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

David Toman and Grant Weddell

School of Computer Science, University of Waterloo, Canada
{david,gweddell}@uwaterloo.ca

Abstract. We define a boolean complete description logic dialect called $\mathcal{DLFD}_{\mathrm{reg}}$ 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 $\mathcal{DLFD}_{\mathrm{reg}}$ to express functional dependencies over sets of possibly infinite 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 $\mathrm{Datalog}^{\vee,\neg}_{nS}$ and in turn to the Ackermann case of the decision problem.

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 [5]. In particular, structural equality is 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 [5]. 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 and, 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 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 a number of query rewrite rules that relate to duplicate elimination [23], to sort operator elimination [31], and to a variety of other optimizations [29,33] 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 constraints have been considered in [2,3,8,17]. 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 compliment earlier work by Calvanese et al. [9] that was first to propose 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.

In [23,24] we considered a very simple DL dialect that included an "fd" concept constructor for capturing path functional dependencies [7,33]. The implication problem for this dialect was complete for DEXPTIME despite the fact that it did not allow defined concepts in terminologies. This result was obtained by comparing the problem to the recognition problem for Datalog_{nS}.

The remainder of the paper is organized as follows. Section 2 defines the syntax and semantics for $\mathcal{DLFR}$, and introduces $\text{Datalog}_{nS}^{\vee,\neg}$. Section 3 studies the complexity associated with reasoning in $\mathcal{DLFD}_{\text{reg}}$; Subsection 3.1 shows DEXPTIME hardness for a fragment of $\mathcal{DLFD}_{\text{reg}}$, then Subsections 3.2 and 3.3 present DEXPTIME decision procedure for $\mathcal{DLFD}_{\text{reg}}$. Section 4 outlines how $\mathcal{DLFD}_{\text{reg}}$ can be used to reason about structural equality in XML. We conclude with a summary and a list of conjectures and open questions in Section 5.

2 Definitions

Apart from *concepts*, which are descriptions of sets, description logics that derive from $\mathcal{ALC}$ [30], such as $\mathcal{SHIQ}$ [20] and $\mathcal{DLR}$ [11], 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* (or *features*). 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 $\text{Datalog}_{nS}^{\vee,\neg}$, a *logic programming* language of monadic predicates and functions [12,13]. Second, the use of $\text{Datalog}_{nS}^{\vee,\neg}$ 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 provides a rich set of *efficient implementation* techniques [14,16].

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. 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 3. A concept formed by an application of the final production in the grammar is* equality generating*. The derived concept descriptions for $\mathcal{DLF}_{\text{reg}}$ are those in $\mathcal{DLFD}_{\text{reg}}$ that are not equality generating.*

	Syntax	Semantics: Defn of "$(\cdot)^{\mathcal{I}}$"
$D ::=$	C	$(C)^{\mathcal{I}} \subset \Delta$
$\mid$	$D_1 \sqcap D_2$	$(D_1)^{\mathcal{I}} \cap (D_2)^{\mathcal{I}}$
$\mid$	$\neg D$	$\Delta \setminus (D)^{\mathcal{I}}$
$\mid$	$\forall L.D$	$\{x \in \Delta : (\mathsf{Pf})^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}$ for all $\mathsf{Pf} \in \mathcal{L}(L)\}$
$\mid$	$\exists L.D$	$\{x \in \Delta : (\mathsf{Pf})^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}$ for some $\mathsf{Pf} \in \mathcal{L}(L)\}$
$E ::=$	D	
$\mid$	$D : L \to L'$	$\Big\{x \in \Delta : \forall y \in (D)^{\mathcal{I}}. \Big(\bigwedge_{\mathsf{Pf} \in \mathcal{L}(L)} (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\Big) \Rightarrow \Big(\bigwedge_{\mathsf{Pf} \in \mathcal{L}(L')} (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\Big)\Big\}$

Fig. 1. Syntax and Semantics of $\mathcal{DLF}_{\text{reg}}$ and $\mathcal{DLFD}_{\text{reg}}$.

An inclusion dependency $\mathcal{C}$ *is an expression of the form* $D \sqsubseteq E$.

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 3.*

An interpretation satisfies an inclusion dependency $\mathcal{C}(\equiv D \sqsubseteq E)$ *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 constraint in* $\mathcal{T}$ *must also satisfy* $\mathcal{C}$ *(the posed question).*

Notation 4 We simplify the notation for path expressions in the rest of the paper by omitting the trailing Id, and also allow a syntactic composition $\mathsf{Pf}_1\,.\,\mathsf{Pf}_2$ of path expressions that stands for their concatenation.

Definition 5 (Datalog$_{nS}^{\vee,\neg}$ [13]) *A* Datalog$_{nS}^{\vee,\neg}$ program *is a finite set of clauses constructed from monadic predicates* P_i *and their negations, unary function symbols* f_i*, constants, and a single variable* x *using the usual well-formedness rules. A* satisfiability problem *for a Datalog$_{nS}^{\vee,\neg}$ program* Π *is to determine if* Π *has a well-founded model.*

A Datalog$_{nS}$ program is a finite set of definite Horn *Datalog$_{nS}^{\vee,\neg}$ clauses. A* recognition problem *for a Datalog$_{nS}$ program* Π *and a ground atom* q *is to determine if* q *is true in all models of* Π *(i.e., if* $\Pi \cup \{\neg q\}$ *is not satisfiable).*

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

Notation 6 $\overline{\mathsf{Pf}}(t)$ denotes a Datalog$_{nS}^{\vee,\neg}$ term of the form "$f_k(\cdots f_1(t)\cdots)$". This notation corresponds to the $\mathsf{Pf} = f_1.\cdots.f_k$ notation for path functions.

It is known that a $\mathrm{Datalog}_{nS}^{\vee,\neg}$ program has a model if and only if it has a Herbrand model [12,13,27]; this allows the use of syntactic techniques for model construction. To establish the complexity bounds we use the following result for the satisfiability of $\mathrm{Datalog}_{nS}^{\vee,\neg}$ programs [13,18].

Proposition 7 *The complexity of satisfiability for $\mathit{Datalog}_{nS}^{\vee,\neg}$ is DEXPTIME-complete. The lower bound holds for $\mathit{Datalog}_{nS}$ programs.*

For the remainder of the paper, we use general first-order syntax for $\mathrm{Datalog}_{nS}^{\vee,\neg}$ formulas instead of the clausal syntax to improve readability. However, each of the $\mathrm{Datalog}_{nS}^{\vee,\neg}$ programs in the paper can be presented in a clausal form.

3 Decision Procedures and Complexity Bounds

Relationships between $\mathcal{DLF}_{\mathrm{reg}}$, $\mathcal{DLFD}_{\mathrm{reg}}$ and $\mathrm{Datalog}_{nS}^{\vee,\neg}$ are the concerns of this section. The main underlying idea lies in relating $\mathcal{DLFD}_{\mathrm{reg}}$ concept descriptions to monadic predicates and in simulating $\mathcal{DLFD}_{\mathrm{reg}}$ *structural properties* as assertions in $\mathrm{Datalog}_{nS}^{\vee,\neg}$.

3.1 $\mathcal{DLF}_{\mathrm{reg}}$: Lower Bounds

We first show that every $\mathrm{Datalog}_{nS}$ recognition problem can be simulated by a $\mathcal{DLF}_{\mathrm{reg}}$ implication problem.

Definition 8 *Let Π be a $\mathit{Datalog}_{nS}$ program and $P(\overline{\mathsf{Pf}}(0))$ a ground atom (a goal). We construct an implication problem for $\mathcal{DLF}_{\mathrm{reg}}$ as follows: a terminology from clauses in Π,*

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

and the posed question from ground facts in $Q_i(\overline{\mathsf{Pf}}_i(0)) \in \Pi$ and the goal $G = P(\overline{\mathsf{Pf}}(0))$,

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

where, assuming $\overline{\mathsf{Pf}}(t) = f_k(\cdots f_1(t)\cdots)$, $\forall\, \mathsf{Pf}\, .D$ denotes $\forall f_1. \ldots .\forall f_k.D$.

Theorem 9 *Let Π be a $\mathit{Datalog}_{nS}$ program and G a goal. Then $\Pi \models G \iff \mathcal{T}_\Pi \models \mathcal{C}_{\Pi,G}$.*

The lower complexity bound then follows from [13,18].

Corollary 10 *The implication problem for $\mathcal{DLF}_{\mathrm{reg}}$ is DEXPTIME hard; this remains true for the fragment of $\mathcal{DLF}_{\mathrm{reg}}$ in which all concept constructors are conjunctions and restrictions of the form "$\forall f.D$".*

This complements the hardness result for path functional dependencies [24]: while that result was obtained in a class-free setting, here we need multiple class labels. Note that the complexity is inherently connected with a $\forall f.D$ construct. Without it, the problem reduces to propositional satisfiability which is NP-complete.

3.2 A Decision Procedure for $\mathcal{DLF}_{\text{reg}}$

We now consider the other direction: can $\mathcal{DLF}_{\text{reg}}$ in turn be (naturally) simulated by $\text{Datalog}_{nS}^{\vee,\neg}$? In this subsection, we show that this is indeed possible. In particular, we show how to construct a $\text{Datalog}_{nS}^{\vee,\neg}$ satisfiability problem that is equivalent to a given $\mathcal{DLF}_{\text{reg}}$ implication problem. In the simulation, $\mathcal{DLF}_{\text{reg}}$'s concept descriptions D are modeled by monadic predicates $P_D(x)$. The construction proceeds in three steps. The first encodes the structural properties of regular expression sets as $\text{Datalog}_{nS}^{\vee,\neg}$ assertions.

Definition 11 (Constraints for Regular Path Expressions) *Let $\langle N, F, R, S\rangle$ denote a right-linear grammar G for a given regular language L consisting of nonterminal symbols N unique to L, productions R and start symbol $S \in N$.*[1] *For formula φ with one free variable x not containing any predicate symbols of the form $N^{B,\psi}$ and G we define a $\text{Datalog}_{nS}^{\vee,\neg}$ program*

$$\Pi_{\mathcal{L}} = \left\{\forall x.N^{A,\varphi}(x) \leftarrow \mathcal{M}(\varphi,\alpha) : A \to \alpha \in G\right\}$$

where

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

Intuitively, the atomic formula $N^{A,\varphi}(t)$ is true for a term t if and only if there is $\mathsf{Pf} \in \mathcal{L}(A)$ such that $\varphi(\overline{\mathsf{Pf}}(t))$ is true ($\mathcal{L}(A)$ is the language generated by G from the nonterminal A). Note that it is essential to use the *minimal model semantics* to define $N^{A,\varphi}(x)$. However, the rules that define the atoms associated with G'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*[2].

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

Definition 12 ($\mathcal{DLF}_{\text{reg}}$ Concept Formation Constraints) *Let D, D_1, and D_2 range over concept descriptions and f over attribute names. We define*

$$\Pi_{\mathcal{DLF}_{\text{reg}}} = \Pi_{\mathcal{L}} \cup \left\{ \begin{array}{l} \forall x.(P_D(x) \vee P_{\neg D}(x)), \forall x.\neg(P_D(x) \wedge P_{\neg D}(x)) \\ \forall x.P_{D_1 \sqcap D_2}(x) \leftrightarrow (P_{D_1}(x) \wedge P_{D_2}(x)) \\ \forall x.P_{\exists L.D}(x) \leftrightarrow N^{S,P_D}(x)) \\ \forall x.P_{\forall L.D}(x) \leftrightarrow \neg N^{S,\neg P_D}(x)) \end{array} \right\}.$$

where S is the start symbol in the grammar for L.

[1] A grammar is right regular iff each production is of the form $A \to \mathsf{Id}$, $A \to a$ or $A \to bB$, 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 [19].

[2] This fact guarantees total models for the theories in this paper and avoids difficulties connected with unfounded recursion in general $\text{Datalog}_{nS}^{\vee,\neg}$ programs. For a comprehensive survey of semantics for disjunctive logic programs see [15,26,28]. A thorough exploration of that subject for $\text{Datalog}_{nS}^{\vee,\neg}$ is beyond the scope of this paper.

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

Definition 13 *Let $\mathcal{T}$ and $\mathcal{C} \equiv D \sqsubseteq E$ be a $\mathcal{DLF}_{\text{reg}}$ terminology and an inclusion constraint, respectively. We define*

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

and

$$\Pi_{\mathcal{C}} = \{P_D(0), P_{\neg E}(0)\}.$$

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

Theorem 14 *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]

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

Corollary 15 *The implication problem for $\mathcal{DLF}_{\text{reg}}$ is DEXPTIME-complete.*

3.3 Adding Regular Functional Dependencies: $\mathcal{DLFD}_{\text{reg}}$

We now consider $\mathcal{DLFD}_{\text{reg}}$ which adds equality generating concepts to $\mathcal{DLF}_{\text{reg}}$. In this more general setting, an inclusion dependency that contains such a concept is called a *regular path functional dependency* (regular PFD). Such dependencies have the form $(D_1 \sqsubseteq D_2 : L_1 \to L_2)$.

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

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 decision procedure from Section 3.2.

Lemma 16 *Let $\mathcal{T}'$ be the set of inclusion dependencies in a $\mathcal{DLFD}_{\text{reg}}$ terminology $\mathcal{T}$ that are not regular PFDs. Then if $\mathcal{T}'$ has a model it also has a Herbrand model. A Herbrand model is also a model of $\mathcal{T}$.*

[3] Note that we are assuming that satisfiability for $\text{Datalog}^{\vee,\neg}_{nS}$ is defined with respect to well-founded models.

Thus, the implication problem reduces to the problem in Section 3.2 since, by the above Lemma, the regular PFDs in $\mathcal{T}$ do not interfere with the decision procedure.

Case 2: $\mathcal{C}(= D_1 \sqsubseteq D_2 : L_1 \to L_2)$ is a regular PFD. 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 [21,33]. However, as Herbrand terms are essentially the same in the two copies, it is sufficient to distinguish them by renaming the predicate symbols [24]. 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:

$$\Pi_{\mathcal{DLFD}_{\text{reg}}} = \Pi^G_{\mathcal{DLF}_{\text{reg}}} \cup \Pi^B_{\mathcal{DLF}_{\text{reg}}} \cup \left\{ \begin{array}{l} \forall x.\mathsf{Eq}(x) \to \mathsf{Eq}(f(x)) \\ \forall x.\mathsf{Eq}(x) \to (P^G_D(x) \leftrightarrow P^B_D(x)) \end{array} \right\},$$

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

Definition 17 *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

$$\Pi_{\mathcal{C}} = \{P^G_{D_1}(0), P^B_{D_2}(0), \neg N^{S_1,\neg\mathsf{Eq}}(0), N^{S_2,\neg\mathsf{Eq}}(0)\}$$

and

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

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

Theorem 18 *Let $\mathcal{T} \models \mathcal{C}$ be a $\mathcal{DLFD}_{\text{reg}}$ implication problem in which $\mathcal{C}$ is a regular PFD. 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.*

And since all $\mathcal{DLF}_{\text{reg}}$ implication problems are also $\mathcal{DLFD}_{\text{reg}}$ implication problems, we have:

Corollary 19 *The implication problem for $\mathcal{DLFD}_{\text{reg}}$ is DEXPTIME-complete.*

4 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 20 (XML Forests and Trees) *Let* String *be a set of strings. We define the set* XF *of XML forests inductively by*

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

where s is in String*, 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 $\mathcal{DLFD}_{\text{reg}}$ is achieved by mapping ordered forests corresponding to XML documents to $\mathcal{DLFD}_{\text{reg}}$ *interpretations.* As in [9], 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 [25]. However, to be able to reason about structural equality, we add a third *label* edge connecting a node with its string label. Infinite completions of such tree yields an $\mathcal{DLFD}_{\text{reg}}$ interpretation. Formally:

Definition 21 *Let $\mathcal{F} \in$ XF be a XML forest. We define an $\mathcal{DLFD}_{\text{reg}}$ interpretation that represents this forest in two steps.*

1. *Let* String $\subset \Delta$ *be the set of all strings. For every document tag* `<a>` *we define a primitive class C_{a} interpreted by $(C_{\mathsf{a}})^{\mathcal{I}} = \{$*`<a>`*$\} \subset$* String.
2. *The tree structure of the XML document is then captured by defining the interpretation for an additional primitive concept C_{XML}, satisfying $(C_{\text{XML}})^{\mathcal{I}} \cap$ String $= \emptyset$, 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 $(C_{\text{XML}})^{\mathcal{I}}_{\mathcal{F}}$ and an auxiliary $r_{\mathcal{F}}$ constant (denoting the root of the encoded document $\mathcal{F}$).*
   – *$\mathcal{F}$ is an empty forest. Then $(C_{\text{XML}})^{\mathcal{I}}_{\mathcal{F}} = \emptyset$ and $r_{\mathcal{F}}$ is an arbitrary element $n \in \Delta -$ String.*
   – *$\mathcal{F}$ is a tree $\mathsf{XNode}(s,\mathcal{F}')$. Then*

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

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

[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>
```

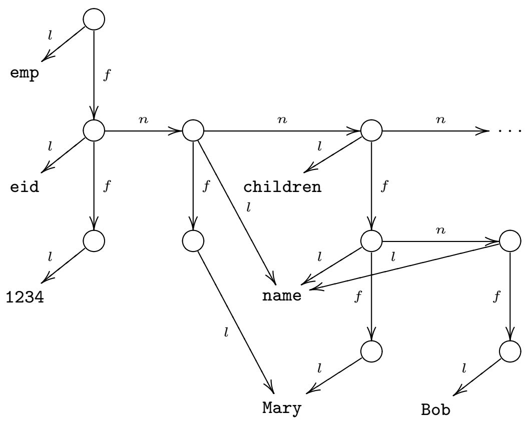

Fig. 2. An XML Document and $\mathcal{DLFD}_{\text{reg}}$ Interpretation

– *$\mathcal{F}$ 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 feature values not explicitly defined in this construction are roots of complete ternary trees, the nodes of which do not belong to interpretations of primitive concepts defined above.

Theorem 22 *Let $o_1, o_2 \in (C_{\text{XML}})^{\mathcal{I}}$ 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 $\mathsf{Pf} \in \mathcal{L}((f,n) * .l)$.*

Example 23 In Figure 2, we illustrate an XML fragment together with the $\mathcal{DLFD}_{\text{reg}}$ interpretation that corresponds to this fragment. Now consider how to express that `<eid>`s are always integer values and are the first components of `<emp>`s. This is indirectly accomplished by including the following inclusion dependency in a terminology constraining $C_{\mathtt{emp}}$:

$$\forall l.C_{\mathtt{emp}} \sqsubseteq (\forall f.l.C_{\mathtt{eid}}) \sqcap (\forall f.f.l.C_{\mathtt{int}})$$

Finally, consider how to express that `<eid>`s are keys for employee subdocuments. This is also indirectly accomplished by including the following regular path functional dependency in the same terminology:

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

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

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

Now, as a consequence of Theorem 22, we can replace the structural equality in the `where` clause of the XQuery in Example 1 by a much more efficient comparison of integer values:

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

In addition the typing constraint above specifies *the location* of the `<eid>` component in the XML tree (the first child of `<emp>`) and the location (first child of `<eid>`) and structure (integer) of the identifier itself.

The *positional* nature of specifying structural relationships between elements is essential to reasoning about structural equality. In particular *keyword-based* functional dependencies (i.e., those based on element tag names) [2,3] cannot distinguish documents `<a>1</a><b>2</b>` and `<b>2</b><a>1</a>`.

5 Summary

Structural equality is an important performance issue for XML data models and query languages. We have presented a description logic called $\mathcal{DLFD}_{\rm reg}$ that can be used for reasoning about structural equality in such models, and have outlined how $\mathcal{DLFD}_{\rm reg}$ can be applied in the case of XML and XQuery. This application depends on a more powerful version of an `fd` concept constructor in $\mathcal{DLFD}_{\rm reg}$ 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 compliments earlier work [9] in which a description logic is used to reason about XML document type definitions.

We have also presented a decision procedure for $\mathcal{DLFD}_{\rm reg}$. The procedure is based on mapping implication problems in $\mathcal{DLFD}_{\rm reg}$ to satisfaction problems in the logic programming language $\text{Datalog}_{nS}^{\vee,\neg}$. It is worth noting that this can in turn be reduced to the classical decision problem for the Ackermann ($\exists^*\forall\exists^*$) prefix [1].

Definition 24 (Monadic Ackerman Formulae) *Let P_i be monadic predicate symbols and x, y_i, z_i variables. A* monadic first-order formula in the Ackermann class *is a formula of the form $\exists z_1 \ldots \exists z_k \forall x \exists y_1 \ldots \exists y_l.\varphi$ where φ is a quantifier-free formula over the symbols P_i.*

To establish this final relationship, we can appeal to the results in [22] in which the authors outline a mapping of satisfiability problems for the μ-calculus to a closed disjunctive fixpoint free fragment of the μ-calculus. Note that it is straightforward to map $\text{Datalog}_{nS}^{\vee,\neg}$ satisfiability problems, in turn, to μ-calculus when recursion in $\text{Datalog}_{nS}^{\vee,\neg}$ passes through even numbers of negations.

5.1 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 [21] to allow the regular path functional dependencies in $\mathcal{DLFD}_{\text{reg}}$ 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 [21] 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 $\mathcal{DLFD}_{\text{reg}}$ 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 $\mathcal{DLFD}_{\text{reg}}$. Although the details are beyond the scope of this paper, it is possible to adapt the undecidability result of [10] to show that $\mathcal{DLFD}_{\text{reg}}$ 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 [10] 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 [4] and to integrate results in [32] 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. Wilhelm Ackermann. *Solvable Cases of the Decision Problem.* Studies in Logic and the Foundations of Mathematics. North-Holland, 1954.
2. Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On Verifying Consistency of XML Specifications. In *ACM Symposium on Principles of Database Systems*, pages 259–270, 2002.
3. Marcelo Arenas and Leonid Libkin. Normal Form for XML Documents. In *ACM Symposium on Principles of Database Systems*, pages 85–96, 2002.
4. Marainne Baudinet, Jan Chomicki, and Pierre Wolper. Constraint-Generating Dependencies. In *International Conference on Database Theory*, 1995.

5. S. Boag, D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery 1.0: An XML Query Language. Technical report, W3C, 2001.
6. Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer-Verlag, 1997.
7. Alexander Borgida and Grant E. Weddell. Adding uniqueness constraints to description logics. In *International Conference on Deductive and Object Oriented Databases, DOOD*, pages 85–102, 1997.
8. Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan. Keys for XML. In *World Wide Web*, pages 201–210, 2001.
9. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and Reasoning on XML Documents: A Description Logic Approach. *Journal of Logic and Computation*, 9(1):295–318, 1999.
10. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *International Joint Conference on Artificial Intelligence*, pages 155–160, 2001.
11. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description Logic Framework for Information Integration. In *Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.
12. Jan Chomicki. *Functional Deductive Databases: Query Processing in the Presence of Limited Function Symbols.* PhD thesis, Rutgers University, 1990. Laboratory for Computer Science Research LCSR-TR-142.
13. Jan Chomicki and Tomasz Imieliński. Finite Representation of Infinite Query Answers. *ACM Transactions on Database Systems*, 18(2):181–223, June 1993.
14. Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative Problem-solving using the `dlv` System. In Jack Minker, editor, *Logic-Based Artificial Intelligence.* Kluwer Academic Publishers, 2000.
15. Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
16. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR System `dlv`: Progress Report, Comparisons and Benchmarks. In *Principles of Knowledge Representation and Reasoning, KR'98*, pages 406–417, 1998.
17. Wenfei Fan and Leonid Libkin. On XML integrity constraints in the presence of DTDs. In *Symposium on Principles of Database Systems*, 2001.
18. Martin Fürer. Alternation and the Ackermann Case of the Decision Problem. *L'Enseignement Math.*, 27:137–162, 1981.
19. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
20. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical Reasoning for Expressive Description Logics. In *Logic Programming and Automated Reasoning, LPAR'99*, pages 161–180, 1999.
21. Minoru Ito and Grant Weddell. Implication Problems for Functional Constraints on Databases Supporting Complex Objects. *Journal of Computer and System Sciences*, 49(3):726–768, 1994.
22. David Janin and Igor Walukiewicz. Automata for the μ-calculus and related results. In *Mathematical Foundations of Computer Science*, pages 552–562, 1995.
23. Vitaliy L. Khizder, David Toman, and Grant Weddell. Reasoning about Duplicate Elimination with Description Logic. In *Rules and and Objects in Databases, DOOD 2000 (part of Computational Logic 2000)*, pages 1017–1032, 2000.

24. Vitaliy L. Khizder, David Toman, and Grant E. Weddell. On Decidability and Complexity of Description Logics with Uniqueness Constraints. In *International Conference on Database Theory ICDT'01*, pages 54–67, 2001.
25. Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Welsley (2ed), 1998.
26. Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.
27. John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.
28. Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, Cambridge, MA, 1992.
29. Alberto O. Mendelzon and Peter T. Wood. Functional Dependencies in Horn Clause Queries. *TODS*, 16(1):31–55, 1991.
30. Manfred Schmidt-Schauß and Gert Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26, 1991.
31. David E. Simmen, Eugene J. Shekita, and Timothy Malkemus. Fundamental Techniques for Order Optimization. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 57–67, 1996.
32. David Toman and Grant 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*, pages 76–85, 2001.
33. Grant E. Weddell. Reasoning about Functional Dependencies Generalized for Semantic Data Models. *TODS*, 17(1):32–64, 1992.

A Adding Roles to $\mathcal{DLF}_{\mathbf{reg}}$

Traditionally, description logics allow *roles*—binary relations between concepts. 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 since two $\forall$'s are needed (then adding unary function symbols leads to undecidable theories [6]).

Therefore we 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 25 ($\exists$-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* $\mathrm{Rank}(\mathcal{T},\mathcal{C})$.

Now let $\mathcal{T}$ and $\mathcal{C}$ be fixed, and ρ and $\delta_0,\ldots,\delta_l$, where $l = \mathrm{Rank}(\mathcal{T},\mathcal{C})$, 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:

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

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

Theorem 26 *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 later 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.

A.1 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 assertions (similarly for $P_{\forall R^{-1}.D}$ and $P_{\exists R^{-1}.D}$):

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

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 26 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 [11, 20].