcal{F}} := \bigcup_{0<i\leq k} (C_{\text{XML}})^{\mathcal{I}}_{\mathcal{T}_i}.$$

In addition, we modify the interpretation of the primitive features asserting that $(n)^{\mathcal{I}}(r_{t_i}) = r_{t_{i+1}}$ *for* $0 < i < k$ *and set* $r_{\mathcal{F}} = r_{t_1}$*.*

Without loss of generality, we assume that values for features f and n not explicitly defined in this construction are roots of complete binary trees in f and n, the nodes of which do not belong to interpretations of primitive concepts defined above. We also assume that interpretations satisfy the following pair of inclusion dependencies: [6]

- $(\top \sqsubseteq \exists(n*.l).C_\epsilon)$, nodes have a finite number of children that are nonempty strings; and
- $(C_\epsilon \sqsubseteq \forall(f,n).l).C_\epsilon)$, any child of an empty string is an empty string.

An implication problem in $\mathcal{DLFD}_{\text{reg}}$ that "encodes" useful questions about XML (sub)documents, including questions about structural equality, should always include these dependencies as part of the terminology.

Example 31 Figure 2 illustrates an XML fragment together with the $\mathcal{DLFD}_{\text{reg}}$ interpretation that corresponds to this fragment. Note that the elipsis denotes arbitrary XML fragments encoding additional information about a given employee.

---

[5] It is always possible to pick disjoint sets to interpret nodes in distinct trees.

[6] The dependencies capture part of what are called *general structural properties* in [10].

```
<!DOCTYPE emps [
    <!ELEMENT emp (eid, (name)?, children, any)>
    <!ELEMENT eid (#PCDATA)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT children ((name)*)>
]>
```

(a) An XML DTD for personnel.

$$\begin{aligned}
\forall l.C_{\mathtt{emp}} &\sqsubseteq \forall f.l.C_{\mathtt{eid}} \\
\forall l.C_{\mathtt{emp}} &\sqsubseteq \forall f.n.((\forall l.C_{\mathtt{children}}) \sqcup ((\forall l.C_{\mathtt{name}}) \sqcap (\forall n.l.C_{\mathtt{children}}))) \\
\forall l.(C_{\mathtt{eid}} \sqcup C_{\mathtt{name}}) &\sqsubseteq (\forall f.l.\,\mathsf{PCDATA}) \sqcap (\forall f.n.l.C_{\epsilon}) \\
\forall l.C_{\mathtt{children}} &\sqsubseteq \forall f.n * .l.(C_{\mathtt{name}} \sqcup C_{\epsilon})
\end{aligned}$$

(b) A corresponding set of inclusion dependencies.

Fig. 3. On translating DTDs.

In Figure 3(a), we illustrate a DTD for part of the structure of employee subdocuments in which, say for security reasons, data encoded after information about dependents is left abstracted. Figure 3(b) illustrates how additional inclusion dependencies can be added to a terminology to capture the constraints on subdocuments for each of the four element tags that comprise the DTD. [7] Note that $(D_1 \sqcup D_2)$ is used as shorthand in the figure for $\neg(\neg D_1 \sqcap \neg D_2)$. Also note that, among other things, the constraints ensure that `<eid>`s are the first components of `<emp>`s.

Finally, consider how one might capture the constraint that `<eid>`s are keys for employee subdocuments. This can also be indirectly accomplished by adding a regular PFD with the following form to a terminology:

$$\forall l.C_{\mathtt{emp}} \sqsubseteq (\forall l.C_{\mathtt{emp}}) : f.f.l \rightarrow (f, n) * .l.$$

Observe that this additional dependency requires no knowledge of the structure of employee subdocuments beyond the fact noted above that an `<eid>` element is the first child of every such document.

Now, as a consequence of Theorem 32 below, we can replace the structural equality in the `where` clause of the XQuery in Example 1 by a much more efficient check for label equality:

```
where x=y      ⇒      where x/eid/data()=y/eid/data().
```

[7] It is straightforward to devise a procedure that can generate such constraints automatically from any given DTD.

The fact that label equality suffices also follows from the "typing constraints" in Figure 3(b) which ensure that an `<eid>` has one and only one child that must be in PCDATA.

Theorem 32 *Let $o_1, o_2 \in (C_{\mathrm{XML}})^{\mathcal{I}}$ be two nodes in the interpretation $(\Delta, (.)^{\mathcal{I}})$ that correspond to the roots of XML forests $\mathcal{F}_1, \mathcal{F}_2$, respectively. Then $\mathcal{F}_1 = \mathcal{F}_2$ structurally if and only if $(\mathsf{Pf})^{\mathcal{I}}(o_1) = (\mathsf{Pf})^{\mathcal{I}}(o_2)$ for all $\mathsf{Pf} \in \mathcal{L}((f, n) * .l)$.*

Proof: Follows from our definitions by a straightforward induction on the size of the forests. □

Finally, it is worth noting that the *positional* nature of specifying structural relationships between elements is essential to reasoning about structural equality. Observe in particular that *keyword-based* functional dependencies (i.e., those based on element tag names) [6,7] cannot distinguish documents `<a>1</a><b>2</b>` and `<b>2</b><a>1</a>`.

6 Summary

Structural equality is an important performance issue for XML data models and query languages. We have presented a description logic called $\mathcal{DLFD}_{\mathrm{reg}}$ that can be used for reasoning about structural equality in such models, and have outlined how $\mathcal{DLFD}_{\mathrm{reg}}$ can be applied in the case of XML and XQuery. This application depends on a more powerful version of an fd concept constructor in $\mathcal{DLFD}_{\mathrm{reg}}$ that has a novel and essential ability to express functional dependencies over sets of possibly infinite feature paths defined by regular languages. Thus, our work complements earlier work [10] in which a description logic is used to reason about XML document type definitions.

6.1 Future Work

There are several avenues of work currently under way that we believe will enhance the results of this paper. In particular, we are exploring the possibility of adapting results in [34] to allow the regular path functional dependencies in $\mathcal{DLFD}_{\mathrm{reg}}$ to have empty left-hand-sides, a serious possibility in view of the fact that element tags in XML are not (a least apriori) "isa" related. Such constraints can be used to (almost) simulate the incorporation of nominals in a terminology. Another topic we are exploring relates to finite models. Although [34] has shown that any object model with path functional dependencies does not have the finite model property, we believe an acyclicity property that underlies XML document type definitions can be exploited to recover the finite model property for related terminologies.

So how "close to the cliff or the valley" have we come? One of the remaining limitations of $\mathcal{DLFD}_{\rm reg}$ is the lack of an ability to define roles that are inverse attributes. This would represent a first opportunity for roles and functional dependencies to interact in $\mathcal{DLFD}_{\rm reg}$. Although the details are beyond the scope of this paper, it is possible to adapt the undecidability result of [37] to show that $\mathcal{DLFD}_{\rm reg}$ extended with inverse attributes will render its implication problem undecidable. However, some limited capacity to express inverse roles while ensuring decidability is still very desirable. Syntactic restrictions on regular path functional dependencies along the lines considered in [37] merit particular consideration. Second, the consequences of granting full first-order status to such dependencies are not clear. Indeed, the epistemological significance of either a negated fd or even the disjunction of two fds is unclear. Finally, we plan to investigate more general decidable constraint theories as the basis for path dependencies [38] and to integrate results in [39] that relate to ordering dependencies.

Acknowledgments

The authors gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada, the Communications and Information Technology of Ontario and Nortel Networks Ltd. for their support of this research.

References

[1] S. Boag, D. Chamberlin, D. Florescu, J. Robie, J. Simeon, M. Stefanescu, XQuery 1.0: An XML Query Language, Tech. rep., W3C (2001).

[2] V. L. Khizder, D. Toman, G. Weddell, Reasoning about Duplicate Elimination with Description Logic, in: Rules and and Objects in Databases, DOOD 2000 (part of Computational Logic 2000), 2000, pp. 1017–1032.

[3] D. E. Simmen, E. J. Shekita, T. Malkemus, Fundamental Techniques for Order Optimization, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 1996, pp. 57–67.

[4] A. O. Mendelzon, P. T. Wood, Functional Dependencies in Horn Clause Queries, TODS 16 (1) (1991) 31–55.

[5] G. E. Weddell, Reasoning about Functional Dependencies Generalized for Semantic Data Models, TODS 17 (1) (1992) 32–64.

[6] M. Arenas, W. Fan, L. Libkin, On Verifying Consistency of XML Specifications, in: ACM Symposium on Principles of Database Systems, 2002, pp. 259–270.

[7] M. Arenas, L. Libkin, Normal Form for XML Documents, in: ACM Symposium on Principles of Database Systems, 2002, pp. 85–96.

[8] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, W. C. Tan, Keys for XML, in: World Wide Web, 2001, pp. 201–210.

[9] W. Fan, L. Libkin, On XML integrity constraints in the presence of DTDs, Journal of the ACM 49 (3) (2002) 368–406.

[10] D. Calvanese, G. D. Giacomo, M. Lenzerini, Representing and Reasoning on XML Documents: A Description Logic Approach, Journal of Logic and Computation 9 (1) (1999) 295–318.

[11] A. Borgida, G. E. Weddell, Adding uniqueness constraints to description logics, in: International Conference on Deductive and Object Oriented Databases, DOOD, 1997, pp. 85–102.

[12] V. L. Khizder, D. Toman, G. E. Weddell, On Decidability and Complexity of Description Logics with Uniqueness Constraints, in: International Conference on Database Theory ICDT'01, 2001, pp. 54–67.

[13] M. Schmidt-Schauß, G. Smolka, Attributive Concept Descriptions with Complements, Artificial Intelligence 48 (1) (1991) 1–26.

[14] I. Horrocks, U. Sattler, S. Tobies, Practical Reasoning for Expressive Description Logics, in: Logic Programming and Automated Reasoning, LPAR'99, 1999, pp. 161–180.

[15] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, R. Rosati, Description Logic Framework for Information Integration, in: Principles of Knowledge Representation and Reasoning (KR'98), 1998, pp. 2–13.

[16] J. Chomicki, Functional deductive databases: Query processing in the presence of limited function symbols, Ph.D. thesis, Rutgers University, laboratory for Computer Science Research LCSR-TR-142 (1990).

[17] J. Chomicki, T. Imieliński, Finite Representation of Infinite Query Answers, ACM Transactions on Database Systems 18 (2) (1993) 181–223.

[18] T. Eiter, W. Faber, N. Leone, G. Pfeifer, Declarative Problem-solving using the `dlv` System, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer Academic Publishers, 2000.

[19] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, F. Scarcello, The KR System `dlv`: Progress Report, Comparisons and Benchmarks, in: Principles of Knowledge Representation and Reasoning, KR'98, 1998, pp. 406–417.

[20] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[21] J. Ullman, Principles of Database and Knowledge-Base Systems, Vol. 1, Computer Science Press, 1988.

[22] J. W. Lloyd, Foundations of Logic Programming, Springer-Verlag, 1987.

[23] M. Gelfond, V. Lifschitz, The Stable Model Semantics for Logic Programming, in: Proc. Fifth International Conference and Symposium on Logic Programming, ICLP/SLP, 1988, pp. 1070–1080.

[24] J. A. Fernández, J. Lobo, J. Minker, A. Rajasekar, Disjunctive LP + Integrity Constraints = Stable Model Semantics, Annals of Mathematics and Artificial Intelligence 5 (2-4) (1993) 89–131.

[25] J. Lobo, J. Minker, A. Rajasekar, Foundations of Disjunctive Logic Programming, MIT Press, Cambridge, MA, 1992.

[26] J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 1988.

[27] J. Minker, A. Rajasekar, A Fixpoint Semantics for Non-Horn Logic Programs, Journal of Logic Programming To appear.

[28] A. Rajasekar, J. Minker, Procedural Interpretation of Non-Horn Logic Programs, in: International Conference on Automated Deduction, Springer-Verlag, LNCS, 1988.

[29] W. Ackermann, Solvable Cases of the Decision Problem, Studies in Logic and the Foundations of Mathematics, North-Holland, 1954.

[30] M. Fürer, Alternation and the Ackermann Case of the Decision Problem, L'Enseignement Math. 27 (1981) 137–162.

[31] E. Emerson, C. Jutla, Tree Automata, Mu-Calculus and Determinacy, in: IEEE Symposium on Foundations of Computer Science, 1991, pp. 368–377.

[32] D. Janin, I. Walukiewicz, Automata for the μ-calculus and related results, in: Mathematical Foundations of Computer Science, 1995, pp. 552–562.

[33] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[34] M. Ito, G. Weddell, Implication Problems for Functional Constraints on Databases Supporting Complex Objects, J. Comput. Syst. Sci. 49 (3) (1994) 726–768.

[35] E. Börger, E. Grädel, Y. Gurevich, The Classical Decision Problem, Perspectives in Mathematical Logic, Springer-Verlag, 1997.

[36] D. E. Knuth, The Art of Computer Programming: Sorting and Searching, Vol. 3, Addison-Welsley (2ed), 1998.

[37] D. Calvanese, G. D. Giacomo, M. Lenzerini, Identification Constraints and Functional Dependencies in Description Logics, in: International Joint Conference on Artificial Intelligence, 2001, pp. 155–160.

[38] M. Baudinet, J. Chomicki, P. Wolper, Constraint-Generating Dependencies, in: International Conference on Database Theory, 1995.

[39] D. Toman, G. E. Weddell, On Attributes, Roles, and Dependencies in Description Logics and the Ackermann Case of the Decision Problem, in: Proceedings of Description Logics, CEUR-WS, vol.49, 2001, pp. 76–85.