Abstract

Several classes of path constraints for semistructured data are analysed and a number of decidability and complexity results proved for such constraints. While some of these decidability results were known before, it is believed that the improved complexity bounds are new. Proofs are based on techniques from modal logic and automata theory. This modal logic perspective sheds additional light on the reasons for previously known decidability and complexity results.

Keywords: Semistructured data, path constraints, modal logic, automata theory.

1 Introduction

In recent years, a lot of interesting work has been done to extend database techniques to semistructured collections of data, in particular the World Wide Web or fragments of it; an overview of this work can be found in [1]. It is generally agreed that the appropriate data model for semistructured data is an edge-labelled graph. More specifically, the web can be viewed as a set of objects linked by labelled edges; an object represents a page, and the labelled edges represent hypertext links.

Query languages proposed for semistructured data and querying the web, such as WebSQL [35], Lorel [2], and UnQL [11] are similar in spirit if not in syntax, and all include a form of recursion (regular expressions). Making effective use of whatever information is available about the format of data is obviously a very important issue. In the context of the web, it is often useful to know that everything accessible by a given sequence of links is cached, or available locally; or that the site reachable by a given sequence of links is mirrored elsewhere, etc. To express such information, one can use so-called path constraints, that is: statements about paths in the graph. It is reasonable to expect that the language of constraints forms a well-behaved (preferably decidable) sublanguage of the query language.

In this paper we build on results in [3, 13], and embed several classes of path constraints that have been considered in the literature into well-known modal logics. Earlier work has
considered the modal $\mu$-calculus, but we consider variants of PDL (see [28]). Our embeddings into a flavour of PDL establish a number of things; they shed light on known decidability results and give rise to new ones and to new complexity bounds. In some cases, the complexity bounds obtained by translation are not the tightest possible; in such cases we provide tighter bounds by using other methods (see Theorems 5.2 and 5.3).

But perhaps more importantly, adopting a logical perspective on data modelling and description languages often yields conceptual clarity, as demonstrated, for instance by Hayes’ paper on a model-theoretic semantics for RDF and RDFs [30]. In our case, we think that the main benefit of our modal perspective on path constraints lies in the insights it yields on the way various constraints relate to each other. Of course, rephrasing reasoning tasks on constraints in terms reasoning inside a suitable logic will not always give the sharpest possible complexity bounds: in some cases the corresponding logic simply has not been explored yet.

This paper is an extended and updated version of [4]. It is organized as follows. Section 2 provides background information on data models and query languages and it introduces several kinds of path constraints. In Section 3 we introduce logical formalisms to capture such constraints. In Section 4 we state our complexity and decidability results for logical problems. In Section 5, we establish corresponding results for reasoning problems on path constraints. We conclude in Section 6.

## 2 Path constraints

Semistructured data is often represented as an edge-labelled graph. In particular, the World Wide Web can be modelled as a graph where the vertices are uniquely identified by URLs and the labels are hypertext links between them [1]; richer structures able to deal with the frame structure of the pages can be found in [5]. An important special class of graphs are deterministic graphs. A graph is called deterministic if for every node $u$ and label $a$ there is at most one node $v$ such that $u \xrightarrow{a} v$ holds. In the case of the web (unlike the case of most object-oriented databases) it is reasonable to expect a graph to be deterministic.

In this paper, we will restrict attention to rooted connected graphs: that is, one of the nodes in the graph is designated as the root and every other node is accessible from the root by a directed path of edges. Intuitively, this is because we consider the web from the point of view of browsing, i.e. only the sites accessible from the current site (the root) really matter.

Languages for querying semistructured data use so-called path queries. These have emerged as an important class of browsing-style queries, and in their simplest version they are of the form ‘find all objects reachable by paths whose edge labels form a regular expression over some given alphabet of labels.’

Let us make things more formal. Let $L$ be a countable set of edge labels. An $L$-structure $G$ is a tuple of the form $G = \langle V, rt, (R_a)_{a \in L} \rangle$ such that $V$ is a set of nodes, $rt$ is a distinguished element of $V$ (the root), and $(R_a)_{a \in L}$ is a family of binary relations on $V$ acting as links between nodes. We say that $G$ is finite whenever $V$ is finite and $R_a$ is nonempty for only finitely many labels. In that case, the size of $G$, written $|G|$, is $|V| + \sum_{a \in L} |R_a|$. In other words, $L$-structures are rooted labelled transitions systems over the (possibly infinite) alphabet $L$.

**Definition 2.1**

Let $L$ be a countable set of edge labels. A label $a \in L$, the empty path $\epsilon$ and wildcard $\#$ are path expressions. If $p_1$ and $p_2$ are path expressions, then so are $p_1 ; p_2$ (sequential composition), $p_1 + p_2$ (union), and $p^*$ (finite iteration).
Given an \(L\)-structure \(G\), we define transition relations \(tr(p)\) on \(V \times V\) corresponding to the path expressions \(p\):

\[
\begin{align*}
tr(a) &= R_a \text{ for } a \in L \\
tr(p^*) &= \text{the reflexive transitive closure of } tr(p) \\
tr(\#) &= \bigcup_{a \in L} R_a \\
tr(\epsilon) &= \{ \langle u, u \rangle : u \in V \} \\
tr(p_1 ; p_2) &= \{ \langle u, v \rangle : \exists z (tr(p_1)(u, z) \land tr(p_2)(z, v)) \} \\
tr(p_1 + p_2) &= tr(p_1) \cup tr(p_2).
\end{align*}
\]

As usual, given a binary relation \(R\) on \(V\), we write \(R(u)\) to denote the set of \(R\)-successors of \(u\): \(\{ v \in V : \langle u, v \rangle \in R \}\).

In the absence of information about the format of data, evaluating queries with regular expressions can be very inefficient. A natural way to express useful information about the data represented as a graph is to impose constraints on possible paths in the graph, such as ‘all objects reachable by a path \(p\) are also reachable by a path \(q\),’ where \(p\) and \(q\) are sequences of labels, possibly involving regular expressions. Examples of constraints which may be useful for query optimization in the context of the web are constraints saying that everything accessible by such-and-such sequence of labels is also accessible locally; that the answer to such-and-such query is cached; that such-and-such site is mirrored elsewhere, and so on. All these examples can be expressed by means of path inclusion constraints as defined in [3] (see below).

The motivation of the work in [13] is more database-related than the work in [3]. Indeed, one important difference between the constraints considered in [3] and those studied in [13] is that the former correspond to unary properties and are evaluated relative to a node. The latter are closed sentences and can be evaluated anywhere and don’t have to mention the root; the motivation for the latter is much more database-related than the work in [3]. Another difference is that the constraints from [3] can contain regular expressions, while those in [13] are strictly first-order definable.

**Definition 2.2**

Let \(p\) and \(q\) be two path expressions. A **path inclusion constraint** is a statement of the form \(p \subseteq_f q\). Let \(G\) be an \(L\)-structure. A path inclusion constraint \(p \subseteq_f q\) is **true at \(G\)**, written \(G \models p \subseteq_f q\), if \(tr(p)(rt) \subseteq tr(q)(rt)\): every node \(y\) reachable from \(rt\) by a path whose labels form a word described by \(p\) (i.e. a \(p\)-path), is reachable from \(rt\) by a path whose labels form a word described by \(q\) (i.e. a \(q\)-path). See Figure 1(a).

The path inclusion constraints defined above are sometimes referred to as **forward constraints**.
In [13], backward constraints are introduced. We generalize their definition to a language containing regular expressions.

**Definition 2.3**
Let $D_4$ and $D_5$ be two path expressions. A backward path constraint is a statement of the form $D_4 \subseteq_b D_5$. Let $B Z$ be an $C_4$-structure. A backward path constraint $D_4 \subseteq_b D_5$ is true at $B Z$, written $B Z \models D_4 \subseteq_b D_5$, if $r \models (tr(p)(rt))$ where $-1$ denotes the converse operator on binary relations. See Figure 1(b).

Notice that a backward constraint can be rewritten as an inclusion constraint, and vice versa, by rewriting the regular expressions involved in the presence of the converse operator.

A standard path constraint (notation: $p \subseteq q$) is either a forward constraint or a backward constraint. The next class of constraints is a generalization of path constraints as defined in [13] to a language containing regular expressions.

**Definition 2.4**
Let $D_4$, $D_5$, and $D_6$ be path expressions. A lollipop path constraint is an expression of the form $D_6 \sim p \subseteq q$. Let $B Z$ be an $C_4$-structure. A lollipop path constraint $D_6 \sim p \subseteq q$ is true at $B Z$, written $B Z \models D_6 \sim p \subseteq q$, if for every $x \in \{ r \mid (tr(p)(rt)) \}$.

See Figure 1(c).

Obviously, a forward constraint $p \subseteq_f q$ is a lollilop path constraint $r \sim p \subseteq_f q$ with $r = \epsilon$. In the sequel, a lollipop path constraint will simply be called a path constraint.

Our syntax for talking about path constraints is obviously much more abstract than, for example, the XPath syntax [45, 29], which was designed to describe paths in XML trees. XPath involves predicates specific to attributes and names, and it allows navigation along the sibling axis. However, many constraints which are formulated using XPath expressions are very similar to the path constraints we are interested in. Consider the following example (taken from [15]):

This is the same as the following path inclusion constraint:

$$\text{Adverts; ProductName} \subseteq_f \text{Products; Name.}$$

A further exploration of the formal logical aspects of XPath is beyond the scope of this paper; we refer the reader to [25, 37].

Now that we have formulated path constraints, we take a closer look at important reasoning tasks involving them.
The query evaluation problem for a class \( C \) of path constraints is defined as follows:

**instance:** a finite \( L \)-structure \( G \) and a constraint \( c \) in \( C \);

**question:** \( G \models c \)?

A more difficult problem is the containment problem for the class \( C \) of path constraints. It is defined as follows:

**instance:** constraints \( c_1, \ldots, c_{n+1}, n \geq 0 \), in \( C \);

**question:** is it the case that for every \( L \)-structure \( G \), \( G \models c_1 \) and \( \ldots \) and \( G \models c_n \) imply \( G \models c_{n+1} \)? (If so, we write \( c_1, \ldots, c_n \rightarrow c_{n+1} \).

Variants of the above problems can be defined by considering only (finite, deterministic, ...) \( L \)-structures or/and by distinguishing the class of constraints for \( c_1, \ldots, c_n \) and \( c \) separately (if applicable).

In the sequel we investigate the decidability and complexity issues of the problems we have just introduced, mostly from a modal logic point of view.

### 3 Capturing path constraints by modal means

To determine the computational costs of reasoning tasks on path constraints, we recast them as model checking, validity, and satisfiability tasks in some logic. The logic used for this purpose should be such that it can easily encode problems on path constraints, especially the containment problem for the class of standard path constraints. Moreover, we wish to reduce the complexity gap between problems on path constraints and logical problems. In this way, a model checker or a theorem prover for such a logic will allow us to solve problems on path constraints in an efficient manner.

#### 3.1 Choosing a logic

A modal flavour

Which logic (or logics) should we use for capturing path constraints? Many formalisms have been proposed for reasoning about graphs. As we will see below, many decidable classes of constraints can be defined in terms of suitable modal logics, while constraints that lack a modal flavour (such as the ones studied in [13]) are generally undecidable. Rather than the presence or absence of regular expressions or even the need for two vs. three variables to express a constraint, the ‘modal flavour’ of constraints seems to be important. By this we mean the fact that modal formulas can only express local properties and the fact that the quantification implicit in modal formulas is ‘guarded’ [6]. We opt for Propositional Dynamic Logics (PDL, [28]) here since these are modal logics that incorporate regular expressions. Thus, we translate constraints into formulas of a flavour of PDL and reformulate reasoning tasks for constraints as reasoning tasks within this flavour of PDL.

PDL-like features

The language of PDL has two kinds of primitive symbols: propositional symbols and atomic transitions. Propositional symbols stand for properties that are true or false of a node in a graph; we only need three propositional symbols: \( \top \) (tautology), \( \bot \) (falsum), and \( \text{root} \) (to denote the root of the graph). Atomic transitions are used to label edges; we include a distinguished label \( e \) to denote the diagonal relation. Compound transition terms correspond
to path expressions and are built from atomic ones using ;, + and *. We indifferently use ‘path expression’, ‘transition term’ and ‘relational term’ as synonyms.

In addition to these traditional ingredients of PDL, we add a wildcard # and a converse operator (·)−1: # is a transition term, and if t is a transition term, then so is (t)−1. For instance, the satisfaction of the forward constraint c = p ⋙ f q at rt in the L-structure G will be expressed as G, rt ⊨ [p]((q)−1)root. The node rt is the only node w in G satisfying G, w ⊨ root. Hence, our symbol root is an example of a so-called nominal (a proposition letter that is true of at most a single node of a model). PDL with converse is called converse PDL (CPDL). We obtain CPDL with nominals (see [20]) by extending CPDL with nominals.

3.2 Defining the logic PDLpath

The logic PDLpath we study is a fragment of CPDL with nominals augmented with the wildcard #. Here’s a more precise definition.

**Definition 3.1 (PDLpath)**
The path expressions of PDLpath considered are those introduced in Definition 2.1 with the inclusion of the converse operator (·)−1; they are denoted by p, q and also by t. Formulas of PDLpath are typically denoted by φ:

\[ \phi ::= T | \bot | \text{root} | \neg \phi | \phi \land \phi | \langle t \rangle \phi | [t] \phi. \]

Given a formula φ, define |φ|, the length of φ, as the number of symbols in φ. A formula \langle t \rangle φ is read as ‘after some transition t, φ holds,’ or, more precisely, as ‘there is a sequence of labels which forms a word in a regular language defined by t and it leads to a node where φ holds.’ Dually, [t]φ is definable as \neg \langle t \rangle \neg φ and means ‘after every transition t, φ holds,’ that is: ‘if labels of a path form a word in t, then at the end of the path φ holds.’

To give an example, \langle a^* \rangle \neg \langle b \rangle T means that after 0 or finitely many a links one can reach a node which has no outgoing links labelled b.

**Definition 3.2 (Semantics)**
PDLpath is interpreted on L-structures. We now define truth of a formula φ at a node w in a structure G (notation: G, w ⊨ φ). For atomic propositional symbols, T is true at all nodes, \bot is false at all nodes, and root is true only at the root of the graph. Further, \neg φ is true if φ is false, and φ \land ψ is true if both φ and ψ are true. For modalities, we shall use the transition relations tr(p). We just need to extend tr(·) such that tr((p)−1) = \{⟨u, v⟩ : ⟨v, u⟩ ∈ tr(p)\}.

We say that v is accessible from u by a transition t if ⟨u, v⟩ ∈ tr(t). Then, for modal formulas, ⟨t⟩φ is true at a node u if there exists a node v accessible from u by t such that φ is true at v. Dually, [t]φ is true at u if for every v accessible from u by t, φ is true. A PDLpath formula φ is true on an L-structure G if it is true at the root of G. A PDLpath formula φ is satisfiable if it is true on some L-structure G. A formula is valid if it is true on all L-structures.

Roughly speaking, L-structures equipped with tr(t) for every relational term t in PDLpath can (almost) be viewed as PDL-models with a unique proposition letter interpreted by a singleton. This is not quite true because of the presence of #, but we will show below that it is correct when nondeterministic L-structures are considered (see the proof of Theorem 4.3). Hence, modulo the presence of #, PDLpath can be viewed as a fragment of CPDL with nominals [20] or as a fragment of the hybrid μ-calculus [43]. Furthermore, the constructive
A Modal Perspective on Path Constraints

$\mu$-calculus introduced in [5] also contains nominals (i.e. proposition letters interpreted by
singletons) as well as a form of recursion. We don’t need the full expressive power of the
$\mu$-calculus, however: we are interested in path queries, and regular expressions from PDL are
sufficient to express standard path constraints.

Just like the modal logic HML [33], PDL$_{path}$ has no proposition letters except the truth constant $\top$ and the unique nominal $root$. Indeed, PDL$_{path}$ is designed to reason about relations.

A final comment: one of our main aims is to reduce as much as possible the complexity
gaps between problems on path constraints and logical problems. We will show that PDL$_{path}$
is well-designed in this respect.

3.3 Standard logical reasoning tasks

For the purposes of this paper, the following logical reasoning tasks (involving PDL$_{path}$) are
important:

- The model checking problem for PDL$_{path}$ is:
  - instance: a finite $L$-structure $G$ and a formula $\phi$;
  - question: Is $\phi$ true at $G$?
- The validity problem for PDL$_{path}$ is defined as follows:
  - instance: a formula $\phi$;
  - question: Is it the case that for every $L$-structure $G$, $\phi$ is true at $G$?
- The satisfiability problem is defined accordingly in the usual way.

Before we explore the computational costs of the above reasoning tasks for PDL$_{path}$ we will show that PDL$_{path}$ suffices for expressing the reasoning tasks on the standard path constraints that we defined earlier.

3.4 From path constraints to PDL$_{path}$ and back

Given a forward constraint $c = p \subseteq_f q$ (or a backward constraint $c = p \subseteq_b q$), we write $\phi_c$ to
denote the formula $[p](q)^{-1}r root$ (or $[p]q r root$, respectively). Lemma 3.3 relates reasoning
with path constraints to reasoning tasks with PDL$_{path}$.

**Lemma 3.3**

1. Let $G$ be an $L$-structure and $c = p \subseteq q$ a standard path constraint. Then $G \models c$ iff $\phi_c$ is true at $G$.
2. Let $c_1, \ldots, c_{n+1}$ be standard path constraints. Then $c_1, \ldots, c_n \rightarrow c_{n+1}$ iff $(\phi_{c_1} \land \cdots \land \phi_{c_n}) \Rightarrow \phi_{c_{n+1}}$ is PDL$_{path}$ valid.

The proof of Lemma 3.3 is by an easy verification.

By contrast, there is no way to express (lollipop) path constraints $r \leadsto p \subseteq q$ by PDL$_{path}$
formulas in a similar fashion since the containment problem for the class of (lollipop) path
constraints is undecidable [13] and PDL$_{path}$ validity is decidable, as we will see below. As
an aside, (lollipop) path constraints can be expressed in modal logics with reference pointers
(see [24]), but the validity problem of such logics is usually undecidable.

By using Lemma 3.3, one can easily establish the following results.
Let $\mathcal{C}$ be either the full class of $L$-structures or the class of deterministic $L$-structures.

1. The query evaluation problem for standard path constraints is LOGSPACE reducible to the model checking problem for PDL$^{path}$.
2. The containment problem for forward constraints restricted to $L$-structures in $\mathcal{C}$ is LOGSPACE reducible to the validity problem for PDL$^{path}$ restricted to $L$-structures in $\mathcal{C}$.
3. The containment problem for backward constraints restricted to $L$-structures in $\mathcal{C}$ is LOGSPACE reducible to the validity problem for PDL$^{path}$ without converse and restricted to $L$-structures in $\mathcal{C}$.

To conclude this section, we will briefly contrast our modal formalization of path constraints with others in the literature. In [37], a fragment of the Computation Tree Logic CTL is shown to be equivalent to $\mathcal{L}^{tree}$, a tree pattern language; more precisely, the authors prove equivalence between the implication problem in this fragment of CTL and the containment problem for $\mathcal{L}^{tree}$. In addition, in [25], Core XPath is translated into full CTL. It is important to notice that, unlike the authors of the papers just quoted, we consider graph structures — and not just tree structures.

In [16], a spatial logic is introduced for reasoning about labelled directed graphs; the logic is closely related to monadic second-order logic (MSO) for graphs (see [19]), and its main use is in querying structures from the semistructured data model. Both logics have a model checking problem in PSPACE, whereas satisfiability for MSO is undecidable. By contrast, our logic PDL$^{path}$ is better attuned to the reasoning tasks about path constraints and we do not actually need the full power of MSO-like formalisms, especially if we wish to deduce complexity upper bounds of problems on path constraints from translations into logical problems.

## 4 The complexity of reasoning with PDL$^{path}$

We now consider complexity and decidability results for PDL$^{path}$ problems. In the next section we will use these results, together with Lemma 3.4, to derive complexity and decidability results for path constraints.

### 4.1 The model checking problem

The model checking problem for PDL$^{path}$ is no more expensive than for PDL.

**Theorem 4.1**

Let $G = (V, rt, (R_a)_{a \in L})$ be an $L$-structure, $w \in V$, and $\phi$ a PDL$^{path}$ formula. Checking whether $G, w \models \phi$ can be done in time $O(|G| \times |\phi|)$.

**Proof.** There is a simple linear reduction of the model checking problem for PDL$^{path}$ to the model checking problem for PDL. The latter problem is in time $O(|G| \times |\phi|)$. This follows from the fact that model checking for the alternation-free modal $\mu$-calculus is in linear time [18].

The linear time reduction works as follows. First, given $G$, we construct a new $L$-structure $G'$. $G'$ has the same vertices and root, and contains all the edges which $G$ has plus, for every edge $u \xrightarrow{a} v$ in $G$ we add three more edges to $G'$:

$$u \xrightarrow{\#} v, v \xrightarrow{(#)^{-1}} u, \text{ and } v \xrightarrow{(a)^{-1}} u.$$
The construction of $G'$ is obviously linear in the size of $G$. Second, we rewrite $\phi$ so that all occurrences of $(\cdot)^{-1}$ are on the atomic labels. This can be done in linear time by using the following standard equivalences:

\[
(p + q)^{-1} = (p)^{-1} + (q)^{-1} \\
(p ; q)^{-1} = (q)^{-1} ; (p)^{-1} \\
(p^*)^{-1} = ((p)^{-1})^* \\
((p)^{-1})^{-1} = p.
\]

Note that the resulting formula $\phi'$ is linear in the size of $\phi$. Finally, it is easy to show that $G, w \models_{PDL_{path}} \phi$ iff $G', w \models_{PDL} \phi'$.

**Corollary 4.2**

The model checking problem for $PDL_{path}$ is P-complete.

The polynomial upper bound is a consequence of Theorem 4.1 and the polynomial lower bound can easily be obtained by a reduction from the P-complete problem SAM2CVP (synchronous alternating monotone fanout 2 circuit value problem; see [26, p. 123]), as has been done to show the P-hardness of CTL model-checking restricted to $\forall X$ and $\exists X$, a folklore result in model checking. The proof for CTL can easily be adapted to the model-checking problem for $PDL_{path}$ restricted to deterministic $L$-structures, as soon as $|L| \geq 2$.

### 4.2 The satisfiability problem

The satisfiability problem for $PDL_{path}$ can be proved to be decidable by a reduction to the (decidable) satisfiability problem of CPDL with nominals.

**Theorem 4.3**

The satisfiability and validity problems for $PDL_{path}$ are decidable in EXPTIME.

**Proof.** We use the fact that CPDL with nominals is decidable [20, Theorem 49] and reduce satisfiability in $PDL_{path}$ to satisfiability in CPDL with nominals. If the set of labels $L$ is finite, $\#$ can be replaced by the finite union of labels from $L$ and the reduction to CPDL with nominals is immediate.

Suppose that $L$ is infinite, in which case $\#$ is a nontrivial addition to the language. We proceed as follows. Given a $PDL_{path}$ formula $\phi$ which uses labels $\{a_1, \ldots, a_n\}$ and possibly $\#$, we construct a CPDL formula $\psi$ by replacing $\#$ with $(a_1 + \cdots + a_{n+1})$ in $\phi$, and we show that $\phi$ is satisfiable (which means $\phi$ is true at the root of some $L$-structure) iff $root \land \psi$ is.

Suppose $G = \langle V, rt, (R_a)_{a \in L} \rangle, rt \models \phi$. We will define an $L$-structure $G'$ and view it as a partial description of a model for CPDL with nominals. Let $G' = \langle V, rt, (R'_a)_{a \in L} \rangle$ be the $L$-structure defined as follows:

1. for every $1 \leq i \leq n$, $R'_{a_i} = R_{a_i}$;
2. $R'_{a_{n+1}} = \bigcup_{a \in L} R_a$.

It is easy to show that $G, rt \models \phi$ iff $G', rt \models root \land \psi$ since $tr(\#)$ in $G$ is equal to $tr(a_1 + \cdots + a_{n+1})$ in $G'$.

Now, suppose $G = \langle V, rt, (R'_a)_{a \in L} \rangle, rt \models root \land \psi$ for some partial description of a model for CPDL with nominals. Let $G = \langle V, rt, (R_a)_{a \in L} \rangle$ be the $L$-structure defined as follows:
1. for every $1 \leq i \leq n$, $R_{a_i} = R_{a_i}^i$;
2. for every $a \in (L \setminus \{a_1, \ldots, a_n\})$, $R_a = R_{a_{n+1}}$.

It is now easy to show that $G', r t \models \text{root} \land \psi$ iff $G, r t \models \phi$. 

By itself, Theorem 4.3 does not imply an analogous result for deterministic $L$-structures, which remains an open problem to date. However, if the set of edge labels $L$ is finite, deterministic CPDL with nominals is decidable only if on deterministic $L$-structures, the satisfiability problem for PDL$^{path}$ is decidable.

In the nondeterministic case, we can actually do better than Theorem 4.3, and obtain matching lower and upper bounds for the complexity of the satisfiability problem for PDL$^{path}$.

THEOREM 4.4
The satisfiability problem for PDL$^{path}$ is EXPTIME-complete whenever $|L| \geq 1$.

PROOF. To see that the satisfiability problem for PDL$^{path}$ is decidable in exponential time, recall that, by [20, p. 98], the satisfiability problem for CPDL with nominals is EXPTIME-complete. The reduction of the PDL$^{path}$ satisfiability problem to the satisfiability problem for CPDL with nominals is polynomial in the size of the input formula.

As to the lower bound, we use a reduction from the global satisfiability problem for the standard modal logic K. A formula $\phi$ is globally satisfiable if there exists a model such that $\phi$ is satisfied in every state of the model. The global satisfiability problem for K is known to be EXPTIME-hard [17, 31]. Our reduction only uses a restricted form of the PDL$^{path}$ satisfiability problem, restricted to the modal operators $\langle a_0 \rangle$ and $\langle a_0^{-1} \rangle$. We use the spy-point technique as described in [10] by adapting the proof of [7, Theorem 2]. As part of this technique, we introduce a node (the ‘spy point’) in the model that can see any other node, and, therefore, universal quantification can be simulated by exploring the successors of the spy point. The only difficulty is to use the spy-point technique and simultaneously encode the proposition letters which occur in the K-formula.

We set out to define a map $f$ from K-formulas into PDL$^{path}$-formulas such that $\phi$ is globally satisfiable iff $f(\phi)$ is PDL$^{path}$ satisfiable. In order to show the ‘only-if’ direction, given a Kripke structure $\mathcal{M} = \langle V, R, m \rangle$ such that $\mathcal{M} \models \phi$, we construct an $L$-structure $G = \langle V^*, r t, R_{a_0} \rangle$ such that $f(\phi)$ is true at $G$ and the fact that $\mathcal{M}, w \models p_i$ is encoded by the existence of an edge $i \xrightarrow{a_0} w$ in $G$ where $i$ is a node in $G$ associated with the proposition letter $p_i$.

To simplify notation we now write $\rightarrow$ instead of $\xrightarrow{a_0}$ The structure $\langle V^*, R_{a_0} \rangle$ consists of:

- a copy of $\langle V, R \rangle$, that is to say, the restriction of $\langle V^*, R_{a_0} \rangle$ to $V$ is $\langle V, R \rangle$;
- an extra node $r t$ (the spy point) that can see any node in $V$ (i.e. for every $w \in V$, $r t \rightarrow w$ in $G$);
- extra nodes $1, \ldots, N$ that are used to encode the satisfaction of proposition letters $p_1, \ldots, p_N$ in such a way that $\mathcal{M}, w \models p_i$ iff $i \rightarrow w$ is in $G$;
- a chain $r t \leftarrow 1 \rightarrow 2 \rightarrow \ldots \rightarrow N$ that allows us to identify the elements of $V^* \setminus V$:
  1. $r t$ is the only element $w$ in $V^*$ such that $G, w \models \text{root}$ (not related to the chain);
  2. 1 is the only element $w$ of $V^*$ such that $r t \rightarrow w$ and $w \rightarrow r t$;
  3. $i + 1 (1 \leq i \leq N - 1)$ is the only element $w$ in $V^*$ such that $i \rightarrow w$, $w \neq r t$ (just here for the case $i = 1$), and not $r t \rightarrow w$. 

Consequently, the elements of $V$ in $G$ are precisely the elements $w$ in $V^*$ such that $rt \not\rightarrow w$ and not $w \not\rightarrow rt$.

Let us define a family $(\pi_i)_{i \in \mathbb{N}}$ of PDL$_{Path}$ formulas encoding proposition letters.

$$\pi_1 := \langle a_0^{-1} \rangle (\langle a_0 \rangle \text{root} \land \langle a_0^{-1} \rangle \text{root})$$

$$\pi_{i+1} := \langle a_0^{-1} \rangle (\neg \text{root} \land \neg \langle a_0^{-1} \rangle \text{root} \land \pi_i).$$

Encodings of proposition letters can be also found in [27, 32], but our encoding is of a different nature since we tailor it to the spy-point technique. The mapping $f$ is now defined as follows:

$$f(p_i) = \pi_i \text{ for } i \geq 1$$

$f$ commutes with the Boolean connectives

$$f(\Box \phi) = \langle a_0 \rangle (\langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root} \land f(\phi)).$$

Now, let $\phi$ be a K-formula. We will show that $\phi$ is globally K-satisfiable iff the following formula is PDL$_{Path}$-satisfiable:

$$[a_0](\langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root} \rightarrow f(\phi)).$$

Without loss of generality, we can assume that if $N$ distinct proposition letters occur in $\phi$, then they are $p_1, \ldots, p_N$.

(‘Only-if’) Assume that $M \models \phi$ for some Kripke structure $M = (V, R, m)$. Let $G = (V^*, rt, R_{a_0})$ be the L-structure such that

- $V^* := V \cup \{rt\} \cup \{1, \ldots, N\}$ where $rt \not\in V \cup \{1, \ldots, N\}$ and $V \cap \{1, \ldots, N\} = \emptyset$;
- $R_{a_0} := R \cup \{\langle rt, w \rangle : w \in V\} \cup \{(1, 2), \ldots, (N-1, N)\} \cup \{(1, rt), \langle rt, 1 \rangle\} \cup \{(i, w) : M, w \models p_i\};$
- the interpretation of root is $rt$.

In Figure 2, we give a simple example of the construction. Let us show that for all $w \in V$, for all $\psi \in \text{sub}(\phi)$ (the set of subformulas of $\phi$), $M, w \models \psi$ iff $G, w \models f(\psi)$. For the base case (for $i \in \{1, \ldots, N\}$, $M, w \models p_i$ iff $G, w \models \pi_i$), one can easily show by induction on $i$ that $\{w \in V^* : G, w \models \pi_i\} = \{w \in V : M, w \models p_i\} \cup (\{i + 1\} \setminus\{N + 1\})$.

We now turn to the case $\psi = \Box \psi'$. Observe that $M, w \models \Box \psi'$ iff there is $w' \in R(w)$ such that $M, w' \models \psi'$ iff (i) there is $w' \in V$ such that $\langle w, w' \rangle \in R_{a_0}, \langle r, w' \rangle \in R_{a_0}, \langle w', r \rangle \notin R_{a_0}$, and $M, w' \models \psi'$. Furthermore, (i) holds iff there is $w' \in R_{a_0}(w)$ such that $G, w' \models f(\psi') \land \langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root}$ if $G, w \models f(\psi)$. So for all $w \in V$, we have $G, w \models f(\psi)$ since $M \models \psi$. Moreover, $R_{a_0}(rt) \cap \{w \in V^* : G, w \models \langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root}\} = V$.

So, $G, rt \models [a_0](\langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root} \rightarrow f(\psi))$.

(‘If’) Assume that $G, rt \models [a_0](\langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root} \rightarrow f(\phi))$ for some L-structure $G = (V, rt, R_{a_0})$. Define a model $\mathcal{M} = (V^*, R, m)$ such that

- $V^* := \{w \in V : \langle rt, w \rangle \in R_{a_0}, \langle w, rt \rangle \not\in R_{a_0}\};$
Let us show that for all \( w \in V^* \) and for \( \psi \in \text{sub}(\phi) \), \( G, w \models f(\psi) \) iff \( M, w \models \psi \).

The base case is obvious. We treat the case \( \psi = \Diamond \psi' \) in a bit more detail. We have \( G, w \models \langle a_0 \rangle(f(\psi') \land \langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root}) \) iff there is \( w' \in R_{a_0}(w) \cap V^* \) such that \( G, w' \models f(\psi') \) iff there is \( w' \in R(w) \) such that \( M, w' \models \psi' \) iff \( M, w \models \psi \). Since \( G, rt \models \langle a_0 \rangle (\langle a_0^{-1} \rangle \text{root} \land \neg \langle a_0 \rangle \text{root} \rightarrow f(\psi)) \), we obtain \( M, w \models \psi \) for all \( w \in R_{a_0}(rt) \cap V^* \). Since \( V^* \subseteq R_{a_0}(rt) \), it follows that \( M, w \models \phi \) for all \( w \in V^* \).

The minimal tense logic (the logic \( K_t \) in [41]) is a bimodal logic with modal operators \([a_0], [a_0^{-1}]\), and a countably infinite set of proposition letters. As a corollary to the proof of Theorem 4.4, the minimal tense logic augmented with a single nominal but without proposition letters has an EXPTIME-hard satisfiability problem. This improves the result in [7], which states that the minimal tense logic with one nominal and \( \text{proposition letters} \) is EXPTIME-hard. From the proof of Theorem 4.4, it is clear that the main ingredient for EXPTIME-hardness is not the presence of regular expressions in PDL\textsuperscript{path} but rather the presence of a unique nominal with future and past-time operators.

To conclude this section we summarize the results we have obtained so far, and situate them among related results in the literature; see Table 1.

5 The complexity of reasoning with path constraints

In order to characterize the complexity of reasoning problems on path constraints, we will either use translations into PDL\textsuperscript{path} (mainly via Lemma 3.4) or use direct methods.

5.1 The query evaluation problem

An immediate corollary to Lemma 3.4, item 1, and Theorem 4.1 is that checking whether \( G \models p \subseteq q \) can be done in time \( O(|G| \times (|p| + |q|)) \). The rest of this section is devoted to showing that this tractability result can be improved significantly. We need the following lemma.

**Lemma 5.1**
The problem below is in NLOGSPACE in \( |G| \) and \( |t| \):
TABLE 1. A summary of results on logical reasoning tasks

<table>
<thead>
<tr>
<th>Model checking problem</th>
<th>nondeterministic graphs</th>
<th>deterministic graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDL</td>
<td>P-complete; see e.g. [18]</td>
<td>P-complete; see e.g. [18]</td>
</tr>
<tr>
<td>PDL_{pa}</td>
<td>P-complete; this paper, Corollary 4.2</td>
<td>P-complete; this paper, Corollary 4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfiability problem</td>
<td>nondeterministic graphs</td>
<td>deterministic graphs</td>
</tr>
<tr>
<td>PDL</td>
<td>EXPTIME-complete [22, 40]</td>
<td>EXPTIME-complete [39, 9]</td>
</tr>
<tr>
<td>PDL with nominals</td>
<td>EXPTIME-complete [23]</td>
<td>EXPTIME-complete [23]</td>
</tr>
<tr>
<td>CPDL</td>
<td>EXPTIME-complete [22, 40]</td>
<td>EXPTIME-complete [44]</td>
</tr>
<tr>
<td>CPDL with nominals</td>
<td>EXPTIME-complete [20, 7]</td>
<td>open</td>
</tr>
<tr>
<td>PDL_{pa}</td>
<td>EXPTIME-complete; this paper, Theorem 4.4</td>
<td>open</td>
</tr>
</tbody>
</table>

instance: a finite $L$-structure $G$, a path expression $t$, and $u,v \in V$;
question: is $\langle u,v \rangle \in \text{tr}(t)$?

PROOF. Without loss of generality, we can assume that $t$ does not contain $\#$, since $G$ is finite. We write $A_{G,u,v}$ to denote the finite state automaton obtained from $G$ in the obvious way with initial state $u$ and final state $v$. The following facts are known:

1. constructing a finite state automaton $A(t)$ recognizing $L(t)$ (the language generated by $t$) can be done in LOGSPACE in $|t|$;
2. constructing a product automaton $B$ recognizing the intersection of the languages from $A(t)$ and $A_{G,u,v}$ can be done in LOGSPACE in $|A(t)| + |G|$;
3. the class of LOGSPACE transformations is closed under composition (see [8, Theorem 3.37]).

Now, note that the question whether $\langle u,v \rangle \in \text{tr}(t)$ is equivalent to checking whether $L(B)$ is nonempty. By [8, Theorem 3.36], we get that the latter can be done in NLOGSPACE in $|B|$.

Lemma 5.1 is an improvement of [36], which only states that the problem formulated in Lemma 5.1 is in P.

In the proof of Lemma 5.1, if $t$ contains $\#$, then we consider $t'$ instead of $t$ by replacing every occurrence of $\#$ by $(a_1 + \cdots + a_k)$, where either $a_i$ occurs in $t$ or $R_{a_i}$ is nonempty in $G$. Hence, $|t'|$ is in $O(|G| \times |t|)$ which guarantees that we also have an NLOGSPACE upper bound in this case.

THEOREM 5.2
The query evaluation problem for the class of path constraints is NLOGSPACE-complete in $|G|$ and $|t|$.

PROOF. The Graph Accessibility Problem (GAP) can easily be reduced to the query evaluation problem, which provides NLOGSPACE-hardness. Indeed, let $G = \langle V, E \rangle$ be a graph and $u,v \in V$. We have $\langle u,v \rangle \in R^*$ iff $G' = \langle V, u, R_{a_1}, R_{a_2} \rangle \models a_2 \subseteq f a_1^*$ with $R_{a_1} = R$ and $R_{a_2} = \{\langle u,v \rangle\}$.
952 A Modal Perspective on Path Constraints

Now, let us establish the NLOGSPACE upper bound. Let \( c = r \sim p \subseteq f \ q \) (respectively, \( c = r \sim p \subseteq b \ q \)) be a path constraint and \( G \) a finite L-structure. We provide an NLOGSPACE algorithm to check whether \( G \not\models c \). Since NLOGSPACE = co-NLOGSPACE, we are done. The steps are the following:

1. nondeterministically choose \( u \) and \( v \) in \( V \);
2. check in NLOGSPACE whether \( \langle rt, u \rangle \in tr(r) \) (see Lemma 5.1);
3. check in NLOGSPACE whether \( \langle u, v \rangle \in tr(p) \) (see Lemma 5.1);
4. check in NLOGSPACE whether \( \langle u, v \rangle \not\in tr(q) \) (respectively \( \langle v, u \rangle \not\in tr(q) \)).

To see that the final step can also be done in NLOGSPACE, use Lemma 5.1 together with the fact that NLOGSPACE = co-NLOGSPACE.

Notice that the proof for NLOGSPACE-hardness is actually a lower bound for forward constraints. Moreover (and by taking \( R_{a_0} = \{ \{ u, v \} \} \)) it can be adapted for backward constraints. In a similar manner, the proof can be adapted for deterministic structures. Summarizing, then, we have the following:

**Theorem 5.3**
The query evaluation problems for the classes of forward constraints, backward constraints, and path constraints \( c \) are all NLOGSPACE-complete in \(|G|\) and \(|c|\), for both deterministic and nondeterministic graphs \( G \).

As an aside, we designed the logic PDL_{path} in such a way that we can express reasoning problems for standard path constraints as easily as possible. We can ‘measure’ how well PDL_{path} fits this purpose by comparing the complexity results of reasoning in PDL_{path} to those of reasoning in other logics, and hopefully getting as close as possible to the complexity of the corresponding problem on standard path constraints. For instance, the hybrid \( \mu \)-calculus has a model-checking problem in NP \( \cap \coNP \); given Theorems 5.2 and 5.3 this result disqualifies the hybrid \( \mu \)-calculus as a logic that nicely fits the descriptive requirements of standard path constraints; on the other hand, the alternation-free \( \mu \)-calculus would fit better.

### 5.2 The containment problem

Our next aim is to obtain sharp complexity results for containment problems for classes of path constraints. We start by considering nondeterministic \( L \)-structures, and the first step is to show the following:

**Theorem 5.4**
The containment problem for forward constraints is decidable in exponential time, while it is at least \( \text{PSPACE} \)-hard whenever \(|L| \geq 2\).

**Proof.** The EXPTIME upper bound is a corollary of Lemma 3.4(2) and Theorem 4.4. As to the lower bound \( \text{PSPACE} \), each relational term \( t \) built with \( +, *, \) over the atomic terms \( a_0, a_1, \ldots \) can obviously be viewed as regular expressions and we write \( L(t) \) to denote the language generated by \( t \). By [34, Theorem 2.12(c)], the problem of checking whether \( L((a_0 + a_1)^*) \not\subseteq L(t) \), where \( t \) is a relational term built over \( \{ a_0, a_1 \} \), is \( \text{PSPACE} \)-complete. The complement problem belongs to the same complexity class. One can show that for any regular expression \( t, t', L(t) \subseteq L(t') \) iff for any structure \( \langle V, r, R_{a_0}, R_{a_1}, tr(t) \subseteq tr(t') \). Consequently, it is easily verified that for any relational term \( t \) built over \( \{ a_0, a_1 \} \),
$L((a_0 + a_1)^*) \subseteq L(t) \iff (a_0 + a_1)^* \subseteq_f t$. Hence, the containment problem restricted to two labels and without forward constraints as premisses, is already PSPACE-hard.

For backward constraints one can obtain results similar to those for forward constraints.

**Theorem 5.5**
The containment problem for backward constraints is decidable in exponential time, while it is at least PSPACE-hard whenever $|L| \geq 3$.

**Proof.** Since the converse construction $(\cdot)^{-1}$ is not present in the path expressions, we cannot simply use the proof of Theorem 5.4. However, one can easily show that for any regular expression $p$ built over \{a_0, a_1\}, (i) $L((a_0 + a_1)^*) \subseteq L(p)$ iff (ii) for any $L$-structure $G$, $G \models p \subseteq_b a_2$ implies $G \models (a_0 + a_1)^* \subseteq_b a_2$, and from this we easily get our theorem. So let us prove that the equivalence (i) iff (ii) holds.

(Only-if) If $L((a_0 + a_1)^*) \subseteq L(p)$, then obviously for every $L$-structure $G$, $G \models p \subseteq_b a_2$ implies $G \models (a_0 + a_1)^* \subseteq_b a_2$.

(If) Assume that (ii) holds and suppose that there is a finite word $w \in L((a_0 + a_1)^*) \setminus L(p)$ such that $w = b_1 \cdots b_n$ for some $n \geq 0$ ($n = 0$ if $w = \epsilon$). Let $G_0 = \langle \{1, \ldots, n + 1\}, 1, R_{a_0}, R_{a_1}, R_{a_2} \rangle$ be the structure such that

- for every $i \in \{0, 1\}$, for all $k, k' \in \{1, \ldots, n + 1\}$, $k R_{a_i} k'$ iff $k' = k + 1$ and $b_k = a_i$;
- $R_{a_2} := \{ \langle k + 1, 1 \rangle : k \in \{0, \ldots, n\}, b_1 \cdots b_k \in L(f) \}$.

So $G_0 \models p \subseteq_b a_2$ but not $G \models (a_0 + a_1)^* \subseteq_b a_2$ since $\langle 1, n + 1 \rangle \in (R_{a_0} \cup R_{a_1})^*$ and $\langle n + 1, 1 \rangle \not\in R_{a_2}$, a contradiction.

We now restrict attention to deterministic $L$-structures, which makes a substantial difference. The containment problem with (lollipop) path constraints of the form $r \leadsto p \subseteq_f q$ restricted to deterministic $L$-structures is undecidable even if $L$ contains only two labels [12]. However, in the lollipop constraints used in the proof, the operator $\ast$ occurs in $r$, and this is used to encode the word problem. Moreover by imposing restrictions of $r$, $p$, and $q$ (for instance by forbidding $\ast$), decidable restrictions of the containment problem on deterministic $L$-structures have been found (again, see [12]). Using the results in this paper, we are able to identify a new decidable case. By combining Lemma 3.4, item 3, and Theorem 4.3, we get

**Lemma 5.6**
The containment problem for backward constraints restricted to deterministic $L$-structures for finite sets of labels $L$ is decidable in exponential time.

Lemma 5.6 above provides a positive answer to an open question from [12]. However, in spite of Lemma 3.4 it is open whether the containment problem for forward constraints restricted to deterministic $L$-structures is decidable. Similarly, the decidability of the containment problem for backward constraints restricted to deterministic $L$-structures (without restrictions on $L$) is also open.

### 6 Conclusions

By moving back and forth between reasoning tasks for PDL$_{path}$ and reasoning tasks for semistructured data, we have given new and transparent decidability proofs for the forward constraints proposed in [3] for optimizing queries on semistructured data, mostly in the context of the web. In some cases we have obtained sharp upper and lower bounds that are better...
TABLE 2. A summary of results on reasoning tasks with path constraints

<table>
<thead>
<tr>
<th>Query evaluation problem</th>
<th>nondeterministic graphs</th>
<th>deterministic graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward constraints</td>
<td>NLOGSPACE-complete;</td>
<td>NLOGSPACE-complete;</td>
</tr>
<tr>
<td></td>
<td>this paper, Theorem 5.3</td>
<td>this paper, Theorem 5.3</td>
</tr>
<tr>
<td>backward constraints</td>
<td>NLOGSPACE-complete;</td>
<td>NLOGSPACE-complete;</td>
</tr>
<tr>
<td></td>
<td>this paper, Theorem 5.3</td>
<td>this paper, Theorem 5.3</td>
</tr>
<tr>
<td>constraints</td>
<td>NLOGSPACE-complete;</td>
<td>NLOGSPACE-complete;</td>
</tr>
<tr>
<td></td>
<td>this paper, Theorem 5.2</td>
<td>this paper, Theorem 5.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Containment problem</th>
<th>nondeterministic graphs</th>
<th>deterministic graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward constraints</td>
<td>PSPACE-hard, in EXPTIME;</td>
<td>open</td>
</tr>
<tr>
<td></td>
<td>this paper, Theorem 5.4</td>
<td></td>
</tr>
<tr>
<td>backward constraints</td>
<td>PSPACE-hard, in EXPTIME;</td>
<td>in EXPTIME ($L$ finite);</td>
</tr>
<tr>
<td></td>
<td>this paper, Theorem 5.5</td>
<td>this paper, Lemma 5.6</td>
</tr>
<tr>
<td>constraints</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
<tr>
<td></td>
<td>[13, Theorem 3.1]</td>
<td>[12, Theorem 6.1]</td>
</tr>
</tbody>
</table>

than previously known ones (see e.g. Theorem 5.2 and 5.3), and in other cases we have improved known bounds (Theorems 5.4 and 5.5). Tables 1 and 2 summarize the complexity and (un-) decidability results for the reasoning tasks considered in this paper.

It is worth observing that some of our decidability results were obtained by re-using the results of [20]. More generally, there are many areas in computer science in which describing and reasoning about finite graphs is a key issue. There exists a large body of work in feature structures [42], process algebra [38], or knowledge representation [21] that can be usefully applied in the theory of semistructured data. But there are differences in the kind of questions asked and in the emphasis in descriptions of linguistic structures, processes, or knowledge on the one hand, and in descriptions of database schemas on the other hand; these differences make the present application interesting and nontrivial.

Our modal logic perspective on standard path constraints moves many decidability and complexity issues for semistructured data into the realm of PDL-like logics. Here are just some of the many remaining open problems.

1. Complexity of the containment problem for the class of forward constraints (respectively backward constraints) (we know PSPACE-hardness and the EXPTIME upper bound). The containment problem for the class of forward constraints cannot be expressed naturally by query containment between a conjunctive two-way regular path query with constants and a tree two-way regular path query with constants, although this problem is in PSPACE [14]. Query containment between conjunctive two-way regular path queries with constants is roughly about the validity of questions of the form $\exists q_1 \cap \cdots \cap q_m \subseteq p_1 \cap \cdots \cap p_n$.

2. decidability of the containment problem for forward constraints restricted to deterministic $L$-structures.

3. Decidability of $\text{PDL}_{\text{path}}$ restricted to deterministic $L$-structures; decidability of PDL with converse and determinism is a long-standing open problem [43].

4. Extending our results to a richer path expressions language containing additional predicates such as, for example, XPath [45].
Acknowledgements

We are grateful to Ulrike Sattler and the anonymous referees for useful comments. We are especially grateful to Philippe Schnoebelen, who gave us the proof for Lemma 5.1. This research was supported by the British Council/NWO UK-Dutch Joint Scientific Research Programme under grant JRP548. Natasha Alechina was supported by the EPSRC grant M98050. Maarten de Rijke was supported by grants from the Netherlands Organization for Scientific Research (NWO), under project numbers 612-13-001, 365-20-005, 612.069.006, 612.000.106, and 220-80-001.

References


Received 8 November 2001