Extending SPARQL with Regular Expression Patterns (for Querying RDF) ⋆

Faisal Alkhateeb [a], Jean-François Baget [b], Jérôme Euzenat [a]

[a]*INRIA Grenoble Rhône-Alpes and LIG, Montbonnot, France*

[b]*INRIA Sophia-Antipolis Méditéranée and LIRMM, Montpellier, France*

Abstract

RDF is a knowledge representation language dedicated to the annotation of resources within the framework of the semantic web. Among the query languages for RDF, SPARQL allows querying RDF through graph patterns, *i.e.*, RDF graphs involving variables. Other languages, inspired by the work in databases, use regular expressions for searching paths in RDF graphs. Each approach can express queries that are out of reach of the other one. Hence, we aim at combining these two approaches. For that purpose, we define a language, called PRDF (for "Path RDF") which extends RDF such that the arcs of a graph can be labeled by regular expression patterns. We provide PRDF with a semantics extending that of RDF, and propose a correct and complete algorithm which, by computing a particular graph homomorphism, decides the consequence between an RDF graph and a PRDF graph. We then define the PSPARQL query language, extending SPARQL with PRDF graph patterns and complying with RDF model theoretic semantics. PRDF thus offers both graph patterns and path expressions. We show that this extension does not increase the computational complexity of SPARQL and, based on the proposed algorithm, we have implemented a correct and complete PSPARQL query engine.

Key words: semantic web, query language, RDF, SPARQL, regular expression patterns

⋆ This work has been partially supported by the Knowledge Web network of excellence (IST-2004-507482)

Email addresses:
`Faisal.Alkhateeb@inrialpes.fr` (Faisal Alkhateeb),
`Jean-Francois.Baget@inria.fr` (Jean-François Baget),
`Jerome.Euzenat@inrialpes.fr` (Jérôme Euzenat).

1 Introduction

RDF (Resource Description Framework [49]) is a knowledge representation language dedicated to the annotation of documents and more generally of resources within the semantic web. It represents knowledge as a graph relating resources (see Fig. 1). Nowadays, more resources are annotated via RDF due to its simple data model, formal semantics, and a sound and complete inference mechanism. A query language that provides a range of querying paradigms is therefore needed.

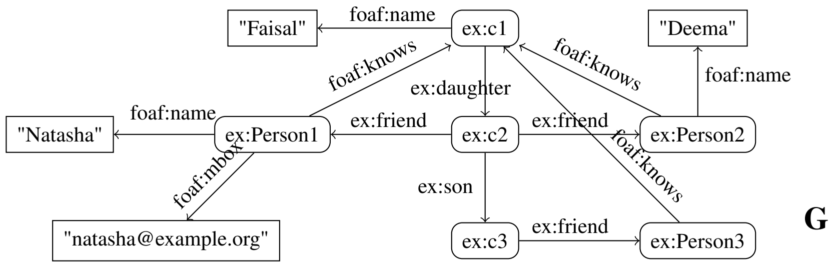


Fig. 1. An RDF graph.

Several languages have been developed for querying RDF (*cf.* [37] for a comparison of query languages for RDF). Among them, SPARQL [56] is a W3C recommendation for querying RDF. Answers to SPARQL queries can be computed by a kind of graph homomorphisms known as projection in conceptual graphs [50]. More precisely, the answer to a SPARQL query Q relies on calculating the set of possible homomorphisms from the basic graph pattern(s) of Q into the RDF graph representing the knowledge base (see Example 1). Unfortunately, SPARQL lacks the ability of expressing paths, which is necessary for many applications (see Example 2).

Example 1 *SPARQL graph patterns allow to match a query graph against an actual RDF graph. Fig. 1(a) presents such a graph pattern. It can be used for finding the name and email address of any one related in any way, i.e., not family restricted, to a daughter of a person named "Faisal". If this pattern is used in a SPARQL query against the graph G of Fig. 1, it will return "Natasha" (with email "natasha@example.org").*

Another approach, that has been successfully used in databases [25,28,48,59,62] but little in the context of the semantic web, uses path queries, *i.e.*, regular expressions, for finding regular paths in a database graph. The answer to a path query R over a database graph G, is the set of all pairs of nodes in G satisfying the language denoted by R, *i.e.*, all pairs connected by a directed path such that the concatenation of the labels of the arcs along the path forms a word that belongs to the language denoted by R (see Example 2).

Example 2 *Assuming an RDF graph representing a social network, i.e., a graph representing relations between people, like the graph G of Fig. 1, the regular expression* $(\texttt{ex:son}|\texttt{ex:daughter})^{+}\cdot ?b4$, *when used as a query, searches all pairs of* *nodes connected by paths with a sequence of son and daughter relations followed by any relation (not restricted to family relation). Applied to node* `ex:c1` *of G, it should match the paths leading to* `ex:Person1`*,* `ex:Person2`*,* `ex:Person3` *and* `ex:c3`*. This query, as it represents paths of unknown length, cannot be expressed in SPARQL. On the other hand, the graph of Fig. 1(a), which represents a basic graph pattern of a SPARQL query, cannot be expressed by a regular expression.*

None of these approaches can be reduced to the other, *i.e.*, some queries that can be expressed in one approach cannot be expressed in the other. As shown in Fig. 1(a), a query whose homomorphic image in the database is not a path cannot be expressed by a regular expression, while RDF does not allow expressing paths of undetermined length. Furthermore, regular expressions provide a simple way to capture additional information along paths that is not be provided by SPARQL graph patterns, but they are not powerful enough as a query language.

Therefore, an approach that combines the advantages of both SPARQL and path queries is herein investigated. This combined approach, in which the arcs of the SPARQL graph patterns may be labeled with regular expression patterns, supports path queries (see Example 3).

Example 3 *Assuming that we are interested in finding, among the persons related in any way, i.e., not family restricted, to Faisal's descendants, people who know Faisal, and we want to know their names and email addresses. This query can be expressed using graph patterns labeled with regular expression patterns, as shown graphically in Fig. 1(b).*

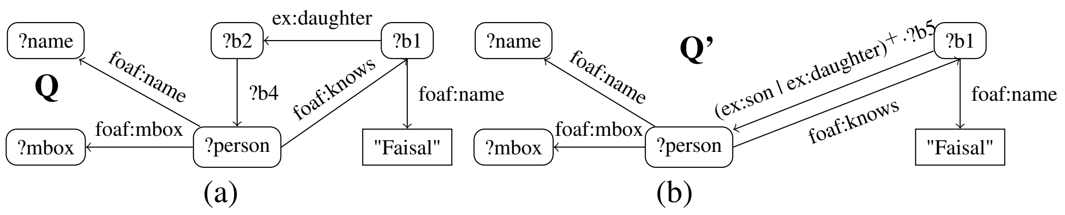


Fig. 2. A SPARQL graph pattern (a) and a PSPARQL graph pattern (b).

In order to formally define that language, we first introduce Path RDF (PRDF) as an extension of RDF in which arcs of the graphs can be labeled by regular expression patterns. Because we want to ground the definition of our language on the semantics of RDF, and we want to leave the door open to further extensions, we define the semantics of PRDF on top of RDF semantics and we provide a sound and complete algorithm for checking if a PRDF graph is entailed by some RDF graph. However, those readers who are not interested in the semantic justification of this extension, and only require syntactic definitions, can skip Sections 2.2 and 4.2 (GRDF and PRDF semantics) and trust Theorem 1 and Theorem 3 for grounding our language semantically (see Appendix A).

PRDF graphs are then used to define an extension to SPARQL, called PSPARQL, that replaces RDF graph patterns used in SPARQL by PRDF graph patterns, *i.e.*, graph patterns with regular expression patterns. We present the syntax and the semantics of PSPARQL. We provide algorithms, which are sound and complete for evaluating PSPARQL graph patterns over RDF graphs. We establish complexity results on evaluating PSPARQL graph patterns over RDF graphs. We have implemented a PSPARQL query engine.

Paper outline. This paper is organized as follows: we introduce simple RDF in Section 2. Section 3 presents the two approaches mentioned so far for querying RDF graphs. In Section 4, we give the syntax and the semantics of PRDF, as well as a sound and complete inference mechanism for querying RDF graphs with PRDF queries. Section 5 defines the syntax of the PSPARQL language and establishes the complexity results of PSPARQL query evaluation. Section 6 presents sound and complete algorithms for answering a PSPARQL query, *i.e.*, for enumerating the set of all answers to a PSPARQL query. We provide the first experimental results with an implementation of a PSPARQL query evaluator (Section 7). After a review of related work (Section 8), we conclude in Section 9. The proof of the most important results are given in Appendix.

2 Simple RDF

This section is devoted to the presentation of the *Simple RDF* knowledge representation language. We first recall (Section 2.1) its abstract syntax [23], its semantics (Section 2.2), using the notions of simple interpretations, models, simple entailment of [39]), then Section 2.3 uses homomorphisms to characterize simple RDF entailment (as done in [15] for a graph-theoretic encoding of RDF, and in [36] for a database encoding), instead of the equivalent interpolation lemma of [39].

2.1 RDF syntax

To define the syntax of RDF, we need to introduce the *terminology* over which RDF graphs are constructed.

Terminology The RDF *terminology* $\mathcal{T}$ is the union of three pairwise disjoint infinite sets of *terms* [39]: the set $\mathcal{U}$ of *urirefs*, the set $\mathcal{L}$ of *literals* (itself partitioned into two sets, the set $\mathcal{L}_p$ of *plain literals* and the set $\mathcal{L}_t$ of *typed literals*), and the set $\mathcal{B}$ of *variables*. The set $\mathcal{V} = \mathcal{U} \cup \mathcal{L}$ of *names* is called the *vocabulary*. From now on, we use different notations for the elements of these sets: a variable will be prefixed by `?` (like `?b1`), a literal will be between quotation marks (like `"27"`), and the rest will be urirefs (like `foaf:Person` — `foaf`: is the prefix used for identifying the "Friend of a friend" name space used for representing personal information — or `ex:friend`).

RDF graphs are usually constructed over the set of urirefs, blanks, and literals [23]. "Blanks" is a vocabulary specific to RDF. Because we want to stress the compatibility of the RDF structure with classical logic, we will use the term *variable* instead. The specificity of a blank with regard to variables is their quantification. Indeed, a blank in RDF is an existentially quantified variable. We prefer to retain this classical interpretation which is useful when an RDF graph is put in a different context. When switching to SPARQL, variables and blanks have different behaviors in complex cases. For example, a blank shared in different simple patterns of a group query pattern has a local scope which is easier to describe as changing the quantification scope of a variable than changing a blank into a variable. So, for the purpose of this paper and without loss of generality, we have chosen to follow [54] to not distinguish between variables and blanks, and speak of variables instead.

Definition 1 (RDF graph) *An* RDF triple *is an element of* $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times \mathcal{T}$. *An* RDF graph *is a finite set of RDF triples.*

Excluding variables as predicates and literals as subject was an unnecessary restriction in the RDF design, that has been relaxed in many RDF extensions. Relaxing these constraints simplifies the syntax specification and neither changes RDF semantics nor the computational properties of reasoning. In consequence, we adopt such an extension introduced in [41] and called *generalized RDF graphs*, or simply GRDF graphs.

Definition 2 (GRDF graph) *A* GRDF triple *is an element of* $\mathcal{T} \times (\mathcal{U} \cup \mathcal{B}) \times \mathcal{T}$. *A GRDF graph is a finite set of GRDF triples.*

Notations If $\langle s, p, o\rangle$ is a GRDF triple, s is called its *subject*, p its *predicate*, and o its *object*. We denote by $subj(G)$ the set $\{s \mid \langle s, p, o\rangle \in G\}$ the set of elements appearing as a subject in a triple of a GRDF graph G. $pred(G)$ and $obj(G)$ are defined in the same way for predicates and objects. We call $nodes(G)$ the *nodes* of G, the set of elements appearing either as subject or object in a triple of G, i.e., $subj(G) \cup obj(G)$. A *term* of G is an element of $term(G) = subj(G) \cup pred(G) \cup obj(G)$. If $\mathcal{Y} \subseteq \mathcal{T}$ is a set of terms, we denote $\mathcal{Y} \cap term(G)$ by $\mathcal{Y}(G)$. For instance, $\mathcal{V}(G)$ is the set of names appearing in G.

A *ground* GRDF graph G is a GRDF graph with no variable, *i.e.*, $term(G) \subseteq \mathcal{V}$.

GRDF graphs as graphs: A *simple GRDF graph* can be represented graphically as a directed labeled multigraph$\langle N, E, \gamma, \lambda\rangle$ where the set of nodes N is the set of terms appearing as a subject or object in at least one triple of G, the set of arcs E is the set of triples of G, γ associates to each arc a pair of nodes (its extremities) $\gamma(e) = \langle \gamma_1(e), \gamma_2(e)\rangle$ where $\gamma_1(e)$ is the source of the arc e and $\gamma_2(e)$ its target; finally, λ labels the nodes and the arcs of the graph: if s is a node of N, *i.e.*, a term, then $\lambda(s) = s$, and if e is an arc of E, *i.e.*, a triple $\langle s, p, o\rangle$, then $\lambda(e) = p$.

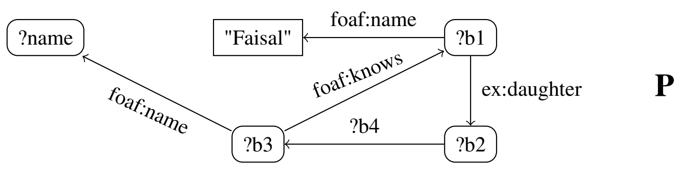


Fig. 3. A GRDF graph.

When drawing such graphs, the nodes resulting from literals are represented by rectangles while the others are represented by rectangles with rounded corners. In what follows, we do not distinguish between the two views of the RDF syntax (as sets of triples or directed labeled multigraphs). We will then speak interchangeably about their nodes, their arcs, or the triples which make it up.

Example 4 *The GRDF graph defined by the set of triples $\{\langle$`?b1`, `foaf:name`, `"Faisal"`$\rangle$, $\langle$`?b1`, `ex:daughter`, `?b2`$\rangle$, $\langle$`?b2`, `?b4`, `?b3`$\rangle$, $\langle$`?b3`, `foaf:knows`, `?b1`$\rangle$, $\langle$`?b3`, `foaf:name`, `?name`$\rangle\}$ is represented graphically in Fig. 3. Intuitively, this GRDF graph means that there exists an entity named (`foaf:name`) `"Faisal"` that has a daughter (`ex:daughter`) that has some relation with another entity whose name is non determined, and that knows (`foaf:knows`) the entity named `"Faisal"`.*

2.2 Simple RDF semantics

[39] introduces different semantics for RDF graphs. Since RDF and RDFS entailments can be polynomially reduced to simple entailment via RDF or RDFS rules [36,39,41], we are only interested in the *simple semantics* without RDF/RDFS vocabulary [18]. The definitions of interpretations, models, satisfiability, and entailment correspond to the *simple interpretations*, *simple models*, *simple satisfiability*, and *simple entailments* of [39].

Definition 3 (Interpretation of a vocabulary) *Let $V \subseteq \mathcal{V}$ be a vocabulary. An interpretation of V is a 5-tuple $I = \langle IR, IP, I_{EXT}, I_S, I_L\rangle$, where:*

- *IR is a set of resources containing plain literals of $\mathcal{L}_p$;*
- *$IP \subseteq IR$ is a set of properties;*
- *$I_S : \mathcal{U} \rightarrow IR$, maps each uriref to a resource;*
- *$I_L : \mathcal{L}_t \rightarrow IR$, maps each typed literal to a resource;*
- *I_{EXT}: $IP \rightarrow 2^{(IR\times IR)}$, maps each property p to a set of pairs of resources called the extension of p.*

In order to simplify the notations, and without loss of generality, we assume that $IP \subseteq IR$, which is true for RDF, but not necessarily for Simple RDF. If $I = \langle IR$, IP, I_{EXT}, I_S, $I_L\rangle$ is an interpretation of a vocabulary V, we also denote by I the mapping defined by:

– $\forall x \in \mathcal{U}$, $I(x) = I_S(x)$;
– $\forall x \in \mathcal{L}_t$, $I(x) = I_L(x)$;
– $\forall x \in \mathcal{L}_p$, $I(x) = x$.

We have defined the interpretation of a vocabulary. Now, we want to specify the conditions under which an interpretation I is a model for a GRDF graph G, *i.e.*, G is satisfied by the interpretation I. For that matter, we need to extend the interpretations of a vocabulary to interpret the variables in G.

Definition 4 (Extension to variables) *Let I be an interpretation of a vocabulary $V \subseteq \mathcal{V}$, and $B \subseteq \mathcal{B}$ a set of variables. An extension of I to B is a mapping $I' : \mathcal{V} \cup B \to IR$ such that $\forall x \in V$, $I'(x) = I(x)$.*

This definition implies that a variable can be interpreted (or mapped) to any resource of IR.

Definition 5 (Model of a GRDF graph) *Let G be a GRDF graph. An interpretation $I = \langle IR, IP, I_{EXT}, I_S, I_L\rangle$ of a vocabulary $V \supseteq \mathcal{V}(G)$ is a model of G if and only if there exists an extension I' of I to $\mathcal{B}(G)$ such that for each triple $\langle s, p, o\rangle \in G$, $\langle I'(s), I'(o)\rangle \in I_{EXT}(I'(p))$. The mapping I' is called a* proof *of G.*

This definition is necessary for GRDF graphs having variables as predicates [41], and PRDF graphs (*cf.* Section 4). It is equivalent to the standard definition [39], *i.e.*, $\langle I'(s), I'(o)\rangle \in I_{EXT}(I(p))$, in the case of RDF graphs.

The notions of satisfiability and entailment are then defined as usual.

Definition 6 (Satisfiability) *A GRDF graph G is satisfiable iff there exists a model of G.*

Definition 7 (Entailment) *Let G and H be two GRDF graphs. Then G* entails *H (denoted by $G \models_{RDF} H$) iff every model of G is also a model of H.*

The definitions of satisfiability and entailment will be the same when we extend the syntax and the semantics of GRDF. Two GRDF graphs G, H are said equivalent if and only if $G \models_{RDF} H$ and $H \models_{RDF} G$. We associate to this semantics the decision problem called SIMPLE RDF ENTAILMENT:

SIMPLE RDF ENTAILMENT
Instance: two GRDF graphs G and H.
Question: Does $G \models_{RDF} H$?

SIMPLE RDF ENTAILMENT is an NP-complete problem for RDF graphs [36]. For GRDF graphs, its complexity remains unchanged. Polynomial subclasses of the problem have been exhibited based upon the structure or labeling of the query:

– when the query is ground [40], or more generally when it has a bounded number

of variables,
– when the query is a tree or admits a bounded decompositions into a tree, according to the methods in [35] as shown in [15].

2.3 *Simple RDF entailment as a graph homomorphism*

SIMPLE RDF ENTAILMENT [39] can be characterized as a kind of graph homomorphism. A *graph homomorphism* from an RDF graph H into an RDF graph G, as defined in [15,36], is a mapping π from the nodes of H into the nodes of G preserving the arc structure, *i.e.*, for each node $x \in H$, if $\lambda(x) \in \mathcal{U} \cup \mathcal{L}$ then $\lambda(\pi(x)) = \lambda(x)$; and each arc $x \xrightarrow{p} y$ is mapped to $\pi(x) \xrightarrow{\pi(p)} \pi(y)$. This definition is similar to the projection used to characterize entailment of conceptual graphs (CGs) [50] (*cf.* [26] for precise relationship between RDF and CGs). We modify this definition to the following equivalent one that maps $term(H)$ into $term(G)$.

Definition 8 (Map) *Let $V_1 \subseteq \mathcal{T}$, and $V_2 \subseteq \mathcal{T}$ be two sets of terms. A* map *from V_1 to V_2 is a mapping $\mu : V_1 \rightarrow V_2$ such that $\forall x \in (V_1 \cap \mathcal{V})$, $\mu(x) = x$.*

The *map* defined in [36,54] is a particular case of Definition 8. An RDF homomorphism is a map preserving some structure (here, the arc structure).

Definition 9 (RDF homomorphism) *Let G and H be two GRDF graphs. An RDF homomorphism from H into G is a map π from $term(H)$ to $term(G)$ such that $\forall \langle s, p, o \rangle \in H$, $\langle \pi(s), \pi(p), \pi(o) \rangle \in G$.*

Example 5 (RDF homomorphism) *Fig. 4 shows two GRDF graphs Q and G (note that the graph Q is the graph P of Fig. 3, to which the following triple is added $\langle$`?b3, foaf:mbox, ?mbox`$\rangle$. The map π_1 defined by $\{\langle$`"Faisal", "Faisal"`$\rangle$, $\langle$`?b1, ex:c1`$\rangle$, $\langle$`?name, "Natasha"`$\rangle$, $\langle$`?mbox, "natasha@example.org"`$\rangle$, $\langle$`?b2,ex:c2`$\rangle$, $\langle$`?b4, ex:friend`$\rangle$, $\langle$`?b3, ex:Person1`$\rangle\}$ is an RDF homomorphism from Q into G. And the map π_2 defined by $\{\langle$`"Faisal", "Faisal"`$\rangle$, $\langle$`?b1, ex:c1`$\rangle$, $\langle$`?name, "Deema"`$\rangle$, $\langle$`?b3, ex:Person2`$\rangle$, $\langle$`?b4, ex:friend`$\rangle$, $\langle$`?b2, ex:c2`$\rangle\}$ is an RDF homomorphism from P into G. Note that π_2 cannot be extended to an RDF homomorphism from Q into G since there is no mailbox for `"Deema"` in G.*

Theorem 1 *Let G and H be two GRDF graphs, then $G \models_{RDF} H$ if and only if there is an RDF homomorphism from H into G.*

The definition of RDF homomorphisms (Definition 9) is similar to the *map* defined in [36] for RDF graphs. [36] provides without proof an equivalence theorem (Theorem 3) between RDF entailment and maps. A proof is provided in [15] also for RDF graphs, but the homomorphism involved is a mapping from nodes to nodes, and not from terms to terms. In RDF, the two definitions are equivalent. However, the terms-to-terms version is necessary to extend the theorem of RDF (Theorem 1) to the PRDF graphs studied in Section 3.1. The proof of Theorem 1 will be a particular case of the proof of Theorem 3 for PRDF graphs.

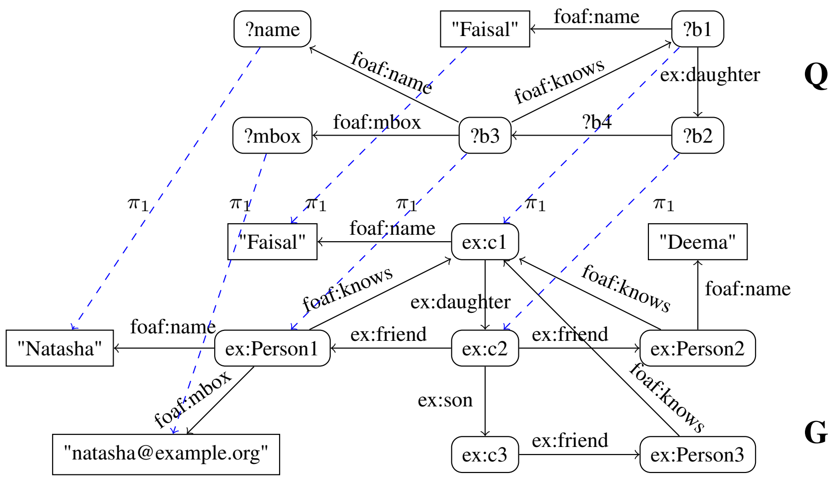


Fig. 4. An RDF homomorphism π from Q into G.

This equivalence between the semantic notion of entailment and the syntactic notion of homomorphism is the ground by which a correct and complete query answering procedure can be designed.

3 Querying RDF graphs

This section presents two approaches for querying RDF graphs. In Section 3.1, a simplified version of the SPARQL query language, insisting on its development on top of RDF, is given. Section 3.2 shows how “path queries” developed in databases, and which use regular expression patterns, can be used to query RDF knowledge bases. Lastly, in Section 3.3, we discuss the significance of the combined approach which will be the goal of this paper.

3.1 The SPARQL query language

SPARQL is the RDF query language developed by the W3C [56]. SPARQL query answering is characterized by defining a mapping from the query to the queried RDF graph.

We think that query languages for a semantically defined language like RDF should be defined semantically. This ensures the correct interpretation of the queried database, *e.g.*, guaranteeing that querying two semantically equivalent graphs will yield the same result in the sense that the application of the variable assignments to the graph patterns would return equivalent sets of graphs. This also preserves the opportunity to extend this language beyond what can be defined through simple maps or even homomorphisms, *e.g.*, querying modulo an OWL ontology.

Hence, we ground the definition of answers to a (P)SPARQL query on consequence, *i.e.*, we show that a GRDF graph G contains an answer σ to a (P/G)RDF graph H if and only if G entails $\sigma(H)$.

In this section we first define SPARQL semantically, *i.e.*, we characterize SPARQL answers with regard to entailment. Theorem 1 shows that this definition is conform to the classical definition of SPARQL. As a benefit, this provides a standard way to extend SPARQL – by changing the entailment relation – and to define new query evaluation mechanisms – by proving them sound and complete with regard to the definition. This is what we will do in Section 5.

The basic building blocks of SPARQL queries are *graph patterns* which are shared by all SPARQL query forms. Informally, a *graph pattern* can be a triple pattern, *i.e.*, a GRDF triple, a basic graph pattern, *i.e.*, a GRDF graph, the union of graph patterns, an optional graph pattern, or a constraint (*cf.* [56] for more details).

Definition 10 (SPARQL graph pattern) *A* SPARQL graph pattern *is defined inductively in the following way:*

- *every GRDF graph is a SPARQL graph pattern;*
- *if P, P' are SPARQL graph patterns and R is a SPARQL constraint, then (P `AND` P'), (P `UNION` P'), (P `OPT` P'), and (P `FILTER` R) are SPARQL graph patterns.*

Our proposal is based upon extending these graph patterns, and leaving the remainder of the query forms unchanged. So, we illustrate our extension using the SELECT query form [1]. For a complete version of SPARQL, the reader is referred to the SPARQL specification [56] or to [54,55] for formal semantics of SPARQL.

A SPARQL SELECT query is of the form `SELECT` $\vec{B}$ `FROM` u `WHERE` P where u is the URL of an RDF graph G, P is a SPARQL graph pattern and $\vec{B}$ is a tuple of variables appearing in P. Intuitively, an answer to a SPARQL query is an instantiation σ of the variables of $\vec{B}$ by terms of the RDF graph G.

Such an instantiation, called a variable assignment, is a map from a set of variables to terms. Any homomorphism is a map too (the converse is not true). Hence, we can define operations on maps that will be used for maps, assignments and homomorphisms.

[1] SPARQL provides several result forms that can be used for formating the query results. For example, CONSTRUCT that can be used for building an RDF graph from the set of answers, ASK that returns TRUE if there is a answer to a given query and FALSE otherwise, and DESCRIBE that can be used for describing a resource RDF graph.

Operations on maps. If μ is a map, then the domain of μ, denoted by $dom(\mu)$, is the subset of $\mathcal{T}$ where μ is defined. The restriction of μ to a set of terms X is defined by $\mu|_X = \{\langle x, y\rangle \in \mu |\ x \in X\}$ and the completion of μ to a set of terms X is defined by $\mu|^X = \mu \cup \{\langle x, \texttt{null}\rangle |\ x \in X$ and $x \notin dom(\mu)\}$.

If P is a graph pattern, then $\mu(P)$ is the graph pattern obtained by the substitution of $\mu(b)$ to each variable $b \in \mathcal{B}(P)$. Two maps μ_1 and μ_2 are *compatible* when $\forall x \in dom(\mu_1) \cap dom(\mu_2)$, $\mu_1(x) = \mu_2(x)$. If μ_1 and μ_2 are two compatible maps, then we denote by $\mu = \mu_1 \oplus \mu_2 : T_1 \cup T_2 \rightarrow \mathcal{T}$ the map defined by: $\forall x \in T_1, \mu(x) = \mu_1(x)$ and $\forall x \in T_2, \mu(x) = \mu_2(x)$. Analogously to [54] we define the *join* of two sets of maps Ω_1 and Ω_2 as follows:

– *(join)* $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \oplus \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2$ are compatible$\}$;
– *(difference)* $\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall \mu_2 \in \Omega_2, \mu_1$ and μ_2 are not compatible$\}$.

[55] defines different semantics for the join operation when the maps contain `null` value, and their effects in the answers are outlined.

Definition 11 (Answers to a SPARQL graph pattern) *Let P be a SPARQL graph pattern and G be an RDF graph. The set $\mathcal{S}(P, G)$ of answers to P in G is defined inductively in the following way:*

$$\begin{aligned}
\mathcal{S}(P, G) &= \{\sigma|_{\mathcal{B}(P)}|\ G \models \sigma(P)\} \text{ if } P \text{ is a GRDF graph} && (1)\\
\mathcal{S}((P \texttt{ AND } P'), G) &= \mathcal{S}(P, G) \bowtie \mathcal{S}(P', G) && (2)\\
\mathcal{S}(P \texttt{ UNION } P', G) &= \mathcal{S}(P, G) \cup \mathcal{S}(P', G) && (3)\\
\mathcal{S}(P \texttt{ OPT } P', G) &= (\mathcal{S}(P, G) \bowtie \mathcal{S}(P', G)) \cup (\mathcal{S}(P, G) \setminus \mathcal{S}(P', G)) && (4)\\
\mathcal{S}(P \texttt{ FILTER } R, G) &= \{\sigma \in \mathcal{S}(P, G) \mid \sigma(R) = \top\} && (5)
\end{aligned}$$

As usual for this kind of query languages, an answer to a query is an assignment of distinguished variables (those variables in the `SELECT` part of the query). Such an assignment is a map from variables in the query to nodes of the graph. The defined answers may assign only one part of the variables, those sufficient to prove entailment. The answers are these assignments extended to all distinguished variables.

Definition 12 (Answers to a SPARQL query) *Let $Q =$ `SELECT` $\vec{B}$ `FROM` u `WHERE` P be a SPARQL query, G be the RDF graph identified by the URL u, and $\mathcal{S}(P, G)$ is the set of answers to P in G, then the answers to the query Q are the restriction and completion to $\vec{B}$ of answers to P in G, i.e., $ANS(Q) = \{\sigma|_{\vec{B}}^{\vec{B}}|\ \sigma \in \mathcal{S}(P, G)\}$.*

From Theorem 1, this definition corresponds to the definition given in [54]:

Consequence 1 (Answers to SPARQL graph patterns and homomorphisms) *Let P be a GRDF graph and G be an RDF graph. The set $\mathcal{S}(P, G)$ of answers to P in*

G is

$$\mathcal{S}(P,G) = \{\pi|_{\mathcal{B}(P)} \mid \pi \textit{ is an RDF homomorphism from } P \textit{ into } G\}$$

Example 6 *Consider the following SPARQL query Q:*

SELECT `?name ?mbox`
FROM < `http://example.org/index1.ttl` >
WHERE $\{P$ *OPT* $\{($`?b2`, `foaf:mbox`, `?mbox`$)\}\}$

such that P is the GRDF graph of Fig. 3, and the RDF graph identified by the uriref of the FROM clause is the graph G of Fig. 4. We construct the answer to the query by taking the join of homomorphism Q into G and the homomorphism from the optional triple into G; i.e., the homomorphisms from Q into G, e.g., the homomorphism π_1 of Example 5, and the homomorphisms from P into G that cannot be extended to include the optional triple, e.g., the homomorphism π_2 of Example 5. There are therefore two answers to the query:

?name	?mbox
"Deema"	null
"Natasha"	"natasha@example.org"

Hence, what we have done so far is only to provide a semantic definition of answers to SPARQL queries (through Definition 11 and 12) and to show that this definition exactly corresponds to the original SPARQL definition (though Consequence 1). We have gained two benefits in doing this:

– Finding an RDF homomorphism is now one way to find SPARQL answers which is complete and correct; other ways may be designed.
– Extensions of SPARQL can be defined semantically and the proposed evaluation strategies, like finding a particular kind of homomorphism, can be compared with this definition. So extensions can be defined in the same way as SPARQL.

This is what we will do in the next sections.

SPARQL complexity. To evaluate the complexity of SPARQL, we consider the decision problem which consists of checking if a given assignment is an answer and is called SPARQL QUERY EVALUATION:

SPARQL QUERY EVALUATION [54]
Instance: an RDF graph G, a graph pattern P and an assignment σ.
Question: Is $\sigma \in \mathcal{S}(P,G)$?

SPARQL QUERY EVALUATION is shown to be PSPACE-complete for SPARQL graph patterns [54]. Restricting the graph pattern constructs to the AND, FILTER and

UNION operators makes it NP-complete. This problem is equivalent to that of checking the existence of a solution.

3.2 Regular expression patterns for path queries

Regular expressions are the usual way for expressing path queries [27,28,21,2,31,46]. Informally, the set of answers to a path query R over a database graph G is the set of all pairs of nodes in G connected by a directed path such that the concatenation of the labels of the arcs along the path forms a word that belongs to the language denoted by R.

3.2.1 Languages and regular expression patterns

Let Σ be an alphabet. A *language* over Σ is a subset of Σ^*: its elements are sequences of elements of Σ called *words*. A (non empty) word $(a_1, \ldots, a_k)$ is denoted by $a_1 \cdot \ldots \cdot a_k$. If $A = a_1 \cdot \ldots \cdot a_k$ et $B = b_1 \cdot \ldots \cdot b_q$ are two words over Σ, then $A \cdot B$ is the word over Σ defined by $A \cdot B = a_1 \cdot \ldots \cdot a_k \cdot b_1 \cdot \ldots \cdot b_q$. Regular expressions can be used for defining languages over Σ.

Definition 13 (Regular expression) *Let Σ be an alphabet, the set $\mathcal{R}(\Sigma)$ of regular expressions is inductively defined by:*

– $\forall a \in \Sigma$, $a \in \mathcal{R}(\Sigma)$ *and* $!a \in \mathcal{R}(\Sigma)$*;*
– $\epsilon \in \mathcal{R}(\Sigma)$*;*
– *If* $A \in \mathcal{R}(\Sigma)$ *and* $B \in \mathcal{R}(\Sigma)$ *then* $A|B$, $A \cdot B$, A^*, $A^+ \in \mathcal{R}(\Sigma)$.

such that $!a$ is the complement of a over Σ, $A|B$ denotes the disjunction of A and B, $A \cdot B$ the concatenation of A and B, A^* the Kleene closure, and A^+ the positive closure.

We have restricted regular expressions to atomic negation in order to have a reasonable time complexity in the query language that we are building. However, the semantics, soundness and completeness results as well as the algorithms defined throughout this paper still work with non-atomic regular expressions [12].

Introduction of variables. More general forms of regular expressions are the ones that include variables, we call them *regular expression patterns*. Their combined power and simplicity contribute to their wide use in different fields. For example, in [31], in which they are called *universal regular expressions*, they are used for compiler optimizations. In [46], they are called *parametric regular expressions*, and are used for program analysis and model checking. The use of variables in regular expression patterns is different from the use of variables in Unix ("regular expressions with back referencing" in [4]). A variable appearing in a regular expression pattern matches any symbol of the alphabet or any variable, while a variable in regular expressions with back referencing can match strings. Matching strings with regular expressions with back referencing has been shown to be NP-complete [4].

Definition 14 (Regular expression pattern) *Let Σ be an alphabet, X be a set of variables, the set $\mathcal{R}(\Sigma, X)$ of regular expression patterns is inductively defined by:*

– *$\forall a \in \Sigma$, $a \in \mathcal{R}(\Sigma, X)$ and $!a \in \mathcal{R}(\Sigma, X)$;*
– *$\forall x \in X$, $x \in \mathcal{R}(\Sigma, X)$;*
– *$\epsilon \in \mathcal{R}(\Sigma, X)$;*
– *If $A \in \mathcal{R}(\Sigma, X)$ and $B \in \mathcal{R}(\Sigma, X)$ then $A|B$, $A \cdot B$, A^*, $A^+ \in \mathcal{R}(\Sigma, X)$.*

The language generated by a regular expression pattern R, denoted by $L^*(R)$, is given in the following definition.

Definition 15 (Language defined by a regular expression pattern) *Let Σ be an alphabet, X be a set of variables, and $R, R' \in \mathcal{R}(\Sigma, X)$ be regular expression patterns. $L^*(R)$ is the set of words of $(\Sigma \cup X)^*$ defined by:*

$$\begin{aligned}
L^*(\epsilon) &= \{\epsilon\};\\
L^*(a) &= \{a\};\\
L^*(!a) &= \Sigma \setminus \{a\};\\
L^*(x) &= \Sigma \cup X;\\
L^*(R \mid R') &= \{w \mid w \in L^*(R) \cup L^*(R')\};\\
L^*(R \cdot R') &= \{w \cdot w' \mid w \in L^*(R) \textit{ and } w' \in L^*(R')\};\\
L^*(R^+) &= \{w_1 \cdot \ldots \cdot w_k \,|\forall i \in [1 \ldots k], w_i \in L^*(R)\};\\
L^*(R^*) &= \{\epsilon\} \cup L^*(R^+).
\end{aligned}$$

With regard to a more traditional definition of the language generated by a regular expression, our definition ranges over $\Sigma \cup X$. This is necessary because variables may match variables in GRDF graphs.

3.2.2 Paths in graphs and languages

Informally, a pair of nodes $\langle x, y\rangle$ in a given graph satisfies a language $L^*(R)$ if there exists a directed path from x to y in the graph such that the word obtained from the concatenation of arc labels along the path is in $L^*(R)$. We define this notion more precisely. First we define the notion of a path in a graph and the word associated to this path.

Definition 16 (Path in a directed graph) *Let $G = \langle N, E, \gamma, \lambda\rangle$ be a directed labeled graph, let x and y be two nodes of N, a* path *from x to y is a non-empty list of arcs $(a_1, \ldots, a_k)$ of E such that $\gamma_1(a_1) = x$, $\gamma_2(a_k) = y$, and for all $1 \leq i < k$, $\gamma_2(a_i) = \gamma_1(a_{i+1})$.*

Definition 17 (Word associated to a path) *Let $G = \langle N, E, \gamma, \lambda \rangle$ be a labeled directed graph, whose arcs are labeled over an alphabet $\Sigma \cup X$, the* word $\lambda(P) = \lambda(a_1) \cdot \ldots \cdot \lambda(a_k)$ *over* $(\Sigma \cup X)^*$ is associated *to the path $P = (a_1, \ldots, a_k)$ of G.*

Then, we establish when a path satisfies a regular expression patterns as defined above.

Definition 18 (Satisfaction of a regular expression pattern) *Let $G = \langle N, E, \gamma, \lambda \rangle$ be a directed labeled graph where the arcs are labeled by elements of an alphabet Σ and X a set of variables, a pair $\langle x, y \rangle$ of nodes of G* satisfies *a regular expression pattern over Σ and X, if one of the following conditions is satisfied:*

– *$\epsilon \in L^*(R)$ and $x = y$; or*
– *there exists a path P from x to y in G and a map μ from $\Sigma \cup X$ to $term(G)$ such that $\lambda(P) \in L^*(\mu(R))$.*

The definition involves a map μ which ensures that variables which are matched against several arcs in the path – because of multiple occurrences in the regular expression pattern or because of repetitions like in x^+ – match the same label (predicate or variable) in G. In fact, the same definition without the introduction of μ would treat variables in regular expression patterns as wildcards, *i.e.*, each occurrence of a variable would behave independently of the other ones, *e.g.*, $x \cdot a \cdot x$ would behave exactly like $x \cdot a \cdot y$. Our definition of regular expression patterns is thus more powerful than regular expression with wildcards which is more powerful than simple regular expressions.

3.2.3 Regular expression patterns as queries

Let G be an RDF graph, and R be a regular expression pattern over Σ and X. An answer to R in G is a triple (x, y, μ) (where x et y are two nodes and μ is a map from terms of R to terms of G) such that there exists a path P from x to y and a word $w \in L^*(R)$ with $\lambda(P) = \mu(w)$.

Example 7 *Consider the RDF graph G of Fig. 4, and the regular expression pattern $R = (\texttt{ex:son}|\texttt{ex:daughter})^+ \cdot \texttt{?b5}$. Intuitively, this regular expression pattern encodes the paths from the entity x to the entity y such that y has a relation, by any predicate, with a descendant of x. The answers to R are:*

$$\begin{gathered}\{\langle \texttt{ex:c1}, \texttt{ex:c3}, \{\langle \texttt{?b5}, \texttt{ex:son}\rangle\}\rangle,\\ \langle \texttt{ex:c1}, \texttt{ex:Person1}, \{\langle \texttt{?b5}, \texttt{ex:friend}\rangle\}\rangle,\\ \langle \texttt{ex:c1}, \texttt{ex:Person2}, \{\langle \texttt{?b5}, \texttt{ex:friend}\rangle\}\rangle,\\ \langle \texttt{ex:c1}, \texttt{ex:Person3}, \{\langle \texttt{?b5}, \texttt{ex:friend}\rangle\}\rangle\}\end{gathered}$$

The decision problem for the satisfiability of regular expression patterns is defined as follows:

PATH SATISFIABILITY [61]
Instance: a directed labeled graph G, two nodes x, y of G, and a regular expression pattern $R \in \mathcal{R}(\Sigma, X)$, where $\Sigma \supseteq \mathcal{V}(G)$.
Question: Is there a map μ from $\Sigma \cup X$ to $term(G)$ such that the pair $\langle x, y \rangle$ satisfies $L^*(\mu(R))$?

We have established the complexity results for two classes of regular expression patterns. First, when paths are reduced to regular expressions, they satisfiability can be checked efficiently.

Proposition 1 PATH SATISFIABILITY *in which* $X = \emptyset$ *(*$R \in \mathcal{R}$ *is a regular expression that does not contain variables) can be decided in* NLOGSPACE *in* G *and* R*.*

Then, when paths contain variables, checking satisfiability requires to find a map from the regular expression pattern to the graph. This increases complexity.

Proposition 2 PATH SATISFIABILITY *is in* NP.

3.3 Discussion

We have presented in this section the SPARQL query language and we have provided its semantics with regard to RDF entailment. We have also presented regular expression patterns. Although regular expression patterns can easily capture information along paths in a graph (they are good for graph traversals), they are not powerful enough as a query language for RDF and for processing queried information. Furthermore, both approaches are incomparable, *i.e.*, there are some queries that can be expressed by one approach and cannot be expressed by the other (*cf.* Section 1).

In order to benefit from the query capabilities of both query frameworks, we will extend SPARQL with regular expression patterns. To that extent, we will replace the graph patterns of SPARQL queries with graph patterns embedding regular expression patterns that we call Path RDF graphs or simply *PRDF graphs*. This will require extending RDF syntax, semantics, and the inference mechanism used for SPARQL, *i.e.*, RDF homomorphism, with path semantics as we will see in the following section.

4 Path RDF graphs: syntax, semantics, and inference mechanism

PRDF graphs are GRDF graphs where predicates in the triples are regular expression patterns constructed over the set of urirefs and the set of variables. We extend the RDF semantics to take into account these constructs. Section 4.1 presents its abstract syntax, and its semantics is presented in Section 4.2. Section 4.3 presents an inference mechanism for checking if a PRDF graph is a consequence of a GRDF graph. This mechanism will be used for calculating the set of answers to a PRDF graph over a GRDF graph when using PRDF graphs for constructing PSPARQL graph patterns.

4.1 PRDF syntax

Since arcs in GRDF graphs are labeled by the elements of $\mathcal{U} \cup \mathcal{B}$, path queries will be defined by regular expression patterns over $\mathcal{U}$ and $\mathcal{B}$.

We denote by Path RDF, or PRDF, the extension to GRDF with regular expression patterns used in the predicate position of PRDF triples.

Definition 19 (PRDF graph) *A* PRDF triple *is an element of* $\mathcal{T} \times \mathcal{R}(\mathcal{U}, \mathcal{B}) \times \mathcal{T}$. *A PRDF graph is a set of PRDF triples.*

All PRDF graphs with atomic predicates are not necessarily RDF graphs, but they are GRDF graphs [41]. A PRDF graph can be represented graphically in the same way as a GRDF graph in which arcs can be labeled by elements of $\mathcal{R}(\mathcal{U}, \mathcal{B})$.

Notations Let R be a regular expression patterns, $u \in \mathcal{U}(R)$ if $u \in U$ and U is the smallest set such that $R \in \mathcal{R}(U, \mathcal{B})$. In the same way, $b \in \mathcal{B}(R)$ if $b \in B$ and B is the smallest set such that $R \in \mathcal{R}(\mathcal{U}, B)$. Let G be a PRDF graph, $pred(G)$ is the set of regular expression patterns appearing as a predicate in a triple of G. Let $\mathcal{UB}(\mathcal{R})$ $= \mathcal{U}(R) \cup \mathcal{B}(R), \forall R \in pred(G)$. Then $term(G) = subj(G) \cup \mathcal{UB}(\mathcal{R}) \cup obj(G)$.

For example, the graph P of Fig. 5 searches among any related one to Faisal's descendants, the names and email addresses of people who know Faisal.

4.2 PRDF semantics: interpretations and models

Since the terminology of RDF is the one used for PRDF, RDF interpretations remain unchanged in the case of PRDF. However, an RDF interpretation has specific conditions to be a model for a PRDF graph. These conditions are the transposition of the classical path semantics within the RDF semantics.

Definition 20 (Support of a regular expression pattern) *Let $I = \langle IR, IP, I_{EXT}, I_S, I_L \rangle$ be an interpretation of a vocabulary $V = U \cup L$, I' be an extension of I to $B \subseteq \mathcal{B}$, and $R \in \mathcal{R}(U, B)$, a pair $\langle x, y \rangle$ of $(IR \times IR)$ supports R in I' if and only if one of the two following conditions is satisfied:*

(i) the empty word $\epsilon \in L^*(R)$ and $x = y$;
(ii) there exists a word of length $n \geq 1$ $w = w_1 \cdot \ldots \cdot w_n$ where $w \in L^*(R)$ and $w_i \in U \cup B$ $(1 \leq i \leq n)$, and a sequence of resources of IR $x = r_0, \ldots, r_n = y$ such that $\langle r_{i-1}, r_i \rangle \in I_{EXT}(I'(w_i))$, $1 \leq i \leq n$.

Instead of considering paths in RDF graphs, Definition 20 considers paths in the interpretations of PRDF graphs, *i.e.*, paths are now relating resources. This definition is the semantic substitute for the satisfaction of a regular expression pattern by two nodes (Definition 18). It has the same function: ensuring that variables have only one image. This is achieved by the "extension to variables" (I') which plays the same role as μ in Definition 18.

It is used in the following definition of PRDF models in which it replaces the direct correspondences that exists in RDF between a relation and its interpretation (see Definition 5), by a correspondence between a regular expression pattern and a sequence of relation interpretations. This allows to match regular expression patterns, *e.g.*, r^+, with variable length paths.

Definition 21 (Model of a PRDF graph) *Let G be a PRDF graph, and $I = \langle IR, IP, I_{EXT}, I_S, I_L \rangle$ be an interpretation of a vocabulary $V \supseteq \mathcal{V}(G)$. I is a PRDF model of G if and only if there exists an extension I' of I to $\mathcal{B}(G)$ such that for every triple $\langle s, R, o \rangle \in G$, $\langle I'(s), I'(o) \rangle$ supports R in I'.*

This definition extends the definition of RDF models (Definition 5), and they are equivalent when all regular expression patterns R are reduced to atomic terms, *i.e.*, urirefs or variables. Moreover, GRDF graphs are PRDF graphs with the regular expression patterns used to label the arcs restricted to atomic regular expression patterns.

Proposition 3 *If G is a PRDF graph with $pred(G) \subseteq \mathcal{U} \cup \mathcal{B}$, i.e., G is a GRDF graph, and I be an interpretation of a vocabulary $V \supseteq \mathcal{V}(G)$, then I is an RDF model of G (Definition 5) iff I is a PRDF model of G (Definition 21).*

Complexity of PRDF-GRDF entailment. We associate the following decision problems to the entailment between PRDF graphs.

PRDF ENTAILMENT
Instance: a PRDF graph G and a PRDF graph H.
Question: Does $G \models_{PRDF} H$?

We have studied independently the PRDF ENTAILMENT problem [8] which is useful if one wants to consider query containment for instance. For the purpose of defining a query language, we will only deal with the simpler PRDF-GRDF ENTAILMENT problem:

PRDF-GRDF ENTAILMENT
Instance: a GRDF graph G and a PRDF graph H.
Question: Does $G \models_{PRDF} H$?

This problem is at least NP-hard, since it contains SIMPLE RDF ENTAILMENT, an NP-complete problem. However, when the entailed graph, *i.e.*, the query, is *ground*, this problem can be decided in NLOGSPACE.

Theorem 2 *Let G be a GRDF graph and H be a* ground *PRDF graph, then* PRDF-GRDF ENTAILMENT *is in* NLOGSPACE.

The following section shows the complexity of the latter problem through the equivalence between PRDF-GRDF ENTAILMENT and PRDF-GRDF HOMOMORPHISM.

4.3 PRDF homomorphisms

In order to answer queries, it is necessary to find homomorphisms between PRDF graph patterns and the database. We consider that the database is made of simple GRDF graphs, so we only investigate homomorphisms between PRDF graphs and GRDF graphs.

This section presents a restriction of PRDF homomorphism for checking if a PRDF graph is a consequence of an RDF graph. It extends RDF homomorphisms to deal with nodes connected with regular expression patterns, that can be mapped to nodes connected by paths. PRDF homomorphism will then be used for answering PRDF graphs over RDF graphs.

Definition 22 (PRDF homomorphism) *Let G be a GRDF graph, and H be a PRDF graph. A PRDF homomorphism from H into G is a map π from $term(H)$ into $term(G)$ such that: $\forall \langle s, R, o\rangle \in H$, either*

(i) the empty word $\epsilon \in L^*(R)$ and $\pi(s) = \pi(o)$; or
(ii) $\exists \langle n_0, p_1, n_1\rangle, \ldots, \langle n_{k-1}, p_k, n_k\rangle$ in G such that $n_0 = \pi(s)$, $n_k = \pi(o)$, and $p_1 \cdot \ldots \cdot p_k \in L^*(\pi(R))$.

Definition 22 is equivalent to $\forall \langle s, R, o\rangle \in H$, $\langle \pi(s), \pi(o)\rangle$ satisfies $\pi(R)$ in G (Definition 18). This means that we can reformulate the definition using Definition 18. If R is a regular expression pattern, then $\pi(R)$ is the regular expression pattern obtained by substituting $\pi(x)$ to each atom x in R. Also (thanks to Definition 8),

$\pi(x) = x$ where $x \in \mathcal{U}$: no mapping is needed in that case.

Example 8 *Fig. 5 shows a PRDF homomorphism from the PRDF graph P into the RDF graph G. Note that the path satisfying the regular expression pattern of P is one of those given in Example 7.*

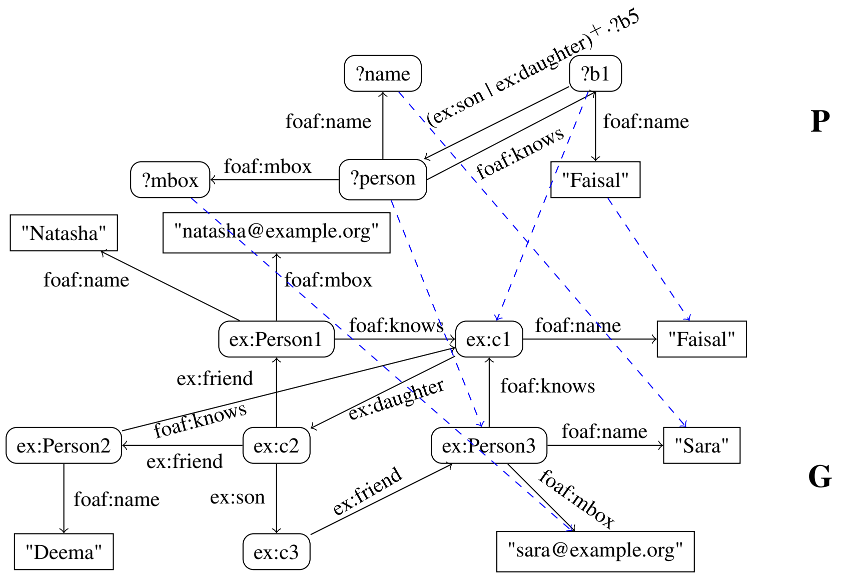


Fig. 5. A PRDF homomorphism from a PRDF graph to a GRDF graph represented in dashed lines.

The existence of a PRDF homomorphism is exactly what is needed for deciding entailment between GRDF and PRDF graphs:

Theorem 3 *Let G be a GRDF graph, and H be a PRDF graph, then there is a PRDF homomorphism from H into G iff $G \models_{PRDF} H$.*

This result, which proof is in appendix, shows that, as for RDF, there is an equivalence between PRDF homomorphisms and entailment of a PRDF graph by a GRDF graph. So, testing the entailment between PRDF graphs and RDF graphs, can be reduced to the PRDF-GRDF HOMOMORPHISM problem.

PRDF-GRDF HOMOMORPHISM
Instance: a PRDF graph H and a GRDF graph G.
Question: Is there a PRDF homomorphism from H into G?

Since any solution can be checked by checking as many times as there are edges in the query an instance of the PATH SATISFIABILITY problem, the problem is subject to its satisfaction checking. Since PATH SATISFIABILITY is in NP and PRDF-GRDF HOMOMORPHISM contains RDF HOMOMORPHISM which is equivalent to SIMPLE RDF ENTAILMENT, an NP-complete problem, then PRDF-GRDF HOMOMORPHISM is NP-complete.

The next section presents how this framework is used to extend the SPARQL query language, and Section 6 presents algorithms for enumerating the answers to such queries, *i.e.*, computing PRDF homomorphisms.

5 The PSPARQL query language

We have defined, in the previous section, the syntax and the semantics of PRDF, where regular expression patterns can be used in the predicate position of PRDF graphs. The PSPARQL query language is built on top of PRDF in the same way that SPARQL is built on top of RDF. Section 5.1 presents the syntax of PSPARQL. Section 5.2 defines the answer to a given PSPARQL query following the framework of [54], as well as an evaluation algorithm. Finally, Section 5.3 presents the complexity study of evaluating PSPARQL graph patterns.

5.1 PSPARQL syntax

PSPARQL graph patterns are built on top of PRDF in the same way that SPARQL graph patterns are built on top of RDF, by building on basic graph patterns which are here PRDF graphs instead of GRDF graphs.

Definition 23 (PSPARQL graph patterns) *A* PSPARQL graph pattern *is defined inductively in the following way:*

- *every PRDF graph is a PSPARQL graph pattern;*
- *if P, P' are PSPARQL graph patterns and R is a SPARQL constraint, then (P `AND` P'), (P `UNION` P'), (P `OPT` P'), and (P `FILTER` R) are PSPARQL graph patterns.*

Example 9 *The following PSPARQL graph pattern P*

```
{ { ex:Paris (ex:train|ex:plane)+ ?City . }
  {
    { ?City ex:capitalOf ?Country . }
      UNION
    { ?City ex:populationSize ?Population .
      FILTER (?Population > 200000)
    }
  }
}
```

consists of the following basic graph patterns, i.e., PRDF graphs, and constraint:

$P = (P_1$ `AND` $(P_2$ `UNION` $(P_3$ `FILTER` $R)))$, where

$P_1 = \{$ `ex:Paris (ex:train|ex:plane)+ ?City.` $\}$ *that finds cities reachable*

from Paris by a sequence of trains or planes;
$P_2 = \{$ `?City ex:capitalOf ?Country.` $\}$ *that finds capital cities together with their countries;*
$P_3 = \{$ `?City ex:populationSize ?Population.` $\}$ *that finds cities and their population size;*
$R =$ `Filter (?Population > 20000)` *is a constraint that restricts the values of the variable* `?Population` *to be greater than* 200000.

PSPARQL query. A *PSPARQL query* is of the form SELECT $\vec{B}$ FROM u WHERE P. The only difference with a SPARQL query is that, this time, P is a PSPARQL graph pattern, *i.e.*, a PRDF graph. The use of variables in PRDF regular expression patterns is a generalization of the use of variables as predicates in the basic graph patterns of SPARQL.

As PSPARQL introduces PRDF graph patterns, we give in Table 1 the necessary modifications to the SPARQL grammar [56] in the extended Backus-Naur form, where the production rule [21'] replaces [21] in SPARQL, and all other rules are added to SPARQL grammar to have a complete grammar for PSPARQL (see also `psparql.inrialpes.fr`).

Example 10 *The following PSPARQL query:*

```
SELECT ?First
WHERE { (?First) rdf:rest*.rdf:first "X" . }
```

searches the first element of a collection (a list) containing an element `"X"`. *For instance, if queried against an RDF graph reduced to the list* [`"A""B""X""C"`], *it will return the answers* $\{\langle$`?First`, `"A"`$\rangle$, $\langle$`?First`, `"B"`$\rangle$, $\langle$`?First`, `"X"`$\rangle\}$. *However, if we use + instead of *, then* `"X"` *is not an answer.*

5.2 Evaluating PSPARQL queries

As in the case of GRDF, the answer to a query reduced to a PRDF graph is also given by an assignment to distinguished variables. The definition of an answer to a PSPARQL query will thus be identical to that given for SPARQL in Section 3.1 (but it will use PRDF entailment).

Definition 24 (Answers to a PSPARQL graph pattern) *Let P be a PSPARQL graph pattern and G be an RDF graph, the set $\mathcal{S}(P, G)$ of answers to P in G is defined inductively in the following way:*

- *if P is a PRDF graph, $\mathcal{S}(P, G) = \{\sigma|_{\mathcal{B}(P)} \mid G \models_{PRDF} \sigma(P)\}$;*
- *(2,3,4,5) are the rules used to define answers to SPARQL graph patterns of Definition 11.*

[21’]	$\langle TriplesBlock \rangle$	::=	$\langle PathTriples1 \rangle$
		$\vert$	(‘.’ $\langle PathTriples1 \rangle$?)*
[30.1]	$\langle PathTriples1 \rangle$	::=	$\langle VarOrTerm \rangle$ $\langle PathPropLNE \rangle$
		$\vert$	$\langle PathTripleNode \rangle$ $\langle PathPropL \rangle$
[31.1]	$\langle PathPropL \rangle$	::=	$\langle PathPropLNE \rangle$?
[32.1]	$\langle PathPropLNE \rangle$	::=	$\langle PathVerb \rangle$ $\langle PathObL \rangle$ (‘;’ $\langle PathPropL \rangle$)?
[33.1]	$\langle PathObL \rangle$	::=	$\langle PathGraphNode \rangle$ (‘,’ $\langle PathObL \rangle$)?
[34.1]	$\langle PathVerb \rangle$	::=	$\langle RegularExp \rangle$
[35.1]	$\langle PathTripleNode \rangle$	::=	$\langle PathCollection \rangle$
		$\vert$	$\langle PathBNodePropL \rangle$
[36.1]	$\langle PathBNodePropL \rangle$	::=	‘[’ $\langle PathPropLNE \rangle$ ‘]’
[37.1]	$\langle PathCollection \rangle$	::=	‘(’ $\langle PathGraphNode \rangle$+ ‘)’
[38.1]	$\langle PathGraphNode \rangle$	::=	$\langle VarOrTerm \rangle$
		$\vert$	$\langle PathTripleNode \rangle$
[39.1]	$\langle RegularExp \rangle$	::=	$\langle Rexp \rangle$ ((‘$\vert$’ $\vert$ ‘·’) $\langle Rexp \rangle$)*
[39.2]	$\langle Rexp \rangle$	::=	(‘+’ $\vert$ ‘*’)? $\langle Atom \rangle$
[39.3]	$\langle Atom \rangle$	::=	‘!’ $\langle IRIref \rangle$
		$\vert$	$\langle VarOrIRIref \rangle$
		$\vert$	‘(’ $\langle RegularExp \rangle$ ‘)’

Table 1
PSPARQL graph pattern grammar.

The answers to a PSPARQL query are defined from the answers to PSPARQL graph patterns, exactly like answers to a SPARQL query in Definition 12.

As a consequence of Theorem 3, PSPARQL answers can be computed through PRDF homomorphisms because they corresponds to PRDF-GRDF ENTAILMENT.

Consequence 2 (Answers to PSPARQL graph patterns and PRDF homomorphisms) *Let P be a PRDF graph and G be an RDF graph, the set $\mathcal{S}(P, G)$ of answers to P in G is*

$$\mathcal{S}(P, G) = \{\pi|_{\mathcal{B}(P)} | \ \pi \textit{ is a PRDF homomorphism from } P \textit{ into } G\}$$

This means that it is possible to obtain answers to PSARQL queries by computing PRDF homomorphisms.

Example 11 *According to Consequence 2, the set of answers to the PSPARQL graph pattern P of Example 9 in a given RDF graph G is defined as:*

$$P = (\mathcal{S}(P_1, G) \bowtie (\mathcal{S}(P_2, G) \cup (\{\sigma \in \mathcal{S}(P_3, G) \mid \sigma(R) = \top\})))$$

In words, the set of maps, i.e., PRDF homomorphisms, from P_1 into G joined with the union of that from P_2 into G and those from P_3 into G that satisfy the constraint R.

Example 12 *The following PSPARQL query that uses the graph pattern P of Example 9:*

```
SELECT ?City
WHERE { P }
ORDER BY Asc(?City)
```

returns in an ascending order the set of cities reachable from Paris by a sequence of trains and planes, which are either capital cities or have a population size greater than 200000.

5.3 PSPARQL complexity

We define the PSPARQL QUERY EVALUATION decision problems for PSPARQL in the same way as for SPARQL. This problem depends on calculating PRDF homomorphisms, and hence it is parametrized by the PRDF HOMOMORPHISM problem.

PSPARQL QUERY EVALUATION
Instance: an RDF graph G, a PSPARQL graph pattern P and a assignment σ.
Question: Is $\sigma \in \mathcal{S}(P, G)$?

We have studied the PSPARQL QUERY EVALUATION problem for basic graph patterns. We have first considered ground graph patterns, which is reduced to checking if a given map is a PRDF homomorphism. So there is no need to seek such a map, and the REGULAR PATH problem is considered in this case (see Appendix). Theorem 4 shows that PSPARQL QUERY EVALUATION for ground basic graph patterns is no more difficult than REGULAR PATH (defined in Appendix).

Theorem 4 PSPARQL QUERY EVALUATION *is in* NLOGSPACE *for ground basic graph patterns and* NP*-complete for basic graph patterns.*

The complexity of PSPARQL QUERY EVALUATION for basic graph patterns is thus the same as SPARQL QUERY EVALUATION for basic graph patterns [36]. Since PSPARQL queries are the same as SPARQL queries with the difference of the kind of basic graph patterns and since PSPARQL QUERY EVALUATION for PRDF graphs is in NP, our extension does not increase the worst case complexity of SPARQL, *i.e.*, PSPACE-complete [54].

6 Answering PSPARQL queries: algorithms for PRDF homomorphism

To answer a PSPARQL query Q involving PRDF graphs as basic graph patterns, mandates to enumerate all PRDF homomorphisms from the graph pattern(s) of Q into the data RDF graph of Q. So, we are interested in an algorithm which, given a PRDF graph H and an RDF graph G, answers the following problems:

(1) Is there a PRDF homomorphism from H into G? (PRDF-GRDF HOMOMORPHISM)
(2) Exhibit, if it exists, a PRDF homomorphism from H into G.
(3) Enumerate all PRDF homomorphisms from H into G.

Two possible methods can be used for solving these problems: a method based on evaluating the PRDF graph triple-by-triple is presented in Section 6.1; a backtracking method based on the standard backtrack techniques is presented in Section 6.2.

6.1 Triple-by-triple evaluation

One possible method to enumerate all PRDF homomorphisms from a given PRDF graph H into an RDF graph G is to evaluate the graph H triple-by-triple and take the join of the intermediate results. This method is similar to the edge-by-edge evaluation method presented in [28].

[46,31] present the algorithm $Reach(G, R, s, \mu_i)$ (see also Appendix B), where G is a graph (for us, an RDF graph), R is a regular expression patterns, and s is a node of G. This algorithm calculates the set of triples $\langle s, o, \mu \rangle$, where o is a node of G and μ is a map from terms of R into terms of G such that there exists a path P from s to o in G and a word $w \in L^*(R)$ with $\lambda(P) = \mu(w)$ and μ_i is compatible with μ.

The $Reach(G, R, s, \mu_i)$ algorithm is used by the algorithm $Evaluate$ (Algorithm 1), which, given an RDF graph G and a PRDF triple $\langle x, R, y \rangle$, calculates the set of maps μ such that $\langle \mu(x), \mu(y) \rangle$ satisfies R in G with the map μ (it is said that μ satisfies $\langle x, R, y \rangle$ in G).

The results of the algorithm $Evaluate$ are used to calculate the PRDF homomorphisms of a PRDF graph P into an RDF graph G by successive joins in the algorithm $Eval$ (Algorithm 2), whose initial call will be $Eval(P, G, \{\mu_\emptyset\})$, where $\mu_\emptyset$ is the map with the empty domain.

The $Eval$ algorithm is given for evaluating PRDF graphs, and can be extended to evaluate PSPARQL graph patterns following the $Eval$ algorithm for evaluating SPARQL graph patterns [54].

Algorithm 1. $Evaluate(t, G)$.

```
Data: An RDF graph $G$, a PRDF triple $t = (x, R, y)$.
Result: The set of maps $\mu$ satisfying $t$ in $G$.
if $x \in \mathcal{U}$ then
    $S_G(t) \leftarrow Reach(G, R, x, \emptyset)$;
else
    $S_G(t) \leftarrow \bigcup_{s \in G} Reach(G, R, s, \{\langle x, s\rangle\})$;
end if
if $y \in \mathcal{B}$ then
    $S_G(t) \leftarrow \{(s, y, \mu) \in S_G(t)\}$;
else
    $S_G(t) \leftarrow \{(s, o, \mu') \mid (s, o, \mu) \in S_G(t)$, $(\mu, (y \leftarrow o))$ are compatible, and
    $\mu' \leftarrow \mu \oplus \{(y \leftarrow o)\}\}$
end if
return $\{\mu \mid (s, o, \mu) \in S_G(t)\}$;
```

Algorithm 2. $Eval(P, G, \Omega)$.

```
Data: An RDF graph $G$, a set of maps, a PRDF graph $P$.
Result: The set of PRDF homomorphisms from $P$ into $G\}$.
if $P = \{t\}$ then
    return $\Omega \bowtie Evaluate(t, G)$;
    $S_G(t) \leftarrow Reach(G, R, x, \emptyset)$;
else if $P = (t \cup P')$ then
    return $Eval(\{t\}, G, Eval(P', G, \Omega))$;
end if
```

Algorithmic time complexity. The $Reach$ algorithm has worst-case time complexity $\mathcal{O}(|G| \times |R_i| \times maps \times (predicateSize + vars(R_i)))$ (the notations used in Table 2 are reformulated from [46] and adapted to our problem). Now, for each triple $\langle x, R_i, y\rangle$ in P, the $Reach$ algorithm is called by the $Evaluate$ algorithm once if x is a constant, *i.e.*, a uriref or a literal if it is allowed in the subject position; otherwise it is called for each node in G multiplied by the number of variables in P in the subject position. So, the $Evaluate$ algorithm has overall worst-case time complexity $\mathcal{O}((vars_s(P) \times subj(G) + const_s(P)) \times |G| \times |R_i| \times maps$ $\times (predicateSize + vars(R_i)))$, where $vars_s(P)$ (respectively, $const_s(P)$) is the number of variables (respectively, constants) appearing in the subject position in a triple of P.

This result shows an exponential complexity with respect to the number of variables in the regular expression patterns of the PRDF graph representing the query ($\mathcal{O}(pred(G)^{vars(R)})$)). However, the size of the query, and in particular, the number of variables is usually considered very small with regards to the knowledge base. Hence, the number of variables in each regular expression pattern can be assumed a constant. With this assumption, the data complexity, which is defined as the com-

Name	Meaning
$vars$	the number of variables.
$predicateSize$	the maximum predicate size appearing in G or as a term in R.
$maps$	the number of possible maps from variables and variables of R into terms of G that match some path in G with some path in R; the worst case is $pred(G)^{vars(R)}$.

Table 2
Notations for complexity analysis

plexity of query evaluation for a fixed query [60], is $\mathcal{O}(|G|^2)$, *i.e.*, not much worse than the one of SPARQL [54].

Though the above method is correct and complete, it is not efficient, in particular, for testing the existence of a PRDF homomorphism which is sufficient for checking if a PRDF graph is a consequence of an RDF graph. Using this method, we need to perform the join operation for all PRDF triples to have the set of PRDF homomorphism, while we need to test the existence of one PRDF homomorphisms. Consider the PRDF graph P and the RDF graph of Fig. 6. To test if there exists a PRDF homomorphism from P into G, we need to solve PATH SATISFIABILITY N^2 times for the regular expression pattern R in P, where N is the number of nodes of G. However, we need to solve PATH SATISFIABILITY only once as it appears in Fig. 6. More precisely, since the extremities of the regular expression R are variables (namely, `?b6` and `?b7`), we need to check for each pair of nodes $\langle x, y\rangle$ of G if they satisfy R in G while, in this example, `?b6` and `?b7` can be only mapped to `ex:c1` and `ex:c2`, respectively. In such a case, it is sufficient to determine whether the pair $\langle$`ex:c1`, `ex:c2`$\rangle$ satisfies R in G.

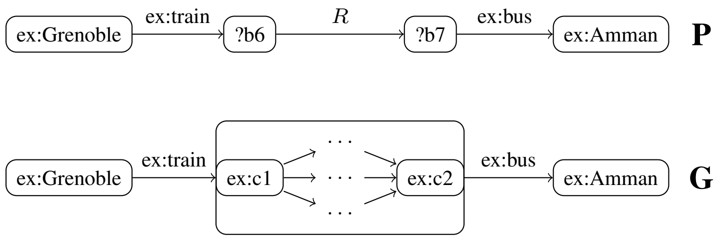


Fig. 6. A case in which the path closure method is not efficient.

The next section presents a backtracking algorithm for calculating the set of PRDF homomorphisms from PRDF graph into an RDF graph. This algorithm has the same worst-case time as the triple-by-triple method, but it is more efficient in practice since in some cases there is no need to traverse all the backtrack tree to find the first PRDF homomorphism.

Algorithm 3. $Extendhomomorphism(H, G, partialProj)$.

```
Data: a PRDF graph $H$, an RDF graph $G$, and a partial map $partialProj$
        from $term(H)$ to $term(G)$.
Result: extends the partial map to a set of PRDF homomorphisms.
if $complete(partialProj)$ then
  return $solution\text{-}Found(partialProj)$;
end if
$x \leftarrow chooseTerm(nodes(H))$;
for each $\langle y, \theta \rangle \in candidates(partialProj, x, G)$ do
  $Extendhomomorphism(H, G, partialProj \bowtie \{\langle x, y \rangle\} \bowtie \theta)$;
end for
```

6.2 A backtrack algorithm for calculating PRDF homomorphisms

An alternative method for evaluating PSPARQL graph patterns, *i.e.*, enumerating all PRDF homomorphisms from the PRDF graph of a given PSPARQL query into the data graph, is based on a backtracking technique that generates each possible map from the current one by traversing the parse tree in a depth-first manner and using the intermediate results to avoid unnecessary computations.

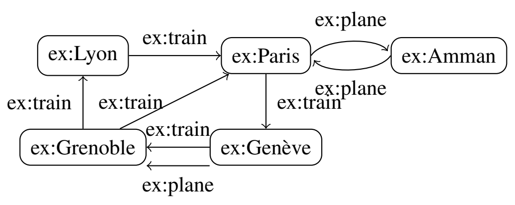
Algorithm 3 is a simple recursive version of the basic Backtrack algorithm [34]. The input of this algorithm is: a PRDF graph, an RDF graph, and a partial map, denoted by $partialProj$. $partialProj$ includes a set of pairs $\{\langle x_i, y_i \rangle\}$ such that x_i is a term of H, *i.e.*, $x_i \in term(H)$, and y_i is the image of x_i in G, *i.e.*, $y_i \in term(G)$.

The other parts of the algorithm perform as follows (see [10] for a full description of the algorithm):

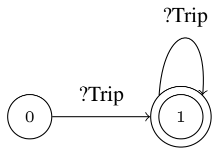
$complete(partialProj)$ checks if each term $x \in nodes(H)$ is mapped to a term in G. It returns TRUE if all $x \in nodes(H)$ are mapped, and FALSE otherwise.

$chooseTerm(nodes(H))$ chooses a term $x \in nodes(H)$.

$candidates(partialProj, x, G)$ calculates all possible candidate images in G for the current term x satisfying the partial map $partialProj$. It returns all sets of pairs $\langle y, \theta \rangle$ such that y is a possible image of x, and θ is the possible map from the terms of each regular expression pattern R_i appearing in a triple with x and one of the terms in $nodes(H)$ already mapped in $partialProj$. That is, if there is no term in $nodes(H)$ involved in a triple with x, then the possible candidate images of x are all y in $nodes(G)$ such that x can be mapped to y (*cf.* the definition of mapping Definition 8). Otherwise, there exists a set of terms $z_1, \ldots, z_k \in nodes(H)$ involved in a triple with x, which are already mapped in $partialProj$. In this case, $image(z_i)$ and y satisfies $\theta(R_i)$, where R_i is the regular expression pattern appearing in the predicate position of the triple between z_i and x. The order in which the two nodes $image(z_i)$ and y satisfy $\theta(R_i)$ depends on the order in which x and z_i appear in the triple, that is, if the triple is $\langle z_i, R_i, x \rangle$ then



(a) An RDF graph with cycles.



(b) A NDFA of `(?Trip)`$^+$.

Fig. 7. An RDF graph and a NDFA.

$\langle image(z_i), y\rangle$ satisfies $\theta(R_i)$ in G, otherwise $\langle y, image(z_i)\rangle$ satisfies $\theta(R_i)$ in G. θ maps the terms appearing in the regular expression patterns of H into the terms appearing along the paths in G with respect to $partialProj$, that is, θ is a possible map such that θ and $partialProj$ are compatible.

Then the algorithm takes each candidate y of the current term $x \in nodes(H)$ and the possible map θ, put y in the $image(x)$, and tries to generate the possible candidates of y with the current map $partialProj \bowtie \{\langle x, y\rangle\} \bowtie \theta$ (note that $partialProj$, $\{\langle x, y\rangle\}$ and θ are compatible, since the set $\langle y, \theta\rangle$ is calculated with respect to $partialProj$). This is done recursively in a depth-first manner through the call of $Extendhomomorphism(H, G, partialProj \bowtie \{\langle x, y\rangle\} \bowtie \theta)$. At the end of the algorithm, we have a tree that contains one level with a term from H, *i.e.*, a node from H, and one level with the possible images of that term in G. The input to each node of each level is the current map. Each possible path in the tree from the root to a leaf labeled by a term of G represents a possible PRDF homomorphism.

If we call $Extendhomomorphism(H, G, partialProj_{\emptyset})$ where $partialProj_{\emptyset}$ denotes the empty map, then at the end of the algorithm we have all PRDF homomorphisms from the PRDF graph H into the RDF graph G.

Proposition 4 *Algorithm 3 is correct and complete for enumerating all PRDF homomorphisms from a given PRDF graph into an RDF graph.*

This can be proved inductively because, at the beginning, the set of all homomorphisms is complete for the empty set, and at each step the partial homomorphism, *i.e.*, $partialProj$, are completely extended for the current node if Algorithm 4 [46] is complete, and the number of nodes being finite. The procedure ends having a homomorphic image for each node in H.

In our case, we do not need to enumerate all paths but instead we search the existence of paths satisfying (C)PRDF homomorphisms.

Example 13 *Consider the following PSPARQL query:*

```
SELECT ?Trip
WHERE { ex:Paris (?Trip)+ ex:Paris . }
```

and the RDF graph of Fig. 7(a). As it is shown in this graph, there are several cycles (going through Amman and Genève) that can generate an infinite number of paths. For example, considering non-simple paths, we can generate:
$\{\langle$`ex:Paris, ex:plane, ex:Amman, ex:plane, ex:Paris`$\rangle\}$
$\{\langle$`ex:Paris, ex:plane, ex:Amman, ex:plane, ex:Paris, ex:plane, ex:Amman, ex:plane, ex:Paris`$\rangle\}$
etc.

To overcome this problem, i.e., to cut cycles, our evaluation algorithm calculates all possible finite maps (or homomorphisms in the case of (C)PRDF graphs).

To this end, we can go from Paris with a state 0 *of the automata corresponding to the regular expression (see Fig. 7(b)) to Amman with a state* 1 *and a map* $\{\langle$`?Trip, ex:plane`$\rangle\}$*, then we can return to Paris since the state is different from the first visit to Paris (with a state* 1*). A possible answer therefore is:*

`?Trip`$\rightarrow \{\langle$`ex:plane`$\rangle\}$

A second answer is to go from Paris to Genève through Grenoble, and then Paris with a map $\{\langle$`?Trip, ex:train`$\rangle\}$ *(we can take Paris since the map is different from the first answer):*

`?Trip`$\rightarrow \{\langle$`ex:train`$\rangle\}$

Now, we can also go from Paris to Genève, through Grenoble, Lyon and then Paris. However, this path is not explored since Paris is already visited with the same map and state (second answer). Similarly, when we arrive at Genève or Amman for the second time, we cut the cycles since they are already visited with the same map and/or state.

For illustrating non-simple paths, consider the following PSPARQL query:

```
SELECT ?City
WHERE { ex:Paris (ex:train.ex:plane)+ ?City . }
```

In simple paths, nodes must not be visited more than once. If we consider simple paths in this example, then we cannot retrieve Amman since we cannot go through the path Paris, Genève, Grenoble, Paris, and then Amman (Paris has been visited twice).

7 Implementation and experiments

We have implemented in Java a PSPARQL query evaluator [2]. It is provided with two parsers: one for parsing PSPARQL queries based upon the syntax of PSPARQL, and the second one for parsing RDF graphs (documents) written in the Turtle language [17].

The algorithm follows the backtrack technique presented before and the evaluation of regular expression patterns generalizes those of [46]. They are used for calculating the satisfiability set of a given regular expression pattern, to take into account the multiple appearances of a given variable in different places of the query, *i.e.*, to take into account the current mappings.

This evaluator successfully passed all test cases designed by DAWG (Data Access Working Group) for the SPARQL query language [3] except the ones that concern the DESCRIBE query format. In addition, the evaluator can parse PRDF graphs and evaluate PSPARQL queries. It is currently being thoroughly tested for performances and practical hard problem detection [10] as well as for the test suite [4].

8 Related work

We divide related works in four areas: graph query languages with path expressions, RDF structural query languages, extensions of SPARQL and work on defining RDF query language semantics.

8.1 (Semi)-Structured Query Languages

Semi-structured data models [20,1] deal with data whose structure is irregular, implicit, and partial, and with schema contained in the data.

Query languages for structured graph data models can be used for querying RDF viewing RDF data as a graph that may contain transitive or repetitive patterns of relations. Among them, G [27] and its extension G+ [28] are two languages for querying structured databases. A simple G+ query has two elements, a query graph that specifies the pattern to be matched and a summary graph that defines graphically how the answers are to be structured and then presented to the user

[2] `http://psparql.inrialpes.fr/`
[3] `http://www.w3.org/2001/sw/DataAccess/tests/`
[4] `http://www.w3.org/2001/sw/DataAccess/tests/r2`

Example 14 *Given a graph that represents relations between people, the G+ query of Fig. 8 finds pairs of people who share a common ancestor.*

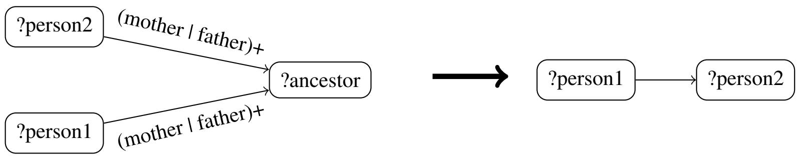


Fig. 8. A G+ query to find common ancestor.

The left hand side of the bold arrow is the pattern to be matched in the knowledge base while the right hand side is the summary graph.

Graphlog — a visual query language which has been proven equivalent to linear Datalog [25] — extends G+ by combining it with the Datalog notation. It has been designed for querying hypertext. A Graphlog query is only a graph pattern containing a distinguished edge or arc, *i.e.*, it is a restructuring edge, which corresponds to the summary graph in G+.

Example 15 *Fig. 9 shows a Graphlog query: dashed lines represent edge labels with the positive closure, a crossed dashed line represents a negated label, e.g.,* `!descendant+` *between* `?person2` *and* `?person3`*,* `person` *is a unary predicate, and finally a bold line represents a distinguished edge that must be labeled with a positive label. The effects of this query is to find all instances of the pattern that occur in the database, i.e., finding descendant of* `?person1` *which are not descendant of* `?person2`*. Then, for each one of them, define a virtual link represented by the distinguished edge.*

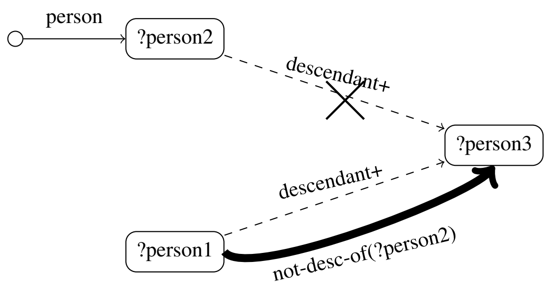


Fig. 9. A Graphlog query.

These query languages (namely G, G+ and Graphlog) support only graphical queries similar to PRDF queries. In contrast to PRDF, they are limited to finding simple paths (cycle-free paths). The main problem with finding only simple paths, is that there are situations in which answers to such queries are all non simple, *e.g.*, if the only paths matching a regular expression pattern have cycles (see end of Example 13 or the example of non-simple paths in [14]). In addition, the complexity of finding simple paths problem is NP-complete even without variables in regular expressions [61]. Moreover, they do not provide complex functionalities, for example, for filtering, ordering, projection, union of graph patterns, optional graph patterns and other useful features (see SPARQL features and examples below).

Lorel [2] is an OEM-based language for querying semi-structured documents. OEM (Object Exchange Model) [53] is based on objects that have unique identifiers, and property value that can be simple types or references to objects. However, labels in the OEM model cannot occur in both nodes (objects) and edges (properties). Lorel is a powerful query language which uses regular expression patterns for traversing object hierarchy paths, restricted to simple path semantics (or acyclic paths; see why this matters in Example 13). UnQL [21] is a language closely related to Lorel for querying semi-structured data. It is based on a data model similar to OEM [22]. A particular aspect of the language is that it allows some form of restructuring even for cyclic structures. A traverse construct allows one to transform a database graph while traversing it, *e.g.*, by replacing all labels A by the label A_0. This powerful operation combines tree rewriting techniques with some control obtained by a guided traversal of the graph. For instance, one could specify that the replacement occurs only if a particular edge is encountered on the way from the root. STRUQL [32], a query language for a web-site management system, incorporates regular expressions and has precisely the same expressive power as stratified linear Datalog.

As stated in [43], these query languages are not well suited for RDF because they do not take into account its specific semantics. This is not a real problem for RDF itself: the semantics being simple enough this roughly manifests only in the lack of blank interpretation and the enforcing of strict typing constraints. However, this prevents to extend further the query language towards ontology languages such as RDFS or OWL.

8.2 RDF Query Languages

Several query languages have been proposed for RDF [37]. Most of them use a query model based on *relational algebra* [24], where RDF graphs are viewed as a collection of triples and the queries are triple-based formulas expressed over a single relation. In spite of the benefits gained from the existing relational database systems such as indexing mechanisms, underlying storage of triples as relations [38], query optimization techniques, and others; relational queries cannot express recursive relations and even the most simple form, the transitive closure of a relation [6], directly inherited from the graph nature of RDF triples.

There are many real-world applications, inside and outside the domain of the semantic web, requiring data representation that are inherently recursive. For that reason, there has been several attempts to extend relational algebra to express complex query modeling. Outside the domain of the semantic web, [3] extends the relational algebra to represent transitive closure and [42] to represent query hierarchies. In the domain of RDF, some query languages such as RQL [43] attempt to combine the relational algebra with some special class hierarchies. It supports a form of transitive expressions over RDFS transitive properties, *i.e.*, subPropertyOf and subClassOf, for navigating through class and property hierarchies. Versa [52], RxPath [58], PRDF [11,9] and [47] are all path-based query languages for RDF that are well suited for graph traversal but do not support SQL-like functionalities. WILBUR [45] is a toolkit that incorporates path expressions for navigation in RDF graphs.

SQL-like query languages for RDF include SeRQL [19], RDQL [57] and its current successor, the SPARQL recommendation [56].

In contrast to all the above mentioned languages, PSPARQL uses regular expression patterns, *i.e.*, regular expressions with variables, and is not restricted to finding simple paths. This provides polynomial classes of the satisfiability problem of regular expressions, *e.g.*, when they do not contain variables. The originality of our proposal lies in our adaptation of RDF model-theoretic semantics to take into account regular expression patterns, effectively combining the expressiveness of these two languages. In addition, the integration of this combination on top of SPARQL provides a wider range of querying paradigms than the above mentioned languages.

8.3 SPARQL extensions

Two extensions of SPARQL, which are closely similar to PSPARQL, have been recently defined after our initial proposal [11]: SPARQLeR and SPARQ2L.

SPARQLeR [44] extends SPARQL by allowing query graph patterns involving path variables. Each path variable is used to capture simple, *i.e.*, acyclic, paths in RDF graphs, and is matched against any arbitrary composition of RDF triples between given two nodes. This extension offers good functionalities like testing the length of paths and testing if a given node is in the found paths. Since SPARQLeR is not defined with a formal semantics, its use of path variables in the subject position is unclear, in particular, when they are not bound. Even when this is the case, multiple uses of the same path variable is not fully defined: it is not specified which path is to be returned or if the variable occurences have to match the same path.

SPARQ2L [14] also allows using path variables in graph patterns and offers good features like constraints in nodes and edges, *i.e.*, testing the presence or absence of nodes and/or edges; constraints in paths, *e.g.*, simple or non-simple paths, presence of a pattern in a path. This extension is also not described semantically. One can only try to guess what is the intuitive semantics of the constructs. It seems that the algorithms are not complete with regard to their intuitive semantics, since the set of answers can be infinite in absence of constraints for using shortest or acyclic paths. Moreover, this extension is quite restricted: it does not allow using more than one triple pattern having a path variable. Relaxing this restriction requires adapting radically the evaluation algorithm which otherwise is inoperative. This occurs due to the compatibility function that does not take into account the use of the same path variable in multiple triple patterns.

In both cases, the proposal adds expressivity to PSPARQL, in particular due to the use of path variables. However, the lack of a clearly defined semantics raises questions about what should be the returned answers and this does not allow to assess the correctness and completeness of proposed procedures.

8.4 Work on SPARQL

[29] presents a relational model of SPARQL, in which relational algebra operators (join, left outer join, projection, selection, etc.) are used to model SPARQL SELECT clauses. The authors propose a translation system between SPARQL and SQL to make a correspondence between SPARQL queries and relational algebra queries over a single relation. [38] presents an implementation of SPARQL queries in a relational database engine, in which relational algebra operators similar to [3] are used. [30] addresses the definition of mapping for SPARQL from a logical point of view. [33], in which we can find a preliminary formalization of the semantics of SPARQL, defines an answer set to a basic graph pattern query using partial functions. The authors use high level operators (Join, Optional, etc.) from sets of mappings to sets of mappings, but currently they do not have formal definitions for them, stating only their types. [55] provides translations from SPARQL to Datalog with negation as failure, some useful extensions of SPARQL, like *set difference and nested queries*, are proposed. Finally, [54] presents the semantics of SPARQL using traditional algebra, and gives complexity bounds for evaluating SPARQL queries. The authors use the graph pattern facility to capture the core semantics and complexities of the language, and discussed their benefits. We followed their framework to define the answer set to PSPARQL queries.

9 Conclusion and future work

In this paper, we have extended SPARQL with regular expressions patterns allowing path queries. In order to achieve this goal, we have provided an extension of RDF graphs, called PRDF, in which regular expression patterns are used as predicates. We provided the syntax and semantics of PRDF and proved the validity of the homomorphism approach for finding entailment. Then PRDF graphs have been used as graph patterns in SPARQL queries yielding the PSPARQL query language. We have defined semantically SPARQL and shown that PSPARQL is an actual extension of SPARQL replacing RDF entailment by PRDF entailment. We have shown that finding PRDF homomorphism is a sound and complete approach for answering PSPARQL queries over RDF graphs and we have provided algorithms for calculating these answers. Finally, we proved that the problem of PRDF-GRDF entailment is NP-complete, as is RDF entailment, and thus answering PSPARQL queries over RDF graphs, remains PSPACE-complete.

This work, because it is grounded on a semantic redefinition of SPARQL, is the first stone on which further extensions can be safely built. Indeed, the semantic definition of the query language allows to define simply new query languages either by changing data and query languages or by changing the entailment regime of the query language. The query answering procedure can also change and is not restricted to simple homomorphisms. This approach is worth because regular expression patterns are not the only extension that could be considered for SPARQL. We mention below some work that we have already done and other work that would be worth pursuing.

We have shown here that using PRDF homomorphisms, which are more complex than simple graph homomorphisms, works for PSPARQL. We have also investigated various possible regular expression pattern generators [12] and considered PRDF ENTAILMENT with regard to these generators [8,10]. We have also been able to add constraints to PSPARQL in CPSPARQL [13]. However, for dealing with queries against ρdf (a subset of RDFS introduced in [51]), we have found a procedure finding (C)PRDF homomorphisms between a transformed query and the queried graph [10]. But when considering expressive ontology languages, allowing the use of disjunction and negation in class definitions, the procedure will have to go beyond homomorphisms.

Finally, PSPARQL itself has been extended with path variables following SPARQ2L and SPARQLeR but providing a clear semantics for the path variables [13]. This extension is implemented in the PSPARQL prototype.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proceeding of the 6th International Conference on Database Theory (ICDT). Volume 1186 of LNCS., Springer-Verlag*, pages 1–18, 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L.Wiener. The Lorel query language for semistructured data. *Journal on Digital Libraries*, 1(1):68–88, 1997.

[3] R. Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. *IEEE Transactions on Software Engineering*, 14(7):879–885, 1988.

[4] A. V. Aho. Pattern matching in strings. In R. V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 325–347. Academic Press, New York (NY US), 1980.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading (MA US), 1974.

[6] A. V. Aho and J. D. Ullman. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL 1979)*, pages 110–119, New York, NY, USA, 1979. ACM.

[7] N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13:1–18, 2003.

[8] F. Alkhateeb. Graphes à chemins: Graphes RDF/RDFS étiquetés par des expressions algébriques. Master's thesis, Université de Joseph Fourier/INRIA Rhône-Alpes, 2005.

[9] F. Alkhateeb. Une extension de RDF avec des expressions régulières. In *actes de 8e Rencontres Nationales des Jeunes Chercheurs en Inteligence Artificielle (RJCIA)*, pages 1–14, July 2007.

[10] F. Alkhateeb. *Querying RDF(S) with Regular Expressions*. PhD thesis, Université Joseph Fourier, Grenoble (FR), 2008.

[11] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Complex path queries for RDF graphs. In *Poster proceedings of the 4th International Semantic Web Conference (ISWC'05), Galway (IE)*, 2005.

[12] F. Alkhateeb, J.-F. Baget, and J. Euzenat. RDF with regular expressions. Research report 6191, INRIA, Montbonnot (FR), 2007.

[13] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Constrained regular expressions in SPARQL. In *Proceedings of the 2008 International Conference on Semantic Web and Web Services (SWWS'08), Las Vegas (NV US)*, pages 91–99, 2008.

[14] K. Anyanwu, A. Maduko, and A. P. Sheth. SPARQ2L: towards support for subgraph extraction queries in RDF databases. In *Proceedings of the 16th international conference on World Wide Web (WWW'07)*, pages 797–806, 2007.

[15] J.-F. Baget. RDF entailment as a graph homomorphism. In *Proceedings of the 4th International Semantic Web Conference (ISWC'05), Galway (IE)*, pages 82–96, 2005.

[16] J. L. Balcazar, J. Diaz, and J. Gabarro. *Structural complexity 1*. Springer-Verlag, New York (NY, USA), 1988.

[17] D. Beckett. Turtle - terse RDF triple language. Technical report, Hewlett-Packard, Bristol (UK), 2006.

[18] D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, 2004. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[19] J. Broekstra. SeRQL: Sesame RDF query language. In *SWAP Delivrable 3.2 Method Design*, 2003.

[20] P. Buneman. Semistructured data. In *Tutorial in Proceedings of the 16th ACM Symposium on Principles of Database Systems*, pages 117–121, 1997.

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

[22] P. Buneman, S. B. Davidson, and D. Suciu. Programming constructs for unstructured data. In *Proceedings of the 1995 International Workshop on Database Programming Languages*, page 12, 1995.

[23] J. J. Carroll and G. Klyne. RDF concepts and abstract syntax. W3C recommendation, W3C, February 2004.

[24] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[25] M. P. Consens and A. O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 404–416, 1990.

[26] O. Corby, R. Dieng, and C. Hébert. A conceptual graph model for W3C resource description framework. In *Proceedings of the International Conference on Conceptual Structures*, pages 468–482, 2000.

[27] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 323–330, New York, NY, USA, 1987.

[28] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G+: Recursive queries without recursion. In *Proceedings of Second International Conference on Expert Database Systems*, pages 355–368, 1988.

[29] R. Cyganiak. A relational algebra for SPARQL. Technical Report HPL-2005-170, HP Labs, 2005. `http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html`.

[30] J. de Bruijn, E. Franconi, and S. Tessaris. Logical reconstruction of normative RDF. In *International Workshop on OWL: Experiences and Directions (OWLED 2005), Galway, Ireland*, 2005.

[31] O. de Moor and E. David. Universal regular path queries. *Higher-Order and Symbolic Computation*, 16(1-2):15–35, 2003.

[32] M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. A query language for a web-site management system. *ACM SIGMOD Record*, 26(3):4–11, 1997.

[33] E. Franconi and S. Tessaris. The semantics of SPARQL. W3C working draft, November 2005. `http://www.inf.unibz.it/krdb/w3c/sparql/`.

[34] S. W. Golomb and L. D. Baumert. Backtrack programming. *Journal of the ACM*, 12(5):516–524, 1965.

[35] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 394–399, 1999.

[36] C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 95–106, 2004.

[37] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of RDF query languages. In *Proceedings 3rd International Semantic Web Conference*, pages 502–517, Hiroshima (JP), 2004.

[38] S. Harris and N. Shadbolt. SPARQL query processing with conventional relational database systems. In *Web Information Systems Engineering (WISE'05 Workshops)*, pages 235–244, 2005.

[39] P. Hayes. RDF semantics. W3C Recommendation, February 2004.

[40] H. J. t. Horst. Extending the RDFS entailment lemma. In *Proceedings of the Third International Semantic web Conference (ISWC2004)*, pages 77–91, 2004.

[41] H. J. t. Horst. Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2):79–115, 2005.

[42] H. V. Jagadish. Incorporating hierarchy in a relational model of data. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989*, pages 78–87. ACM Press, 1989.

[43] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A declarative query language for RDF. In *Proceedings of the 11th International Conference on the World Wide Web (WWW2002)*, 2002.

[44] K. Kochut and M. Janik. SPARQLeR: Extended SPARQL for semantic association discovery. In *Proceedings of 4th European Semantic Web Conferenc (ESWC'07)*, pages 145–159, 2007.

[45] O. Lassila. Taking the RDF model theory out for a spin. In *Proceedings of the First International Semantic Web Conference on The Semantic Web (ISWC 2002)*, pages 307–317, London, UK, 2002. Springer-Verlag.

[46] Y. A. Liu, T. Rothamel, F. Yu, S. Stoller, and N. Hu. Parametric regular path queries. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, pages 219–230, 2004.

[47] A. Matono, T. Amagasa, M. Yoshikawa, and S. Uemura. A path-based relational RDF database. In *Proceedings of the 16th Australasian database conference (ADC'05)*, pages 95–103, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[48] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[49] E. Miller, R. Swick, and D. Brickley. Resource description framework RDF. Recommendation, W3C, 2004.

[50] M.-L. Mugnier and M. Chein. Conceptual graphs: Fundamental notions. *Revue d'intelligence artificielle*, 6(4):365–406, 1992.

[51] S. Muñoz, J. Pérez, and C. Gutierrez. Minimal deductive systems for rdf. In *Proceedings of the 4th European Semantic Web Conference(ESWC 2007)*, pages 53–67, 2007. `http://www.springerlink.com/content/g8w64n1264874118/`.

[52] M. Olson and U. Ogbuji. Versa: Path-based RDF query language, 2002. `http://copia.ogbuji.net/files/Versa.html`.

[53] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In P. S. Yu and A. L. P. Chen, editors, *Proceedings of 11th Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, 1995. IEEE Computer Society.

[54] J. Perez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *Proceedings of the 5th International Semantic Web Conference*, pages 30–43, Athens (GA US), 2006.

[55] A. Polleres. From SPARQL to rules (and back). In *Proceedings of the 16th World Wide Web Conference (WWW)*, pages 787–796, 2007.

[56] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, January 2008.

[57] A. Seaborne. RDQL - a query language for RDF. Member submission, W3C, 2004.

[58] A. Souzis. RxPath specification proposal, 2004. `http://rx4rdf.liminalzone.org/RxPathSpec`.

[59] R. E. Tarjan. Fast algorithms for solving path problems. *Journal of (ACM)*, 28(3):594–614, 1981.

[60] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the fourteenth annual ACM symposium on Theory of computing STOC'82*, pages 137–146, 1982.

[61] P. T. Wood. *Queries on Graphs*. PhD thesis, Department of Computer Science, University of Toronto, 1988.

[62] M. Yannakakis. Graph-theoretic methods in database theory. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 230–242, 1990.

A Proofs

Theorem 1 *Let G and H be two GRDF graphs, then $G \models_{RDF} H$ if and only if there is an RDF homomorphism from H into G.*

Proof. The proof of this theorem is an immediate consequence of the proof of Theorem 3, since each GRDF graph is a PRDF graph. Moreover, any PRDF homomorphism between GRDF graphs is an RDF homomorphism and, by Proposition 3, PRDF entailment applied to GRDF graphs is equivalent to RDF entailment.

□

Consequence 1 *Let P be a GRDF graph and G be an RDF graph. The set $\mathcal{S}(P, G)$ of answers to P in G is*

$$\mathcal{S}(P, G) = \{\pi|_{\mathcal{B}(P)} \mid \pi \textit{ is an RDF homomorphism from } P \textit{ into } G\}$$

Proof. We must show that to each RDF homomorphism π corresponds an assignment σ such that $G \models_{RDF} \sigma(P)$ coinciding on $\mathcal{B}(P)$ assignments and vice versa.

($\Rightarrow$) Let G and P be two GRDF graphs and π be an RDF homomorphism from P into G. We want to show that there exists an assignment σ from $X \subseteq \mathcal{B}$ to $\mathcal{T}(G)$ such that $G \models_{RDF} \sigma(P)$ and $\sigma|_{\mathcal{B}(P)} = \pi|_{\mathcal{B}(P)}$.

Assume an RDF homomorphism π from P into G. Since, π preserves constants, its application is equivalent to that of the assignment $\sigma = \pi|_{\mathcal{B}(P)}$. Then, there exists the identity RDF homomorphism from $\sigma(P)$ into G, then, by Theorem 1, $G \models_{RDF} \sigma(P)$.

($\Leftarrow$) Let G and P be two GRDF graphs and σ be an assignment from $X \subseteq \mathcal{B}$ to $\mathcal{T}(G)$ such that $G \models_{RDF} \sigma(P)$. We want to show that there exists an RDF homomorphism π from P to G such that $\sigma|_{\mathcal{B}(P)} = \pi|_{\mathcal{B}(P)}$.

Let $\sigma : X \longrightarrow \mathcal{T}(G)$ be an assignment such that $G \models_{RDF} \sigma(P)$. According to Theorem 1, there exists an RDF homomorphism π_1 from $\sigma(P)$ into G. Consider the map π_2 from P to $\sigma(P)$, such that if $x \in dom(\sigma)$, then $\pi_2(x) = \sigma(x)$, otherwise, $\pi_2(x) = x$. It is clear that π_2 is an RDF homomorphism from P into $\sigma(P)$ since $\forall\langle s, p, o\rangle \in P$, $\langle\pi_2(s), \pi_2(p), \pi_2(o)\rangle \in \sigma(P)$. Now, the map defined by $\pi = \pi_1 \circ \pi_2$ is an RDF homomorphism from P into G.

□

Proposition 1 PATH SATISFIABILITY *in which* $X = \emptyset$ *(*$R \in \mathcal{R}$ *is a regular expression that does not contain variables) is in* NLOGSPACE *in* G *and* R*.*

Proof. The labels of paths between x and y form a regular language $P_{x,y}$ [62]. So, construct a non-deterministic finite automaton A_G accepting the regular language $P_{x,y}$ with initial state x and final state y (G can be transformed to an equivalent NDFA in NLOGSPACE). Constructing a NDFA M accepting $L^*(R)$, the language generated by R, can be done in NLOGSPACE. Constructing the product automaton $\mathcal{P}$, that is, the intersection of G and M, can be done in NLOGSPACE. Checking if the pairs $\langle x, y\rangle$ satisfies $L^*(R)$ is equivalent to checking whether $L^*(\mathcal{P})$ is not empty, and each of these operations can be done in NLOGSPACE in $\mathcal{P}$ [48,7] (with the fact that the class of LOGSPACE transformations is closed under composition [16]). An automaton for the intersection of $L^*(R)$ with M is constructed by taking the product of the automaton for the two languages. That is, the states of the product automaton are of the form $\langle s, u\rangle$ such that s is a state of M and u is a node of G; and there exists a transition on letter a (respectively, letter b) from a state $\langle s, u\rangle$ to another state $\langle t, v\rangle$ if M has a transition on a (respectively, on letter $!a$[5]) from s to t and $\langle u, a, v\rangle \in G$ (respectively, $\langle u, b, v\rangle \in G$ and $b \neq a$). The construction is similar to the one presented in [62] without atomic negation.

□

When regular expressions do not contain variables, there is no need to guess a map and the problem is reduced to the following decision problem [48,7]:

REGULAR PATH [48]
Instance: a directed labeled graph G, two nodes x, y of G, a map μ, and a regular expression pattern $R \in \mathcal{R}(U, B)$.
Question: Does the pair $\langle x, y\rangle$ satisfies $L^*(\mu(R))$?

Proposition 2 PATH SATISFIABILITY *is in* NP.

Proof. PATH SATISFIABILITY is in NP, since each variable in the regular expression pattern R can be mapped (assigned) to p terms, where p denotes the number of terms appearing as predicates in G. If the number of variables in R is n, then there are (p^n) possible assignments (mappings) in all. Once an assignment of terms to variables is fixed, the problem is reduced to PATH SATISFIABILITY ($\Sigma \subseteq \mathcal{U}$), which is in NLOGSPACE.

It follows that a non-deterministic algorithm needs to guess a map μ and check in NLOGSPACE if the pair $\langle x, y\rangle$ satisfies $L^*(\mu(R))$.

□

[5] $!a$ is an atomic negation, *i.e.*, a negated uriref.

Proposition 3 *If G is a PRDF graph with $pred(G) \subseteq \mathcal{U} \cup \mathcal{B}$, i.e., G is a GRDF graph, and I be an interpretation of a vocabulary $V \supseteq \mathcal{V}(G)$, then I is an RDF model of G (Definition 5) iff I is a PRDF model of G (Definition 21).*

Proof. We prove both directions of the proposition.

($\Rightarrow$) Assume that I is an RDF model of G, then there exists an extension I' of I to $\mathcal{B}(G)$ such that $\forall \langle s, p, o\rangle \in G$, $\langle I'(s), I'(o)\rangle \in I_{EXT}(I'(p))$ (Definition 5). Since $pred(G) \subseteq \mathcal{U} \cup \mathcal{B}$, $\langle I'(s), I'(o)\rangle$ supports p in I' (Definition 20)(with a word $w = p$), *i.e.*, I is also a PRDF model (Definition 21).

($\Leftarrow$) Assume that I is a PRDF model of G, then there exists an extension I' of I to $\mathcal{B}(G)$ such that $\forall \langle s, p, o\rangle \in G$, $\langle I'(s), I'(o)\rangle$ supports p in I' (Definition 21). Since $pred(G) \subseteq \mathcal{U} \cup \mathcal{B}$, $\epsilon \notin L^*(p)$. So there there exists a word of $length = 1$ where $w \in L^*(p)$, $w = p$, and a sequence of resources of IR $I'(s) = r_0$, $I'(o) = r_1$ such that $\langle r_0, r_1\rangle \in I_{EXT}(I'(w))$ (Definition 20). So $\forall \langle s, p, o\rangle \in G$, $\langle I'(s), I'(o)\rangle \in I_{EXT}(I'(p))$ (by replacing r_0 with $I'(s)$, r_1 with $I'(o)$, and w with p). So I is also an RDF model (Definition 5).

□

Theorem 2 *Let G be a GRDF graph and H be a* ground *PRDF graph, then* PRDF-GRDF ENTAILMENT *is in* NLOGSPACE.

Proof. We prove first that the PRDF-GRDF HOMOMORPHISM problem is in NLOGSPACE. If H is ground, for each node x in H, $\pi(x)$ is determined in G. Then it remains to verify independently, for each triple $\langle s, R, o\rangle$ in H, if $\langle \pi(s), \pi(o)\rangle$=$\langle s, o\rangle$ satisfies $\pi(R) = R$. Since each of these operations corresponds to the case of PATH SATISFIABILITY, in which $\Sigma \subseteq \mathcal{U}$ and $X = \emptyset$, the complexity of each of them is NLOGSPACE (see Proposition 1) (Since H is ground, R does not contain variables). So, the total time is also NLOGSPACE. Given the equivalence between PRDG-GRDF ENTAILMENT and checking the existence of PRDF homomorphism (Theorem 3), PRDF-GRDF ENTAILMENT is thus in NLOGSPACE.

□

Theorem 3 *Let G be a GRDF graph, and H be a PRDF graph, then there is a PRDF homomorphism from H into G iff $G \models_{PRDF} H$.*

We have proven Theorem 3 via a transformation to hypergraphs following the proof framework in [15]. Since this requires a long introduction to hypergraphs, we prefer here to give a simple direct proof to Theorem 3.

The following lemma (called satisfiability lemma) will be used to prove the theorem. We prove it through the construction of the isomorphic model. A similar proof can be found in [40].

Lemma (Satisfiability lemma [40]) *Each GRDF graph is satisfiable.*

Proof. To each GRDF graph G we associate an interpretation of $\mathcal{V}(G)$, noted $ISO(G)$, called an isomorphic model of G. We prove that $ISO(G)$ is a model of G. It follows that every GRDF graph admits a model, so it is satisfiable.

(1) Construction of $ISO(G)$.
To each term $x \in term(G)$, we associate a distinct resource $\iota(x)$ (if $x \in L_p$, $\iota(x) = x$) :
  (i) $IR = \{\iota(x) \mid x \in term(G)\}$, note that ι is a bijection between $term(G)$ and IR;
  (ii) $IP = \{\iota(x) \mid x \in pred(G)\}$;
  (iii) $\forall x \in \mathcal{U}(G) \cup \mathcal{L}_t(G), I(x) = \iota(x)$;
  (iv) $\forall p \in IP$, $I_{EXT}(p) = \{\langle x, y\rangle \in IR \times IR \mid \langle \iota^{-1}(x), \iota^{-1}(p), \iota^{-1}(y)\rangle \in G\}$.

(2) Let us prove that $ISO(G)$ is a model of G.
  (a) $ISO(G)$ is an interpretation of $\mathcal{V}(G)$ (Definition 3).
  (b) ι is an extension of ISO to $\mathcal{B}(G)$ (Definition 4).
  (c) It remains to prove (Definition 5), that for all $\langle s, p, o\rangle \in G$, $\langle \iota(s), \iota(o)\rangle \in I_{EXT}(\iota(p))$. If $\langle s, p, o\rangle \in G$, then $\iota(p) \in IP$ $(1.ii)$. Then $I_{EXT}(\iota(p)) = \{\langle x, y\rangle \in IR \times IR \mid \exists s, o \in term(G)$ with $\iota(s) = x, \iota(o) = y$ and $\langle s, p, o\rangle \in G\}$ $(1.iv)$, *i.e.*, $\langle \iota(s), \iota(o)\rangle \in I_{EXT}(\iota(p))$.

□

Proof. We prove both directions of Theorem 1.

($\Rightarrow$) Assume that there exists a PRDF homomorphism π from H into G ($\pi : term(H) \rightarrow term(G)$). We want to prove that $G \models_{PRDF} H$, *i.e.*, that every model of G is a model of H. Consider the interpretation I of a vocabulary $V = U \cup L$.

If I is a model of G, then there exists an extension I' of I to $\mathcal{B}(G)$ such that $\forall \langle s, p, o\rangle \in G$, $\langle I'(s), I'(o)\rangle \in I_{EXT}(I'(p))$ (Definition 5). We want to prove that I is also a model of H, *i.e.*, that there exists an extension I'' of I to $\mathcal{B}(H)$ such that $\forall \langle s, R, o\rangle \in H$, $\langle I''(s), I''(o)\rangle$ supports R in I''.

Let I'' be the map defined by:

$$\forall x \in \mathcal{T}, I''(x) = \begin{cases} (I' \circ \pi)(x), & \text{if } \pi \text{ is defined;} \\ x, & \text{otherwise.} \end{cases}$$

.

We show that I'' verifies the following properties:

(1) I'' is an interpretation of $\mathcal{V}(H)$ (in particular, $I''(x) = I(x), \forall x \in (\mathcal{V}(H) \cap nodes(H)))$[6].
(2) I'' is an extension to variables of H, *i.e.*, $\forall x \in \mathcal{V}(H)$, $I''(x) = I(x)$ (Definition 4).
(3) I'' satisfies the conditions of PRDF models (Definition 21), *i.e.*, for every triple $\langle s, R, o\rangle \in H$, the pair of resources $\langle I''(s), I''(o)\rangle$ supports R in I''.

Now, we prove the satisfaction of these properties:

(1) From the definition of I'', I'' interprets all $x \in \mathcal{V}(H)$. Moreover, since each term $x \in (\mathcal{V}(H) \cap nodes(H))$ is mapped by π to a term $x \in \mathcal{V}(G)$ and $\pi(x) = x$, $I''(x) = I'(x) = I(x)$.
(2) $\forall x \in \mathcal{V}(H)$, $I''(x) = (I' \circ \pi)(x)$ (definition of I''). Now, if $\pi(x)$ is defined, $I''(x) = I'(x)$ (since $\pi(x) = x$ by Definition 22). Since $I'(x) = I(x)$ ($\forall x \in \mathcal{V}(H)$, Definition 4), $I''(x) = I(x)$. Otherwise, $I''(x) = I'(x) = I(x)$ ($\forall x \in \mathcal{V}(H)$, definition of I'')
(3) It remains to prove that for every triple $\langle s, R, o\rangle \in H$, the pair of resources $\langle I''(\pi(s)), I''(\pi(o))\rangle$ supports R in I'' (by Definition 20):
    (i) If the empty word $\epsilon \in L^*(R)$ and $\pi(s) = \pi(o) = y$ ($y \in term(G)$, Definition 22), then $I''(s) = (I' \circ \pi)(s) = I'(y)$, and $I''(o) = (I' \circ \pi)(o)$ $= I'(y)$. So $I''(s) = I''(o) = I'(y)$. Hence, $\langle I''(s), I''(o)\rangle$ supports R in I'' (Definition 21).
    (ii) If $\exists\langle n_0, p_1, n_1\rangle, \ldots, \langle n_{k-1}, p_k, n_k\rangle$ in G such that $n_0 = \pi(s)$, $n_k = \pi(o)$, and $p_1 \cdot \ldots \cdot p_k \in L^*(\pi(R))$ (*cf.* Definition 22). It follows that $\langle I'(\pi(s)), I'(n_1)\rangle \in I_{EXT}(I'(p_1))$, …, $\langle I'(n_{k-1}), I'(\pi(o))\rangle \in I_{EXT}(I'$ $(p_k))$ (Definition 5). So the two resources $\langle I'(\pi(s))$, $I'(\pi(o))\rangle$ supports $\pi(R)$ in I'. $\langle I'(\pi(s)), I'(\pi(o))\rangle$ supports $\pi(R)$ in I'' (since $I'' = (I' \circ \pi)$, we have $\forall x \in term(H), I''(x) = I'(\pi(x))$ and $\pi(x) \in term(G)$. Moreover, we can choose every variable b appearing in H to be interpreted by the resource of $\pi(b)$). Hence, $\langle I''(s), I''(o)\rangle$ supports R in I'' (since for every word $w \in \pi(R), w \in R$).

($\Leftarrow$) Assume that $G \models_{PRDF} H$. We need to prove that there is a PRDF homomorphism from H into G. Every model of G is also a model of H. In particular, the isomorphic model $ISO = \langle IR,IP,I_{EXT}, I_S, I_L\rangle$ of G, where there exists a bijection ι between $term(G)$ and IR (*cf.* the satisfiability lemma). ι is an extension of ISO to $\mathcal{B}(G)$ such that $\forall\langle s, p, o\rangle \in G$, $\langle\iota(s), \iota(o)\rangle \in I_{EXT}(\iota(p))$ (Definition 5). Since ISO is a model of H, there exists an extension I' of ISO to $\mathcal{B}(H)$ such that $\forall\langle s, R, o\rangle$, $\langle I'(s), I'(o)\rangle$ supports R in I' (Definition 21). Let us consider the function $\pi = (\iota^{-1} \circ I')$. To prove that π is a PRDF homomorphism from H into G, we must prove that:

[6] An interpretation can be a model of a PRDF graph H even if it does not interpret all terms of H. This is due to the disjunction operator that occurs inside constrained regular expressions.

(1) π is a map from $term(H)$ into $term(G)$;
(2) $\forall x \in \mathcal{V}(H)$, $\pi(x) = x$;
(3) $\forall \langle s, R, o \rangle \in H$, either
 (i) the empty word $\epsilon \in L^*(R)$ and $\pi(s) = \pi(o)$; or
 (ii) $\exists \langle n_0, p_1, n_1 \rangle, \ldots, \langle n_{k-1}, p_k, n_k \rangle$ in G such that $n_0 = \pi(s)$, $n_k = \pi(o)$, and $p_1 \cdot \ldots \cdot p_k \in L^*(\pi(R))$.

(1) Since I' is a map from $term(H)$ into IR and ι^{-1} is a map from IR into $term(G)$, $\pi = (\iota^{-1} \circ I')$ is clearly a map from $term(H)$ into $term(G)$ ($term(H) \xrightarrow{I'} IR \xrightarrow{\iota^{-1}} term(G)$).
(2) $\forall x \in \mathcal{V}(H)$, $I'(x) = \iota(x)$ (Definition 4 and the satisfiability lemma. $\forall x \in \mathcal{V}(H)$, $(\iota^{-1} \circ I')(x) = (\iota^{-1} \circ \iota)(x) = x$.
(3i) If $\epsilon \in L^*(R)$ and $I'(s) = I'(o) = r \in IR$ (Definition 20), then $\pi(s) = (\iota^{-1} \circ I')(s) = \iota^{-1}(r)$, and $\pi(o) = (\iota^{-1} \circ I')(o) = \iota^{-1}(r)$. So $\pi(s)$ =$\pi(o)$= $\iota^{-1}(r)$.
(3ii) If there exists a word of length $n \geq 1$ $w = a_1 \cdot \ldots \cdot a_n$ where $w \in L^*(R)$ and $a_i \in U \cup \mathcal{B}(G)$ $(1 \leq i \leq k)$, and there exists a sequence of resources of IR $I'(s) = r_0, \ldots, r_k = I'(o)$ such that $\langle r_{i-1}, r_i \rangle \in I_{EXT}(I'(a_i))$, $1 \leq i \leq k$ (Definition 20). It follows that $\langle n_{i-1}, p_i, n_i \rangle \in G$ with $n_i = \iota^{-1}(r_i)$, and $p_i = (\iota^{-1} \circ I')(a_i)$ (construction of $ISO(G)$, the satisfiability lemma). So $(\iota^{-1} \circ I')(s) = \iota^{-1}(r_0) = n_0$, $(\iota^{-1} \circ I')(o) = \iota^{-1}(r_k) = n_k$, and $p_1 \cdot \ldots \cdot p_k \in L^*((\iota^{-1} \circ I')(R))$.

□

Consequence 2 *Let P be a PRDF graph and G be an RDF graph. The set $\mathcal{S}(P, G)$ of answers to P in G is*

$$\mathcal{S}(P, G) = \{\pi|_{\mathcal{B}(P)} \mid \pi \textit{ is a PRDF homomorphism from } P \textit{ into } G\}$$

Proof. The proof is exactly the same as that of Consequence 1 by replacing $\models_{RDF}$ by $\models_{PRDF}$, RDF homomorphisms by PRDF homomorphisms and Theorem 1 by Theorem 3.

□

Theorem 4 PSPARQL QUERY EVALUATION *is in* NLOGSPACE *for ground basic graph patterns and* NP*-complete for basic graph patterns.*

Proof. The first assertion (NLOGSPACE for ground PRDF graphs) follows directly from Theorem 2. For the second assertion (NP-complete), when reduced to PRDF graphs, PSPARQL QUERY EVALUATION is equivalent to PRDF-GRDF HOMOMORPHISM (Definition 12). Indeed, PRDF-GRDF HOMOMORPHISM can be reduced to PSPARQL QUERY EVALUATION with the empty SELECT clause ($\vec{B} = \emptyset$). In such a case, PSPARQL QUERY EVALUATION is true when there exist a PRDF homomorphism between P and G. On the other way, PRDF-GRDF EVALUATION is reduced to PRDF-GRDF HOMOMORPHISM between G and $\sigma(P)$. Since PRDF-GRDF HOMOMORPHISM is NP-complete, then PSPARQL QUERY EVALUATION is NP-complete for PRDF graphs.

$\square$

B Reach algorithm

Algorithm 4 reuses the definition of matching two regular expression patterns found in [46].

Matching. Let R_1 and R_2 be two regular expression patterns, then R_2 *matches* R_1 under the mapping μ, denoted by $match(R_2, R_1, \mu)$, if one of the following conditions holds: (1) $R_1 = \mu(R_2)$; (2) $R_2 \in \mathcal{B}$ and $R_2 \notin dom(\mu)$; (3) $R_1, R_2 \in \mathcal{B}$ and ($\mu(R_2) = R_1$ or $R_2 \notin dom(\mu)$); (4) $R_2 = \#$; (5) $R_2 =!R_3$, and recursively, R_1 does not match R_3; (6) $R_1 = (e_1, \ldots, e_k)$, $R_2 = (a_1, \ldots, a_k)$, and recursively e_i matches a_i, $\forall 1 \leq i \leq k$, where e_i, a_i are the atomic elements of R_1, R_2, respectively. For example, the regular expression pattern $(\texttt{?z} \cdot \texttt{?y})$ matches the regular expression pattern $(\texttt{ex:train} \cdot \texttt{ex:plane})$ with the mapping $\{\langle\texttt{?z}, \texttt{ex:train}\rangle, \langle\texttt{?y}, \texttt{ex:plane}\rangle\}$.

Algorithm 4 uses a non deterministic finite automaton, denoted by NDFA, that recognizes a language equivalent to a given regular expression pattern. It can be constructed in the usual way (*cf.* [5]).

Algorithm 4. $Reach(G, R, v_0, \mu_i)$

```
Construct the NDFA $A = \langle S, s_0, \delta, F \rangle$ accepting $L^*(R)$;
Eliminate $\epsilon$-transitions from $A$;
$R \leftarrow \{\}$;
$W \leftarrow \{\}$;
**for** $\langle v_0, el, v \rangle \in G$ **do**
  **for** $\langle s_0, tl, s \rangle \in A$ **do**
    **if** $match(tl, el, \mu_i)$ **then**
      $\mu \leftarrow \{\langle tl, el \rangle\}$;
      **if** $\mu$ and $\mu_i$ are compatible **then**
        $\mu_2 \leftarrow \mu \bowtie \mu_i$;
        $W \leftarrow W \cup \{\langle v, s, \mu_2 \rangle\}$;
      **end if**
    **end if**
  **end for**
**end for**
**while** exists $\langle v, s, \mu \rangle \in W$ **do**
  $R \leftarrow R \cup \{\langle v, s, \mu \rangle\}$;
  $W \leftarrow W - \{\langle v, s, \mu \rangle\}$;
  **for** $\langle v, el, v_1 \rangle \in G$ **do**
    **for** $\langle s, tl, s_1 \rangle \in A$ **do**
      **if** $match(tl, el, \mu)$ **then**
        $\mu_1 \leftarrow \{\langle tl, el \rangle\}$;
        **if** $\mu$ and $\mu_1$ are compatible **then**
          $\mu_2 \leftarrow \mu \bowtie \mu_1$;
          **if** $\langle v_1, s_1, \mu_2 \rangle \notin R$ **then**
            $W \leftarrow W \cup \{\langle v_1, s_1, \mu_2 \rangle\}$;
          **end if**
        **end if**
      **end if**
    **end for**
  **end for**
  **if** $s \in F$ **then**
    $S(G) \leftarrow S(G) \cup \{\langle v_0, v, \mu \rangle\}$;
  **end if**
**end while**
```