

Path Constraints in semistructured data

Y. André, A.C. Caron, D. Debarbieux, Y. Roos, and S. Tison

{andre, caronc, debarbie, yroos, tison}@lifl.fr

Mostrare project, RU INRIA Futurs

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 multi rooted edge-labeled directed graphs, and path inclusion constraints on these graphs. A path inclusion constraint $p \preceq q$ is satisfied by a semistructured data if any node reached by the regular query p is also reached by the regular query q .

In this paper, two problems are mainly studied: the implication problem and the problem of the existence of a finite exact model.

- We give a new decision algorithm for the implication problem of a constraint $p \preceq q$ by a set of bounded path constraints $p_i \preceq u_i$ where p, q , and the p_i 's are regular path expressions and the u_i 's are words, improving in this particular case, the more general algorithms of S. Abiteboul and V. Vianu, and N. Alechina et al. In the case of a set of word equalities $u_i \equiv v_i$, we provide 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.. We prove that, in this case, implication for non-deterministic models is equivalent to implication for (complete) deterministic ones.
- We introduce the notion of exact model: an exact model of a set of path constraints \mathcal{C} satisfies the constraint $p \preceq q$ if and only if this constraint is implied by \mathcal{C} . We prove that any set of constraints has an exact model and we give a decidable characterization of data which are exact models of bounded path inclusion constraints sets.

Key words: Semistructured data graph; path inclusion; prefix rewriting; (finite) exact model

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. A presentation of this model and an overview of works done in this context can be found in [1]. In order to treat composition of queries, it can be useful to consider multi rooted data graphs, instead of single rooted ones. In this paper,

we consider the multi rooted case, and the results can often be easily adapted to the single rooted case.

Let us consider the data graph figure 1 which represents a journal. This journal contains articles and each article is written by one or two authors.

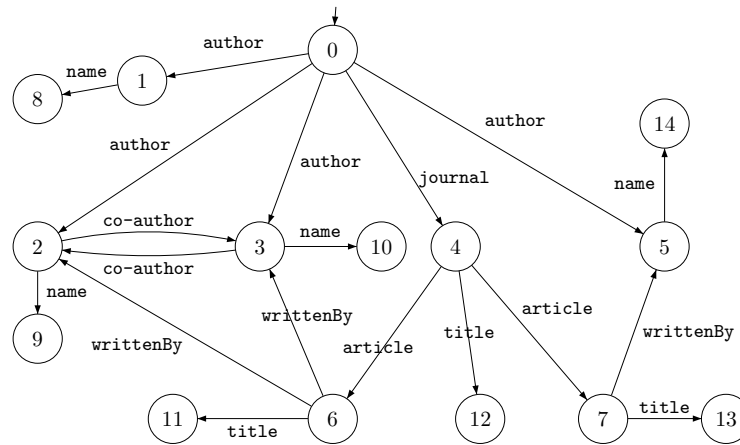


Fig. 1. Example of a semistructured data

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 to be non-deterministic. In the deterministic case, the outgoing edges of a given node must have distinct labels. Moreover, if any node has at least one outgoing edge for each label of the alphabet, the graph is said complete. Note that XML documents are usually non deterministic and non complete.

Path : Basic query mechanisms proposed for semistructured data are based on path expressions. UnQL [14] the language defined by Buneman et al for querying data organized as edge-labeled graph or Lorel [2] defined by Abiteboul et al as part of the Lore project are examples of such query languages. Propositions for querying XML-data such as XML-QL [26], XQL [33], Quilt or the most recent one XQuery [12] all use the XPath language [10] to select nodes in the documents. The large number of features of the full XPath language makes it unwieldy for theoretical study and so fragments of XPath, as Tree pattern queries (or twigs [7]) for example, are usually investigated.

In this paper, we study regular path expressions (or regular queries) which are regular expressions 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(s) by a path labeled by any word u of q .

For example, `author`, `author.name`, `journal.article.title` are paths of the data graph 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 (or to approximate) path queries, it can be useful to use structural information about the data graph. Some of these are called path constraints since they give restrictions on the 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 [16], [30], [11] 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 . When p and q describe finite languages, these constraints are called *finite path inclusions*.

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 data graph 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 words. We call *word equality* this kind of constraint. Similarly, the constraint $u \preceq v$ is called *word inclusion*. More generally, when p is a regular path query and u is a word, the constraint $p \preceq u$ is called *bounded path inclusion*.

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 data 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 $\mathcal{C} \models p \preceq q$.

Let us note that sometimes the data are seen as single rooted graphs (see [1], [5], [15]) and sometimes they are seen as multi rooted graphs (see for instance [32]). In this paper we work with multi rooted graphs but in many cases, the results and the proof techniques are similar to the single rooted case. For instance we prove that the implication problem with single rooted graphs is equivalent to the implication with multi rooted graphs as soon as each query of the set \mathcal{C} is ε -free. As in the single rooted case [3,5] the implication problem is EXPTIME. Moreover 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 words. We prove that this decision problem is PSPACE complete .

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 [15], the authors give a cubic decision algorithm in the case of the implication for deterministic models (with more general forward constraints). Here, we build a quasi-linear algorithm and we prove that, in this very particular case of word equalities, the implication problem for non-deterministic models is equivalent to the implication problem for (complete and) deterministic ones.

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 $\text{author.co-author} \equiv \text{author.co-author.co-author}$, the regular query $\text{author.co-author}^+$ has the strong boundedness property. Since it is easier to answer 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, it is possible to approximate q with a finite query f since the answer to f is a superset of the answer to q . We extend the result of [8] giving an algorithm which computes such a finite query f when it exists. Moreover, this result applies on the strong boundedness property.

Exact (finite) models: In this paper, we are mainly interested in models of a set of constraints. Of course, any set of path inclusions has a model (e.g. the complete model reduced to one root which models any path inclusion); here we focus on the notion of exact model: a model of \mathcal{C} is said to be exact if it models only constraints satisfied conjointly by every model of \mathcal{C} . In other words, a data D_c is an exact model of \mathcal{C} if D_c is a model of $p \preceq q$ if and only if \mathcal{C} implies $p \preceq q$. We prove that the existence of an exact model is ensured for any set of constraints. So a natural question arises: is there a finite exact model? Having effectively a finite exact model of the set of constraints provides an effective manner of checking whether a (regular) path inclusion is implied by the set of constraints. It ensures also that every query is strongly bounded.

First, we consider the case of bounded path inclusions ($p \preceq u$ where p is a regular query and u is a word). In this case, we propose a decidable characterization of sets \mathcal{C} which have a finite exact model. Moreover, we give an effective way of computing such a model when it exists.

Secondly we consider only word equalities ($u \equiv v$ where u and v are words). In this case, we give a more efficient algorithm for deciding existence of a finite exact model and for constructing such a model.

2 Path constraints

In this section we give the framework of the paper: we define semistructured data graphs, regular queries and path inclusion constraints. Then we introduce implication problem and the notion of exact model.

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 multi rooted data graph is a triple $D = \langle N, \text{Root}, T \rangle$ where*

- N is a set of nodes,
- Root , the set of roots, is a subset of N
- T , the set of transitions, is a subset of $N \times A \times N$.

and N, T are enumerable sets.

As N and T are enumerable sets, any node of a data graph may have an infinite number of ingoing edges and an infinite number of outgoing edges. If the set of nodes N is finite then the data graph D is said to be finite. If \mathbf{Root} is reduced to a singleton then D is said to be **single rooted**.

Definition 2. A data graph $\langle N, \mathbf{Root}, T \rangle$ is said

- deterministic if it is single rooted and for all n in N , for all a in A , there is at most one transition (n, a, n') in T .
- complete if, for all n in N , for all a in A , there is at least one transition (n, a, n') in T .

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

Definition 3.

- Given D a data graph, u a word of A^* , let $\text{result}_D(u)$ be defined by:
 1. if $u = \varepsilon$, then $\text{result}_D(u) = \mathbf{Root}$
 2. if $u = u'a$ ($u' \in A^*$ and $a \in A$) then

$$\text{result}_D(u) = \{n \in N \mid \exists n' \in \text{result}_D(u'), (n', a, n) \in T\}$$

- A regular query p is a regular expression over A .
- $L(p)$ denotes the regular language described by p . If ε does not belong to $L(p)$ then p is said ε -free.
- The result of a regular query p over a data graph D is the set

$$\text{result}_D(p) = \bigcup_{u \in L(p)} \text{result}_D(u)$$

In the following, we shall only consider that any node of D is reachable from some root. Now, we formally define path inclusions.

Definition 4.

- 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 $(p \preceq q) \wedge (q \preceq p)$.
- A data graph D satisfies (is a model of) a path inclusion $p \preceq q$, denoted $D \models p \preceq q$, if the set of nodes $\text{result}_D(p)$ is included in $\text{result}_D(q)$.
- D satisfies a set \mathcal{C} of path inclusions, denoted $D \models \mathcal{C}$, if D satisfies each path inclusion of \mathcal{C} .
- Two sets of path inclusions \mathcal{C} and \mathcal{C}' are said to be equivalent if for each data graph D , $D \models \mathcal{C}$ if and only if $D \models \mathcal{C}'$.

From now, we only consider **finite** sets of path inclusions .

2.1 Implication Problem

In this section, we introduce the implication problem and the equivalence problem and prove they are decidable in EXPTIME.

First, let us define the implication of a constraint $p \preceq q$ by a set of constraints \mathcal{C} .

Definition 5. *A set of path inclusions \mathcal{C} implies a path inclusion $p \preceq q$ (denoted by $\mathcal{C} \models p \preceq q$) if for each data graph D*

$$D \models \mathcal{C} \Rightarrow D \models p \preceq q$$

Definition 6.

- *The implication problem is to decide given a set \mathcal{C} of path inclusions, and two regular queries p, q , whether $\mathcal{C} \models p \preceq q$.*
- *The equivalence problem is to decide given a set \mathcal{C} of path inclusions, and two regular queries p, q , whether $\mathcal{C} \models p \equiv q$.*
- *The size of $p \preceq q$ is the sum $|p| + |q|$, where $|p|$ is the length of the regular query p , and the size of the set of constraints \mathcal{C} , denoted by $|\mathcal{C}|$, is the sum of the sizes of its constraints.*

In [3], S. Abiteboul and V. Vianu have proved the following result with single rooted data graphs in which any node has a finite number of outgoing edges. Their proof can easily be adapted to our kind of data graphs (see appendix):

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

This proposition and its proof provide a way of deciding the implication problem. Indeed, if a set of path inclusions \mathcal{C} does not imply a path inclusion $p_0 \preceq q_0$ there exists a finite data graph D_f s.t. D_f models \mathcal{C} but D_f does not model $p_0 \preceq q_0$. Moreover, we know from the proof of proposition 1 that we can bound the size of D_f exponentially w.r.t. $|\mathcal{C}| + |p_0| + |q_0|$. So, in order to decide whether $\mathcal{C} \models p_0 \preceq q_0$ it is sufficient to check $D \models p_0 \preceq q_0$ for D whose size is bounded exponentially w.r.t. $|\mathcal{C}| + |p_0| + |q_0|$. This proves that the implication problem is decidable in co-NEXPTIME.

An other way of deciding the implication problem is to express it in Propositional Dynamic Logics with converse (converse-PDL), following N. Alechina, S. Demri and M. de Rijke in [6]. Indeed, let us associate with the constraint $C = p \preceq q$ the converse-PDL formula $\Phi_C = \neg root \vee [p](q^{-1})root$ where $root$ is a propositional variable; intuitively, D_f models C if and only if Φ_C is valid in the corresponding structure; then the constraint C_0 is implied by the constraints $\{C_1, \dots, C_n\}$ if and only if the set of axioms $\{\Phi_{C_1}, \dots, \Phi_{C_n}\}$ implies the formula Φ_{C_0} ; this can be reformulated in terms of satisfiability of a converse-PDL formula [18,31] and therefore decided in EXPTIME [27]:

Theorem 1. *The implication problem is decidable in EXPTIME.*

Remark 1. The finite model property of converse-PDL yields another proof of proposition 1.

The following proposition and remark compare the single rooted model and the multi rooted model. We denote by \models^1 the implication in the single rooted model (i.e. $\mathcal{C} \models^1 p \preceq q$ if for any finite **single rooted** data graph D , $D \models \mathcal{C}$ implies $D \models p \preceq q$). See [3] or [9] for more details.

Proposition 2. *Let $\mathcal{C} = \{p_i \preceq q_i, 1 \leq i \leq n\}$ be a set of ε -free constraints. For any regular queries p_0 and q_0 , we get*

$$\mathcal{C} \models p_0 \preceq q_0 \text{ if and only if } \mathcal{C} \models^1 p_0 \preceq q_0$$

Proof. Obviously $\mathcal{C} \models p_0 \preceq q_0$ implies that $\mathcal{C} \models^1 p_0 \preceq q_0$. We have to prove the converse. Let $D = \langle N, R, T \rangle$ be a multi rooted document model of \mathcal{C} . Let us consider the single rooted model $D_r = \langle N \cup \{r\}, \{r\}, T \cup \{(r, x, n) \mid \exists r' \in R \wedge (r', x, n) \in T\} \rangle$ (this construction corresponds to the standardization in the automata theory). We remove from N the non accessible nodes. It can be easily proved by induction that

$$\forall u \in A^+ \text{ result}_D(u) = \text{result}_{D_r}(u) \quad (1)$$

For a query q , q^ε denotes a query s.t. $L(q^\varepsilon) = L(q) \setminus \{\varepsilon\}$. It follows from 1 that for any queries p and q , $D \models p^\varepsilon \preceq q^\varepsilon$ if and only if $D_r \models p^\varepsilon \preceq q^\varepsilon$. As any query of \mathcal{C} is ε -free, D models \mathcal{C} if and only if D_r models \mathcal{C} .

Let us suppose $\mathcal{C} \models^1 p_0 \preceq q_0$ and D models \mathcal{C} . Then D_r models \mathcal{C} and then $D_r \models p_0 \preceq q_0$. As the root of D_r is the only node reached by ε and is only reached by ε , $D_r \models p_0 \preceq q_0$ implies $D_r \models p_0^\varepsilon \preceq q_0^\varepsilon$, and so $D \models p_0^\varepsilon \preceq q_0^\varepsilon$. Now, if $p_0 = p_0^\varepsilon$, we get $D \models p_0 \preceq q_0^\varepsilon$ and then $D \models p_0 \preceq q_0$. Otherwise, ε belongs to $L(p_0)$; then, the root of D_r is reached by p_0 and then by q_0 . As the root of D_r has no ingoing edge, ε belongs to $L(q_0)$ too. So $D \models p_0^\varepsilon \preceq q_0^\varepsilon$ implies $D \models p_0 \preceq q_0$. So, we have proved that $\mathcal{C} \models^1 p_0 \preceq q_0$ implies that $\mathcal{C} \models p_0 \preceq q_0$.



Fig. 2. Differences between \models and \models^1

Remark 2. Proposition 2 does not hold when \mathcal{C} contains queries with the empty word. Indeed for any word u , for any single rooted document D , $D \models u \preceq \varepsilon$

means that either $\text{result}_D(u)$ is empty or $\text{result}_D(u)$ is the root of the document. It follows that $\{a \preceq \varepsilon\} \models^1 a \equiv aa$ whereas $\{a \preceq \varepsilon\} \not\models a \equiv aa$ as shown in figure 2.

If ε appears in the left hand-side of a path inclusion, the proposition 2 does not hold either. For instance $\{\varepsilon \preceq a + b\} \models^1 \varepsilon \preceq aa + bb$ since any single rooted data graphs D which satisfies $\{\varepsilon \preceq a + b\}$ satisfies either $\varepsilon \preceq a$ or $\varepsilon \preceq b$. However $\{\varepsilon \preceq a + b\} \not\models \varepsilon \preceq aa + bb$ (figure 2).

2.2 Models

In the previous subsection we have introduced the definition of path inclusions and the notion of model of a set of path inclusions. Of course, any set of path inclusions has a model: the complete model reduced to one root which models any path inclusion (figure 3). In this subsection we define exact models of path inclusions and prove that any set of path inclusions has an exact model.

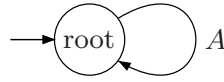


Fig. 3. Any set of path inclusions has a model

Definition 7. Let \mathcal{C} be a set of path inclusions. A data graph D_c is an exact model of \mathcal{C} if

$$D_c \models p \preceq q \text{ if and only if } \mathcal{C} \models p \preceq q$$

A well known property is that an implication problem can be reduced to an equivalence problem. Indeed, as $\text{result}_D(p) \cup \text{result}_D(q)$ is equal to $\text{result}_D(p+q)$, D models $p \preceq q$ if and only if D models $p + q \equiv q$ and then:

$$\mathcal{C} \models p \preceq q \text{ if and only if } \mathcal{C} \models p + q \equiv q \quad (2)$$

It follows that we could give the equivalent definition of an exact model:

Corollary 1. Let \mathcal{C} be a set of path inclusions and D_c be a document. D_c is an exact model of \mathcal{C} if and only if

$$D_c \models p \equiv q \text{ if and only if } \mathcal{C} \models p \equiv q$$

We are now able to prove:

Proposition 3. Any set \mathcal{C} of path inclusions has an exact model.

Proof. Let M_c be the countable set of all finite models of \mathcal{C} (we do not differentiate two isomorphic data graphs). An element d of M_c is of the form $\langle N_d, R_d, T_d \rangle$. Without loss of generality, we assume that the intersection $N_d \cap N_{d'}$ is empty if d and d' are different.

We now prove that $D = \langle \bigcup_{d \in M_c} N_d, \bigcup_{d \in M_c} R_d, \bigcup_{d \in M_c} T_d \rangle$ is an exact model of \mathcal{C} .

It is clear that

$$D \models p \preceq q \text{ if and only if } \forall d \in M_c \ d \models p \preceq q$$

So D is a model of \mathcal{C} .

Let p and q be two queries s.t. $\mathcal{C} \not\models p \preceq q$. We already know from proposition 1 that there exists a finite data graph D_f s.t. $D_f \models \mathcal{C}$ and $D_f \not\models p \preceq q$. By definition of M_c , D_f belongs to M_c and then D does not model $p \preceq q$.

Remark 3. It is already known from [24] that the proposition doesn't hold for single rooted model: $\{a \preceq \varepsilon, b \preceq \varepsilon\}$ has no single rooted exact model.

Let us consider now the ‘‘converse’’ problem: is every finite data graph D an exact model of a set of path inclusions? The answer is yes:

Proposition 4. *For every finite data graph D , there exists a (finite) set of finite path inclusions $\mathcal{C}(D)$ such that $\mathcal{C}(D) \models p \preceq q$ if and only if $D \models p \preceq q$*

Proof. Let $D = \langle N, R, T \rangle$ and $S(D) = \{\text{result}_D(u) \mid u \in A^*\}$. $S(D)$ is included in 2^N and can be exponentially bigger than N .

For any set s of $S(D)$, $\text{lex}(s)$ will denote a word of $\{u \mid \text{result}_D(u) = s\}$. For ensuring the correction of the construction, the only restriction is that $\text{lex}(R)$ must be ε . Here, we will take the shortest and the first word in alphabetical order, which will provide a bound of the size of the constraints.

So let us define $\mathcal{C}(D)$ as $\mathcal{C}(D) = \{\text{lex}(s)x \equiv \text{lex}(s') \mid s' = \{n' \mid \exists n \in s \wedge (n, x, n') \in T\}\} \cup \{\text{lex}(s_1) \preceq \text{lex}(s'_1) + \dots + \text{lex}(s'_k) \mid s_1 \subseteq \bigcup_{1 \leq i \leq k} s'_i, k \leq N\}$.

(By $\text{lex}(s)x \equiv \text{lex}(s')$, we mean that we add $\text{lex}(s)x \preceq \text{lex}(s')$ and $\text{lex}(s') \preceq \text{lex}(s)x$.)

If $\mathcal{C}(D) \models p \preceq q$ then $D \models p \preceq q$ holds since, by construction, D is a model of $\mathcal{C}(D)$.

Conversely, we have to prove that D is exact. As $\text{lex}(R)$ is ε , we can prove by induction that $\mathcal{C}(D) \models u \equiv \text{lex}(\text{result}_D(u))$ and then

$$\mathcal{C}(D) \models q \equiv \bigoplus_{u \in L(q)} \text{lex}(\text{result}_D(u))$$

Now, if $D \models p \preceq q$,

$$\forall u \in L(p), \text{result}_D(u) \subseteq \bigcup_{u \in L(q)} (\text{result}_D(u))$$

As $\text{result}_D(u)$ contains at most N states, it follows that, for every u in $L(p)$ we can find u_1, \dots, u_k in $L(q)$ with $k \leq N$ such that:

$$\text{result}_D(u) \subseteq \bigcup_{1 \leq i \leq k} (\text{result}_D(u_i))$$

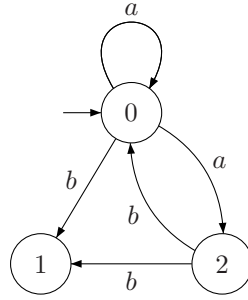
Therefore,

$$\{\text{lex}(\text{result}_D(u)) \preceq \text{lex}(\text{result}_D(u_1)) + \dots + \text{lex}(\text{result}_D(u_k))\}$$

belongs to $\mathcal{C}(D)$. So, for every u in $L(p)$, $\mathcal{C}(D) \models u \preceq q$ and then $\mathcal{C}(D) \models p \preceq q$: D is an exact model of $\mathcal{C}(D)$.

Moreover, an easy analysis shows that the cardinal of $\mathcal{C}(D)$ is bounded by 2^{N^2+2N} , and so its size is bounded by $2 \times N^2 \times (2^{N^2+2N})$, as we can choose for each $\text{lex}(s)$ a word of length at most $N - 1$. This provides an *EXPTIME* algorithm for the construction of $\mathcal{C}(D)$.

Example 1. Let us consider the following data graph D with one root named 0:



$S(D) = \{\{0\}, \{0, 2\}, \{0, 1\}, \{1\}, \emptyset\}$,
 $\text{lex}(\{0\}) = \varepsilon$, $\text{lex}(\{0, 2\}) = a$, $\text{lex}(\{0, 1\}) = ab$, $\text{lex}(\{1\}) = b$, $\text{lex}(\emptyset) = ba$ and
 after removing trivial equivalences,
 $\mathcal{C}(D) = \{a \equiv aa, aba \equiv a, abb \equiv b, bb \equiv ba,$
 $ba \preceq \varepsilon, ba \preceq a, ba \preceq ab, ba \preceq b$
 $\varepsilon \preceq a, \varepsilon \preceq ab, b \preceq ab, ab \preceq \varepsilon + b\}$.

3 Bounded Path Inclusions

From now on we will consider exclusively the case of a finite set of inclusions of the form $p \preceq u$ where p is a regular expression and u is a word: we shall call such path inclusions *bounded path inclusions*. This kind of constraints have been introduced in [8]. In this case, following and slightly generalizing [3] and [9], we associate with a set \mathcal{C} of bounded path inclusions a prefix rewriting system such that there is a prefix rewriting from u to v , if and only if $\mathcal{C} \models u \preceq v$. This technique provides also an uniform way for deciding implication of constraints or properties such as (strong) boundedness.

3.1 Prefix rewriting

First, let us associate with a set of bounded constraints a binary relation on A^* as follows:

Definition 8. Let $\mathcal{C} = \{p_i \preceq u_i, 1 \leq i \leq n\}$ be a finite set of bounded path inclusions over an alphabet A . We consider the relation on words defined by $u \xrightarrow{\mathcal{C}} v$ if and only if there exists i such that u belongs to $L(p_i)$ and $v = u_i$. By extension, we denote also $\xrightarrow{\mathcal{C}}$ its right congruence closure. Then $\xrightarrow{\mathcal{C}^*}$ denotes the reflexive, transitive closure of $\xrightarrow{\mathcal{C}}$.

This relation is a prefix rewriting relation as defined in [19] based on an infinite rewrite system. As proved by the following proposition, the relation simulates the constraints on words:

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

The proof uses an exact model of \mathcal{C} which is close to the one defined in [3]:

Definition 9. With a set of bounded path constraints \mathcal{C} , we associate an infinite data graph $I_{\mathcal{C}} = \langle N, R, T \rangle$ defined by:

- $N = \{u \mid u \in A^*\}$
- $R = \{u \mid u \xrightarrow{\mathcal{C}^*} \varepsilon\}$
- $T = \{(u, x, v) \mid v \xrightarrow{\mathcal{C}^*} ux\}$

Example 2. Let $\mathcal{C} = \{ab^* \preceq ba, b^+ \preceq a, a(aa)^*b \preceq a, b \preceq \varepsilon\}$. Figure 4 shows a finite part of the infinite data graph $I_{\mathcal{C}}$ of definition 9. It follows from $b \xrightarrow{\mathcal{C}^*} \varepsilon$ that b is one of the root. It follows from $bb \xrightarrow{\mathcal{C}^*} a \xrightarrow{\mathcal{C}^*} ba \xrightarrow{\mathcal{C}^*} aa$ that $(a, a, bb) \in T$. It follows from $b \xrightarrow{\mathcal{C}^*} a \xrightarrow{\mathcal{C}^*} ba$ that $(b, a, b) \in T$.

Then, we get immediately by induction on the length of u :

Lemma 1. $\text{result}_{I_{\mathcal{C}}}(u) = \{v \mid v \xrightarrow{\mathcal{C}^*} u\}$

Now, let us suppose $I_{\mathcal{C}} \models u \preceq v$, i.e. $\text{result}_{I_{\mathcal{C}}}(u) \subseteq \text{result}_{I_{\mathcal{C}}}(v)$; by the preceding lemma u belongs to $\text{result}_{I_{\mathcal{C}}}(u)$, so u belongs to $\text{result}_{I_{\mathcal{C}}}(v)$; once again by the preceding lemma $u \xrightarrow{\mathcal{C}^*} v$. Conversely, let us suppose now $u \xrightarrow{\mathcal{C}^*} v$. So for any w , if $w \xrightarrow{\mathcal{C}^*} u$ then $w \xrightarrow{\mathcal{C}^*} v$; by the preceding lemma $\text{result}_{I_{\mathcal{C}}}(u) \subseteq \text{result}_{I_{\mathcal{C}}}(v)$, that is to say $I_{\mathcal{C}} \models u \preceq v$:

Lemma 2. $I_{\mathcal{C}} \models u \preceq v$ if and only if $u \xrightarrow{\mathcal{C}^*} v$

We obtain now

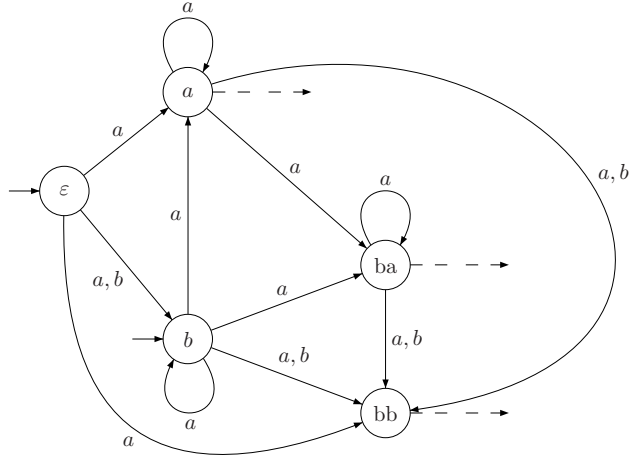


Fig. 4. A part of the model I_C

Lemma 3. $I_C \models u \preceq v$ if and only if $\mathcal{C} \models u \preceq v$

Proof. First, it is easy to get that $I_C \models \mathcal{C}$ and so that if $\mathcal{C} \models u \preceq v$, then $I_C \models u \preceq v$. Now, let us suppose $I_C \models u \preceq v$ that is to say $u \xrightarrow{\mathcal{C}^*} v$. Let $\preceq_{\mathcal{C}}$ defined by $u \preceq_{\mathcal{C}} v$ if $\mathcal{C} \models u \preceq v$. The relation $\preceq_{\mathcal{C}}$ contains $\xrightarrow{\mathcal{C}}$, is transitive and closed by right-congruence: it contains $\xrightarrow{\mathcal{C}^*}$; so $u \xrightarrow{\mathcal{C}^*} v$ implies $u \preceq_{\mathcal{C}} v$ i.e. $\mathcal{C} \models u \preceq v$.

By the two last lemmas, we obtain proposition 5.

Lemma 4. If $I_C \models u \preceq q$, there is some word v in q such that $I_C \models u \preceq v$

Proof. Indeed, if $I_C \models u \preceq q$, $\text{result}_{I_C}(u)$ is included in $\text{result}_{I_C}(q)$; in particular, the state u belongs to $\text{result}_{I_C}(q)$, i.e. there is some v in $L(q)$ such that u belongs to $\text{result}_{I_C}(v)$; then, by lemma 1, $\text{result}_{I_C}(u)$ is included in $\text{result}_{I_C}(v)$, i.e. $I_C \models u \preceq v$.

The following property summarizes the preceding lemmas and holds only in the bounded case:

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 word v in $L(q)$ such that $\mathcal{C} \models u \preceq v$
- there is some word v in $L(q)$ such that $u \xrightarrow{\mathcal{C}^*} v$

Proof. If $\mathcal{C} \models u \preceq q$, as finite and infinite implication are equivalent and as $I_{\mathcal{C}} \models \mathcal{C}$ we get $I_{\mathcal{C}} \models u \preceq q$. So, there is some word v in $L(q)$ such that $I_{\mathcal{C}} \models u \preceq v$, and $\mathcal{C} \models u \preceq v$.

Let us now suppose $\mathcal{C} \models u \preceq v$, for some word v in $L(q)$: $I_{\mathcal{C}} \models u \preceq v$ and then $u \xrightarrow[\mathcal{C}]{}^* v$.

Lastly, let us suppose $u \xrightarrow[\mathcal{C}]{}^* v$, for some word v in $L(q)$: then $I_{\mathcal{C}} \models u \preceq v$. So, $\mathcal{C} \models u \preceq v$ and we get $\mathcal{C} \models u \preceq q$.

In some sense this means that the set of word inclusions we can deduce from a set of bounded path inclusions \mathcal{C} is the closure of \mathcal{C} by right congruence, reflexivity, transitivity.

Let us remark that this implies that $I_{\mathcal{C}}$ is an exact model of \mathcal{C} :

Corollary 2. $I_{\mathcal{C}} \models p \preceq q$ if and only if $\mathcal{C} \models p \preceq q$

In the case of bounded path inclusions, we can extend proposition 2 to constraints $p_i \preceq u_i$ with p_i not necessarily ε -free. Indeed we have already proved in [25] that if \mathcal{C} is a set of bounded path inclusions where no u_i is the empty word then $\mathcal{C} \models^1 u \preceq v$ if and only if $u \xrightarrow[\mathcal{C}]{}^* v$.

Corollary 3. Let $\mathcal{C} = \{p_i \preceq u_i, 1 \leq i \leq n\}$ where the u_i 's are non empty words. Let p be a query and u a word. Then $\mathcal{C} \models p \preceq u$ if and only if $\mathcal{C} \models^1 p \preceq u$

The following example shows that proposition 6 does not hold if we consider general path inclusions:

Example 3. The following data graph (figure 5) satisfies $a^+ \preceq (b + c)$ but does not satisfy $a \preceq b$ neither $a \preceq c$.

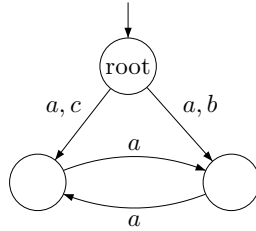


Fig. 5. $\mathcal{C} \models a^+ \preceq (b + c)$ but $\mathcal{C} \not\models a \preceq b$ and $\mathcal{C} \not\models a \preceq c$

Now, as constraints have been simulated by prefix rewriting, properties of constraints can be expressed by properties of the corresponding prefix rewriting. As not only the theory of prefix rewriting is decidable ([23]), but also the monadic theory of prefix rewriting is decidable ([20]), we get directly decidability of many

properties of constraints. The obtained results were mostly already known, but this approach provides an uniform way to get them. Moreover, it proves decidability for strong boundedness, which is new to our knowledge.

Definition 10. *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^* is bounded (by a), whereas the query ba^* is not.

The following theorem summarizes some results we get by using prefix rewriting:

Theorem 2. *Let $\mathcal{C} = \{p_i \preceq u_i, 1 \leq i \leq n\}$ be a set of bounded path inclusions, p and q two queries.*

1. *The implication problem for \mathcal{C} , p and q is decidable,*
2. *The equivalence problem for \mathcal{C} , p and q is decidable,*
3. *The boundedness property for \mathcal{C} and p is decidable,*
4. *The strong boundedness property for \mathcal{C} and p is decidable.*

Proof. Each property can be expressed by a formula of (monadic) prefix rewriting theory:

1. $\mathcal{C} \models p \preceq q$ is expressed by: $\forall u_p \in L(p) \exists u_q \in L(q) \mid u_p \xrightarrow[\mathcal{C}]{*} u_q$.
2. $\mathcal{C} \models p \equiv q =_{def} \mathcal{C} \models p \preceq q \wedge \mathcal{C} \models q \preceq p$.
3. $Bounded(p)$ is defined by

$$\exists F \mid finite(F) \wedge \forall u_p \in L(p) \exists u_f \in L(F) \mid u_p \xrightarrow[\mathcal{C}]{*} u_f$$
4. $StrongBounded(p)$ is defined by

$$\exists F \mid finite(F) \wedge \forall u_p \in L(p) \exists u_f \in L(F) \mid u_p \xrightarrow[\mathcal{C}]{*} u_f \wedge \forall u_f \in L(F) \exists u_p \in L(p) \mid u_f \xrightarrow[\mathcal{C}]{*} u_p$$

Formula 1 and formula 2 are obtained directly from lemma 4 and proposition 6. They are first-order formulae and this provides simple and rather efficient algorithms based on word automata for deciding them. E.g., this would lead to PSPACE algorithms for deciding these two formulae.

Formula 3 and formula 4 are then directly obtained from the definition of the boundedness property. These formulae are second-order ones; thus the “canonical” decision algorithms associated with them are based on infinite trees automata. Let us remark that this provides an effective way to compute one such F , when p is (strongly) bounded w.r.t. \mathcal{C} ; moreover, formulae can be enriched e.g. to exhibit a F minimal for inclusion.

3.2 Models

In proposition 3 and lemma 3, we have proved that any set of bounded path inclusions has an exact model, but the model we construct is always infinite.

In this section we give a characterization of the sets of bounded path inclusions which have a finite exact model. Then we propose an algorithm to decide if a set \mathcal{C} has this property.

In order to characterize the sets of constraints which have an exact finite model, we introduce the following equivalence relation:

Definition 11. *Let \mathcal{C} be a set of path inclusions. We will denote \equiv_c the relation on $A^* \times A^*$ defined by $u \equiv_c v$ if $\mathcal{C} \models u \equiv v$.*

Clearly, \equiv_c is an equivalence relation. We will denote by $[u]_c$ the equivalence class of the path u for the relation \equiv_c and we define the data graph $D_c = \langle N, R, T \rangle$ by:

- $N = \{[u]_c \mid u \in A^*\}$
- $R = \{[u]_c \mid \mathcal{C} \models u \preceq \varepsilon\}$
- $T = \{([u]_c, x, [v]_c) \mid \mathcal{C} \models v \preceq ux\}$

D_c is the quotient of I_c by the relation \equiv_c .

The following lemma, which characterizes $\text{result}_{D_c}(u)$ for any word u , can easily be proved by induction on the length of u :

Lemma 5. $\forall u \in A^* \text{ result}_{D_c}(u) = \{[v]_c \mid \mathcal{C} \models v \preceq u\}$

We are now able to prove:

Proposition 7. *For any set \mathcal{C} of bounded path inclusions, D_c is an exact model of \mathcal{C} .*

Proof. Let us first prove that D_c is a model of \mathcal{C} : let $(p \preceq u) \in \mathcal{C}$, v be a word in $L(p)$ and $[w]_c \in \text{result}_{D_c}(v)$. It follows from lemma 5 that $\mathcal{C} \models w \preceq v$. Since, by transitivity, we get $\mathcal{C} \models w \preceq u$ and using again lemma 5, we obtain $[w]_c \in \text{result}_{D_c}(u)$ and $D_c \models p \preceq u$.

Let us prove now that D_c is exact. Let us suppose that $D_c \models p \preceq q$ for some regular expressions p and q . Let u be a word of $L(p)$. Since $[u]_c$ is in $\text{result}_{D_c}(u)$ and $D_c \models p \preceq q$, there exists a word v of $L(q)$ such that $[u]_c$ is in $\text{result}_{D_c}(v)$. It follows from lemma 5 that $\mathcal{C} \models u \preceq v$ and then $\mathcal{C} \models p \preceq q$.

Clearly if \equiv_c is of finite index then, by construction, D_c is finite. Conversely, if there exists a finite exact model of \mathcal{C} then \equiv_c is of finite index. It follows:

Theorem 3. *Let \mathcal{C} be a set of bounded path inclusions. Then \mathcal{C} has a finite exact model if and only if \equiv_c is of finite index.*

Corollary 4. *A set of bounded path inclusion \mathcal{C} has an exact finite model if and only if D_c is finite.*

By using prefix rewriting, we get:

Proposition 8. *Let \mathcal{C} be a set of bounded path inclusions. Deciding whether $\equiv_{\mathcal{C}}$ is of finite index is PSPACE in the size of \mathcal{C} .*

Proof. The property can easily be expressed with a formula of prefix rewriting theory:

$$\exists F(\text{finite}(F) \wedge \forall u \in A^* \exists v \in L(F)(u \xrightarrow{\mathcal{C}}^* v \wedge v \xrightarrow{\mathcal{C}}^* u))$$

In order to get an EXPTIME decision algorithm, we shall *transform* this second-order formula into a first-order one. The idea is to transform the question *Does it exist a finite set F ?* into the question *Is the set X finite?* where X is defined as the set collecting each *minimal* word of each equivalence class of $\equiv_{\mathcal{C}}$.

Let us first define a notion of *minimal* word which can be expressed with prefix rewrite systems. We will use the reverse alphabetical order, that we denote by $<^R$, and which is defined by: $u <^R v$ if u is a suffix of v or $u = u'xw$ and $v = v'yw$ with $x < y$ for some words u', v', w and some letters x and y . Now, let S_1 be the prefix rewrite system defined over A^* by the set of rules $\{x \rightarrow \varepsilon \mid x \in A\}$ and S_2 be the prefix rewrite system defined over A^* by the set of rules $\{x \rightarrow y \mid x, y \in A, x < y\}$. Then we have $u <^R v$ if and only if $v \xrightarrow{S_1}^+ u$ or $\exists u' \exists v' \mid u \xrightarrow{S_1}^* u' \wedge v \xrightarrow{S_1}^* v' \wedge u' \xrightarrow{S_2} v'$.

We can define now X as the complement of the following set $Y = \{u \in A^* \mid \exists v \in A^*(u \xrightarrow{\mathcal{C}}^* v \wedge v \xrightarrow{\mathcal{C}}^* u \wedge v <^R u)\}$ and it follows that $\equiv_{\mathcal{C}}$ is of finite index if and only if the set Y is cofinite. Using only prefix rewrite systems, the set Y can be defined by:

$$Y = \{u \in A^* \mid \exists v \in A^*(u \xrightarrow{\mathcal{C}}^* v \wedge v \xrightarrow{\mathcal{C}}^* u \wedge (u \xrightarrow{S_1}^+ v \vee \exists u' \exists v' \mid v \xrightarrow{S_1}^* u' \wedge u \xrightarrow{S_1}^* v' \wedge u' \xrightarrow{S_2} v'))\}$$

Using prefix rewriting theory, it is possible to build an automaton which recognizes the set Y in polynomial time in the size of \mathcal{C} . Finally, testing whether Y is cofinite can be done in PSPACE in the size of an automaton recognizing Y , this decision problem being very close to the universality problem.

In proposition 4, we have proved that any finite data graph is an exact model of a finite set of finite path inclusions. A natural question arises: is any finite data graph an exact model of a set of bounded path inclusions? The answer is no as shown in figure 5. Nevertheless we can characterize data graphs which have this property.

Proposition 9. *Let D be a finite data graph. We can decide in EXPTIME whether there exists a set of bounded path inclusions \mathcal{C} s.t. D is an exact model of \mathcal{C}*

Proof. Let $\mathcal{C}(D)$ be the set of path inclusions defined in the proof of proposition 4. There exists $\mathcal{C}_b(D)$ a set of bounded path inclusions equivalent to $\mathcal{C}(D)$ if and only if

$$\begin{aligned} & \forall (\text{lex}(s_1) \preceq \text{lex}(s'_1) + \dots + \text{lex}(s'_k)) \in \mathcal{C}(D), \\ & \exists 1 \leq j \leq k \mid (\text{lex}(s_1) \preceq \text{lex}(s'_j)) \in \mathcal{C}(D) \end{aligned}$$

The condition is obviously sufficient. Conversely, if there exists $\mathcal{C}_b(D)$ equivalent to $\mathcal{C}(D)$ then for every $\text{lex}(s_1) \preceq \text{lex}(s'_1) + \dots + \text{lex}(s'_k)$ in $\mathcal{C}(D)$, $\mathcal{C}_b(D) \models \text{lex}(s_1) \preceq \text{lex}(s'_1) + \dots + \text{lex}(s'_k)$. Then, from lemma 4, there exists some j s.t. $\mathcal{C}_b(D) \models \text{lex}(s_1) \preceq \text{lex}(s'_j)$. As $\mathcal{C}_b(D)$ is equivalent to $\mathcal{C}(D)$, we have also $\mathcal{C}(D) \models \text{lex}(s_1) \preceq \text{lex}(s'_j)$. Then, from the proof of proposition 4, $D \models \text{lex}(s_1) \preceq \text{lex}(s'_j)$ i.e. $\text{result}_D(\text{lex}(s_1))$ is included in $\text{result}_D(\text{lex}(s'_j))$. Therefore s_1 is included in s'_j and $(\text{lex}(s_1) \preceq \text{lex}(s'_j))$ belongs to $\mathcal{C}(D)$ by definition of $\mathcal{C}(D)$.

As the cardinal of $\mathcal{C}(D)$ is bounded by 2^{N^2+2N} , this provides an *EXPTIME*-algorithm for deciding whether there exists a set of bounded path inclusions \mathcal{C} s.t. D is an exact model of \mathcal{C} .

3.3 Implication Problem

We have already proved that the implication problem is decidable in PSPACE using a first order formula of the theory of prefix rewiring. Nevertheless, we propose now another PSPACE algorithm based on the computation of the ancestors in a prefix rewrite system. In the case of word equality constraints, this construction will allow us to give a more efficient algorithm than the one given in the proof of theorem 2.

Theorem 4. *Let $\mathcal{C} = \{p_1 \preceq u_1, \dots, p_n \preceq u_n\}$ 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.*

Let $\mathcal{C} = \{p_i \preceq u_i, 1 \leq i \leq n\}$ be a finite set of bounded path inclusions, and q a regular query, we define the set $\text{ancestor}_{\mathcal{C}}(q) = \{u \mid \exists w_q \in L(q), u \xrightarrow[\mathcal{C}]{*} w_q\}$. Then we can state:

Lemma 6. *Let $\mathcal{C} = \{p_i \preceq u_i, 1 \leq i \leq n\}$ be a finite set of bounded path inclusions, and p, q two regular queries, then $\mathcal{C} \models p \preceq q$ if and only if $L(p) \subseteq \text{ancestor}_{\mathcal{C}}(q)$.*

Proof. This is a direct consequence of proposition 6: $\mathcal{C} \models p \preceq q$ if and only if $\forall u_p \in L(p), \mathcal{C} \models u_p \preceq q$. From proposition 6 this is equivalent to $\forall u_p \in L(p), \exists u_q \in L(q), \mathcal{C} \models u_p \preceq u_q$ and thanks to the same proposition it is equivalent to $\forall u_p \in L(p), \exists u_q \in L(q), u_p \xrightarrow[\mathcal{C}]{*} u_q$ i.e. $L(p) \subseteq \text{ancestor}_{\mathcal{C}}(q)$.

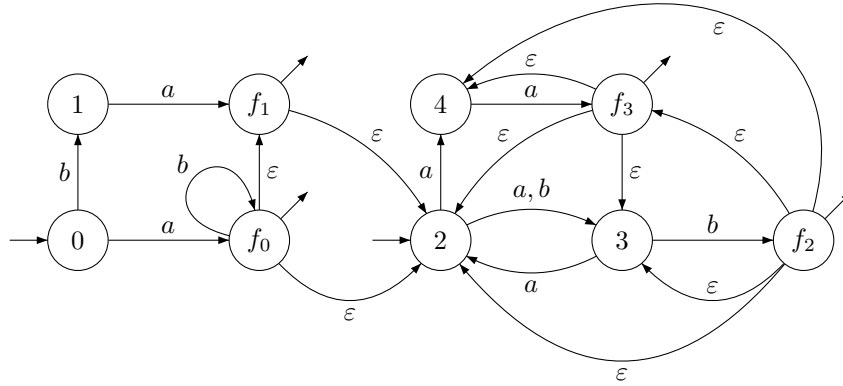
In order to compute $\text{ancestor}_{\mathcal{C}}(q)$ for any regular query q , we first build a finite automaton $\mathcal{A}_{\mathcal{C}}$ (with ε -moves) which recognizes the language $R_{\mathcal{C}} = \{v \in A^* \mid \exists i, v \xrightarrow[\mathcal{C}]{*} u_i\}$. It is already known from [17], [19], [21] that $R_{\mathcal{C}}$ is a recognizable

language. We give here a different construction: for each i with $1 \leq i \leq n$, let $\mathcal{M}_i = (\mathcal{A}, \mathcal{Q}, \mathcal{I}, \mathcal{F}, \delta_i)$ be an automaton recognizing the language $L(p_i + u_i)$. 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 $k > 0$, $\Delta_k = \Delta_{k-1} \cup \{(q, \varepsilon, q') \mid q \neq q' \wedge \exists i \leq n, q \in F_i, q' \in \text{result}_{\mathcal{A}_{\mathcal{C}^{k-1}}}(u_i)\}$ where $\mathcal{A}_{\mathcal{C}^{k-1}}$ is the automata $(A, Q, I, F, \Delta_{k-1})$

Since only transitions of the form (q, ε, q') can be added, there exists an integer K such that $\Delta_K = \Delta_{K+1} = \Delta$ for some K . As $K \leq 1 + |Q|^2$, automaton $\mathcal{A}_{\mathcal{C}}$ can be built in polynomial time in $|\mathcal{C}|$, the size of \mathcal{C} .

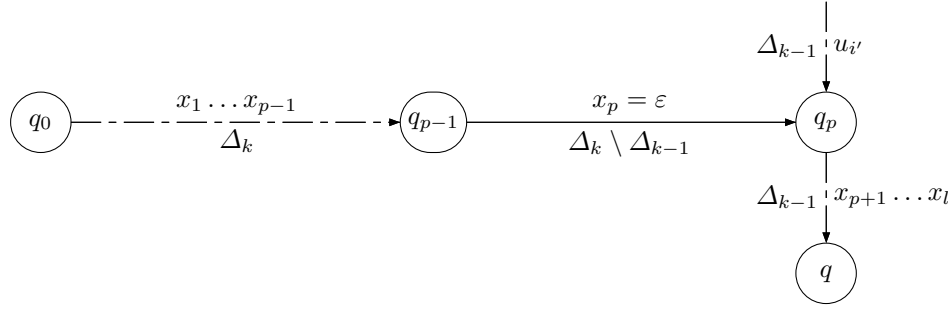
Example 4. Let $\mathcal{C} = \{ab^* \preceq ba, (aa + ba)^*(a + b)b \preceq aa\}$. An automaton $\mathcal{A}_{\mathcal{C}}$ is the following:



We have now to prove that $\mathcal{A}_{\mathcal{C}}$ recognizes $R_{\mathcal{C}}$.

Lemma 7. For any word v in A^* , if $\text{result}_{\mathcal{A}_{\mathcal{C}}}(v) \cap F_i \neq \emptyset$ for some i in $\{1, \dots, n\}$ then $v \xrightarrow{\mathcal{C}}^* u_i$.

Proof. Let $v \in A^*$ and $q \in \text{result}_{\mathcal{A}_{\mathcal{C}}}(v) \cap F_i$ for some i in $\{1, \dots, n\}$. Then there exists a k such that $q \in \text{result}_{\mathcal{A}_{\mathcal{C}^k}}(v)$. We will show that $v \xrightarrow{\mathcal{C}}^* u_i$ by induction on k . If $k = 0$, then $v \in L(p_i + u_i)$ and $v \xrightarrow{\mathcal{C}} u_i$ or $v = u_i$. Suppose now that $k > 0$. There exist q_0, q_1, \dots, q_l in Q and x_1, x_2, \dots, x_l in $A \cup \{\varepsilon\}$ such that $q = q_l$, $v = x_1 x_2 \dots x_l$ and for any j in $\{1, \dots, l\}$, $(q_{j-1}, x_j, q_j) \in \Delta_k$. Let m be the number of such (q_{j-1}, x_j, q_j) which are in $\Delta_k \setminus \Delta_{k-1}$. We shall now make an induction on m . If $m = 0$, then, by induction hypothesis on k , we obtain that $v \xrightarrow{\mathcal{C}}^* u_i$. If $m > 0$, let p be the integer such that $(q_{p-1}, x_p, q_p) \in \Delta_k \setminus \Delta_{k-1}$ and for any j with $p < j \leq l$, (q_{j-1}, x_j, q_j) is in Δ_{k-1} . Then $x_p = \varepsilon$, $q_{p-1} \in F_{i'}$ for some i' in $\{1, \dots, n\}$ and $q_p \in \text{result}_{\mathcal{A}_{\mathcal{C}^{k-1}}}(u_{i'})$:



By induction hypothesis on m , we obtain that $x_1x_2\dots x_{p-1} \xrightarrow{*}_{\mathcal{C}} u_{i'}$ and by induction hypothesis on k , we obtain that $u_{i'}x_px_{p+1}\dots x_l \xrightarrow{*}_{\mathcal{C}} u_i$. It follows that $v = x_1x_2\dots x_l \xrightarrow{*}_{\mathcal{C}} u_{i'}x_px_{p+1}\dots x_l \xrightarrow{*}_{\mathcal{C}} u_i$.

In order to prove the converse of lemma 7, we shall use the following result:

Lemma 8. *Let v and w be two words of A^* . If $v \xrightarrow{*}_{\mathcal{C}} w$, then $\text{result}_{\mathcal{A}_{\mathcal{C}}}(w) \subseteq \text{result}_{\mathcal{A}_{\mathcal{C}}}(v)$.*

Proof. Let j be the length of the derivation $v \xrightarrow{*}_{\mathcal{C}} w$. We shall make an induction on j . If $j = 0$ then $v = w$ and $\text{result}_{\mathcal{A}_{\mathcal{C}}}(w) = \text{result}_{\mathcal{A}_{\mathcal{C}}}(v)$. If $j > 0$, then there exists i in $\{1, \dots, n\}$ and words v_1, v_2 such that $v \xrightarrow{j-1}_{\mathcal{C}} v_1v_2$, $v_1 \in L(p_i)$ and $w = u_iv_2$. By induction hypothesis, we have $\text{result}_{\mathcal{A}_{\mathcal{C}}}(v_1v_2) \subseteq \text{result}_{\mathcal{A}_{\mathcal{C}}}(v)$. Moreover, since $v_1 \in L(p_i)$, there exists a state q in $F_i \cap \text{result}_{\mathcal{A}_{\mathcal{C}}}(v_1)$. Let q' be a state in $\text{result}_{\mathcal{A}_{\mathcal{C}}}(u_i)$ then $(q, \varepsilon, q') \in \Delta$ and $q' \in \text{result}_{\mathcal{A}_{\mathcal{C}}}(v_1)$. As we have $\text{result}_{\mathcal{A}_{\mathcal{C}}}(u_i) \subseteq \text{result}_{\mathcal{A}_{\mathcal{C}}}(v_1)$, we obtain $\text{result}_{\mathcal{A}_{\mathcal{C}}}(w) = \text{result}_{\mathcal{A}_{\mathcal{C}}}(u_iv_2) \subseteq \text{result}_{\mathcal{A}_{\mathcal{C}}}(v_1v_2) \subseteq \text{result}_{\mathcal{A}_{\mathcal{C}}}(v)$.

We are now able to prove:

Proposition 10. *For any word v in A^* , $\text{result}_{\mathcal{A}_{\mathcal{C}}}(v) \cap F_i \neq \emptyset$ for some i in $\{1, \dots, n\}$ if and only if $v \xrightarrow{*}_{\mathcal{C}} u_i$. In other words, automaton $\mathcal{A}_{\mathcal{C}}$ recognizes $R_{\mathcal{C}}$*

Proof. From lemma 7, we have only to prove the if part. Let us consider $v \in A^*$ such that $v \xrightarrow{*}_{\mathcal{C}} u_i$ for some i in $\{1, \dots, n\}$. By definition, $\text{result}_{\mathcal{A}_{\mathcal{C}}}(u_i) \cap F_i \neq \emptyset$, moreover, from lemma 8, $\text{result}_{\mathcal{A}_{\mathcal{C}}}(u_i) \subseteq \text{result}_{\mathcal{A}_{\mathcal{C}}}(v)$ then $\text{result}_{\mathcal{A}_{\mathcal{C}}}(v) \cap F_i \neq \emptyset$.

It is proved in proposition 10 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} . Indeed we have only to consider the automata $\mathcal{A}_{\mathcal{C}}^{u_i} = (A, Q, I, F_i, \Delta)$.

Now, in order to answer to the question $p \subseteq \text{ancestor}_{\mathcal{C}}(q)$, we compute the set of ancestors of q as the language described by the automaton $\mathcal{A}_{\mathcal{C} \cup \{q \preceq \mathcal{S}_q\}}^{\mathcal{S}_q}$ (\mathcal{S}_q is

a new letter). 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 state:

Lemma 9. *For any set $\mathcal{C} = \{p_i \preceq u_i, 1 \leq i \leq n\}$ of bounded path inclusions, and for any regular expressions p and q , the implication problem $\mathcal{C} \models p \preceq q$ is PSPACE.*

Proof. 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 cubic time in $|p|$ a (non deterministic) automaton \mathcal{A}_p which recognizes p (see for instance the Gluskov's algorithm [29]), and in polynomial time in $|q| + |\mathcal{C}|$ an automaton \mathcal{A}_c^q which recognizes $\text{ancestor}_c(q)$.

We are now able to end the proof of theorem 4, 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 10. *For any set $\mathcal{C} = \{p_1 \preceq u_1, \dots, p_n \preceq u_n\}$ of bounded path inclusions, for any regular expression p and for any word u , the implication problem $\mathcal{C} \models p \preceq u$ is PSPACE-complete.*

Proof. Inclusion problem of two regular languages, given by regular expressions p and q is PSPACE-hard [28]. Let us consider the set $\mathcal{C} = \{q \preceq \$\}$ where $\$$ does not appear in q . In this case, $\text{ancestor}_c(q) = L(q)$ and $L(p) \subseteq L(q)$ is equivalent to $L(p) \subseteq \text{ancestor}_c(\$)$. So deciding $L(p) \subseteq L(q)$ is equivalent to decide $p \preceq \$$.

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

Proposition 11. *Let $\mathcal{C} = \{p_1 \preceq u_1, \dots, p_n \preceq u_n\}$ 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.*

Proof. $\mathcal{C} \models u \preceq q$ if and only if $u \in \text{ancestor}_c(q)$. We build an automaton $\mathcal{A}_c^{\$q}$ recognizing $\text{ancestor}_c(q)$ in PTIME and we test the membership of u in $\text{ancestor}_c(q)$ using this automaton.

We summarize our results (\mathcal{C} is a set of bounded path constraints, p and q are two regular queries and u is a word):

bounded path inclusions	new results	already known
$\mathcal{C} \models p \preceq q$	PSPACE (lemma 9)	EXPSPACE [3] EXPTIME [5]
$\mathcal{C} \models p \preceq u$	PSPACE-complete (lemma 10)	
$\mathcal{C} \models u \preceq q$	PTIME (proposition 11)	

4 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 words. Since this case is a particular case of bounded path inclusions, any algorithm presented in section 3 can be used on a set of word equality constraints. As, in this particular case of word equalities, the implication is symmetric (i.e. $\mathcal{C} \models u \preceq v$ implies that $\mathcal{C} \models v \preceq u$) one can improve some of these algorithms: in particular, it is possible to decide in PTIME whether a set of word equality constraints satisfies the strong boundedness property.

4.1 A finite representation of the exact model $D_{\mathcal{C}}$

In section 3, definition 11, we have introduced an equivalence relation over path inclusions, denoted $\equiv_{\mathcal{C}}$, and associated with any set \mathcal{C} of path inclusions. Recall that, for any words u and v in A^* , we have $u \equiv_{\mathcal{C}} v$ if $\mathcal{C} \models u \equiv v$. Now, when \mathcal{C} is a set of word equalities (that is a symmetric relation over A^*), the relation $\equiv_{\mathcal{C}}$ satisfies the following property:

Lemma 11. *Let \mathcal{C} be a set of word equality constraints over an alphabet A . Then $\equiv_{\mathcal{C}}$ is the smallest equivalence relation, closed by right congruence, which contains \mathcal{C} and for any words u and v , if $\mathcal{C} \models u \preceq v$ then $u \equiv_{\mathcal{C}} v$.*

Proof. Clearly, $\equiv_{\mathcal{C}}$ is an equivalence relation which is closed by right congruence and contains \mathcal{C} . Now, if we consider two words u and v such that $\mathcal{C} \models u \preceq v$, then $u \xrightarrow{\mathcal{C}}^* v$ from proposition 5. It follows, from the definition of $\xrightarrow{\mathcal{C}}^*$ that (u, v) belongs to any equivalence relation which is closed by right congruence and contains \mathcal{C} .

In the special case of word equalities, the exact model $D_{\mathcal{C}}$, associated with a set \mathcal{C} of bounded path inclusions, and introduced in section 3.2 is deterministic and complete. Indeed it is defined as:

- $N = \{[u]_{\mathcal{C}} \mid u \in A^*\},$
- $r = \{[\varepsilon]_{\mathcal{C}}\}$
- $T = \{([u]_{\mathcal{C}}, x, [ux]_{\mathcal{C}}) \mid u \in A^*, x \in A\}.$

For any word u in A^* , we get from lemma 5 that $\text{result}_{D_{\mathcal{C}}}(u) = \{[u]_{\mathcal{C}}\}$. Then we can state the following proposition:

Proposition 12. *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 single rooted data graphs.
3. $\mathcal{C} \models u \equiv v$ on the family of deterministic data graphs.
4. $\mathcal{C} \models u \equiv v$ on the family of complete deterministic data graphs.

Proof. Clearly, it is sufficient to prove 4 implies 1. Let u and v be two words such that $\mathcal{C} \models u \equiv v$ on the family of complete deterministic data graphs. Then $D_{\mathcal{C}} \models u \equiv v$, since $D_{\mathcal{C}} \models \mathcal{C}$ and it is complete and deterministic. Now, from proposition 7 which states that $D_{\mathcal{C}}$ is an exact model of \mathcal{C} , it follows that $\mathcal{C} \models u \equiv v$.

Corollary 5. *For any set \mathcal{C} of word equality constraints over an alphabet A , $D_{\mathcal{C}}$ is the unique (complete) deterministic rooted graph D which satisfies: $D \models u \equiv v$ if and only if $\mathcal{C} \models u \equiv v$.*

Generally, the model $D_{\mathcal{C}}$ is an infinite graph. Nevertheless, when \mathcal{C} is a finite set of word equality constraints, it is possible to build a finite deterministic subgraph of $D_{\mathcal{C}}$ in order to decide some properties like implication problem, strong boundedness property or existence of an exact finite model. A quite similar construction has been introduced by Buneman et al. in [15]:

Definition 12. *Let \mathcal{C} be a finite set of word equality constraints over A .*

- Let us denote by W the set of all prefixes of $\{w \in A^* \mid \exists w' \in A^*, (w \equiv w') \in \mathcal{C}\}$.
- For any word in W , let us denote by $[w]$ the equivalence class of w for the restriction of $\equiv_{\mathcal{C}}$ over W .
- We define the finite deterministic graph $D_{\mathcal{C}}^f$ as the graph $D_{\mathcal{C}}^f = \langle N', r, T' \rangle$ 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 consider now the application $f_{\mathcal{C}}$, defined from A^* to $N' \times A^*$, where N' is the set of nodes of $D_{\mathcal{C}}^f$, by: for any word in A^* , $f_{\mathcal{C}}(u) = (\text{result}_{D_{\mathcal{C}}^f}(u_1), u_2)$ where $u = u_1u_2$ and u_1 is the longest prefix of u such that $\text{result}_{D_{\mathcal{C}}^f}(u_1) \neq \emptyset$.

Example 5. 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 6 gives the graph $D_{\mathcal{C}}^f$ for this set of constraints. On this example, $f_{\mathcal{C}}(a^3) = ([bba], \varepsilon)$, $f_{\mathcal{C}}(a^3c) = ([bba], c)$.

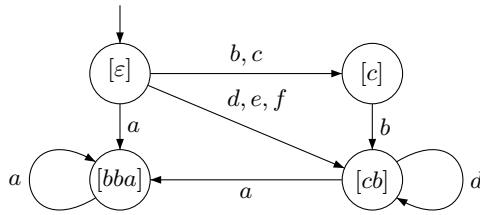


Fig. 6. graph $D_{\mathcal{C}}^f$

Then we can state:

Proposition 13. $C \models u \equiv v$ if and only if $f_c(u) = f_c(v)$.

Proof. $C \models u \equiv v$ if and only if $\text{result}_{D_c}(u) = \text{result}_{D_c}(v)$ if and only if $f_c(u) = f_c(v)$

Now, denoting by $f_c(p)$ the set $\cup_{u \in L(p)} f_c(u)$ for any regular path expression p we can deduce, from the above proposition, lemma 11 and using the fact that D_c is complete and deterministic:

Corollary 6. For any regular path expressions p and q , $C \models p \equiv q$ if and only if $f_c(p) = f_c(q)$.

Moreover, from the above corollary, we obtain:

Corollary 7. A regular path expression p has the strong boundedness property w.r.t. a finite set of word equalities C if and only if $f_c(p)$ is finite.

Concerning existence of an exact finite model for a set C of word equalities, we can state:

Proposition 14. Let C be a set of word equality constraints. C has a finite exact model if and only if D_c^f is a complete graph.

Proof. If D_c^f is complete then \equiv_c is of finite index and then C has a finite exact model (theorem 3). If D_c^f is not complete there exists a word u such that $\text{result}_{D_c^f}(u)$ is empty, that is $f_c(u) = (\text{result}_{D_c^f}(u_1), u_2)$ and u_2 is not the empty word. Then for a label x and for any integer n , $f_c(ux^n) = (\text{result}_{D_c^f}(u_1), u_2x^n)$. It follows from proposition 13 that ux^n and ux^m (where n and m are different) cannot be in the same class. So \equiv_c is not of finite index.

Corollary 8. If \equiv_c is of finite index then the number of classes is bounded by the size of C .

Now, in order to decide efficiently these different properties, it remains to produce an efficient algorithm which can compute the graph D_c^f . This is done in next section.

4.2 A quasi linear algorithm

The aim of this section is to present an algorithm which constructs the graph D_c^f with a quasi linear complexity in the size of the set of equality constraints C . The graph D_c^f is defined to get a finite representation of the relation \equiv_c defined over A^* . By definition, it is also a finite representation of the smallest right congruence which contains the relation $\{(w, w') \mid (w \equiv w') \in C\}$.

The algorithm will construct this congruence in the following way: let the set of equality constraints be $C = \cup_{i=1}^n \{(u_i \equiv v_i)\}$ and W be the set of all prefixes of $\{w \in A^* \mid \exists w' \in A^*, (w \equiv w') \in C\}$. For any integer $1 \leq i \leq n$, let us denote

by R_i the restriction to W of the smallest right congruence, which contains the relation $\cup_{k=1}^i \{(u_k, v_k)\}$ (R_0 denotes the identity relation). We shall denote by $[u]_i$ the equivalence class of a word $u \in W$ for the relation R_i . Then we want to compute R_n , starting from R_0 . At each step i , the algorithm must, for the constraint $u_i \equiv v_i$, merge the equivalence classes $[u_i]_{i-1}$ and $[v_i]_{i-1}$ and compute the right congruence closure.

In order to implement this merging, we need a disjoint-set data structure which provides algorithms for determining which class a word belongs to, and for combining two equivalence classes. The well-known *union-find* algorithm performs these operations (see [35,34] and [22] for data structures that can be used), supporting the primitives **find**(u) - which returns the representative of $[u]$ -, **union**(u, v) - which computes a new class $[u] \cup [v]$ and returns the representative of this new class-, **create**(u) which creates a class with one element, u , requiring that u doesn't belong to any class-.

So, our algorithm builds a graph; initially, the graph is obtained from the prefix tree of words appearing in \mathcal{C} , where nodes are identified with the words of the set W , and edges are labeled by letters. Then, we apply **union** (u, v) for each $u \equiv v$ in \mathcal{C} . But, for ensuring right congruence closure, we will ensure that if there is an edge labeled by x from u to v , there is an edge labeled by x from $\text{find}(u)$ to a node $v' \equiv v$. Thus we can define **merge**($u, v : \text{Node}$) which merges two classes and performs the closure by right congruence:

```
function merge(u , v : Node)
begin
  if (find(u) != find(v)) then
    Node r := union(u , v) ;
    -- w.l.o.g. we suppose r = find(u) or r = find(v)
    for each x ∈ A do
      if there are some edges (find(u),x,s) and (find(v),x,t) then
        merge(s , t) ;
      elsif there is some edge (find(u),x,s) and r = find(v) then
        add a new edge (r , x , s) ;
      elsif there is some edge (find(v),x,t) and r = find(u) then
        add a new edge (r , x , t) ;
      end if ;
    end for ;
  end if ;
end merge ;
```

Finally the algorithm is:

```
for each u ∈ W do create(u); end for ;
for each constraint (u ≡ v) ∈ C do
  merge(find(u), find(v)) ;
end for ;
```

Let us study the complexity of this algorithm. In the worst case, all the nodes belong to the same class; so, the total number of calls to function `union` and calls to function `find` is in $O(|\mathcal{C}|)$. It is well known that by using union by rank and path compression the amortized cost of an operation union or find is quasi-constant, more precisely in $O(\alpha(n))$, where α is the inverse of $f(n) = A(n, n)$ with A the Ackermann function, n the number of nodes [35]. So, complexity of the algorithm is $O(|\mathcal{C}| \cdot \alpha(|\mathcal{C}|))$.

Now, since it is easy to prove that D_C^f is the graph $\langle N, r, T \rangle$ where $N = \{\text{find}(u) \mid u \in W\}$, $r = \{\text{find}(\varepsilon)\}$ and $T = \{(\text{find}(u), x, \text{find}(ux)) \mid ux \in W\}$, we can state:

Proposition 15. *One can compute D_C^f in quasi linear time in $|\mathcal{C}|$.*

Example 6. (example 5 continued) Let $A = \{a, b, c, d, e, f\}$ and $\mathcal{C} = \{a \equiv bba, b \equiv c, cb \equiv dd, d \equiv e, fa \equiv aa, ed \equiv f, e \equiv f, aa \equiv bba\}$. The figure 7 shows the data structure used to compute the graph D_C^f . 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 $\text{find}(a) = \text{find}(fa) = bba$. Since $f \equiv ed \equiv fd$, we get $\text{find}(fd^*) = \text{find}(f)$. Finally, after merging the equivalent nodes, we obtain the graph D_C^f shown in figure 6.

4.3 Some complexity improvements

In this section, we shall use the graph D_C^f in order to improve the complexity of some algorithms for implication problem, strong boundedness property and existence of an exact finite model for a finite set of word equalities \mathcal{C} . Clearly, for this last problem, we can state:

Proposition 16. *Let \mathcal{C} be a set of word equality constraints. Deciding whether \mathcal{C} has a finite exact model is quasi linear in the size of \mathcal{C} .*

Proof. From the proposition 14, \mathcal{C} has a finite exact model if and only if the graph D_C^f is complete. One can compute the graph D_C^f with a quasi linear algorithm and to decide whether D_C^f is complete can be done with an algorithm linear in the size of D_C^f .

Concerning implication problem, we can improve the algorithm of [15] which decides whether a set of word equality constraints implies a word equality constraint. We can also answer whether a regular query p has the strong boundedness property for a finite set of word equalities \mathcal{C} with a PTIME algorithm in the sum of the sizes of p and \mathcal{C} . Nevertheless, we prove that deciding whether a set of word equality constraints implies that a query p is equivalent to a query q is PSPACE complete.

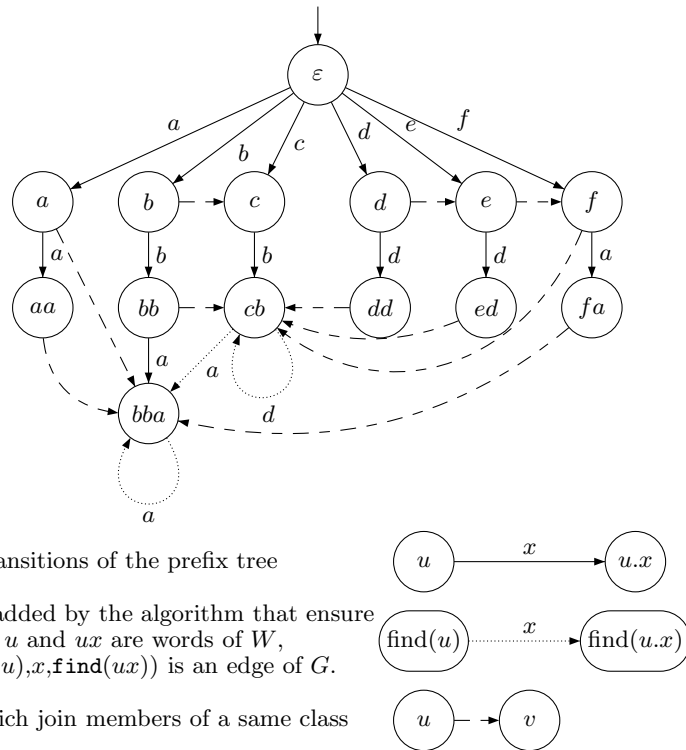


Fig. 7. Steps of the union-find algorithm

Lemma 12. For any regular path expressions p and q , and given the graph D_c^f ,

1. the test of finiteness of $f_c(p)$ can be done in PTIME in the sum of the size of D_c^f and the size of p .
2. a comparison between $f_c(p)$ and $f_c(q)$ can be done in PSPACE in the sum of the size of D_c^f and the size of the two regular expressions.

Proof. We will first show that, for every node n of D_c^f , we can compute an automaton $\mathcal{A}_{c,p}(n)$ in PTIME in the sum of the size of D_c^f and the size of p such that the language recognized by $\mathcal{A}_{c,p}(n)$ is the language $\{w \in A^* \mid (n, w) \in f_c(p)\}$. This construction follows five steps:

1. construct an automaton \mathcal{A}_p which recognizes the language described by p , where all states are accessible and co accessible, this automaton can be constructed in quadratic time w.r.t. $|p|$ (see, for example, [13]).
2. complete the graph D_c^f 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 D_c^f ; this can be done in size of D_c^f .
3. compute 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 D_c^f or equal to \perp , s_1 and s_2 are states of automaton \mathcal{A}_p and x is a letter.
4. remove, in the previous graph, all transitions $((n_1, s_1), x, (n_2, s_2))$ where n_2 is a node of D_c^f (i.e. not equal to \perp).
5. finally $\mathcal{A}_{c,p}(n)$ is obtained from the previous graph, setting the initial states to nodes which are in $\{n\} \times S$ where S is the set of states of automaton \mathcal{A}_p and the final states to nodes which are in $\{\perp\} \times F$ where F is the set of final states of automaton \mathcal{A}_p .

The whole construction can be done in PTIME in the sum of the sizes of \mathcal{A}_p and D_c^f .

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

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

Remark 4. For the comparison of $f_c(p)$ and $f_c(q)$ 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 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 [28] 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 proposition 15, proposition 13, corollary 6, corollary 7 and lemma 12, we obtain:

Theorem 5. *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 $|\mathcal{C}| + |p|$.*

In the case when a regular query q has the strong boundedness property with respect to a finite set of word equality constraints \mathcal{C} , it is possible to produce a regular expression f , denoting a finite language, such that $\mathcal{C} \models q \equiv f$. More precisely, we can state :

Proposition 17. *Let \mathcal{C} be a non empty finite set of word equalities over an alphabet A . One can compute, in quasi linear time in the size of \mathcal{C} , 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 iff p has the strong boundedness property w.r.t. \mathcal{C}

Proof. Let \mathcal{C} be a non empty set of word equalities over an alphabet A . Let us consider the graph $D_{\mathcal{C}}^f = \langle N, r, T_G \rangle$. The nodes of $D_{\mathcal{C}}^f$ are equivalence classes of words for the equivalence relation $\equiv_{\mathcal{C}}$. For any class $[u]$, we will use $\text{find}(u)$, presented in subsection 4.2, as a representative of the class $[u]$. From this graph, we can define a transducer $\tau_{\mathcal{C}} = \langle A, N \cup \{\$, \}, r, N \cup \{\$, \}, T, e \rangle$ where A is the input and the output alphabet, $N \cup \{\$, \}$ with $\$ \notin N$ is the set of states, r is the initial state and all the states are finals. The set of transitions T is defined by $T = \{([u], x, \varepsilon, [ux]) \mid [ux] \in N\} \cup \{([u], x, \text{find}(u)x, \$) \mid [ux] \notin N\} \cup \{(\$, x, x, \$) \mid x \in A\}$ and e is an output function from the final states $N \cup \{\$, \}$ to A^* defined by: $e([u]) = \text{find}(u)$ and $e(\$) = \varepsilon$.

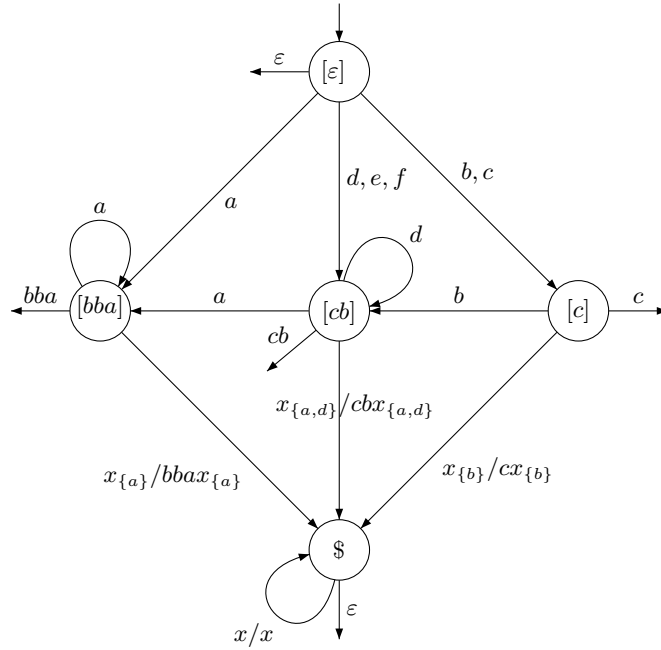
It is easy to see that for all word u , $f_{\mathcal{C}}(u) = ([u_1], u_2)$ if and only if $\tau_{\mathcal{C}}(u) = \text{find}(u_1)u_2$. If q is a path expression then

$$\tau_{\mathcal{C}}(q) = \bigcup_{u \in L(q)} \tau_{\mathcal{C}}(u) = \bigcup_{\substack{u \in L(q) \\ f_{\mathcal{C}}(u) = ([u_1], u_2)}} \text{find}(u_1)u_2$$

As $f_{\mathcal{C}}(\tau_{\mathcal{C}}(q)) = f_{\mathcal{C}}(q)$, it follows from corollary 6 that for all q such that $L(q) = \tau_{\mathcal{C}}(L(p))$, $\mathcal{C} \models p \equiv q$. Finally, from corollary 7 we obtain: $\tau_{\mathcal{C}}(L(p))$ is finite if and only if p has the strong boundedness property w.r.t. \mathcal{C} .

The complexity of the construction of $\tau_{\mathcal{C}}$ is quasi linear, since it is based on the construction of $D_{\mathcal{C}}^f$.

Example 7. (example 5 continued) The transducer τ_c is the following where transitions from a node $[u]$ to the node $\$$ are labeled in the form $x_s \mid ux_s$ with $s \subseteq A$. Such a label $x_s \mid ux_s$ correspond to the set of transitions $\{([u], x, \text{find}([u])x, \$) \mid x \notin s\}$. The set of transitions $\{(\$, x, x, \$) \mid x \in A\}$ is represented by a single transition labeled $x \mid x$ on node $\$$.



- $\tau_c(a^+b) = bbab$ because $f_c(a^+b) = \{([bba], b)\}$. As $L(bbab)$ is finite, a^+b has the strong boundedness property w.r.t \mathcal{C} and $\mathcal{C} \models a^+b \equiv bbab$.
- $\tau_c(f^+) = cbf^*$ since $f_c(f^+) = \{([cb], f^n) \mid n \in \mathbb{N}\}$. As $L(cb f^*)$ is not finite, f^+ has not the strong boundedness property w.r.t. \mathcal{C} . Nevertheless, $\mathcal{C} \models f^+ \equiv cb f^*$ is true.

5 Conclusion

In this paper, we have investigated path constraints on semistructured data modeled as multi rooted edge-labeled directed graphs and we have studied some associated problems such as existence of a finite exact model, implication problem and strong boundedness property.

In the case when path expressions involved in the constraints are full regular expressions, most results we get are straightforward extensions of previous ones for single rooted graphs [3].

When constraints are bounded path inclusions, the problems we consider can be transformed into problems of prefix rewriting systems. So “ad hoc” algorithms

have been developed and we get new results; e.g. we have established that, in this case, the implication problem is PSPACE-complete and that strong boundedness can be decided.

In the special case of word equality constraints, we have proved that it is decidable in quasi-linear time whether a finite set of word equalities has a finite exact model. The word implication problem has been proved to be quasi-linear and strong boundedness property has been proved to be decidable in PTIME. These results use a finite representation of an exact model of a set of word equality constraints.

In further works, some topics deserve investigation. E.g., relations between keys and foreign keys in XML data are a kind of inclusion constraints. So a natural question is to know whether the techniques developed in this paper can be applied in the context of keys.

We are also interested in considering XML query languages, like XPATH or XQuery. We have studied graph queries, a generalization of Tree Pattern Queries. Unfortunately, the evaluation of such queries is NP-complete. It would be interesting to use constraints on the data in order to rewrite a graph query to obtain an equivalent Tree Pattern Query when it is possible.

Acknowledgments

We thank V. Benzaken, S. Demri, F. Gire and J.E. Pin whose constructive comments have helped to improve this article. We also thank anonymous reviewers for their comments.

This research was partially supported by: “CPER 2000-2006, Contrat de Plan état - région Nord/Pas-de-Calais: programme TAC, projet COCOA” and “ACI masse de données TRALALA, FNS”

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. Serge Abiteboul and Victor Vianu. Regular path queries with constraints. In *PODS*, pages 122–133. ACM Press, 1997.
4. A. Aho, J. Hopcroft, and J. Ullman. *The design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Compagny, Reading, Mass., 1974.
5. N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13(6):939 – 956, 2003.

6. Natasha Alechina, Stéphane Demri, and Maarten de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13(6):939–956, 2003.
7. Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Minimization of tree pattern queries. In *SIGMOD Conference*, 2001.
8. Y. André, F. Bossut, and A.C. Caron. On decidability of boundedness property for regular path queries. In *proceedings of DLT'99*, Aachen, Germany, 1999. Development in Language Theory, World Scientific Publishing Co.
9. Y. André, A.C. Caron, D. Debarbieux, Y. Roos, and S. Tison. Extraction and implication of path constraints. In *Proceedings of the 29th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 863–875, 2004.
10. Anders Berglund, Scott Boag, Don Chamberlin, Mary Fernandez, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML Path Language (XPath) 2.0, w3c recommendation. The W3C's Xpath web pages: <http://www.w3.org/TR/xpath/>, 2002.
11. N. Bidoit, S. Cerrito, and V. Thion. A first step towards modelling semistructured data in hybrid modal logic. *Journal of Non Classical Logics*, 2004.
12. Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. Xquery 1.0: An xml query language, w3c recommendation. The W3C's Xquery web pages: <http://www.w3.org/TR/xquery/>, 2002.
13. Anne Brüggemann-Klein. Regular expressions into finite automata. *Theor. Comput. Sci.*, 120(2):197–213, 1993.
14. 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.
15. 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.
16. P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences*, 61(2), 2000.
17. J. Richard Büchi and W.H. Hosken. Canonical systems which produce periodic sets. *Mathematical Systems Theory*, 4(1), 1970.
18. Diego Calvanese and Giuseppe De Giacomo. Expressive description logics. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *Description Logic Handbook*, pages 178–218. Cambridge University Press, 2003.
19. D. Caucal. On the regular structure of prefix rewritings. In Springer, editor, *Selected papers of the conference on Fifteenth colloquium on trees in algebra and programming*, pages 87 – 102, Copenhagen, Denmark, May 1990.
20. D. Caucal. Monadic theory of term rewritings. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, volume 1232, pages 266 – 273. IEEE Computer Society Press, 1992.
21. 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.
22. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
23. M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc 5th IEEE Symp Logic in Computer Science*, pages 242–256, 1990.

24. D. Debarbieux, Y. Roos, and S. Tison. Models of path constraints. In *Proceedings of the 10th Mons Theoretical Computer Science Days*, 2004.
25. D. Debarbieux, Y. Roos, S. Tison, Y. André, and A.C. Caron. Path rewriting in semistructured data. In *proceedings of words'03: 4th International Conference on Combinatorics on Words*, pages 358–369, Turku, Finland, 2003. TUCS General Publication.
26. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for xml. In *Proc. of World Wide Web Conference*, Toronto, 1999.
27. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
28. M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the Theory of NP-completeness*. Freeman, 1978.
29. V.M. Gluskov. the abstract theory of automata. In *Russian mathematical survey*, volume 16, pages 1–53, 1961.
30. 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.
31. Dexter Kozen and Jerzy Tiuryn. Logics of programs. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 789–840. The MIT Press, 1990.
32. Tova Milo and Dan Suciu. Index structures for path expressions. In *ICDT '99: Proceeding of the 7th International Conference on Database Theory*, pages 277–295, London, UK, 1999. Springer-Verlag.
33. J. Rogie, L.Lapp, and D. Schach. Xml query manguage (xql). In *QL'98 - The query manguages Workshop*, 1998.
34. R.E. Shostak. An algorithm for reasoning about equality. *Commun. ACM*, 21(7):583–585, 1978.
35. R. Tarjan. Efficiency of a good but non linear set union algorithm. In *Journal of the ACM*, volume 22 of 2, pages 215 – 225, 1975.

A Proof of proposition 1

The proof of proposition 1 is close to the proof presented in [3].

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

Proof. Clearly, we have only to prove that the condition is sufficient. Let p_0 and q_0 be two queries s.t. $\mathcal{C} \not\models p_0 \preceq q_0$. We are going to construct a finite data graph D_f s.t. $D_f \models \mathcal{C}$ and $D_f \not\models p_0 \preceq q_0$. Since $\mathcal{C} \not\models p_0 \preceq q_0$, there exists a (maybe infinite) data graph $D = \langle N_D, R_D, T_D \rangle$ s.t. $D \models \mathcal{C}$ and $D \not\models p_0 \preceq q_0$.

Let \equiv be the right semi-congruence relation defined on $A^* \times A^*$ by $u \equiv v$ if for any word w , for any path inclusion $p \preceq q \in \mathcal{C}$, uw belongs to $L(q)$ if and only if vw belongs to $L(q)$. Let \bowtie be the equivalence relation defined on $N_D \times N_D$ by $n \bowtie n'$ if

$$- \forall u \in A^*, (n \in \text{result}_D(u) \implies \exists v \in A^* (u \equiv v \wedge n' \in \text{result}_D(v)))$$

$$- \forall v \in A^*, (n' \in \text{result}_D(v) \implies \exists u \in A^* (u \equiv v \wedge n \in \text{result}_D(u)))$$

Denoting $[n]$ the equivalence class of a node n for \bowtie , we can now define the data graph $D_f = \langle N_f, D_f, T_f \rangle$ as

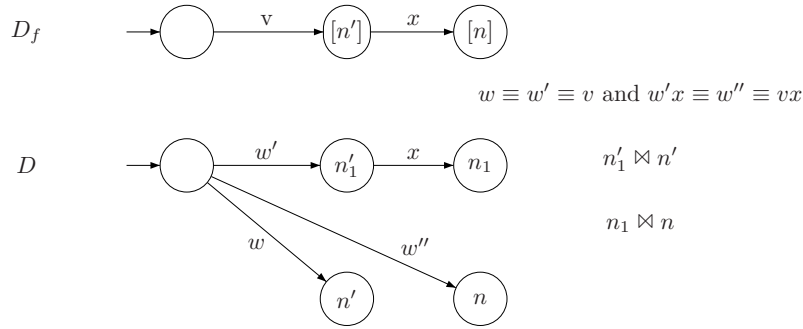
$$\begin{aligned} - N_f &= \{[n] \mid n \in N_D\} \\ - R_f &= \{[n] \mid \exists r \in R_D \ r \bowtie n\} \\ - T_f &= \{([n'], x, [n]) \mid \exists n'_1, n_1 \ (n \bowtie n_1) \wedge (n' \bowtie n'_1) \wedge (n'_1, x, n_1) \in T_D\} \end{aligned}$$

Clearly, the data graph D_f is finite. It remains to prove that $D_f \models \mathcal{C}$ and $D_f \not\models p_0 \preceq q_0$. We shall use the following property:

$$\forall u \in A^*, \forall n \in N_D, ([n] \in \text{result}_{D_f}(u) \Rightarrow \exists v \equiv u \mid n \in \text{result}_D(v)) \quad (1)$$

Let us prove this property by induction on the length of u .

- If u is the empty word then $[n]$ is a root of D_f . So there exists a node r s.t. r is a root of D and $n \bowtie r$. It follows that there is a word v equivalent to ε s.t. n belongs to $\text{result}_D(v)$.
- If $u = vx$ where x is a letter, then there exists a node $[n']$ of D_f s.t. $[n']$ is reached by v and $([n'], x, [n])$ is a transition of T_f . From the definition of T_f , there exist two nodes $n'_1 \bowtie n'$ and $n_1 \bowtie n$ s.t. (n'_1, x, n_1) belongs to T . Moreover, from the induction hypothesis, we know that n' is reached by a word $w \equiv v$. Now, since $n' \bowtie n'_1$, there exists a word $w' \equiv w \equiv v$ s.t. n'_1 belongs to $\text{result}_D(w')$ i.e. n_1 belongs to $\text{result}_D(w'x)$ and since $n \bowtie n_1$, there exists a word $w'' \equiv w'x \equiv vx = u$ s.t. n belongs to $\text{result}_D(w'')$.



We shall also use the following second property:

$$\forall u \in A^*, \forall n \in N_D, (n \in \text{result}_D(u) \Rightarrow [n] \in \text{result}_{D_f}(u)) \quad (2)$$

We prove this property by induction on the length of u . If u is the empty word then n is a root of D then, by definition of D_f , $[n]$ is a root of D_f which is in $\text{result}_{D_f}(u = \varepsilon)$. If $u = vx$ where x is a letter, then there exists a node n' s.t. n' belongs to $\text{result}_D(v)$ and (n', x, n) is in T_D . By induction hypothesis, $[n']$ belongs to $\text{result}_{D_f}(v)$ and by definition of T_f , $([n'], x, [n])$ is a transition of T_f then $[n]$ is reached by u .

Now, from properties 1 and 2 it is easy to obtain that for any regular expression q and for any node n in D :

$$[n] \in \text{result}_{D_f}(q) \text{ if and only if } n \in \text{result}_D(q) \quad (3)$$

And it follows that for any path inclusion $p \preceq q$, $D \models p \preceq q$ if and only if $D_f \models p \preceq q$.