Extraction and Implication of Path Constraints ⋆

Yves André, Anne-Cécile Caron, Denis Debarbieux, Yves Roos, and Sophie Tison

{andre,caronc,debarbie,yroos,tison }@lifl.fr
Laboratoire d'Informatique Fondamentale de Lille,
U.M.R. C.N.R.S. 8022
Université de Lille 1, 59655 Villeneuve d'Ascq Cedex. France.

Abstract. We consider semistructured data as rooted edge-labeled directed graphs, and path inclusion constraints on these graphs. In this paper, we show that we can extract from a finite datum D a finite set $\mathcal{C}_f(D)$ of word inclusions, which implies exactly every word inclusion satisfied by D. Then, we give a new decision algorithm for the implication problem of a constraint $p \preceq q$ by a set of constraints $p_i \preceq u_i$ where p, q, and the p_i's are regular path expressions and the u_i's are non empty paths, improving in this particular case, the more general algorithms of S. Abiteboul and V. Vianu, and Alechina et al. Moreover, in the case of a set of word equalities $u_i \equiv v_i$, we give a more efficient decision algorithm for the implication of a word equality $u \equiv v$, improving the more general algorithm of P. Buneman et al., and we prove that, in this case, the implication problem for non deterministic models or for (complete) deterministic models are equivalent.

1 Introduction

The development of the World Wide Web has led to the birth of semistructured data models with languages adapted to these models. A lot of works have been done to define such models and to extend database techniques to them. In this paper, we see semistructured data as rooted edge-labeled directed graphs: indeed, we can model HTML pages as a graph (a page is a node, an hyper-link is an edge) or we can model XML documents as graphs. A presentation of this model and an overview of works done in this context can be found in [1].
Let us consider the datum figure 1 which represents a journal. This journal contains articles and each article is written by one or two authors.

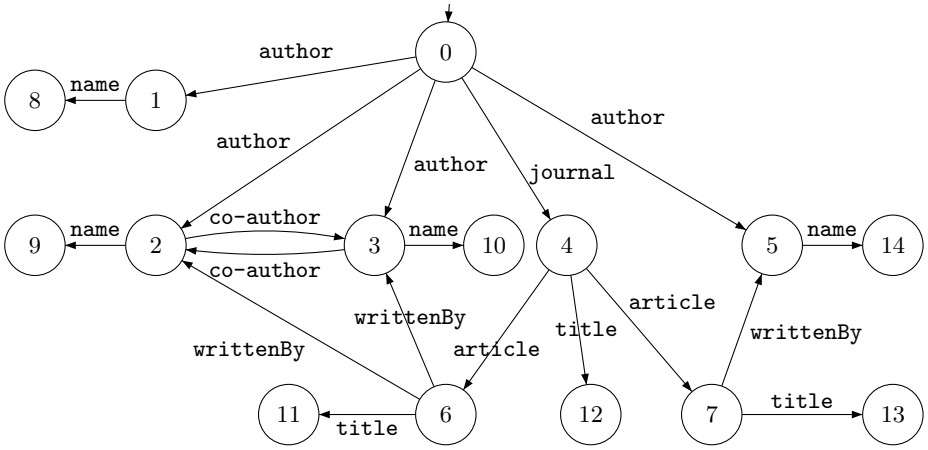


Fig. 1. Example of a semistructured datum

We can remark that some nodes have several outgoing edges with the same label (for example, the root has several "author" edges). In this case, the graph is said non-deterministic.

⋆ This research was partially supported by Inria (MOSTRARE team).

In the deterministic case, the outgoing edges of a given node must have distinct labels. Note that XML documents are usually non-deterministic.

Path : Query languages proposed for semistructured data and querying the web are based on path expressions (see for example Lorel [2], UnSQL [6]). In particular, a regular path expression or regular query is a regular expression on the alphabet of labels appearing in the data. The result of the regular query q, is the set of nodes reached from the root by a path labeled by any word u of q. By extension, this word u is called *path*.

For example, `author`, `author.name`, `journal.article.title` are paths of the datum D (figure 1). The regular expression `author.co-author`* is a regular query whose result on D is $\{1, 2, 3, 5\}$.

Path inclusion: To optimize path queries, it can be useful to use structural informations about the data. Some of these are called path constraints since they give restrictions on the data paths. Certain kinds of integrity constraints found in object-oriented databases and also common in semistructured databases can be expressed with path constraints. These constraints have been introduced by Abiteboul and Vianu in [3]. See for instance [8], [17] or [5] where different classes of path constraints are analyzed. Here, we study path inclusion constraints. A path inclusion constraint is written $p \preceq q$ where p and q are regular path queries, and means that the set of nodes result of p is included in the set of nodes result of q. Continuing the example, since the result of `author.co-author`$^+$ is $\{2, 3\}$ and the result of `journal.article.writtenBy` is $\{2, 3, 5\}$ the path inclusion `author.co-author`$^+ \preceq$ `journal.article.writtenBy` is satisfied. If we denote by $p \equiv q$ the conjunction $p \preceq q \wedge q \preceq p$, the datum D satisfies `author.co-author`$\equiv$ `author.co-author.co-author`. On this example, the constraint is of the form $u \equiv v$ where u and v are paths. We call *word equality* this kind of constraint. Similarly, the constraint $u \preceq v$ is called *word inclusion*.

To take advantage of path inclusions, we must be able to reason about them. In this paper, we study the classical implication problem and the decidability of the boundedness property which is a decision problem close to the implication.

Implication problem: A set of path inclusions $\mathcal{C}$ implies a path inclusion $p \preceq q$ denoted $\mathcal{C} \models p \preceq q$ if every datum model of $\mathcal{C}$ is also a model of $p \preceq q$. Given a set $\mathcal{C}$ of path inclusions, and two regular queries p, q, the implication problem for $\mathcal{C}$, p, q is to decide whether $\mathcal{C} \models p \preceq q$.

In this paper, we give a decision algorithm for the implication problem of a path inclusion $p \preceq q$ by a set of path inclusions $p_i \preceq u_i$, where p, q, the p_i's are regular path expressions, and the u_i's are non-empty paths. This problem is shown decidable in EXPSPACE in [3]. The authors of [5] give an EXPTIME decision algorithm and prove that this problem is PSPACE-hard. We give here a PSPACE algorithm for this decision problem.

In the particular case of deciding if a word equality $u \equiv v$ is implied by a finite set of word equalities $u_i \equiv v_i$ we have an ad-hoc decision algorithm. In [7], the authors give a cubic decision algorithm in the case of the implication for deterministic models (with more general forward constraints). Here, we prove that, in this very particular case of word equalities, implication problem for non-deterministic models is equivalent to the implication problem for (complete and) determistic ones. We build a decision algorithm which complexity is quasi-linear.

Boundedness property: A regular query p has the *boundedness property (strong boundedness property)* w.r.t a set $\mathcal{C}$ of path inclusions if there exists a regular query f such that $\mathcal{C} \models p \preceq f$ ($\mathcal{C} \models p \equiv f$) and $L(f)$, the language described by f, is finite.

On the example, since `author.co-author`$\equiv$ `author.co-author.co-author`, the regular query `author.co-author`$^+$ has the strong boundedness property. Since it is easier to answer to a finite query, we can see the strong boundedness property as a query optimization method. More generally, if a query q has the boundedness property w.r.t $\mathcal{C}$, there exists a finite query f such that $q \preceq f$. So it is possible to approximate q with a finite query f since the answer of f is a superset of the answer of q.

The next section contains formal definitions related to path inclusions. In section 3, we present an algorithm which computes from a finite datum D a set of word inclusions ; this set is finite and implies exactly every word inclusion satisfied by D. Another kind of problems is to study path inclusions independently of the data. In sections 4 and 5 we have a set of path inclusions and we answer to the implication problem. More precisely, in section 4, we use rewriting to define a decision algorithm for the implication problem of a constraint $p \preceq q$ by a set of constraints $p_i \preceq u_i$, where p, q, the p_i's are regular path expressions, and the u_i's

are non-empty paths. In section 5, we give a decision algorithm for the implication problem of a word equality $u \equiv v$ by a set of word equalities $u_i \equiv v_i$. This algorithm is based on the *union-find* algorithm (see [19]). We conclude this paper in section 6.

2 Preliminaries

In the sequel we use the following notions which were introduced in [3]. Let A be a fixed finite alphabet of labels.

Definition 1. – *A semistructured datum is a triple* $D =< N, \texttt{root}, T >$ *where* N *is a set of nodes,* $\texttt{root} \in N$ *is called the root of datum* D *and* $T \subseteq N \times A \times N$ *is the set of transitions.*
– *If* N *is finite, the datum is said finite.*
– *if, for all* n *in* N*, for all* a *in* A*, there is at most one transition (n,a,n') in* T*, then the graph is deterministic.*
– *if, for all* n *in* N*, for all* a *in* A*, there is at least one transition (n,a,n') in* T*, then the graph is complete.*

We are interested in the set of nodes which are reached by some paths in a datum. Then we can define the notions of query and result of query.

Definition 2.
– *A path is a word over the alphabet* A*.*
– *Given* D *a datum,* $u \in A^\star$*, let* $\mathrm{acc}_D(u)$ *be defined by :*
  1. *if* $u = \varepsilon$*, then* $\mathrm{acc}_D(u) = \{\texttt{root}\}$
  2. *if* $u = u'a$ *(*$u' \in A^\star$*,* $a \in A$*)* $\mathrm{acc}_D(u) = \{n \in N \mid \exists n' \in \mathrm{acc}_D(u'), (n', a, n) \in T\}$
– *A regular query* p *is a regular expression over* A*. The result of a query* p *over a datum* D *is the set* $\mathrm{acc}_D(p) = \cup_{u \in L(p)} \mathrm{acc}_D(u)$ *where* $L(p)$ *denotes the regular language described by* p*.*

Now, we formally define path inclusions, path equalities, and notions related to implication.

Definition 3.
– *A path inclusion is an expression of the form* $p \preceq q$ *where* p*,* q *are regular queries.*
– *A path equality* $p \equiv q$ *represents the conjunction* $(p \preceq q) \wedge (q \preceq p)$*.*
– *If* u *and* v *are paths,* $u \preceq v$ *is called word inclusion, and* $u \equiv v$ *is called word equality.*
– *A datum* D *satisfies a path inclusion* $p \preceq q$*, denoted* $D \models p \preceq q$*, if the set of nodes* $acc_D(p)$ *is included in* $acc_D(q)$*.* D *satisfies a set* $\mathcal{C}$ *of path inclusions, denoted* $D \models \mathcal{C}$*, if* D *satisfies each path inclusion of* $\mathcal{C}$*.*
– *A set* $\mathcal{C}$ *of path inclusions implies a path inclusion* $p \preceq q$*, denoted* $\mathcal{C} \models p \preceq q$*, if for each datum* D *such that* $D \models \mathcal{C}$*,* $D \models p \preceq q$*.*
– *Given a set* $\mathcal{C}$ *of path inclusions, and two regular queries* p*,* q*, the implication problem for* $\mathcal{C}$*,* p*,* q *is to decide whether* $\mathcal{C} \models p \preceq q$*.*

In [3], the authors prove that in the context of the implication problem we can restrict ourselvesto finite models, since implication and finite implication of path inclusions are equivalent. It is stated in the following proposition.

Proposition 1. *A set* $\mathcal{C}$ *of path inclusions implies a path inclusion* $p \preceq q$*, denoted* $\mathcal{C} \models p \preceq q$*, if for each* ***finite*** *datum* D *such that* $D \models \mathcal{C}$*,* $D \models p \preceq q$*.*

The implication of word inclusions is closed under right congruence, transitivity and reflexivity :
for all u, v words, $\mathcal{C}$ set of path inclusions, x label,
– $\mathcal{C} \models u \preceq v \;\Rightarrow\; \mathcal{C} \models ux \preceq vx$
– $\mathcal{C} \models u \preceq u$
– $(\mathcal{C} \models u \preceq v) \wedge (\mathcal{C} \models v \preceq w) \;\Rightarrow\; \mathcal{C} \models u \preceq w$
Related to implication, another interesting problem, from a query optimization point of view, is the decidability of the boundedness property.

Definition 4. *A regular query p has the boundedness property (strong boundedness property) w.r.t a set $\mathcal{C}$ of path inclusions if there exists a regular query f such that $\mathcal{C} \models p \preceq f$ ($\mathcal{C} \models p \equiv f$) and $L(f)$ is finite.*

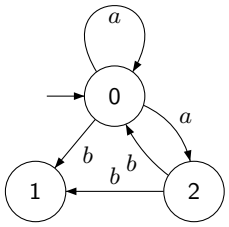
E.g., let $\mathcal{C}$ be $\{a^2 \preceq a\}$; w.r.t. $\mathcal{C}$, the query $a^\star$ is bounded (by a), whereas the query $ba^\star$ is not.

Definition 5. *Where $|p|$ is the length of the regular query p, the size of $p \preceq q$, denoted by $|p \preceq q|$, is the sum $|p| + |q|$, and the size of the set of constraints $\mathcal{C}$, denoted by $|\mathcal{C}|$, is the sum of the sizes of its constraints.*

3 Extraction of a finite set of constraints

Generally, a datum satisfies an infinite number of different path inclusion constraints and, all the more, an infinite number of word inclusion constraints. The goal of this section is to show that one can finitely generate the set of all word inclusion constraints that are satisfied by a given finite datum. More precisely, from a finite datum $D =< N, \texttt{root}, T >$, we want to define a finite set of word inclusion constraints $\mathcal{C}_f(D)$ such that for any words u and v, $\mathcal{C}_f(D) \models u \preceq v$ if and only if $D \models u \preceq v$.
First, let us denote by $S(D)$ the set of all results of word queries over D, that is $S(D) = \{\text{acc}_D(u) \mid u \in A^\star\}$ and, for any member s of $S(D)$, let us denote by $\text{lex}(s)$ the smallest element in the lexicographic order of the set $\{u \in A^\star \mid \text{acc}_D(u) = s\}$. By definition, for any datum D, we always have $\text{lex}(\{\texttt{root}\}) = \varepsilon$. Observe that, if we consider a finite datum D as a finite automaton, the set $S(D)$ corresponds with the set of states of the deterministic automaton equivalent to D that is obtained by the well known subset construction.
We are now able to define the following set of word inclusion constraints :
$\mathcal{C}_f(D) = \mathcal{C}_\preceq(D) \cup \mathcal{C}_\equiv(D)$ where $\mathcal{C}_\preceq(D) = \{\text{lex}(s) \preceq \text{lex}(s') \mid s, s' \in S(D), s \subsetneq s'\}$ and
$\mathcal{C}_\equiv(D) = \{\ \text{lex}(s)x \equiv \text{lex}(s') \mid x \in A, s' = \{n' \in N \mid \exists n \in s, (n, x, n') \in T\},$
$\text{lex}(s)x \neq \text{lex}(s')\ \}.$

Example 1. Let us consider the following semistructured datum D with root 0 :



then $S(D) = \{\{0\}, \{0, 2\}, \{0, 1\}, \{1\}, \emptyset\}$,
$\text{lex}(\{0\}) = \varepsilon$, $\text{lex}(\{0, 2\}) = a$, $\text{lex}(\{0, 1\}) = ab$, $\text{lex}(\{1\}) = b$, $\text{lex}(\emptyset) = ba$ and
$\mathcal{C}_f(D) = \{aba \equiv a, a \equiv aa, abb \equiv b, bb \equiv ba, \varepsilon \preceq a, \varepsilon \preceq ab, b \preceq ab, ba \preceq \varepsilon, ba \preceq a, ba \preceq ab, ba \preceq b\}$

Clearly, for any finite datum D, $\mathcal{C}_f(D)$ is finite, since the cardinality of $\mathcal{C}_f(D)$ is bounded by 2^{2n}, where n is the number of nodes in datum D. We obtain the following proposition :

Proposition 2. *For any datum D , for any words u and v, $D \models u \preceq v$ if and only if $\mathcal{C}_f(D) \models u \preceq v$. Moreover, if D is finite, then $\mathcal{C}_f(D)$ is finite.*

The proof of this proposition uses the next lemma, which enounces an interesting property on the word equalities implied by $\mathcal{C}_\equiv(D)$.

Lemma 1. *For any datum D, and for any word u, $\mathcal{C}_\equiv(D) \models u \equiv \text{lex}(\text{acc}_D(u))$.*

Note that $\mathcal{C}_f(D)$ is finite does not mean that D is finite. For instance, the set $\mathcal{C}_f(D) = \emptyset$ is only satisfied by infinite complete data.

As an other consequence of lemma 1, we obtain :

Proposition 3. *For any finite datum D and for any regular query q, q has the strong boundedness property w.r.t. $\mathcal{C}_{\equiv}(D)$.*

The empty word ε, in semistructured data, is a very particular word : by definition, for any datum D, $\mathrm{acc}_D(\varepsilon)$ is never empty since it is equal to the singleton which contains the root of D. Hence, word constraints involving the empty word ε are particular too : for instance, if $D \models \varepsilon \preceq v$ for some word v then, for any prefix v' of v, $\mathrm{acc}_D(v')$ is not empty. When the empty word appears as the right-hand side of a word inclusion constraint, the consequences are more surprising: let $\mathcal{C}$ be any set of word constraints such that $\mathcal{C}$ contains $u \preceq \varepsilon$ and $\mathcal{C} \models v \preceq w$ for some words u, v and w. It is easy to see that it follows $\mathcal{C} \models uv \preceq uw$, since for any datum D, $D \models u \preceq \varepsilon$ means $\mathrm{acc}_D(u) = \emptyset$ or $\mathrm{acc}_D(u) = \mathrm{acc}_D(\varepsilon)$. Conversely, when a set $\mathcal{C}$ of word inclusion constraints does not contain any constraint where ε appears as the right-hand side, then all constraints implied by $\mathcal{C}$ can be obtained simply in term of reflexive, transitive or right-congruence closure as it is shown in next section. This motivates the following definition of standard datum which corresponds with the well known definition of standard finite automaton[1] :

Definition 6. *A datum $D =< N, \mathtt{root}, T >$ is* standard *if, for any word $u \in A^\star$, $\mathtt{root} \in \mathrm{acc}_D(u)$ implies $u = \varepsilon$.*

Let us remark that the datum of the figure 1 is standart wheras the datum of the example 1 is not.
Let $D =< N, \mathtt{root}, T >$ be a datum, let $\mathrm{std}(D) =< N \cup \{\$\}, \$, T \cup \{(\$, x, n) \mid (\mathtt{root}, x, n) \in T\} >$ where \$ is a new node. Clearly, $\mathrm{std}(D)$ is standard and satisfies the following property :

Lemma 2. *For any datum D and for any paths u and v in A^+, $D \models u \preceq v$ if and only if $\mathrm{std}(D) \models u \preceq v$.*

Observe that in a standard datum $D =< N, \mathtt{root}, T >$ there is no path (except ε) to the root, so in $\mathcal{C}_f(D)$ there will be no constraint with ε as left-hand side. Moreover, the constraint $\mathrm{lex}(\emptyset) \preceq \varepsilon$ is the only constraint with ε as right-hand side which can appear in $\mathcal{C}_f(D)$. Since this constraint appears only if $\mathrm{lex}(\emptyset)$ is defined, that is if the datum D is not complete, we shall use the following notation :

Definition 7. *For any datum D, we denote by $\mathcal{C}_f^+(D)$ the constraint set $\mathcal{C}_f(D) \backslash \{\mathrm{lex}(\emptyset) \preceq \varepsilon\}$ if the datum D is not complete, $\mathcal{C}_f^+(D) = \mathcal{C}_f(D)$ otherwise.*

Then we can state :

Lemma 3. *Let D be a standard semistructured datum then $\forall u, v \in A^+, D \models u \preceq v$ if and only if $\mathcal{C}_f^+(D) \models u \preceq v$*

From lemma 2 and lemma 3, we finally obtain :

Proposition 4. *For any finite datum D over an alphabet A, the finite set of word constraints $\mathcal{C}_f^+(\mathrm{std}(D))$ is included in $A^+ \times A^+$ and satisfies : for any paths u and v in A^+, $D \models u \preceq v$ if and only if $\mathcal{C}_f^+(\mathrm{std}(D)) \models u \preceq v$.*

4 Solving implication problems with rewriting

In the previous section, we have extracted a set of constraints from a given datum D. From now, the problem is different : we want to decide the implication problem. So we have a set $\mathcal{C}$ of path constraints and we want to know if any datum satisfying $\mathcal{C}$ also satisfies a constraint $p \preceq q$ (with restriction on $\mathcal{C}$, p, q).
In order to define a decision algorithm for the implication problem, we introduce in this section a prefix rewrite system built from $\mathcal{C}$ such that u rewrites to v, if and only if $\mathcal{C}$ models $u \preceq v$. These techniques are also used in [3] or in [17]. We explain, in subsection 4.1 the link between path inclusions and rewriting and we give, in subsection 4.2 our algorithm to solve the implication problem. As discussed in the previous section, we do not want constraints with ε as right hand side. For this reason, we will suppose in this section that there is no path constraint of the form $p \preceq \varepsilon$.

[1] In particular, any `XML` document is a standard datum

4.1 Bounded path constraints and Prefix Rewriting

In this subsection, we summarize some results we have shown in [14]. From now on, we will only consider the case of a finite set of inclusions of the form $p \preceq u$ where p is a regular path expression and u is a word: we call such path inclusions *bounded path inclusions*. In this case, following and slightly generalizing [3], we associate with a set $\mathcal{C}$ of bounded path inclusions a prefix rewrite system such that u rewrites to v, if and only if $\mathcal{C}$ models $u \preceq v$.

Definition 8. *Let $\mathcal{C} = \{p_1 \preceq u_1, \ldots, p_n \preceq u_n\}$ be a finite set of bounded path inclusions over an alphabet A. We consider the relation on paths defined by $u \underset{\mathcal{C}}{\longrightarrow} v$ if and only if there exists i such that $u \in L(p_i)$ and $v = u_i$. By extension, we denote also $\underset{\mathcal{C}}{\longrightarrow}$ its right congruence closure. Then $\underset{\mathcal{C}}{\xrightarrow{\star}}$ denotes the reflexive, transitive closure of $\underset{\mathcal{C}}{\longrightarrow}$.*

We can remark that this relation is a prefix rewriting relation as defined in [10] based on an infinite rewrite system. We have the following property:

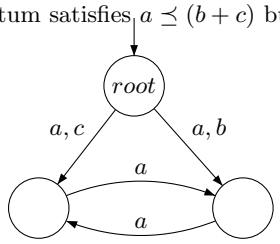
Proposition 5. *Let $\mathcal{C}$ be a set of bounded path inclusions. For any paths u, v, $u \underset{\mathcal{C}}{\xrightarrow{\star}} v$ if and only if $\mathcal{C} \models u \preceq v$.*

Moreover, using the hypothesis that $\mathcal{C}$ contains only bounded inclusions, we have shown that :

Proposition 6. *Let $\mathcal{C}$ be a set of bounded path inclusions, and q a regular query; the following properties are equivalent:*
- *$\mathcal{C} \models u \preceq q$*
- *there is some path v in $L(q)$ such that $\mathcal{C} \models u \preceq v$*
- *there is some path v in $L(q)$ such that $u \underset{\mathcal{C}}{\xrightarrow{\star}} v$*

The following example shows that proposition 6 does not hold when we consider unbounded path inclusions :

Example 2. The following datum satisfies $a \preceq (b + c)$ but satisfies neither $a \preceq b$ nor $a \preceq c$.



The proof of these two propositions uses a kind of (infinite) canonical model $G_{\mathcal{C}}$ of $\mathcal{C}$ which is close to the model defined in [3].
So, using these two propositions, some path constraints properties can be reduced to properties of prefix rewriting relations. We will use extensively this correspondence in the following. The prefix rewriting relation that we use is defined by infinite rewrite systems whereas most known decidability results on prefix rewriting relations deal with finite rewrite systems (e.g see [10]). However our rewrite systems are particular: they are defined by a finite set of rules where the left hand side are regular languages, the right hand side are words. So one could simulate them by a finite rewrite system, expressing left hand sides by the finite set of rules of the corresponding automata. An other approach would be to use ground tree transducers defined in [13]. Indeed, as this class of transducers is closed under transitive closure, it is easy to prove that the prefix left-rational rewrite relation can be realized by such a transducer and so that it is a recognizable relation, as defined in [11]. Then, by expressing the required properties in the theory of recognizable relations, we can get decision procedures for implication of constraints or boundedness properties.
We have chosen here a direct approach, even if the spirit of the techniques we use is the same, mainly computation of right congruence closure. This allows us to get simple and efficient constructions and to obtain tighter complexity results.

4.2 Solving implication problems

Now, we consider $\mathcal{C}$ a set of bounded path inclusions, p, q regular queries, and we study the implication problem $\mathcal{C} \models p \preceq q$. We obtain different complexity results when p and q are regular languages or simply paths. The algorithms use the computation of a set of ancestors by the rewrite system $\xrightarrow[\mathcal{C}]{\star}$. The main result of this section is the following theorem :

Theorem 1. *Let $\mathcal{C} = \{p_1 \preceq u_1, \ldots, p_n \preceq u_n\}$ be a finite set of bounded path inclusions, and p, q two regular queries. The implication problem $\mathcal{C} \models p \preceq q$ is PSPACE-complete.*

To prove this result, we use the rewrite relation $\xrightarrow[\mathcal{C}]{\star}$. First, we define the set of ancestors of a regular query q for this rewrite relation.

Definition 9. *Let $\mathcal{C} = \{p_1 \preceq u_1, \ldots, p_n \preceq u_n\}$ be a finite set of bounded path inclusions, and q a regular query. We define the set* $\text{ancestor}_{\mathcal{C}}(q) = \{u \mid \exists\ w_q \in L(q),\ u \xrightarrow[\mathcal{C}]{\star} w_q\}$.

Then we can state

Lemma 4. *Let $\mathcal{C} = \{p_1 \preceq u_1, \ldots, p_n \preceq u_n\}$ be a finite set of bounded path inclusions, and p, q two regular queries.*

$$\mathcal{C} \models p \preceq q \text{ iff } p \subseteq \text{ancestor}_{\mathcal{C}}(q)$$

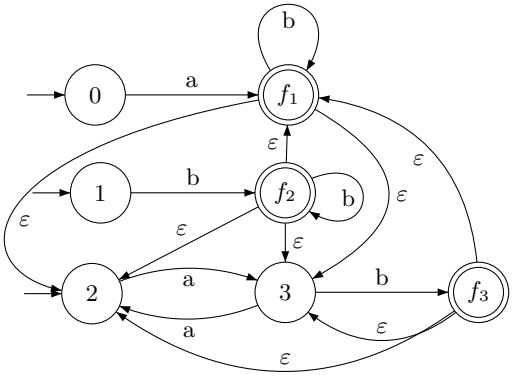
n

In order to compute $\text{ancestor}_{\mathcal{C}}(q)$ for any regular query q, we will use the following construction, introduced in [14]. We build a finite automaton $\mathcal{A}_{\mathcal{C}}$ (with ε-moves) which recognizes the language $R_{\mathcal{C}} = \{v \in A^\star \mid \exists i, v \xrightarrow[\mathcal{C}]{+} u_i\}$. It is already known that $R_{\mathcal{C}}$ is a recognizable language from [9], [11], [10]. We give here a different construction : For each i with $1 \leq i \leq n$, let $\mathcal{M}_i = (A, Q_i, I_i, F_i, \delta_i)$ be an automaton recognizing the language $L(p_i)$. We can assume, without loss of generality, that for different subscripts i and j, the intersection $Q_i \cap Q_j$ is empty. Then we can define $\mathcal{A}_{\mathcal{C}} = (A, Q, I, F, \Delta)$ where $Q = \cup_{i=1}^n Q_i$, $I = \cup_{i=1}^n I_i$, $F = \cup_{i=1}^n F_i$ and $\Delta = \cup_{k \in \mathbb{N}} \Delta_k$ where Δ_k, for $k \in \mathbb{N}$ is defined inductively by :

- $\Delta_0 = \cup_{i=1}^n \delta_i$
- for $k > 0$, $\Delta_k = \Delta_{k-1} \cup \{(q, \varepsilon, q') \mid \exists i \leq n, q \in F_i, q' \in \Delta_{k-1}(I, u_i)\}$

Since Δ is included in $Q \times (A \cup \{\varepsilon\}) \times Q$, it is clear that there exists an integer K such that $\Delta_K = \Delta_{K+1} = \Delta$. Since we have $K \leq |Q|^2$, automaton $\mathcal{A}_{\mathcal{C}}$ can be built in polynomial time in $|\mathcal{C}|$.

Example 3. Let $\mathcal{C} = \{ab^\star \preceq ba, b^+ \preceq a, a(aa)^\star b \preceq a\}$. Automaton $\mathcal{A}_{\mathcal{C}}$ is the following :



It is proven in [14] that $\mathcal{A}_{\mathcal{C}}$ recognizes $R_{\mathcal{C}}$, and it is clear that, from automaton $\mathcal{A}_{\mathcal{C}}$, we easily obtain, for any word u_i, an automaton which recognizes $\text{ancestor}_{\mathcal{C}}(u_i)$ in PTIME in the size of $\mathcal{C}$.

Now, in order to answer to the question $p \subseteq \text{ancestor}_{\mathcal{C}}(q)$, we have to compute the set of ancestors of q. Let us consider $\$_q$ a new letter (i.e. $\$_q \notin A$). We define a new finite set of bounded path inclusion $\mathcal{C}_q = \mathcal{C} \cup \{q \preceq \$_q\}$, and we can prove that a word u is in $\text{ancestor}_{\mathcal{C}}(q)$ if and only if u is in $\text{ancestor}_{\mathcal{C}_q}(\$_q) \cap A^\star$. It follows that an automaton $\mathcal{A}_{\mathcal{C}}^q$ for $\text{ancestor}_{\mathcal{C}}(q)$ can be built in polynomial time in $|q| + |\mathcal{C}|$. In [4] the authors give a decision algorithm for the inclusion of two regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$, given by two automata $\mathcal{A}_1$ and $\mathcal{A}_2$. Using this result, we can enounce :

Lemma 5. *For any set $\mathcal{C} = \{p_1 \preceq u_1, \ldots, p_n \preceq u_n\}$ of bounded path inclusions, and for any regular expressions p and q, the implication problem $\mathcal{C} \models p \preceq q$ is PSPACE.*

We are now able to end the proof of theorem 1, which is a consequence of the following lemma which states that, even when the regular expression q is reduced to a word u, the implication problem $\mathcal{C} \models p \preceq u$ is PSPACE-complete.

Lemma 6. *For any set $\mathcal{C} = \{p_1 \preceq u_1, \ldots, p_n \preceq u_n\}$ of bounded path inclusions, for any regular expressions p and for any word u, the implication problem $\mathcal{C} \models p \preceq u$ is PSPACE-complete.*

Nevertheless, for the implication problem of a constraint $u \preceq q$, we get a polynomial algorithm, since we only check whether $u \in \text{ancestor}_{\mathcal{C}}(q)$:

Proposition 7. *Let $\mathcal{C} = \{p_1 \preceq u_1, \ldots, p_n \preceq u_n\}$ a set of bounded path inclusions, u a word and q a regular query. We can decide the implication problem $\mathcal{C} \models u \preceq q$ in PTIME.*

5 Word equality constraints

In this section, we consider the case of a set of word equality constraints of the form $u \equiv v$ where u and v are paths.
In this case, as in [3], we associate with a set $\mathcal{C}$ of word equality constraints a graph $G_{\mathcal{C}}$ such that for any paths u and v, $\mathcal{C} \models u \equiv v$ if and only if $G_{\mathcal{C}} \models u \equiv v$.

Definition 10. *Let $\mathcal{C}$ be a set of word equality constraints over an alphabet A. Clearly, $\mathcal{C}$ is a symmetric binary relation over $A^\star$. We denote by $\equiv_{\mathcal{C}}$ the smallest equivalence relation, closed by right congruence, which contains $\mathcal{C}$, and, for any path $u \in A^\star$, we denote by $[u]_{\mathcal{C}}$ the equivalent class of the path u for the relation $\equiv_{\mathcal{C}}$.*

Let $G_{\mathcal{C}} =< N, r, T >$ be the labeled rooted directed graph where $N = \{[u]_{\mathcal{C}} \mid u \in A^\star\}$, $r = [\varepsilon]_{\mathcal{C}}$ and $T = \{([u]_{\mathcal{C}}, x, [ux]_{\mathcal{C}}) \mid u \in A^\star, x \in A\}$. Clearly, $G_{\mathcal{C}}$ is deterministic and complete, and for any path $u \in A^\star$, we have $acc_{G_{\mathcal{C}}}(u) = \{[u]_{\mathcal{C}}\}$. It follows :

Proposition 8. *For any set $\mathcal{C}$ of word equality constraints over an alphabet A, the following properties are equivalent :*
1. $\mathcal{C} \models u \equiv v$
2. $G_{\mathcal{C}} \models u \equiv v$
3. $u \equiv_{\mathcal{C}} v$

From definition of $G_{\mathcal{C}}$ and from equivalence between 1 and 3 of proposition 8, we obtain a converse of the problem studied in proposition 2 :

Corollary 1. *For any set $\mathcal{C}$ of word equality constraints over an alphabet A, $G_{\mathcal{C}}$ is the unique (complete) deterministic rooted graph D which satisfies : $D \models u \equiv v$ if and only if $\mathcal{C} \models u \equiv v$.*

Moreover, since $G_{\mathcal{C}}$ is a complete deterministic graph, we have the following result :

Proposition 9. *For any set $\mathcal{C}$ of word equality constraints over an alphabet A, the following properties are equivalent :*
1. $\mathcal{C} \models u \equiv v$.
2. $\mathcal{C} \models u \equiv v$ *on the family of deterministic data.*
3. $\mathcal{C} \models u \equiv v$ *on the family of complete deterministic data.*

Generally, our model $G_{\mathcal{C}}$ is an infinite graph. Nevertheless, when $\mathcal{C}$ is a finite set of word equality constraints, it is possible to build a finite deterministic graph in order to decide implication of word equality constraints. A quite similar construction has been introduced by Buneman et al. in [7] :
Let $\mathcal{C}$ be a finite set of word equality constraints over A. Let us denote by W the set of all prefixes of $\{w \in A^\star \mid \exists w' \in A^\star, (w \equiv w') \in \mathcal{C}\}$. For any path in W, let us denote by $[w]$ the equivalence class of w for the restriction of $\equiv_{\mathcal{C}}$ over W. Let us consider the

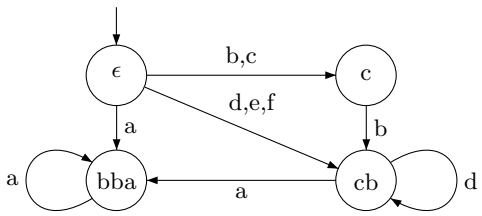


Fig. 2. graph $G_{\mathcal{C}}^f$

finite deterministic graph $G_{\mathcal{C}}^f =< N', r, T' >$ where $N' = \{[w] \mid w \in W\}$, $r = [\varepsilon]$ and $T' = \{([w], x, [wx]) \mid w \in W, wx \in W, x \in A\}$.
Let us remark that if $G_{\mathcal{C}}^f$ is a complete graph then it coincides with graph $G_{\mathcal{C}}$ and it follows that $\mathcal{C}$ has a finite canonical model if and only if $G_{\mathcal{C}}^f$ is complete.
Then we can use $G_{\mathcal{C}}^f$ to decide if a finite set $\mathcal{C}$ of word equality constraints implies a word equality constraint. First, it is clear that, by construction, $G_{\mathcal{C}}^f \models \mathcal{C}$ and, for any $w \in W$, $\mathrm{acc}_{G_{\mathcal{C}}^f}(w) = \{[w]\}$. Then, for any path u in $A^\star$, $\mathrm{acc}_{G_{\mathcal{C}}^f}(u) = \{[w]\}$ if and only if $u \equiv_{\mathcal{C}} w$.
Let us consider now the application $f_{\mathcal{C}}$, defined from $A^\star$ to $N' \times A^\star$, where N' is the set of nodes of $G_{\mathcal{C}}^f$, by : $\forall u \in A^\star, f_{\mathcal{C}}(u) = (\mathrm{acc}_{G_{\mathcal{C}}^f}(u_1), u_2)$ where $u = u_1 u_2$ and u_1 is the least prefix of u such that $\mathrm{acc}_{G_{\mathcal{C}}^f}(u_1) \neq \emptyset$.

Example 4. Let $A = \{a, b, c, d, e, f\}$ and $\mathcal{C} = \{a \equiv bba, b \equiv c, cb \equiv dd, d \equiv e, fa \equiv aa, ed \equiv f, e \equiv f, aa \equiv bba\}$. Figure 2 gives the graph $G_{\mathcal{C}}^f$ for this set of constraints. On this example, $f_{\mathcal{C}}(a^3) = (bba, \epsilon)$, $f_{\mathcal{C}}(a^3c) = (bba, c)$.

Proposition 10. *$C \models u \equiv v$ if and only if $f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v)$.*

Now, denoting by $f_{\mathcal{C}}(p)$ the set $\bigcup_{u \in L(p)} f_{\mathcal{C}}(u)$ for any regular path expression p
we can deduce, from the above proposition, proposition 8 and using the fact that $G_{\mathcal{C}}$ is complete and deterministic :

Corollary 2. *For any regular path expressions p and q, $\mathcal{C} \models p \equiv q$ iff $f_{\mathcal{C}}(p) = f_{\mathcal{C}}(q)$.*

Moreover, from the above corollary, we obtain :

Corollary 3. *A regular path expression p has the strong boundedness property w.r.t. a finite set of word equalities $\mathcal{C}$ if and only if $f_{\mathcal{C}}(p)$ is finite.*

We will now give the complexity of the three decision algorithms which correspond to proposition 10, corollary 2 and corollary 3. We will first give an algorithm which computes the graph $G_{\mathcal{C}}^f$ with a lower complexity than the algorithm given in [7].
This algorithm uses the union-find principle to compute the set of equivalent classes, in a way similar to Shostak algorithm [18]. The union-find principle manages a set of classes with three primitives: create a new class, compute the union of two classes and find a representative of a class (in [12] the authors give the data structures and the algorithms). In the following *find*(u) will denote a representative of the class of u. We first represent W by a prefix tree and consider one class by path of W (i.e. one node is one class). We compute the equivalent classes applying procedure *merge*, for all (u, v) in $\mathcal{C}$:

- If $(u \equiv_{\mathcal{C}} v) \in \mathcal{C}$ then compute the union between the class of u and the class of v
- Compute recursively the union between the class of ux and the class of vx, for x label.

```
procedure merge(in  u,v: Path; out  S:SetOfClasses)
% merge the class of u with the class of v .
union(u,v,S)
for  each u' such that find(u') =find(u) do  % i.e. u and u' are in the same class
   for  each v' such that find(v') =find(v) do
      for  each x ∈ A do
         if  u'x ∈ W and v'x ∈ W then  merge(u'x,v'x,S) end if
      end for end for end for
end
```

Note that W contains $|\mathcal{C}|$ elements, where $|\mathcal{C}|$ is the sum of the sizes of the constraints in $\mathcal{C}$. Merge is an $O(|A| \times |\mathcal{C}|^2)$ algorithm. But we define an appropriate data structure (see example 4) and we obtain an $O(|A|)$ algorithm. We prove that $G^f_{\mathcal{C}}$ is the graph $< N, r, T >$ where $N = \{\mathit{find}(u) \mid u \in W\}$, $r = \{\mathit{find}(\epsilon)\}$ and $T = \{(\mathit{find}(\mathrm{u}), x, \mathit{find}(ux)) \mid ux \in W\}$. In [4] and [19], it is proved that an algorithm using $n-1$ unions and m finds ($m \geq n$) is an $O(n + m.\alpha(n, m))$ algorithm[2]. It follows that our algorithm which computes $G^f_{\mathcal{C}}$ is better than $O(|\mathcal{C}| \times lg^*(|\mathcal{C}|))$, so is in quasi linear time in $|\mathcal{C}|$.

Example 5. (example 4 continued) The figure 3 shows the data structure G used to compute the graph $G^f_{\mathcal{C}}$. We can see the prefix tree (a), some new edges which are helpful to compute efficiently unions (b) and edges which simulate the classes (c). Since $(b \equiv c) \in \mathcal{C}$, b and c are in the same class. Then bb and cb are in the same class. Since $bba \in W$ and $cba \notin W$, we add an edge from the class of cb to the class of bba labeled by a. It follows from $a \equiv bba \equiv cba \equiv dda \equiv eda \equiv fa$ that $\mathrm{acc}_G(a) = \mathrm{acc}_G(fa)$. Since $f \equiv ed \equiv fd$, we get $\mathrm{acc}_G(fd^*) = \mathrm{acc}_G(f)$. Finally, after merging the equivalent nodes, we obtain the graph $G^f_{\mathcal{C}}$ shown in figure 2.

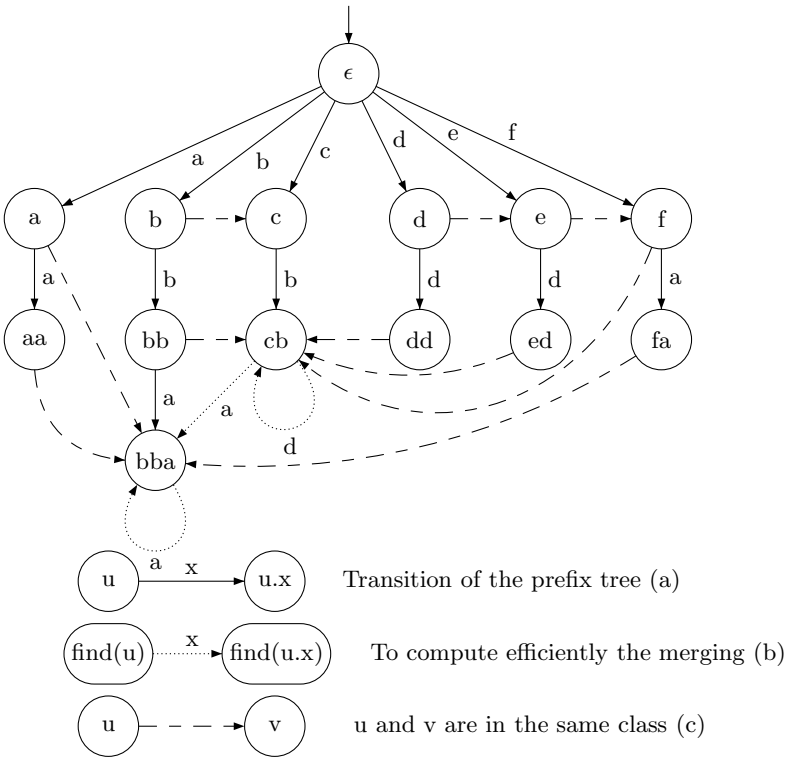


Fig. 3. Steps of the union-find algorithm

Clearly, for any path u, if $G^f_{\mathcal{C}}$ is given, the computation of $f_{\mathcal{C}}(u)$ has a linear complexity in the length of u. If we compute $f_{\mathcal{C}}(p)$ for some regular path expression p, we obtain :

Lemma 7. *For any regular path expression p and given the graph $G^f_{\mathcal{C}}$, the test of finiteness of $f_{\mathcal{C}}(p)$ can be done in PTIME in the sum of the size of $G^f_{\mathcal{C}}$ and the size of p and a comparison between $f_{\mathcal{C}}(p)$ and $f_{\mathcal{C}}(q)$ for some regular path expressions p and q can be done in PSPACE in the sum of the size of $G^f_{\mathcal{C}}$ and the size of the two regular expressions.*

Remark 1. For the comparison of $f_{\mathcal{C}}(p)$ and $f_{\mathcal{C}}(q)$ for some regular path expressions p and q, we cannot obtain a better complexity, since if we consider an empty set $\mathcal{C}$ of word equality

[2] α is the inverse of the Ackermann function. The Ackermann function A is defined by $A(1, j) = 2^j$ $j \geq 1$, $A(i, 1) = A(i-1, 2)$ if $i \geq 2$, $A(i, j) = A(i-1, A(i, j-1))$ if $i, j \geq 2$

constraints, we have $\mathcal{C} \models p \equiv q$ if and only if the language described by p is equal to the language described by q, and it is known from [15] that this problem is PSPACE-complete in the sum of the size of the two regular expressions p and q. It follows that the problem to know whether, given a finite set $\mathcal{C}$ of word equality constraints, we have $\mathcal{C} \models p \equiv q$ for some regular expressions p and q is PSPACE-complete.

Then, summarizing the complexity results of this section, we obtain :

Proposition 11. *For any finite set of word equality constraints $\mathcal{C}$,*
- *it is decidable to know whether $\mathcal{C} \models u \equiv v$ for some paths u and v in quasi-linear time in the sum of $|\mathcal{C}|$ and the size of the constraint $u \equiv v$.*
- *the problem to know whether $\mathcal{C} \models p \equiv q$ for some regular path expressions p and q is PSPACE-complete, in the sum of $|\mathcal{C}|$ and the size of the constraint $p \equiv q$.*
- *it is decidable to know whether some regular path expression p has the strong boundedness property w.r.t. $\mathcal{C}$ in PTIME in the sum $|\mathcal{C}| + |p|$.*

We can deduce from proposition 10 :

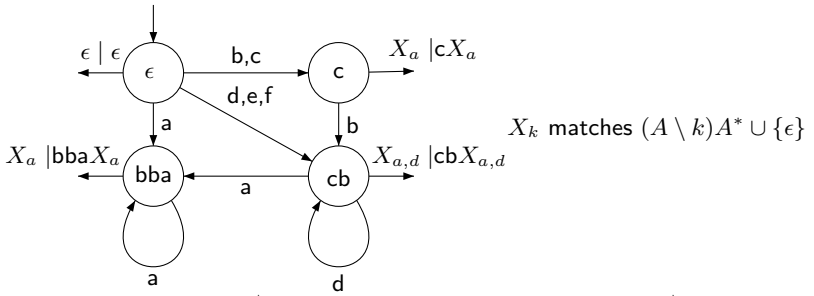
Corollary 4. *For any finite set $\mathcal{C}$ of word equality constraints over an alphabet A, and for any regular path expressions p and q, ($\mathcal{C} \models p \equiv q$) if and only if ($\mathcal{C} \models p \equiv q$ on the family of finite (complete) deterministic data).*

It follows from the proposition 11 that it is possible to check if a regular query q has the strong boundedness property w.r.t $\mathcal{C}$. But we are also able to compute f such that $L(f)$ is finite and $\mathcal{C} \models q \equiv f$.

Proposition 12. *Let $\mathcal{C}$ be a non-empty finite set of word equalities over an alphabet A. We compute, in quasi linear time, a transducer $\tau_{\mathcal{C}}$ such that, for any regular query p over A :*
1. *$\mathcal{C} \models p \equiv \tau_{\mathcal{C}}(L(p))$*
2. *$\tau_{\mathcal{C}}(L(p))$ is finite if and only if p has the strong boundedness property w.r.t. $\mathcal{C}$*

To close this section on word equality constraints, we can study this proposition on an example.

Example 6. (example 4 continued) The transducer $\tau_{\mathcal{C}}$ is :



- $\tau_{\mathcal{C}}(a^+b) = bbab$ because $f_{\mathcal{C}}(a^+b) = (bba, b)$. As $L(bbab)$ is finite, a^+b has the strong boundedness property w.r.t $\mathcal{C}$ and $\mathcal{C} \models a^+b \equiv bbab$.
- $\tau_{\mathcal{C}}(f^+) = cbf^*$ because $f_{\mathcal{C}}(f^+) = (cb, f^*)$. As $L(cbf^*)$ is not finite, f^+ has not the strong boundedness property w.r.t. $\mathcal{C}$. Nevertheless, $\mathcal{C} \models f^+ \equiv cbf^*$ is true.

6 Conclusion

We have proposed an algorithm extracting from a document D a set $\mathcal{C}_f$ of constraints. This set is finite and implies a path inclusion constraint if and only if D satisfies this constraint. Then, implication of a path inclusion by a set of bounded path inclusions, implication of a path equality by a set of word equalities, and the boundedness property in the case of word equalities have been studied.

We summarize our results :

$\mathcal{C}$ = {bounded path inclusions}	new results	already known
$\mathcal{C} \models p \preceq q$	PSPACE (lemma 5)	EXPSPACE [3] EXPTIME,PSPACE hard [5]
$\mathcal{C} \models p \preceq u$	PSPACE-complete (lemma 6)	
$\mathcal{C} \models u \preceq q$	PTIME (proposition 7)	

$\mathcal{C}$ = {word equality constraints}	new results	already known
$\mathcal{C} \models p \equiv q$	PSPACE-complete (proposition 11)	
$\mathcal{C} \models u \equiv v$	quasi linear (proposition 11)	cubic time [7]
boundedness property	PTIME (proposition 11)	

When the set of path inclusions $\mathcal{C}$ contains unbounded path inclusions, the problem of deciding whether a regular query p has the boundedness property w.r.t. $\mathcal{C}$ is still open.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kaufmann Publishers, 2000.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1):68–88, 1997.
3. S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proc. of ACM Symposium on Principles of Database Systems*, 1997.
4. A. Aho, J. Hopcroft, and J. Ullman. *The design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
5. N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, to appear, 2004.
6. P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *SIGMOD*, pages 505–516, Montreal, 1996.
7. P. Buneman, W. Fan, and S. Weinstein. Query optimization for semistructured data using path constraints in a deterministic data model. In *Lecture Notes in Computer Science 1949*, pages 208–223. 7th International Workshop on Database Programming Languages, 1999.
8. P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences*, 61(2), 2000.
9. J. Richard Büchi and W.H. Hosken. Canonical systems which produce periodic sets. *Mathematical Systems Theory*, 4(1), 1970.
10. D. Caucal. On the regular structure of prefix rewritings. *CAAP*, pages 87 – 102, May 1990.
11. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997.
12. T. Cormen, C. Leiserson, and R.Rivest. *Introduction to Algorithms*. MIT Press, 1990.
13. M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. pages 242–248. LICS 90, 1990.
14. D. Debarbieux, Y. Roos, S. Tison, Y. Andre, and A.C. Caron. Path rewriting in semistructured data. In *proceedings of words'03: 4th International Conference on Combinatorics on Words*, pages 358–369, Turkü, Finland, 2003. TUCS General Publication.
15. M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the Theory of NP-completeness*. Freeman, 1978.
16. V.M. Gluskov. the abstract theory of automata. In *Russian mathematical survey*, volume 16, pages 1–53, 1961.
17. G. Grahne and A. Thomo. Query containment and rewriting using views for regular path queries under constraints. In *proceedings of PODS'03*, pages 111–122. Symposium on Principles of Database Systems, ACM, 2003.
18. R.E. Shostak. An algorithm for reasoning about equality. *Commun. ACM*, 21(7):583–585, 1978.
19. R. Tarjan. Efficiency of a good but non linear set union algorithm. In *Journal of the ACM*, volume 22, n°2, pages 215 – 225, 1975.

7 Appendices

Proof of lemma 1 : Let $D = (N, \texttt{root}, T)$. We prove this lemma by induction on the length of u

- If $u = \varepsilon$ then $\mathrm{acc}_D(\varepsilon) = \texttt{root}$ and $\mathrm{lex}(\{\texttt{root}\}) = \varepsilon$, then we have $\mathcal{C}_{\equiv}(D) \models \varepsilon \equiv \mathrm{lex}(\mathrm{acc}_D(\varepsilon))$.
- If $u = u'x$ with $x \in A$, by induction hypothesis we have $\mathcal{C}_{\equiv}(D) \models u' \equiv \mathrm{lex}(\mathrm{acc}_D(u'))$ and by right congruence we obtain $\mathcal{C}_{\equiv}(D) \models u'x \equiv \mathrm{lex}(\mathrm{acc}_D(u'))x$. Moreover $\mathrm{acc}_D(u) = \{n \in N \mid \exists n' \in \mathrm{acc}_D(u'), (n', x, n) \in T\}$. Now, if $\mathrm{lex}(\mathrm{acc}_D(u'))x$ is equal to $\mathrm{lex}(\mathrm{acc}_D(u))$, then by reflexivity $\mathcal{C}_{\equiv}(D) \models \mathrm{lex}(\mathrm{acc}_D(u'))x \equiv \mathrm{lex}(\mathrm{acc}_D(u))$, and if $\mathrm{lex}(\mathrm{acc}_D(u'))x$ and $\mathrm{lex}(\mathrm{acc}_D(u))$ are different, it follows from the definition of $\mathcal{C}_{\equiv}(D)$ that $\mathrm{lex}(\mathrm{acc}_D(u'))x \equiv \mathrm{lex}(\mathrm{acc}_D(u))$ is in $\mathcal{C}_{\equiv}(D)$. Finally, by transitivity, $\mathcal{C}_{\equiv}(D) \models u \equiv \mathrm{lex}(\mathrm{acc}_D(u))$. Observe that, for a member s of $S(D)$, we could have chosen, instead of $\mathrm{lex}(s)$, any word of the set $\{w \in A^\star \mid \mathrm{acc}_D(w) = s\}$ to represent s in a canonical way, assuming $\{\texttt{root}\}$ is represented by ε.

□

Proof of proposition 2 :

1. Let us suppose first that $D \models u \preceq v$, then we have $\mathrm{acc}_D(u) \subseteq \mathrm{acc}_D(v)$. From lemma 1, since $\mathcal{C}_{\equiv}(D) \subseteq C_f(D)$, we know that $\mathcal{C}_f(D) \models u \equiv \mathrm{lex}(\mathrm{acc}_D(u))$ and $C_f(D) \models v \equiv \mathrm{lex}(\mathrm{acc}_D(v))$. If $\mathrm{acc}_D(u) = \mathrm{acc}_D(v)$ then $\mathrm{lex}(\mathrm{acc}_D(u))$ is equal to $\mathrm{lex}(\mathrm{acc}_D(v))$ and, by transitivity, it follows $\mathcal{C}_f(D) \models u \equiv v$. If $\mathrm{acc}_D(u)$ and $\mathrm{acc}_D(v)$ are different then $\mathrm{acc}_D(u) \subsetneq \mathrm{acc}_D(v)$ and $\mathrm{lex}(\mathrm{acc}_D(u)) \preceq \mathrm{lex}(\mathrm{acc}_D(v)) \in \mathcal{C}_f(D)$. By transitivity, we obtain also $\mathcal{C}_f \models u \preceq v$.
2. Le us suppose now that $\mathcal{C}_f(D) \models u \preceq v$. Clearly, it is sufficient to prove that $u \preceq v \in \mathcal{C}_f(D)$ implies that $\mathrm{acc}_D(u) \subseteq \mathrm{acc}_D(v)$. Then we have $u \neq v$. Let us consider two cases:
   (a) There are s and s' in $S(D) = \{\mathrm{acc}_D(w) \mid w \in A^\star\}$ and a letter x in A such that $u = \mathrm{lex}(s)x$, $v = \mathrm{lex}(s')$ and $s' = \{n' \mid \exists n \in s \wedge (n, x, n') \in T\}$. Then $\mathrm{acc}_D(u) = \mathrm{acc}_D(v) = s'$.
   (b) There are s and s' in $S(D)$ such that $s \subsetneq s'$, $u = \mathrm{lex}(s)$ and $v = \mathrm{lex}(s')$. Then $\mathrm{acc}_D(u) \subsetneq \mathrm{acc}_D(v) = s'$.

□

Proof of proposition 3 : Let F be the set $\{\mathrm{lex}(\mathrm{acc}_D(u)) \mid u \in L(q)\}$. As $S(D) = \{\mathrm{acc}_D(u) \mid u \in A^*\}$ is finite, F is finite. From lemma 1, for any u in $L(q)$, there exists v in F such that $\mathcal{C}_{\equiv}(D) \models u \equiv v$ and conversely, for any v in F, there exists u in $L(q)$ such that $\mathcal{C}_{\equiv}(D) \models u \equiv v$. Hence $\mathcal{C}_{\equiv}(D) \models q \equiv F$.

□

Proof of lemma 3 : Let u and v be two words in A^+. Clearly, if $\mathcal{C}_f^+(D) \models u \preceq v$ then $\mathcal{C}_f(D) \models u \preceq v$, and from proposition 2, we have $D \models u \preceq v$. Conversely, let us suppose that $D \models u \preceq v$ then $\mathrm{acc}_D(u) \subseteq \mathrm{acc}_D(v)$. Moreover, as $\mathcal{C}_{\equiv}(D)$ is clearly included in $\mathcal{C}_f^+(D)$, we obtain from lemma 1 that $\mathcal{C}_f^+(D) \models u \equiv \mathrm{lex}(\mathrm{acc}_D(u))$ and $\mathcal{C}_f^+(D) \models v \equiv \mathrm{lex}(\mathrm{acc}_D(v))$. Now, if $\mathrm{lex}(\mathrm{acc}_D(u)) = \mathrm{lex}(\mathrm{acc}_D(v))$, then $\mathcal{C}_f^+(D) \models \mathrm{lex}(\mathrm{acc}_D(u)) \equiv \mathrm{lex}(\mathrm{acc}_D(v))$ by reflexivity, and if $\mathrm{lex}(\mathrm{acc}_D(u))$ and $\mathrm{lex}(\mathrm{acc}_D(v))$ are different then $\mathrm{acc}_D(u) \subsetneq \mathrm{acc}_D(v)$, and since $v \neq \varepsilon$, it follows that $\mathrm{lex}(\mathrm{acc}_D(u)) \preceq \mathrm{lex}(\mathrm{acc}_D(v))$ belongs to $\mathcal{C}_f^+(D)$. Finally, by transitivity, we obtain $\mathcal{C}_f^+(D) \models u \preceq v$. □

Proof of lemma 4 :

- If $p \subseteq \mathit{ancestor}_{\mathcal{C}}(q)$ then $\forall u_p \in p, \exists u_q \in q, u_p \xrightarrow[\mathcal{C}]{\star} u_q$. It follows from proposition 5 that $\forall u_p \in p, \exists u_q \in q, \mathcal{C} \models u_p \preceq u_q$. So it yields that $\mathcal{C} \models p \preceq q$.
- If $\mathcal{C} \models p \preceq q$ then $\forall u_p \in p, \mathcal{C} \models u_p \preceq q$. From proposition 6 we can say that $\forall u_p \in p, \exists u_q \in q, \mathcal{C} \models u_p \preceq u_q$. It follows from the same proposition that $\forall u_p \in p, \exists u_q \in q, u_p \xrightarrow[\mathcal{C}]{\star} u_q$ i.e $p \subseteq \mathit{ancestor}_{\mathcal{C}}(q)$.

□

Proof of lemma 5 : In [4] the authors give a decision algorithm for the inclusion of two regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$, given by two automata $\mathcal{A}_1$ and $\mathcal{A}_2$. This algorithm is in PSPACE in the size of the automata. Moreover, we can construct in linear time in $|p|$ a (non deterministic) automaton $\mathcal{A}_p$ which recognizes p (see for instance the Gluskov's algorithm [16]), and in polynomial time in $|q| + |\mathcal{C}|$ an automaton $\mathcal{A}_{\mathcal{C}}^q$ which recognizes $\mathit{ancestor}_{\mathcal{C}}(q)$.

□

Proof of lemma 6 : Inclusion problem of two regular languages, given by regular expressions p and q is PSPACE-hard [15]. Let us consider the set $\mathcal{C} = \{q \preceq \$\}$ where \$ does not appear in q. In this case, $ancestor_{\mathcal{C}}(q) = q$ and $p \subseteq q$ is equivalent to $p \subseteq ancestor_{\mathcal{C}}(\$)$. So deciding $p \subseteq q$ is equivalent to decide $p \preceq \$$. □

Proof of proposition 7 : $\mathcal{C} \models u \preceq q$ iff $u \in ancestor_{\mathcal{C}}(q)$. We build an automaton $\mathcal{A}^q_{\mathcal{C}}$ recognizing $ancestor_{\mathcal{C}}(q)$ in PTIME and we test the membership of u in $ancestor_{\mathcal{C}}(q)$ using this automaton. □

Proof of proposition 8 : We shall prove first that 1 implies 2 : Let $u_i \equiv v_i$ be any word equality constraint of $\mathcal{C}$. Then $u_i \equiv_{\mathcal{C}} v_i$ and $\mathrm{acc}_{G_{\mathcal{C}}}(u_i) = [u_i]_{\mathcal{C}} = [v_i]_{\mathcal{C}} = \mathrm{acc}_{G_{\mathcal{C}}}(v_i)$. Then $G_{\mathcal{C}} \models u_i \equiv v_i$ and it follows that $G_{\mathcal{C}} \models \mathcal{C}$: if $\mathcal{C} \models u \equiv v$ for some paths u and v, then $G_{\mathcal{C}} \models u \equiv v$.
Clearly, 2 implies 3.
It remains to prove that 3 implies 1 : let u and v be two paths such that $u \equiv_{\mathcal{C}} v$. Let us consider now any datum D such that $D \models \mathcal{C}$ and let us denote by $\equiv_D$ the equivalence relation defined over $A^\star$ by $w \equiv_D w'$ if and only if $\mathrm{acc}_D(w) = \mathrm{acc}_D(w')$. Clearly, $\equiv_D$ is a right congruence, and, since $D \models \mathcal{C}$, $\equiv_D$ contains $\equiv_{\mathcal{C}}$ which is the smallest congruence which contains the relation $\mathcal{C}$. Then $u \equiv_{\mathcal{C}} v$ implies that $u \equiv_D v$ then $\mathcal{C} \models u \equiv v$. □

Proof of proposition 9 : Clearly, it is sufficient to prove 3 implies 1. Let u and v be two paths such that $\mathcal{C} \models u \equiv v$ on the family of complete deterministic data. Then $G_{\mathcal{C}} \models u \equiv v$, since $G_{\mathcal{C}} \models \mathcal{C}$ and it is complete and deterministic. Now, from proposition 8, we obtain $\mathcal{C} \models u \equiv v$ □

Proof of proposition 10 : Clearly, if $f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v)$, then $u \equiv_{\mathcal{C}} v$ and, from proposition 8, $C \models u \equiv v$. Conversely, if $C \models u \equiv v$, then $u \equiv_{\mathcal{C}} v$. If we consider the equivalence relation $\equiv_f$ defined over $A^\star$ by $u \equiv_f v$ if and only if $f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v)$, then $\equiv_f$ clearly contains $\mathcal{C}$. It remains to prove that $\equiv_f$ is closed by right congruence : let u and v be two paths such that $f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v) = ([w_1], w_2)$ and let us consider $f_{\mathcal{C}}(ux)$ and $f_{\mathcal{C}}(vx)$ for some letter x. Then $f_{\mathcal{C}}(ux) = ([w_1], w_2x)$ or $f_{\mathcal{C}}(ux) = ([w'], \varepsilon)$ with $w' \equiv_{\mathcal{C}} w_1w_2x \equiv_{\mathcal{C}} ux$, and $f_{\mathcal{C}}(vx) = ([w_1], w_2x)$ or $f_{\mathcal{C}}(vx) = ([w'], \varepsilon)$ with $w' \equiv_{\mathcal{C}} w_1w_2x \equiv_{\mathcal{C}} vx$. Clearly, it is sufficient to prove that, if $f_{\mathcal{C}}(ux) = ([w'], \varepsilon)$ then $f_{\mathcal{C}}(vx) = ([w'], \varepsilon)$. if $f_{\mathcal{C}}(ux) = ([w'], \varepsilon)$ with $w' \equiv_{\mathcal{C}} w_1w_2x \equiv_{\mathcal{C}} ux$. Since, $f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v)$, it follows that $u \equiv_{\mathcal{C}} v$, then $ux \equiv_{\mathcal{C}} vx$. It follows that $vx \equiv w'$, and then $\mathrm{acc}_{G^f_{\mathcal{C}}}(vx) = [w']$ and $f_{\mathcal{C}}(vx) = ([w'], \varepsilon) = f_{\mathcal{C}}(ux)$. □

Proof of corollary 3 : Clearly, if a regular path expression q describes a finite language, then $f_{\mathcal{C}}(q)$ is finite. Then if p has the strong boundedness property w.r.t. $\mathcal{C}$, there exists a regular path expression q, describing a finite language such that $f_{\mathcal{C}}(p) = f_{\mathcal{C}}(q)$, then $f_{\mathcal{C}}(p)$ is finite. Conversely, if $f_{\mathcal{C}}(p)$ is finite, then the language $L = \{v \in A^\star \mid \exists w \in W, ([w], v) \in f_{\mathcal{C}}(p)\}$ is finite and $\mathcal{C} \models p \equiv q$ for some expression q describing a language included in WL which is finite. □

Sketch of proof of lemma 7 : We will first show that, for every node n of $G^f_{\mathcal{C}}$, we can compute an automaton $\mathcal{A}_{\mathcal{C},p}(n)$ in PTIME in the sum of the size of $G^f_{\mathcal{C}}$ and the size of p such that the language recognized by $\mathcal{A}_{\mathcal{C},p}(n)$ is the language $\{w \in A^\star \mid (n, w) \in f_{\mathcal{C}}(p)\}$: let us consider an automaton $\mathcal{A}_p$ which recognizes the language described by p, where all states are accessible and co-accessible, this can be done in time $|p|$.
Let us complete the graph $G^f_{\mathcal{C}}$ with a hole node $\perp$, and with transitions $(n, x, \perp)$ for each node n and each letter x such that there is no transition labelled by x from n in $G^f_{\mathcal{C}}$, this can be done in size of $G^f_{\mathcal{C}}$. Let us consider now the Cartesian product of this complete graph and automaton $\mathcal{A}_p$: in this graph, the transitions are in the form $((n_1, s_1), x, (n_2, s_2))$ where n_1 and n_2 are nodes of $G^f_{\mathcal{C}}$ or equal to $\perp$, s_1 and s_2 are states of automaton $\mathcal{A}_p$ and x is a letter. Let us remove all transitions $((n_1, s_1), x, (n_2, s_2))$ where n_2 is a node of $G^f_{\mathcal{C}}$ (i.e. not equal to $\perp$), then $\mathcal{A}_{\mathcal{C},p}(n)$ can be obtained from this graph setting the set of initial states to $\{n\} \times S$ where S is the set of states of automaton $\mathcal{A}_p$ and final states to $\{\perp\} \times F$ where F is the set of final states of automaton $\mathcal{A}_p$. The construction can be done in PTIME in the sum of the sizes of $\mathcal{A}_p$ and $G^f_{\mathcal{C}}$.

Now, to answer the question whether $f_{\mathcal{C}}(p)$ is finite, we can check for every node n of $G^f_{\mathcal{C}}$ if automaton $\mathcal{A}_{\mathcal{C},p}(n)$ recognizes a finite language, this leads to a PTIME algorithm in the sum of the sizes of $\mathcal{A}_p$ and $G^f_{\mathcal{C}}$

At last, in order to compare $f_{\mathcal{C}}(p)$ and $f_{\mathcal{C}}(q)$ for some regular path expressions p and q, we can check if, for each node n of $G^f_{\mathcal{C}}$, the automata $\mathcal{A}_{\mathcal{C},p}(n)$ and $\mathcal{A}_{\mathcal{C},q}(n)$ are equivalent. This can be made in PSPACE in the sum of the size of $\mathcal{A}_{\mathcal{C},p}(n)$ and $\mathcal{A}_{\mathcal{C},q}(n)$. □

Proof of proposition 12 : Let $\mathcal{C}$ be a non empty set of path equalities over an alphabet A. We can compute $G^f_{\mathcal{C}} =< N, r, T_G >$. N is set of equivalent classes. If $[u] \in N$, $\mathit{find}([u])$ will denote a representant of the class $[u]$. From this graph, we can define $\tau_{\mathcal{C}} =< A, N \cup \{\$\}, r, N \cup \{\$\}, T, e >$ where A is the input and the output alphabet, $N \cup \{\$\}$ is set of states ($\$$ is a new node i.e $\$ \notin N$) where r is the initial satate. All the states are finals. T is set of transition defined by $T = \{([u], x, \epsilon, [u.x]) \mid [u.x] \in N\} \cup \{([u], x, \mathit{find}([u]).x, \$) \mid [u.x] \notin N\} \cup \{(\$, x, x, \$) \mid x \in A\}$ and e is a output function from the final states $(N \cup \{\$\})$ to A^* defined by: $e([u]) = find([u])$ and $e(\$) = \epsilon$.
It is easy to see that for all path u, $f_c(u) = ([u_1], u_2)$ iff $\tau_c(u) = \mathit{find}([u_1]).u_2$. If q is path expression then $t_{\mathcal{C}}(q) = \bigcup_{u \in q} t_{\mathcal{C}}(u) = \bigcup_{\substack{u \in q \\ f_{\mathcal{C}}(u) = ([u_1], u_2)}} \mathit{find}([u_1]).u_2$. As $f_{\mathcal{C}}(\tau_{\mathcal{C}}(q)) = f_{\mathcal{C}}(q)$ it follows from the corollary 2 that for all q such that $L(q) = \tau_{\mathcal{C}}(L(p))$, $\mathcal{C} \models p \equiv q$. It follows from the corollary 3 that $\tau_{\mathcal{C}}(L(p))$ is finite if and only if p has the strong boundedness property w.r.t. $\mathcal{C}$ As we are able to compute $G^f_{\mathcal{C}}$ in quasi linear time, we can compute $\tau_{\mathcal{C}}$ in quasi linear time. □