Regular Path Queries with Constraints

Serge Abiteboul* Victor Vianu†

Abstract

The evaluation of path expression queries on semi-structured data in a distributed asynchronous environment is considered. The focus is on the use of local information expressed in the form of *path constraints* in the optimization of path expression queries. In particular, decidability and complexity results on the implication problem for path constraints are established.

1 Introduction

Navigational queries on data represented in a graph-like manner have proven to be useful in a variety of database contexts, ranging from hypertext data to object-oriented databases. Typically, navigational queries are expressed using regular expressions denoting paths in the graph representing the data. Such path queries have assumed renewed interest in the context of semi-structured data such as that commonly found in the Web (e.g., [QRS+95, AQM+96, BDHS96, MMM96, Suc96]). In this paper, we consider the processing of path queries in a distributed asynchronous environment. We focus on path query evaluation that takes advantage of local knowledge about the data graph, of the kind that might be available on the Web. We consider such local knowledge represented as *path constraints*. The main contribution of the paper is the study of the implication problem for path constraints, and its use in optimizing the distributed evaluation of path queries.

We use here an abstraction of the Web as a set of objects linked by labeled edges. An object represents a page (and possibly a site), and the labeled edges represent hypertext links. We focus on path queries [CM90, KKS92, dBV93, CACS94, MW95, AQM+96, BDHS96, MMM96, Suc96], which have emerged as an important class of browsing-style queries on the Web. Path queries are of the form *find all objects reachable by paths whose labels form a word in* r, where r is a regular expression over an alphabet of labels.

We present a basic scenario for evaluating such queries in a distributed context, based on simple communication between sites. We show that our technique correctly evaluates the answer, and provide a protocol that also detects termination whenever possible. We also point to an analogy between our evaluation technique and the *magic-set* [BMSU86] or *query-subquery* [Vie87] evaluation of a Datalog program (see also [AHV95]).

*CS Department, Stanford University, abitebou@db.stanford.edu. Permanent position at INRIA-Rocquencourt.

†CS Department, University of California at San Diego, vianu@cs.ucsd.edu. Work supported in part by the National Science Foundation under grant number IRI-9221268.

The distributed processing of path queries can be greatly enhanced by taking advantage of *path constraints*. Path constraints are local; they capture structural information about a Web site (or a collection of sites) or about its physical organization (e.g., cached information). A path constraint is an expression of the form $p \subseteq q$ or $p = q$, where p and q are regular expressions. A path constraint $p \subseteq q$ holds at a given site if the answer to query p applied to that site is included in the answer to q applied to the same site (and similarly for $p = q$). The following are some self-explanatory examples of path constraints:

```
CS-Department DB-group Member Ullman Classes-Taught cs345
                                    = CS-Department Courses cs345
CS-Department * Back-To-CS          = CS-Department
CS-Department Faculty Publications  ⊆ .netscape cache 07.
```

Taking advantage of such information in query processing turns out to be nontrivial. This is the focus of our results. The central technical problem that we address is to decide equivalence (or inclusion) of regular path queries under such constraints. This problem lies at the confluence of language theory, rewriting systems, and logic. We are able to prove that the general implication problem for regular path constraints is decidable in EXPSPACE (with respect to the size of the constraints). This result is rather surprising, since closely related problems in logic and rewriting systems are known to be undecidable. We obtain improved decision procedures of complexity PTIME and PSPACE for two important special cases, and develop along the way several technical tools related to implication. We lastly apply these techniques to the *boundedness* problem for regular path expressions. We show that it is decidable whether a given regular path query is equivalent to a path query without recursion, assuming that a given set of equalities among words is satisfied.

Related work Path queries in graphs have been studied formally in [CM90, MW95]. The language GraphLog, introduced in [CM90], expresses queries using graph patterns, where paths are specified by regular expressions. GraphLog is shown equivalent to stratified linear Datalog and other languages. The complexity of path queries in graphs is studied in [MW95]. Specifically, the problem of finding all pairs of nodes connected by a simple path satisfying a given regular expression is shown to be NP-complete in the size of the graph, and tractable subcases are identified. Path queries in object-oriented databases are considered in [dBV93]; they focus on the concise specification of path queries and the inference of completions of partially specified paths from schema information. Query languages for semi-structured data, that include path expressions, are considered in [KS95, BDHS96, MMM96, Suc96]. The language UnQL and its optimization are discussed in [BDHS96]; the optimizations involve loop fusion and a form of pushing selection. [Suc96] provides an evaluation procedure of UnQL queries in a distributed web-like environment. Using a decomposition technique, it is shown that UnQL queries can be evaluated by shipping the query exactly once to every site, returning the local results to the client site, and assembling the final result at the client site. [MMM96] considers the language WebSQL, which also incorporates path expressions, and provides a theory of query cost based on the notion of query locality.

To our knowledge, no previous work considers path constraints and their use in path query optimization.

The paper is organized as follows. Section 2 provides the background and motivation, and presents a distributed evaluation algorithm for path queries. The implication problem for path constraints is studied in Section 3. Most proofs are provided in an Appendix.

2 Path queries

We first present a simple abstraction of the Web, and introduce path queries expressed using regular expressions. We consider a scenario for the distributed evaluation of such queries, motivated by the Web.

The Web We view the Web as a labeled graph, i.e., as an instance of the relational schema:

$$Ref\ (source\text{: }\mathbf{oid},\ label\text{: }\mathbf{label},\ destination\text{: }\mathbf{oid})$$

where **oid** and **label** are (countable) infinite, disjoint sorts. Intuitively, an object corresponds to a Web page. Labeled edges model labeled links among pages. More precisely, $Ref(o_1, l, o_2)$ indicates that there is an edge/link labeled l ¿from o_1 to o_2. The graph represented by *Ref* is not restricted to be finite (see discussion below). However, in agreement with what is found on the Web, each vertex is of finite outdegree. More precisely, for each object o there are (at most) finitely many tuples in I with o in the first column. The *description* of o in I consists of this finite set of tuples. Thus, the description of an object provides its *outgoing* links. On the other hand, there may be infinitely many objects pointing to some object o, i.e., o may have an infinite indegree.

We call a relation I over *Ref* restricted as above a *Web instance* (an *instance*). We say that object o' is reachable ¿from object o if there is a directed path from o to o' in the labeled graph given by I. The *distance* between two objects is also defined with respect to the I graph.

Note that we consider infinite instances, a departure from database custom. It turns out that viewing the Web as infinite may have certain advantages, as discussed at length in [AV97]. The infiniteness assumption captures the intuition that exhaustive exploration of the Web is (or will soon become) prohibitively expensive. Such a model leads to a focus on querying and computation where exploration of the Web is controlled. However, unlike [AV97], the investigation in the present paper is not tied to the infiniteness assumption. We consider instead both the finite and infinite cases. It turns out that most of our results are independent of (in)finiteness assumptions.

Regular path queries We next recall the notion of regular path query. In the paper, we assume familiarity with basic notions of formal language theory, such as regular expressions and regular languages, (nondeterministic) finite state automata (n)fsa, context-free languages, and pushdown automata (pda), see [HU79].

Let I be a Web instance. A *(regular) path query* is a regular expression over some finite alphabet Σ included in **label**. In keeping with usual notation for regular expressions, "+" represents union and "*" the Kleene closure. Examples of path queries are:

```
section subsection ( paragraph + figure ) caption
engine ( subpart )* name
```

Path queries on the Web are navigational, and are posed relative to some designated source vertex. Thus, the semantics of a path query is determined by an input pair (o, I), where I is a Web instance and o is an object in I. The *answer* of a path query p on input (o, I) is the set of all objects o' reachable from o by some path whose labels spell a word in p. More precisely, o' is in $p(o, I)$ if there is a directed path from o to o' whose edges are labelled $l_1, ..., l_m$ for some word $l_1 ... l_m$ in $L(p)$ (where $L(p)$ denotes the regular language defined by the regular expression p). Two path queries p and q are *equivalent* if $p(o, I) = q(o, I)$ for every input (o, I). Clearly, this holds if only if $L(p) = L(q)$.

Note that if I is finite, $p(o, I)$ is finite and computable (in polynomial time). If I is infinite, $p(o, I)$ may be finite or infinite, and $p(o, I)$ is no longer computable in the usual sense. To model this situation, we developed in [AV97] the notion of *eventually computable* query. The intuition is as follows. Recall that although I is infinite, the description of each object is finite. We also assume that given the description of an object o, one can effectively obtain the description of any object o' such that there is a link from o to o'. Thus, one can follow links from one object to another. A path query p can then be evaluated on input (o, I) by following links starting from o. In general, the possibly infinite answer to the query is never fully computed. However, every object in the answer is eventually produced given enough time. Thus, we say that p is eventually computable. The formal definitions can be found in [AV97].

Extended path queries Our model provides a bare-bones abstraction of the Web and of some query languages recently proposed for the Web. It is worth noting that our framework can be easily adapted to capture some additional aspects not explicitly included in the model. For example, some languages with path expressions (such as Lorel [AQM+96]) view labels as strings of characters, and use regular expressions that work at two levels of granularity: the label (viewed as a string of characters) and the path (viewed as a sequence of labels). For instance, consider the following *extended* path expression:

```
"doc" ( "[sS]ections?" "text" + "[pP]aragraph" )
```

which specifies a path starting with an edge labeled *doc* either followed by an edge labeled *section(s)* (possibly with a capital S to start) and a *text*-edge, or followed by an edge labeled paragraph (possibly with a capital P).

We used here a syntax based on grep E-regular expressions for string patterns, and quotes to separate labels (strings of characters) from paths (sequences of labels).

Call such queries *extended path queries*. We claim that these can essentially be captured by our framework, modulo some preprocessing of labels. Let q be an extended path query and let Π be the set of string patterns occurring in q. We will reduce the problem of the evaluation of the extended path query q on an instance I with possibly infinitely many labels, to the problem of the evaluation of a regular path query $\mu(q)$ on an instance $\mu(I)$ with finitely many labels. For this, consider the equivalence relation on strings defined by: $v \equiv v'$ if v and v' satisfy precisely the same patterns in Π. For each equivalence class $[v]$, let $l_{[v]}$ be a distinct new label and let Σ be the set of such labels. Observe that Σ is finite. Now μ is defined as follows:

1. for each label/string v, $\mu(v) = l_{[v]}$ and $\mu(o) = o$ for each vertex o in I. This defines $\mu(I)$.

2. for each string pattern s, let $\mu(s) = l_{[v_1]}+...+l_{[v_k]}$ where $[v_l], ..., [v_k]$ are the equivalence classes of words satisfying s (i.e., $[v_j] \subseteq L(s)$). This defines $\mu(q)$.

Now we have:

Fact: For each q, o, I and μ as above, $q(o, I) = \mu(q)(o, \mu(I))$.

This allows us to reduce the problem of the evaluation of an extended path query involving potentially infinitely many labels to the evaluation of a regular path query on a finite alphabet of labels, via preprocessing of labels. In the remainder of the paper, we only consider regular path queries.

To conclude this section, we briefly mention another extension. In the Web, pages have *content*. In our context, a page with a string `w` as content can be modeled by a vertex o with outgoing edge labeled `"content=w"` pointing to o itself. Now content-based selections can be specified using the extended path expressions just discussed. For instance, the reachable vertexes that contain the word "SGML" can be retrieved using the extended path query

```
( "(.)*" )* "content=(.)*SGML(.)*"
```

where "(.)*" indicates some arbitrary sequence of characters.

Distributed evaluation of path queries

We next outline a distributed evaluation algorithm for path queries, motivated by the distributed nature of the Web.

We first recall some concepts for regular expressions. For each regular expression p over some Σ and each label l in Σ, the set $\{w \mid l\ w \in L(p)\}$ is regular. We denote by $l|p$ (the left-quotient of p by l) a regular expression for that language. Observe that $L(p) = L(l\ q)$ if $q = l|p$ and that the set of languages that one can construct from a regular language by repeatedly taking such quotients is finite (indeed, an fsa for $l|p$ is obtained simply by changing the start state of the fsa for p).

As outlined in the introduction, we are motivated by a natural scenario for processing path queries in a *distributed environment with asynchronous communication.* In this scenario, objects represent sites. A path query p on input (o, I) is initiated by sending the query to site o. The processing of a path query involves local processing at each site and simple communication between sites. To simplify, we consider the processing of a single query. (If more queries are processed simultaneously, it suffices to prefix all messages with an identifier for the query.) To start with, we assume communication is by messages of the following form:

Q (*query_source, subquery*) and A (*result*).

We assume that every message eventually reaches its destination. The computation of a query p on input (o, I) is initialized by sending to o the message $Q(o, p)$. Each object keeps a list of the messages it receives. When object o_1 receives a subquery $Q(o, q)$ that it has not already processed:

- it sends $A(o_1)$ to o if ε is in $L(q)$.

- for each label l such that $L(l|q) \neq \emptyset$, and for each o_2 with an l-edge from o_1 to o_2, o_1 sends $Q(o, l|q)$ to o_2.

Thus, objects return themselves to the source whenever they find out that they are in the answer to the original query; then they process the first letter(s) of the query and ask their neighbors accesible by links labeled with the appropriate letter to continue the work.

Fact: On input I, for a query $q(o, I)$, object o will receive a (possibly infinite) sequence $A(\omega_1), A(\omega_2), ...$ of messages and $q(o, I) = \{\omega_i\}$.

The basic algorithm just described ensures that all objects in the answer to the query are eventually returned to the source. On the other hand, it does not detect termination. This can be fixed with a slight modification of the basic algorithm, described in Appendix **A.1**.

Remark 2.1 The distributed algorithm outlined above is in the spirit of a Java crawler. In contrast, existing Web crawlers take, for the time being, a centralized approach. In particular, the http protocol does not allow carrying information when travelling from site to site. □

Path queries and Datalog

We point to an analogy between the evaluation of path queries and Datalog evaluation techniques. It is clear that a path query can be expressed as a program in Datalog augmented with some built-in relations providing information related to regular expressions. Two such relations are sufficient: *contains-*ε and *quotient*. For regular expressions p, q and a label l, *contains-*$\varepsilon(p)$ indicates that $\varepsilon \in L(p)$ and *quotient*(p, l, q) indicates that $q = l|p$ (i.e., $L(p) = L(l\ q)$). Using these predicates, one can write the following Datalog program $\mathcal{P}_1$ (P, Q are variables standing for regular expressions and X, X', X'' are variables representing objects) that defines the answer to $p(o, I)$:

```
answer(X,P,X)      :- contains-ε(P)
answer(X,P,X”)     :- Ref(X,L,X’), quotient(P,L,Q), answer(X’,Q,X”)
result(X)          :- answer (o,p,X)
```

Here $answer(X, Q, X')$ indicates that $X' \in Q(X, I)$. Clearly, the $result$ relation defined by the program is $p(o, I)$. However, the fixpoint bottom-up evaluation of this program is not "practical": it involves vertexes and queries that are completely irrelevant to $p(o, I)$; furthermore, even if $p(o, I)$ is finite, the above query may not terminate. A more practical program is obtained by a rewriting of the first in the spirit of Magic Set rewriting [BMSU86] or Query-Subquery evaluation [Vie87]. This would yield the following Datalog program $\mathcal{P}_2$:

```
interesting(o,p)       :-
interesting(X’,Q’)     :- interesting(X,P), Ref(X,L,X’), quotient(P,L,Q)
result(X)              :- interesting(X,P), contains-ε(P)
```

The relation *result* defined by this program is again $p(o, I)$. However, its bottom-up evaluation now involves only potentially relevant objects reachable from o by prefixes of words in $L(p)$. In fact, its bottom-up evaluation proceeds much like our distributed evaluation algorithm. In particular, it terminates if $p(o, I)$ is finite. We note that such properties were studied in detail in [AV97]. In particular, we characterized there Datalog programs which can be evaluated by following links from a source object. This led to a syntactic restriction called *source safety*, which ensures this property. For example, the second program above is source safe.

Optimization of path queries

The basic distributed processing algorithm can be improved in many ways by taking into account additional information that might be available. In keeping with the spirit of the distributed scenario, we assume such information is local to each site. More precisely, we assume that an object o may have local information of the form $p = q$ or $p \subseteq q$, meaning that $p(o, I) = q(o, I)$ or $p(o, I) \subseteq q(o, I)$. We refer to such properties as *path constraints*.

Path constraints may reflect various kinds of information. First, they may reflect structural information about neighboring Web pages. For example, consider the two paths:

p1 = `CS-Department DB-group Member Ullman Classes-Taught cs345`
p2 = `CS-Department Courses cs345`

It may be the case that starting from some site `Stanford`, the paths $p1$ and $p2$ lead to the same object. Thus, the path constraint $p1 = p2$ holds at site `Stanford`. Similarly, at the site `CS-Department` one could have the constraint Σ^* `CS-Main-Page` $= \epsilon$ stating that all paths starting at site `CS-Department` whose final label is `CS-Main-Page` lead back to that site.

Path constraints also naturally arise from caching frequently asked queries. More precisely, the answer to query q at site o could be saved and accessed from o by links labeled l_q. This would yield the equation $q = l_q$ and a rapid way to evaluate q by simply evaluating l_q. Similar constraints arise from the presence of "mirror sites", which are duplications of frequently accessed sites.

How can path constraints be used? The hope is that they may allow more efficient evaluation of path queries. For instance, the query may ask for the page $p1$ (as above) and the system may decide to substitute it with the page $p2$ if this page is available locally and it is known that it contains the same information.

So, in general, the query processor at each site may use the path constraints holding at the site to replace the query to be executed by a simpler query. We are not concerned here with what "simpler" means; this could potentially involve a cost measure using information not captured by our basic model, such as locality information, cost of accessing different sites in the network, etc. Regardless of the cost measure, the basic problem laying at the core of this approach is testing implication of relationships among queries by the given constraints. Thus, we must be able to answer the following question:

> given a finite set E of path constraints of the form $p_i = q_i$ or $p_i \subseteq q_i$ and two path queries p, q, is it true that $p(o, I) = q(o, I)$ or $p(o, I) \subseteq q(o, I)$ for each (o, I) satisfying E?

We examine this problem in detail in the next section. In the remainder of this section we illustrate how such inferences might be used in query optimization.

Examples

1. Suppose we know that every path ending in l returns to the source site, i.e. $\Sigma^* l = \varepsilon$. Suppose query $p = (la + lb)^* d$ must be executed at this site. It can be shown that p is equivalent to $(a + b)d$. This query is likely to be simpler than the original; in particular, it is non-recursive and so is guaranteed to terminate.

2. Suppose the path constraint $ll \subseteq l$ holds at the source site. Consider the query $p = l^*$. It can be shown that $l^* = l + \varepsilon$ so p can be replaced by the query $l + \varepsilon$.

3. Suppose the query $(ab)^*$ has been cached and labeled l, so that the constraint $l = (ab)^*$ holds. Consider the query $p = a(ba)^* c$. One can show that $p = lac$. In other words, p can be evaluated by sending the query ac to the cached objects.

4. Suppose query $(aa)^*$ has been cached. Suppose that the queries $(aaa)^*$ and a^* are launched at this site. Cached data is not sufficient to answer the two new queries. However, $a^* = a + (aa)^* + (aaa)^*$. Thus it suffices to evaluate $(aaa)^*$ and a and then use the cached data to evaluate a^*.

3 Implication of regular path constraints

In this section, we consider the implication problem for path constraints. We first formalize the problem and relate it to well-known problems in rewrite systems and logic. We show the decidability in the general case with EXPSPACE complexity. We then study several natural special cases. These concern constraints between "words" instead of arbitrary regular expressions. We are able to obtain decision procedures of complexity PTIME for the implication of word constraints, and of complexity PSPACE for implication of path constraints by word constraints. As a side effect we develop tools that are of interest in their own right. For example, we use them to show that, given a finite set of word equalities, the boundedness problem for path queries is decidable; that is, it is decidable if a path query is equivalent to a non-recursive path query given a finite set of word equalities.

Path constraints

We now formalize the implication problem for path constraints and mention related problems in logic and rewriting systems. In the following we fix a finite set of labels Σ (See Section 2.)

Definition 3.1 A *(regular) path inclusion* is an expression of the form $p \subseteq q$ where p, q are regular expressions over Σ. An instance (o, I) *satisfies* a path inclusion $p \subseteq q$, denoted $(o, I) \models [p \subseteq q]$, if $p(o, I) \subseteq q(o, I)$; (o, I) satisfies a set E of path inclusions, denoted $(o, I) \models E$, if it satisfies each inclusion in E. A finite set of path inclusions *implies* a constraint $p \subseteq q$, denoted $E \models [p \subseteq q]$, if for each instance (o, I) such that $(o, I) \models E$, $(o, I) \models [p \subseteq q]$.

If p, q are *words*, i.e., simply sequences of labels, the path inclusion $p \subseteq q$ is called a *word inclusion* (e.g., $a\ b\ c \subseteq d\ e$). The expressions obtained by replacing $\subseteq$ by $=$ are called, respectively, *path equalities* (e.g., $a\ (b + c)* = d\ e$) and *word equalities* (e.g., $a\ b\ c = d\ e$). A *path constraint* is a path inclusion or a path equality, and similarly for *word constraint*. Equality constraints can of course be expressed by inclusions constraints, but equality is an important and well-behaved special case.

We start by pointing to two problems in rewrite systems and logic that are related to the implication problem for path constraints.

Rewrite systems Consider first word inclusions. Suppose that we know $u_1 \subseteq u_2$ and $u_2u_3 \subseteq u_4$. Then it seems natural to infer, for instance, that $u_1u_3u_5 \subseteq u_2u_3u_5 \subseteq u_4u_5$. One can look at this as rewriting the word $u_1u_3u_5$ using rewrite rules $u_1 \to u_2$ and $u_2u_3 \to u_4$. We will present a rewrite system that is sound and complete for word constraints. This will then be used to obtain a decision procedure for this case. Note that in the general case, one cannot decide whether a word can rewrite into another word using an arbitrary system of rewrite rules (a semi-Thue system) [HU79]. Our case differs from the general case in that rewrite rules are applied only to prefixes of words. See [DJ90] for a comprehensive survey of rewrite systems.

First-order logic with 2 variables In the particular context of word constraints, the implication problem can be stated in terms of first-order logic. Moreover, only two variables are needed. Then the decidability of the implication problem for word constraints follows from known results about first-order logic with two variables (FO^2). Indeed, satisfiability of FO^2 sentences is decidable [Mor75], and the implication problem for word constraints can be reduced to satisfiability of an FO^2 sentence. However, the complexity of testing FO^2 satisfiability is doubly exponential in the formula [Mor75] and exponential in the model size [GKV]. In contrast, our direct proof provides a PTIME test for word constraint implication (in the size of the words). Furthermore, results about FO^2 and its extensions are no longer of help for implication of full path constraints, where recursion is present in the form of the Kleene closure. Indeed, for the extensions of FO^2 with recursion/fixpoint that have so far been studied, satisfiability was shown to be undecidable [GOR]. In this light, decidability of implication for path constraints comes as a welcome surprise.

Path constraint implication

In this section, we study path constraint implication. We first prove that implication of path constraints is decidable (in EXPSPACE). The idea of the proof is to show that if an implication $E \models p \subseteq q$ is violated by an instance (finite or infinite), then one can find a finite instance witnessing the violation whose size is bounded by an exponential in the size of E, p, q (the construction of such an instance is outlined in the Appendix). Observe that this also demonstrates that for path constraints, finite and unrestricted implication coincide.

Theorem 3.2 (1) If $[\wedge_{i\in[1..m]}\ p_i \subseteq q_i] \not\models p_0 \subseteq q_0$, then there is some instance (ω, J), of size exponential in the total size of $\{p_i, q_i\}_{i\in[0..m]}$, such that $(\omega, J) \models [\wedge_{i\in[1..m]}\ p_i \subseteq q_i]$ and $(\omega, J) \not\models p_0 \subseteq q_0$. (2) Implication of path constraints is decidable in EXPSPACE.

Although the above result shows the decidability of implication, the test of implication it provides has some drawbacks. First, its complexity is high. Second, it does not provide real insight into the interplay of path constraints. Such insight might be better served by a sound and complete axiomatization of path constraint implication. However, obtaining such an axiomatization appears to be highly nontrivial. Note that even an axiomatization of classical regular expression equivalence (in the absence of constraints) is far from obvious (see the set of axioms provided in [Sal66]).

Word constraints We next consider some particular cases of the implication problem. We show that for word constraints, implication is decidable in PTIME. We are then able to extend this result to implication of full path constraints by word constraints, with PSPACE complexity. Note that deciding the equivalence of regular expressions is by itself PSPACE-complete (in absence of constraints) [GJ79], so this is the best one can do. Finally, we consider the special case of word equality.

Whenever we consider a finite set E of word inclusions, we will assume that if $u \subseteq \epsilon$ is in E, the constraint $\epsilon \subseteq u$ is also in E. This is convenient because $\epsilon(o, I)$ always consists of the single vertex o, so $u \subseteq \varepsilon$ and $\varepsilon \not\subseteq u$ would imply that $u = \emptyset$. This would introduce a new category of emptiness constraints that we wish to avoid.

We will prove the following:

Theorem 3.3 (i) Implication of a word constraint by a set if word constraints can be tested in PTIME. (ii) Implication of a path constraint by a set of word constraints can be tested in PSPACE.

The proof of the theorem requires four lemmas and involves a *rewrite system* of words. We associate to each inclusion $u \subseteq v$ in E a rewrite rule $u \xrightarrow{E} v$. Let $\xrightarrow{E}$ be the binary relation on words defined as follows: $z \xrightarrow{E} t$ iff there is a finite sequence of words $w_1 \ldots w_n$ (for $n \geq 1$) such that $z = w_1, t = w_n$, and for each $i, 1 \leq i < n$, $w_i = xw$ and $w_{i+1} = yw$ for some $x \subseteq y$ in E and some word $w \in \Sigma^*$. It is useful to note that $\xrightarrow{E}$ is the reflexive, transitive, right-congruent closure of E.

The first lemma we prove provides a connection between implication of word constraints and derivation by the corresponding rewrite system.

Lemma 3.4 Given a finite set E of word constraints, $\xrightarrow{E}$ is sound and complete for implication of word constraints. That is, for each E and $u, v \in \Sigma^*$, $E \models u \subseteq v$ iff $u \xrightarrow{E} v$.

The soundness of rewriting is immediate. Completeness is proved by constructing an instance (o, I) that essentially captures implication $u \subseteq v$ for u, v of length bounded by some fixed k. More precisely, (o, I) satisfies E and for all u, v of length at most k, $(o, I) \models u \subseteq v$ iff $u \xrightarrow{E} v$. Note that the boundedness restriction cannot be removed. Indeed, there exists a finite set of constraints E such that there is no (finite or infinite) instance satisfying *exactly* the constraints implied by E. To see an example, let $E = \{a^2 \subseteq a\}$. Observe that E implies in particular: $... \subseteq a^i \subseteq a^{i-1} ... \subseteq a$ but not $a^i = a^{i-1}$. However, in each fixed instance there are only finitely many outgoing edges from the source, so each instance satisfying $a^2 \subseteq a$ must also satisfy $a^i = a^{i-1}$ for some i.

The PTIME bound on testing word implication is obtained using the next lemma that focuses on the set of all words that rewrite to a particular word v. More precisely, consider the set of words $\mathit{RewriteTo}(v) = \{u \mid u \in \Sigma^*, u \overset{E}{\to} v\}$. The lemma shows that $\mathit{RewriteTo}(v)$ is a regular language. To see this, note first that one can easily build a pushdown automaton (pda) that accepts $\mathit{RewriteTo}(v)$. The pda works as follows. It first puts the input word u on the stack, then starts simulating the rewrite rules by rewriting prefixes of the stack (using pda moves). The pda is very particular in that it first reads its entire input and places it on the stack. The crux of the proof consists in showing that such a pda can actually be simulated by an nfsa (see Appendix).

Lemma 3.5 Let E be a finite set of word constraints and v a word in Σ^*. The set $\mathit{RewriteTo}(v) = \{u \mid u \in \Sigma^*, u \overset{E}{\to} v\}$ is a regular language recognized by an nfsa constructible in polynomial time from E and v. In particular, $u \overset{E}{\to} v$ can be decided in PTIME.

Lemmas 3.4 and 3.5 together provide the PTIME test for implication of word constraints, and thus prove (i) of Theorem 3.3. To show part (ii) of the theorem, we use two additional lemmas. The first relates implication of path constraints to implication of word constraints:

Lemma 3.6 Let E be a finite set of word constraints and p, q regular expressions. If $E \models p \subseteq q$ then for each $u \in L(p)$ there exists $v \in L(q)$ such that $E \models u \subseteq v$.

It is worth noting that generally an instance (o, I) may satisfy $p \subseteq q$ without it being the case that each word u in $L(p)$ is included in some word v in $L(q)$ (e.g., consider $a \subseteq b + c$, or $a \subseteq b^*$). The above lemma shows however that this must happen *if $p \subseteq q$ is implied by a finite set of word constraints.*

The PSPACE bound is obtained using an extension of Lemma 3.5:

Lemma 3.7 Let E be a finite set of word constraints and p a regular expression over Σ. The set $\mathit{RewriteTo}(p) = \{u \mid u \in \Sigma^*, \exists v \in L(p)(u \overset{E}{\to} v)\}$ is a regular language recognized by an nfsa constructible in polynomial time from E and v.

By Lemmas 3.6 and 3.7, $E \models p \subseteq q$ iff $L(p) \subseteq \mathit{RewriteTo}(q)$. This provides the PSPACE test of implication and proves part (ii) of Theorem 3.3.

Word equalities What is different about equality? Obviously, decidability of the implication problem for path inclusions yields a decision procedure for implication of path equalities. More precisely, this yields a PTIME test for implication of word equalities, a PSPACE test for the implication of path equalities by word equalities, and an EXPSPACE test for implication of path equalities. However, it turns out that equality has some remarkably nice properties. These are due to the following technical facts:

1. for each finite set E of word equalities there exists a "true" Armstrong instance for E, i.e. an infinite instance that satisfies *precisely* the path equalities implied by E; and,

2. the interesting information contained in the Armstrong instance for E occurs at bounded distance from the source, which allows to compute a finite "summary" of the Armstrong instance.

The existence of the Armstrong instance is shown next.

Proposition 3.8 Let E be a finite set of word equalities. There exists an instance (o, I) (that we call *the Armstrong instance of* E) such that for each u, v, $u(o, I) = v(o, I)$ iff $E \models u = v$.

Proof: The instance (o, I) is built as follows. Let $\equiv$ be the smallest equivalence relation over Σ^* that contains E and is a right-congruence. The set of vertexes in I consists of the equivalence classes of $\equiv$, and $o = o_{\widehat{\epsilon}}$. For each u, a, there is an a-edge from $\widehat{u}$ to $\widehat{ua}$.

First observe that this is indeed an instance, i.e., it has finitely many outgoing edges from every vertex: if $\widehat{u} = \widehat{v}$, then $\widehat{ua} = \widehat{va}$ (right congruence), so there is a single outgoing a-edge from every vertex. Also note that $u(o, I) = \widehat{u}$. Therefore $u(o, I) = v(o, I)$ iff $\widehat{u} = \widehat{v}$ iff $u \overset{E}{\leftrightarrow} v$. By Lemma 3.4 $u \overset{E}{\leftrightarrow} v$ iff $E \models u = v$. Thus, $u(o, I) = v(o, I)$ iff $E \models u = v$. $\square$

The Armstrong instance for E is generally infinite. However, we show that all interesting information is contained at some bounded distance from the source. Given an instance (o, I), let the *K-sphere* (*around* o) consist of the restriction of I to vertexes at distance at most K from o. Using this, the structure of the Armstrong instance is very particular and can be captured as follows (see Figure 1):

Lemma 3.9 Let E be a finite set of word equalities and (o, I) the Armstrong instance for E. There exists an integer K such that each vertex outside the K-sphere has indegree 1, and there is no edge with tail outside and head inside the K-sphere.

The significance of the above property is that all word equalities implied by E follow by right-congruence from equalities of words leading to vertexes inside the K-sphere. Thus, all "interesting" information can be found within the K-sphere around o.

This provides a valuable tool for reasoning about implication of path equalities by word equalities, and dramatically simplifies a number of problems. Due to space limitations, this cannot be explored in detail here. We show however how the Armstrong relation can be used to solve the question of boundedness of a path query under given word equalities. We prove that it is decidable whether a path query is equivalent to a query without recursion assuming a given set of word equalities. Furthermore, an equivalent finite query can be effectively constructed if such a query exists. As illustrated in Section 2, this a problem of significant practical interest.

Theorem 3.10 It is decidable, given a finite set E of word equalities and a regular path expression p, whether $E \models [p = q]$ for some regular path query q where $L(q)$ is finite. Furthermore, such q can be constructed in EXPTIME from E and p.

Proof: **(sketch)** Consider the Armstrong relation (o, I) associated with E, and the K-sphere constructed in Lemma 3.9. Recall that all vertexes outside the K-sphere have indegree one and no path that leaves the K-sphere ever returns. This means that all paths leaving the K-sphere which are distinct outside the K-sphere lead to distinct vertexes. So p is bounded iff the set of words in p that yield distinct paths outside the K-sphere is finite. This can be tested by associating an fsa F with (o, I) as follows. Its states are all the vertexes in the K-sphere plus one new state *out*. Its transitions are all the labeled edges within the K-sphere plus, for each o', one a-edge (o', out) if there is an a-edge going from o' to some vertex *outside* the K-sphere. Finally, for each a, there is an a-edge from *out* to *out*. The start state of F is o and the accepting state is *out*. It is easily seen that p is bounded iff $L(p) \cap L(F)|\Sigma^*$ is a finite language. This is decidable, and yields an EXPTIME algorithm to find an equivalent finite path query if p is bounded. □

It remains open whether boundedness of a path query assuming a set of full path constraints is decidable.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading-Massachusetts, 1995.

[AQM+96] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data, 1996. `ftp://db.stanford.edu//pub/papers/lorel96.ps`.

[AV97] S. Abiteboul and V. Vianu. Queries and computation on the Web. In *Proc. of Intl. Conf. on Database Theory*, 1997. To appear. Available at `http://www-cse.ucsd.edu/users/vianu`.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 505–516, 1996.

[BMSU86] F. Bancilhon, D. Maier, Y. Sagiv, and J.D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–15, 1986.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 313–324, 1994.

[CM90] M. Consens and A. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proc. ACM Symp. on Principles of Database Systems*, pages 404–416, 1990.

[dBV93] J. Van den Bussche and G. Vossen. An extension of path expressions to simplify navigation in object-oriented queries. In *Proc. of Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 267–282, 1993.

[DJ90] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 243–320. Elsevier, 1990.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractibilitiy: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[GKV] E. Graedel, P. Kolaitis, and M.Y. Vardi. On the complexity of the decision problem for two-variable first-order logic. To appear.

[GOR] E. Graedel, M. Otto, and E. Rosen. Undecidability results for two-variable logics. Extended abstract to appear in STACS'97.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[KKS92] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 393–402, 1992.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 54–65, 1995.

[MMM96] A. Mendelzohn, G. A. Mihaila, and T. Milo. Querying the World Wide Web. In *Proc. PDIS*, 1996.

[Mor75] M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math*, 21:135–140, 1975.

[MW95] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comp.*, 24(6), 1995.

[QRS+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. Technical report, Stanford University, 1995. Available by anonymous ftp from `db.stanford.edu`.

[Sal66] A. Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966.

[Suc96] D. Suciu. Query decomposition and view maintenance for query languages for unstructured data. In *Proc. of Intl. Conf. on Very Large Data Bases*, pages 227–238, 1996.

[Vie87] L. Vieille. Recursive query processing: the power of logic. *Theoretical Computer Science*, 69(1):1–53, 1987.

Appendix

A.1 Protocol for termination detection

We describe an extension of the basic distributed algorithm for evaluation of path queries which detects termination whenever possible. We assume that all messages have distinct identifiers and we require that messages be acknowledged. When an object o_1 receives a message $\langle messageX, Q(o,p)\rangle$ from some object o', if it has already received that same query, o_1 simply acknowledge the message to o' and does nothing else. Otherwise, it generates a number of messages and when they have all been acknowledged, it sends an acknowledgement for *messageX* to o'. Note that the source query o should also send acknowledgements for each A message it receives. The processing of $p(o,I)$ now works as follows: one sends $Q(o,p)$ to o and when o acknowledges (if it does), terminatation is detected. A possible run of this algorithm with a graph I consisting of 4 vertices o_1, o_2, o_3, o_4 with an a-edge from o_1 to o_2, and b-edges from o_2 to o_3, and from o_3 to o_2 and o_4; and with the query $ab^*(o_1, I)$ is as follows:

#01($\rightarrow o_1$)	Q(o_1,ab^*)				
	#11($o_1 \rightarrow o_2$)	Q(o_2,b^*)			
		#21($o_2 \rightarrow o_1$)	A(o_2) ack#21		
		#22($o_2 \rightarrow o_3$)	Q(o_3,b^*)		
			#31($o_3 \rightarrow o_1$)	A(o_3) ack#31	
			#32($o_3 \rightarrow o_2$)	Q(o_2,b^*) ack#32	
			#33($o_3 \rightarrow o_4$)	Q(o_4,b^*)	
				#41($o_4 \rightarrow o_1$)	A(o_4) ack#41
				ack#33	
			ack#22		
		ack#11			
	ack#01/termination detected				

Fact: The previous algorithm terminates iff the instance is finite or if the set of o' such that for some prefix u of some word in p there is a path from o to o' is finite. Furthermore the message *termination-detected* occurs exactly when the algorithm terminates, after having computed the proper answer.

A.2 Proof sketches

Proof of Theorem 3.2 Let (o,I) be an instance (possibly infinite) such that $(o,I) \models [\wedge_{i\in[1..m]}\ p_i \subseteq q_i]$ and $(o,I) \not\models p_0 \subseteq q_0$.

Consider the nfsa's for the p_i, q_i, $i \in [0..m]$, and the nfsa F that is the product of these nfsa's. Let f be the start state of F and δ_F be its transition function. For each set S of states in F, let o_S be a distinct new vertex. For each vertex o' in I, let $states(o') = \{s \mid$ there is a path u from o to o' such that $s \in \delta_F(f,u)\}$. Consider the graph homomorphism μ that replaces each vertex o' by o_S where *S = states(o')*. Let $\mu(o) = \omega$ and $\mu(I) = J$.

We prove that for each $p = p_i$ or q_i, $0 \leq i \leq m$, and each o', (†) $o' \in p(o,I)$ iff $\mu(o') \in p(\omega, J)$. For suppose that this holds. Then for $i \geq 1$, $p_i(\omega,J) = \mu(p_i(o,I)) \subseteq \mu(q_i(o,I)) = q_i(\omega,J)$; and for o' in $p_0(o,I) - q_0(o,I)$, $\mu(o')$ in $p_0(\omega,J) - q_0(\omega,J)$, so (1) is proved.

Consider (†). Clearly, it is sufficient to show that for each vertex o' in I, $states(o') = states(\mu(o'))$. The inclusion $states(o') \subseteq states(\mu(o'))$ follows immediately by the definition of homomorphism. Consider the inclusion $states(\mu(o')) \subseteq states(o')$. Let $s \in states(\mu(o'))$. There exists a path u from ω to $\mu(o')$ such that $s \in \delta_F(f, u)$. We prove by induction on $|u|$ that $s \in states(o')$. If $u = \varepsilon$ then $s = f$ and $o' = o$. Since $f \in states(o)$ it follows that $s \in states(o')$. Now suppose $u = v\ a$ with $a \in \Sigma$, and the statement holds for words shorter than u. Let f' be a state in $\delta(f, v)$ such that $s \in \delta(f', a))$. There exist vertexes o_1, o_2 in I such that there is an a-link from o_1 to o_2, $\mu(o_2) = \mu(o')$, and $\mu(o_1) \in v(\omega, J)$. By the induction hypothesis, $states(\mu(o_1)) \subseteq states(o_1)$ so there exists v' such that $f' \in \delta_F(f, v')$ and $o_1 \in v'(o, I)$. Consider the path $v'a$ in I; we have that $s \in \delta_F(f, v'a)$ and $v'a$ is a path from o to o_2. Thus, $s \in states(o_2) = states(o')$. This proves (†).

To summarize, we constructed a finite instance (ω, J) satisfying $\wedge_{i\in[1..m]}\ p_i \subseteq q_i]$ and violating $p_0 \subseteq q_0$. Furthermore, the size of (ω, J) is bounded by an exponential in $|E| + |p_0| + |q_0|$. Thus, one can test implication by considering all instances up to this size, which takes EXPSPACE. □

Proof of Lemma 3.4 It is quite obvious that if $u \xrightarrow{E} v$ then $E \models u \subseteq v$ (soundness of rewriting). To prove the converse (completeness), we show that

(†) for each k, there is a finite instance (o, I) that satisfies E and such that for each u, v shorter than k, if $(o, I) \models u \subseteq v$ then $u \xrightarrow{E} v$.

For suppose (†) holds and $E \models u \subseteq v$. Let k be larger than u, v and (o, I) the instance provided for by (†). Since $E \models u \subseteq v$ and $(o, I) \models E$, $(o, I) \models u \subseteq v$. By (†), $u \xrightarrow{E} v$.

To prove (†), let $\approx$ be the equivalence relation on Σ^* defined by $u \approx v$ iff $u \xrightarrow{E} v$ and $v \xrightarrow{E} u$. Let $\widehat{u}$ denote the equivalence class of a word u with respect to $\approx$. Let $\prec$ be the partial order on the equivalence classes of $\approx$ defined by $\widehat{u} \prec \widehat{v}$ iff $u \xrightarrow{E} v$ (note that this is well defined).

Let $\mathcal{C} = \{\widehat{u} \mid |u| \leq k\}$. We build an instance (o, I) by "populating" each class $\widehat{u}$ of $\mathcal{C}$ with a finite set of vertexes $obj(\widehat{u})$ such that $u(o, I) = obj(\widehat{u})$, as follows. For each $\sigma \in \mathcal{C}$, let o_σ be a distinguished vertex. Let $obj(\sigma) = \{o_\psi \mid \psi \in \mathcal{C}, \psi \prec \sigma\}$, for each $\sigma \in \mathcal{C}$. The instance (o, I) is defined as follows: (i) the vertexes are $\{o_\sigma \mid \sigma \in \mathcal{C}\}$; (ii) o is $o_{\widehat{\epsilon}}$; and (iii) for each u, $|u| < k$ and a in Σ, there is an a-edge from $o_{\widehat{u}}$ to each o' in $obj(\widehat{ua})$. It is sufficient to show that

$$(+) \text{ for each } u \in \Sigma^*,\ |u| \leq k,\ u(o, I) = obj(\widehat{u}).$$

For suppose that (+) holds. Then $(o, I) \models u \subseteq v$ implies $u(o, I) \subseteq v(o, I)$ implies $obj(\widehat{u}) \subseteq obj(\widehat{v})$ implies $u \prec v$ implies $u \xrightarrow{E} v$.

We prove (+) by induction. First observe that $\widehat{\epsilon}$ is a least element for $\prec$ since for each $u \subseteq \epsilon$, we also have $\epsilon \subseteq u$. So $obj(\widehat{\epsilon}) = \{o\}$ and $\epsilon(o, I) = obj(\widehat{\epsilon}) = \{o\}$. Now suppose (by induction on the size of u) that $u(o, I) = obj(\widehat{u})$ for $|u| < k$ and let a be in Σ. Then $ua(o, I)$ contains $obj(\widehat{ua})$ by construction of (o, I). Now, let o' be in $ua(o, I)$. Then there exists $v \prec u$ (so that $o_{\widehat{v}}$ is in $obj(\widehat{u})$) and an a-edge from $o_{\widehat{v}}$ to o'. By (iii), o' is in $obj(\widehat{va})$. But since $v \prec u$, $va \prec ua$, so $obj(\widehat{va}) \subseteq obj(\widehat{ua})$. Thus, o' is in $obj(\widehat{ua})$, and $ua(o, I) = obj(\widehat{ua})$. This proves (+). □

Proof of Lemma 3.5 It is convenient to consider the language that consists of the reverse of the words in $\mathit{RewriteTo}(v)$: $L_v = \{u^R \mid u \in \mathit{RewriteTo}(v)\}$. We show that L_v is context-free, so $\mathit{RewriteTo}(v)$ is context-free. Indeed, a pushdown automaton (pda) accepting L_v works as follows. First, it places the input u^R on the stack, thus reversing it. Once u is on the stack, the pda nondeterministically performs a sequence of prefix substitutions using the rewrite rules of E. Lastly, the pda guesses that v has been constructed and pops the stack to check this. The computation accepts if v is found on the stack.

To see that L_v is actually regular, so $\mathit{RewriteTo}(v)$ is regular, consider the computation of the pda once u is on the stack. Suppose that at this time the pda is in some state q_0. The pda then adds and removes symbols from the stack until the stack is empty, at which time the pda accepts or rejects. Thus, each symbol x in u eventually becomes the top of the stack and is popped. So, consider the triples $move(q, x, q')$ meaning that the pda, starting in state q with x on the stack, can reach state q' after popping x. We can compute the relation $move$ and then simulate the pda with u on the stack by an nfsa that reads u and has $move$ for transition function.

We still have to show that this can be done in PTIME. The construction of the pda is clearly in PTIME. Now, for each states q, q', symbol x on the stack, we have to decide whether the pda A in state q, with x on the stack may (eventually) reach state q' after popping x. To check that, we transform the pda A into another pda $A(q, q', x)$ that has a s as start state and f as final state (both s, f new). Automaton $A(q, q', x)$ never reads any letter; it first puts x on its stack and goes to state q; it then simulates A; finally, it moves to f when it has an empty stack and is in state q'. Observe that $move(q, x, q')$ holds if ϵ is accepted by $A(q, q', x)$, which can be checked in PTIME. So, the nfsa can be constructed in PTIME. Finally, one can checked whether u is accepted by the nfsa in PTIME. □

Proof of Lemma 3.6 Suppose $E \models p \subseteq q$. Consider a word $u \in L(p)$. Evidently, $E \models u \subseteq q$. We must show that there is some $v \in L(q)$ such that $E \models u \subseteq v$. Let k be an integer larger than the lengths of u and of any word in E. Consider the instance (o, I) constructed for E and k in the proof of Lemma 3.4, and recall the notation developed there. Since (o, I) satisfies E, it must also satisfy $u \subseteq q$. Note that $u(o, I) \neq \emptyset$ and $w(o, I) = \emptyset$ for each w such that $\widehat{w} \notin \mathcal{C}$. It follows that

$$(\star) \quad u(o, I) \subseteq \cup\{w(o, I) \mid w \in L(q), \hat{w} \in \mathcal{C}\}.$$

Recall that by construction there is a distinguished vertex $o_{\hat{u}}$ in $u(o, I)$ such that for all words w with $\hat{w} \in \mathcal{C}$, $o_{\hat{u}} \in w(o, I)$ iff $u(o, I) \subseteq w(o, I)$. This together with $(\star)$ imply that there must exist $v \in q$ such that $\hat{v} \in \mathcal{C}$ and $o_{\hat{u}} \in v(o, I)$, so $u(o, I) \subseteq v(o, I)$. It follows that $u \prec v$, so $u \xrightarrow{E} v$ and $E \models u \subseteq v$. □

Proof of Lemma 3.7 To show that $\mathit{RewriteTo}(p)$ is regular, we use the same technique as in Lemma 3.5. We place u^R on the stack, nondeterministically perform rewritings using E, guess that a word v in $L(p)$ has been constructed and finally check whether v is in $L(p)$. This shows that $\mathit{RewriteTo}(p)$ is a regular language and that an nfsa recognizing it can be constructed in PTIME.

By Lemma 3.6, to verify that $E \models p \subseteq q$ it suffices then to check that $L(p) \subseteq \mathit{RewriteTo}(q)$. We can construct an nfsa F_p for $L(p)$ and F_q for $\mathit{RewriteTo}(q)$ in PTIME with

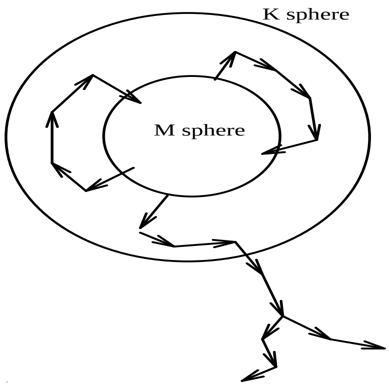

Figure 1: The Armstrong instance

respect to p and q. Next, we can construct in PTIME an nfsa F_{p+q} for $L(p) \cup \mathit{RewriteTo}(q)$. It now suffices to check whether $L(F_q) = L(F_{p+q})$. This can be achieved in PSPACE. (Note that fsa inequivalence is PSPACE-complete by reduction from regular expression non universality [GJ79].) $\square$

Proof of Lemma 3.9 We proceed in two stages. We build a first sphere and then extend it to the desired one (see Figure 1). Let M be the maximum length of of word in E. We prove that:

(*) each vertex outside the M-sphere has indegree 1.

Suppose $\widehat{u}$ is outside the M-sphere and has an incoming a_1-edge from $\widehat{v_1}$ and another incoming a_2-edge from $\widehat{v_2}$. We can assume w.l.o.g. that v_1 and v_2 are the shortest words in their equivalence class. Observe that $|v_1| \geq M$ and $|v_2| \geq M$; otherwise $\widehat{u}$ would be in the M-sphere. Since $\widehat{v_1 a_1} = \widehat{v_2 a_2} = \widehat{u}$, $v_1 a_1 \overset{E}{\rightarrow} v_2 a_2$. Consider the derivation of $v_2 a_2$ from $v_1 a_1$. Observe that $v_1 a_1$ is not allowed to shrink in the course of the derivation, since then $\widehat{u}$ would be in the M-sphere. Since $|v_1|$ is larger or equal than the maximal length of a word occurring in E, the rewriting of $v_1 a_1$ never replaces the last letter. Thus, $a_1 = a_2$ and $v_1 \overset{E}{\rightarrow} v_2$, so $\widehat{v_1} = \widehat{v_2}$.

From (*), one can see that each vertex out of the M-sphere has indegree 1. The lemma is not complete since a path may return to the M-sphere after having left it. However, we show that such paths have bounded length. Then we can find a yet larger sphere that satisfies the lemma.

Consider a path leaving the M-sphere and returning back to it. Let $\widehat{u}$ be the last vertex on the path which is in the M-sphere *before* the path leaves the M-sphere, and $\widehat{v}$ be the first vertex on the path which is next inside the M-sphere. (so $|u| = M, |v| \leq M$). Suppose the path from $\widehat{u}$ to $\widehat{v}$ spells the word w. Thus, no vertex along w is in the M-sphere except the last one which is $\widehat{v}$. We know that the regular language $\mathit{RewriteTo}(v)$ is accepted by an nfsa F that can be constructed from E and v in PTIME. The number N of states in F is polynomial in E and v, i.e. it is polynomial in M. Let us fix $K = M + N$.

We will prove that $|w| \leq N$, so that any point on the path is within the K-sphere. Assume towards a contradiction that $|w| > N$. Observe that $\widehat{uw} = \widehat{v}$, so $uw \in \mathit{RewriteTo}(v)$.

Consider the run of F on input uw. The automaton accepts uw and since $|w|$ is larger than the number of states of F, F goes twice through the same state, say after reading ux and uxy where $w = xyz$ and $y \neq \varepsilon$. We use a pumping argument. The word $uxyyz$ is also accepted by F so $uxyz \stackrel{E}{\leftrightarrow} uxyyz$. Consider the derivation of $uxyyz$ from $uxyz$. Recall that $|u| = M$ and u is the shortest word in $\widehat{u}$. Thus u cannot shrink in the derivation and $u \stackrel{E}{\leftrightarrow} uv'$ for some v' such that $uv'xyz = uxyyz$. It follows that v' is a prefix of xy and so uv' leads to a vertex along the path w. But $\widehat{uv'} = \widehat{u}$ so that vertex is in the M-sphere, which contradicts our assumption that all vertexes along w are outside the M-sphere. Thus $|w| \leq N$, which concludes the proof. $\square$