INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

RDF with regular expressions

Faisal Alkhateeb — Jean-François Baget — Jérôme Euzenat

N° 6191

May 2007

Thème SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6191--FR+ENG

RDF with regular expressions

Faisal Alkhateeb , Jean-François Baget , Jérôme Euzenat

Thème SYM — Systèmes symboliques
Projet EXMO

Rapport de recherche n° 6191 — May 2007 — 32 pages

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 querying an RDF knowledge base, some, such as SPARQL, are based on the formal semantics of RDF and the concept of semantic consequence, others, inspired by the work in databases, use regular expressions making it possible to search the paths in the graph associated with the knowledge base. In order to combine the expressivity of these two approaches, we define a mixed language, called PRDF (for "Paths RDF") in which the arcs of a graph can be labeled by regular expressions. We define the syntax and the semantics of these objects, and propose a correct and complete algorithm which, by a kind of homomorphism, calculates the semantic consequence between an RDF graph and a PRDF graph. This algorithm is the heart of query answering for the PSPARQL query language, the extension of the SPARQL query language which we propose and have implemented: a PSPARQL query allows to query an RDF knowledge base using graph patterns whose predicates are regular expressions.

Key-words: semantic web, query language, RDF, SPARQL, regular expressions.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

RDF avec des expressions régulières
Rapport de recherche
INRIA

Résumé : RDF est un langage de représentation de connaissances dédié à l'annotation de ressources dans le cadre du web sémantique. Parmi les langages de requêtes permettant d'interroger une base de connaissances RDF, certains, tels que SPARQL, s'appuient sur la sémantique formelle de RDF et la notion de conséquence sémantique, d'autres, inspirés par des travaux en bases de données, utilisent des expressions régulières permettant de chercher des chemins dans le graphe associé à la base de connaissances. Afin de conjuguer l'expressivité de ces deux approches, nous définissons un langage mixte, appelé PRDF (pour "Paths RDF") dans lequel les arcs d'un graphe peuvent être étiquetés par des expressions régulières. Nous définissons la syntaxe et la sémantique de ces objets, et proposons un algorithme correct et complet qui, par une sorte d'homomorphisme, calcule la conséquence sémantique entre un graphe RDF et un graphe PRDF. Cet algorithme est au cœur de PSPARQL, l'extension du langage de requêtes SPARQL que nous proposons et avons implémenté: une requête PSPARQL permet d'interroger une base de connaissances RDF en utilisant des patterns dont les prédicats sont des expressions régulières.

Mots-clés : web sémantique, langage de requête, RDF, SPARQL, expressions régulières.

Contents

1 Introduction

RDF (Resource Description Framework [28]) is a knowledge representation language dedicated to the annotation of documents and more generally of resources within the framework of the Semantic Web. Syntactically, an RDF document can be represented indifferently by a set of triples (subject, predicate, object), by an XML document, or a directed labeled graph. An RDF graph is equipped with a model theoretic semantics [23], which formally defines the concept of semantic consequence between RDF graphs. A graph homomorphism makes it possible to calculate this consequence in a sound and complete way [20, 7]. Nowadays, more resources are annotated via RDF due to its simple data model, formal semantics, and a sound and complete inference mechanism.

Several languages for querying RDF have been developed (cf. [21] for a comparison of query languages for RDF). SPARQL [33], based upon RDF semantic consequence, is a W3C candidate recommendation for querying RDF. Answers to SPARQL queries, as specified through RDF entailment [7], can be computed by a kind of graph homomorphisms known as projection in conceptual graphs [29]. More precisely, the answer to a SPARQL query Q relies on calculating the set of possible projections from the basic graph pattern(s) of Q into the RDF graph G representing the database.

Another approach, that has been successfully used in databases [14, 17, 27, 34, 36] but little in the context of the semantic web, uses path queries, *i.e.*, regular expressions, for finding regular paths in a database graph. It is implemented in path languages like G+ [17]. 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.

Both approaches are orthogonal, *i.e.*, some queries that can be expressed in one approach cannot be expressed in the other. As shown in Example 1, a query whose homomorphic image in the database is not a path cannot be expressed by a regular expression, while RDF semantics does not allow expressing paths of undetermined length. Furthermore, regular expressions provide a simple way to capture additional information along paths that may not be provided by SPARQL graph patterns, but they are not powerful enough as a query language.

Example **1** *The regular path expression* $(\texttt{ex:son}|\texttt{ex:daughter})^+ \cdot \texttt{\_:b5}$*, which represents paths of unknown length, cannot be expressed in SPARQL. On the other hand, the query graph of Fig. 1, which represents a basic graph pattern of a SPARQL query, cannot be expressed by a regular path expression.*

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 expressions [5], supports path queries (cf. Example 2). In order to formally define that language, we first introduce Paths RDF (PRDF) as an extension of RDF in which arcs of the graphs can be labeled by regular expressions. We provide sound and complete algorithm for checking that a PRDF graph is entailed by some RDF graph.

Example **2** *Consider the RDF graph G of Fig. 3 representing a social network. Let us suppose that we wish to find among, the entities related to Faisal or the descendants of Faisal, those which know*

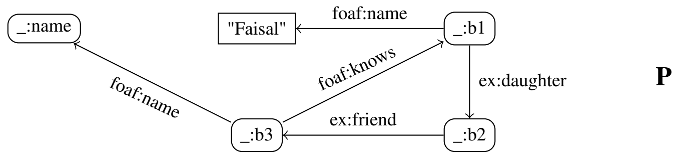

Figure 1: A query graph that cannot be expressed by a path expression.

Faisal. Let us suppose that we want to know their names and their address email. This query can be expressed by using PRDF graph P Fig. 2. Let us note that this query cannot be expressed if one uses only SPARQL graph patterns or regular expressions.

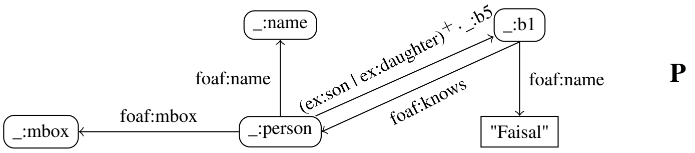

Figure 2: A graph with regular expressions labeling the arcs.

We propose a query language, called PSPARQL, that extends SPARQL graph patterns with regular expression patterns. We present the syntax and the semantics of PSPARQL. We provide a sound and complete inference mechanism for querying RDF graphs with PSPARQL queries. We provide complexity results on evaluating PSPARQL graph patterns on RDF graphs. We give two methods, which are sound and complete for evaluating the basic graph pattern(s) of PSPARQL over RDF graphs.

The report is organized as follows: we present simple RDF in Section 2. Section 3 presents the two approaches mentioned so far for querying RDF graphs. In Section 4, we present the syntax and the semantics of an extension of RDF, called Path RDF or simply PRDF, as well as a sound and complete inference mechanism for querying RDF graphs with PRDF queries. The syntax of the PSPARQL language, an evaluation procedure for PSPARQL queries, and the complexity results of PSPARQL query evaluation are presented in Section 5. In Section 6, we present sound and complete algorithm for answering a PSPARQL query, *i.e.*, for calculating all answers to a PSPARQL query. We provide the first experimental results with an implementation of a PSPARQL query evaluator in Section 7. After a review of related works (Section 8), we conclude in Section 9.

2 Simple entailment of RDF graphs

This section is devoted to the presentation of the *simple RDF* knowledge representation language. We first recall (Section 2.1) its abstract syntax [13], its semantics (Section 2.2), using the notions of simple interpretations, models, simple entailment of [23]), then using homomorphisms in Section 2.3 to characterize simple RDF entailment (as done in [7] for a graph-theoretical encoding of RDF, and in [20] for a database encoding), instead of the equivalent interpolation lemma of [23].

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* [23]: 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 *blanks*. The set $\mathcal{U} \cup \mathcal{L}$ of *names* is called the *vocabulary*. From now on, we use different notations for the elements of these sets: a blank will be prefixed by _: (like _:x1), a literal will be between quotations (like "27"), the rest will be urirefs (like foaf:Person — foaf: is a name space used for representing personal information — or ex:friend).

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.*

A possible extension to RDF syntax, which has no effects in the simple RDF semantics, is to allow blanks as predicates and literals as subjects. We called this extension *generalized RDF graphs*, or simply GRDF graphs. So, 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 note $subj(G) = \{s \mid \langle s, p, o\rangle \in G\}$ 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 the *nodes* of G, denoted $nodes(G) = subj(G) \cup obj(G)$, the set of elements appearing either as subject or object in a triple of G. A *term* of G is an element of $term(G) = subj(G) \cup pred(G) \cup obj(G)$. If $\mathcal{X} \subseteq \mathcal{T}$ is a set of terms, we note $\mathcal{X}(G) = \mathcal{T} \cap term(G)$. As an example, $\mathcal{V}(G)$ is the set of names appearing in G.

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

GRDF graphs as graphs: A *simple (G)RDF graph* can be represented graphically as a directed labeled multigraph[1] (V, E, γ, λ) where the set of nodes V is the set of terms appearing as a subject or object at least in a triple of G, the set of arcs E is the set of triples of G, γ associated to each arc a pair of nodes (its extremities) $\gamma(e) = (\gamma_1(e), \gamma_2(e))$ where $\gamma_1(e)$ is the source of the arc e and $\gamma_2(e)$ its destination; finally, λ label the nodes and the arcs of the graph: if s is the node of V (*i.e.*, a term), then $\lambda(s) = s$, and if e is the arc of E (*i.e.*, a triple (s, p, o)), then $\lambda(e) = p$. By drawing these graphs, the nodes resulting from literals are represented by rectangles while the others

[1] In a directed labeled multigraph there can be many arcs between two given nodes.

are represented by rounded corners. Consequently, we confuse them aimed "multigraphe" or a "set of triples" of a graph RDF, and speak indifferently about its nodes, its arcs, or the triples which make it up.

Example **3** *The RDF graph defined by the set of triples* $\{$*(* `_:b1, foaf:name, "Faisal"`*), (* `_:b1, ex:daughter, _:b2`*), (* `_:b2, ex:friend, _:b3`*), (* `_:b3, foaf:knows, _:b1`*), (* `_:b3, foaf:name, _:name`*)*$\}$ *is represented graphically in Fig. 1. Intuitively, this RDF graph means that an entity named (* `foaf:name`*)* `"Faisal"` *has a daughter (* `ex:daughter`*) that has a friend (* `ex:friend`*) knowing (* `foaf:knows`*) the entity* `"Faisal"`*. The name of this friend is not known.*

2.2 Simple RDF semantics

[23] introduces different semantics for RDF graphs. Since RDF and RDFS entailments can be polynomially reduced to simple entailment via RDF or RDFS rules [20, 23, 25], we are only interested in the *simple semantics* without RDF/RDFS vocabulary [11]. The definitions of interpretations, models, satisfiability, and entailment correspond to the *simple interpretations*, *simple models*, *simple satisfiability*, and *simple entailments* of [23].

Definition 2 (Interpretations) *Let* $V \subseteq \mathcal{V}$ *be a vocabulary. An interpretation of* V *is a 5-tuple* $I = \langle IR, IP, I_{EXT}, I_S, I_L \rangle$ *of* V*, where:*

- IR *is a set of resources containing plain literals of* $\mathcal{L}_p$*;*
- $IP \subseteq IR^2$ *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*.*

If $I = \langle IR, IP, I_{EXT}, I_S, I_L \rangle$ is an interpretation of a vocabulary V, we also note 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 true under the interpretation I. For that matter, we need to extend the interpretations of a vocabulary to interpret the blanks in G.

[2] To facilitate the notations, and without loss of generality, $IP \subseteq IR$, which is true for RDF, but not necessary for simple RDF.

Definition 3 (Extension to Blanks) *Let I be an interpretation of a vocabulary $V \subseteq \mathcal{V}$, and $B \subseteq \mathcal{B}$ a set of blanks. An extension of I to B is a mapping $I' : \mathcal{V} \cup B \rightarrow IR$ such that $\forall x \in V$, $I'(x) = I(x)$.*

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

Definition 4 (Models) *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 $\langle s, p, o \rangle \in G$, $\langle I'(s), I'(o) \rangle \in I_{EXT}(I'(p))$.*

In [22], $\langle I'(s), I'(o) \rangle \in I_{EXT}(I(p))$. This version is equivalent in the case of RDF graphs, and is necessary (see [25]) for GRDF graphs having blanks as predicates and PRDF graphs presented in Section 4.

The following definitions of satisfiability and entailment will be the same when we extend the syntax and the semantics of RDF. Two RDF graphs G, H are said equivalent if and only if $G \models_{RDF} H$ and $H \models_{RDF} G$.

Definition 5 (Satisfiability) *A GRDF graph G is satisfiable iff there exists an interpretation I that is a model of G.*

Lemma 1 *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 2).
   (b) ι is an extension of ISO to $\mathcal{B}(G)$ (Definition 3).
   (c) It remains to prove (Definition 4), 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.*, $\langle \iota(s), \iota(o) \rangle \in I_{EXT}(\iota(p))$.

Definition 6 (Entailment) *Let G and H be two GRDF graphs. Then G* entails *H (we note $G \models_{RDF} H$) iff every model of G is also a model of H.*

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

SIMPLE RDF ENTAILMENT is an NP-complete problem [20]. As polynomial subclasses of the problem, we can consider the case of ground graphs as queries [24], queries whose structure correspond to a polynomial subclass of CSPs [19], thanks to the transformation in [6, 30]. And more generally, the problem is polynomial when the number of blanks in the query is bounded.

2.3 Inference mechanism: graph homomorphism

Simple entailment in RDF [23] 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 [7, 20], 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) [29] (cf. [15] for precise relationship between RDF and CGs). We modify this definition to the following equivalent one that maps $term(H)$ into $term(G)$.

Definition 7 (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 $\forall x \in (V_1 \cap \mathcal{V})$, $\mu(x) = x$.*

Definition 8 (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 $\forall \langle s, p, o \rangle \in H$, $\langle \pi(s), \pi(p), \pi(o) \rangle \in G$.*

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 homomorphism (Definition 8) is similar to the one in [20] given for RDF graphs without proof. A proof is provided in [7] 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 GRDF (Theorem 1) to the PRDF graphs studied in Section 4. The proof of Theorem 1 will be a particular case of the proof of Theorem 2 for PRDF graphs (*cf.* Section 4.3).

Example **4** *Fig. 3 shows two GRDF graphs, Q and G (note that Q is the RDF graph P of Fig. 1, to which we added the following triple (* `_:b3`, `foaf:mbox`, `_:mbox`*)). The map π_1 defined by* {("`Faisal`", "`Faisal`"), (`_:b1`, `_:c1`), (`_:name`, "`Natasha`"), (`_:b3`, `ex:Person1`), (`_:b2`, `_:c2`), (`_:mbox`, "`natasha@yahoo.com`")} *is an RDF homomorphism from Q into G. The map π_2 defined by* {("`Faisal`", "`Faisal`"), (`_:b1`, `_:c1`), (`_:name`, "`Deema`"), (`_:b3`, `ex:Person2`), (`_:b2`, `_:c2`)} *is an RDF homomorphism from P into G. Note that π_2 does not extend to an RDF homomorphism from Q into G.*

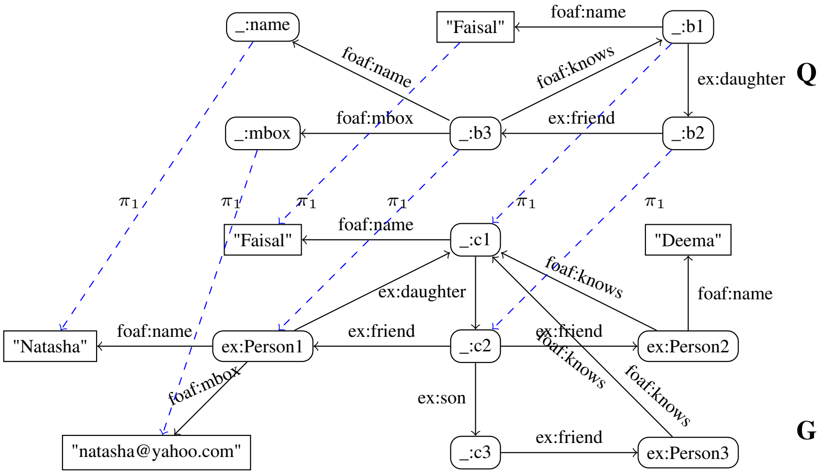

Figure 3: An RDF homomorphism of two GRDF graphs.

3 Querying RDF graphs

In this section we present two approaches for querying RDF graphs. In Section 3.1, a simplified version of the SPARQL query language is given, while insisting on its development in the top of RDF and its semantics. In Section 3.2, we show how "path queries" developed in databases, and which use the regular expressions, can be used to query RDF knowledge bases. Lastly, in Section 3.3, these two types of languages are shown to be orthogonal, and we discuss the significance of the combined approach which will be the main part of this report.

3.1 SPARQL: entailment based queries

We present here a simplified version of SPARQL, which will be enough to present and argue our extension. For complete version of SPARQL, the reader will be able to refer to the SPARQL draft [33] or to [31, 32] for formal semantics of SPARQL.

In SPARQL query language, a set of variables distinct from blanks is used. In the subset of SPARQL presented here, blanks and variables have the same behavior. For the sake of simplicity, we will not introduce a new class of objects and consider only blanks. Moreover,[3] we present only SELECT . . . FROM . . . WHERE . . . queries, and we ignore the key words which allow, for example, to filter (FILTER), to order (ORDER BY), or to limit (LIMIT ou/et OFFSET) the answers of a query.

[3] 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.

The basic building block of SPARQL queries is *graph patterns*. Informally, a *graph pattern* can be a triple pattern (*i.e.*, a GRDF triple), basic graph pattern (*i.e.*, a GRDF graph), union of graph patterns, optional graph pattern, or a constraint (cf. [33] for more details).

Definition 9 (Patterns and SPARQL queries) *A* SPARQL graph pattern *is defined inductively in the following way:*

- *every GRDF graph is a SPARQL graph pattern;*
- *if P_1, P_2 are SPARQL graph patterns and R is a SPARQL constraint, then (P_1 `AND` P_2), (P_1 `UNION` P_2), (P_1 `OPT` P_2), and (P_1 `FILTER` R) are SPARQL graph patterns.*

A *SPARQL query* is of the form `SELECT` $\vec{B}$ `FROM` u `WHERE` P where u is a URL of an RDF graph G, P is a SPARQL graph pattern and $\vec{B}$ is a tuple of blanks appearing in P. Intuitively, an answer to a SPARQL query is an instantiation π to the blanks of $\vec{B}$ by the terms of the RDF graph G such that π is a restriction of a proof that P is semantic consequence of G.

Operations in the 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. If P is a graph pattern, then $\mu(P)$ is the graph pattern obtained by the substitution of $\mu(b)$ to each blank $b \in \mathcal{B}(P)$. Two maps μ_1 and μ_2 are *compatibles* 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 note $\mu = \mu_1 \oplus \mu_2 : T_1 \cup T_2 \rightarrow \mathcal{T}$ the map defined by: $\forall x \in T_1, \mu(x) = \mu_1(x)$ and $\forall x \in T_2, \mu(x) = \mu_2(x)$. Let Ω_1 and Ω_2 be two sets of maps, then we define the following operations:

- *(joint)* $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2$ are compatibles $\}$;
- *(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 compatibles$\}$.

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

- if P is a GRDF graph, $\mathcal{S}(P, G) = \{\mu \mid \mu$ is an RDF homomorphism from P into $G\}$;
- if $P = (P_1$ `AND` $P_2)$, $\mathcal{S}(P, G) = \mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G)$;
- if $P = (P_1$ `UNION` $P_2)$, $\mathcal{S}(P, G) = \mathcal{S}(P_1, G) \cup \mathcal{S}(P_2, G)$;
- if $P = (P_1$ `OPT` $P_2)$, $\mathcal{S}(P, G) = (\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G)) \cup (\mathcal{S}(P_1, G) \setminus \mathcal{S}(P_2, G))$;
- if $P = (P_1$ `FILTER` $R)$, $\mathcal{S}(P, G) = \{\mu \in \mathcal{S}(P_1, G) \mid \mu(R) = \top\}$.

Let $Q =$`SELECT` $\vec{B}$ `FROM` u `WHERE` P be a SPARQL query. Let G be the RDF graph appearing in the URL u, and Ω the set of answers of P in G. Then the answers of the query Q are the homomorphisms of elements of Ω to $\vec{B}$, *i.e.*, for each homomorphism π of Ω, the answer of Q associated to π is $\{(x, y) \mid x \in \vec{B}$ and $y = \pi(x)$ if $\pi(x)$ is defined, otherwise `null`$\}$.

Query answering, *i.e.*, answering basic graph pattern queries (or GRDF graphs), can be reduced to the SIMPLE RDF ENTAILMENT decision problem. For dealing with RDF/RDFS queries, it is sufficient to close the data graph using the RDF/RDFS rules [20, 23, 25]. Then calculate the possible RDF homomorphisms from the basic graph pattern of the query into the closure of the data graph.

Example **5** *Consider the following SPARQL query:*
SELECT `_:name _:mbox`
FROM $<$ `http : //example.org/index1.ttl` $>$
WHERE (P *OPT* $\{($ `_:b2, foaf:mbox, _:mbox`$)\}$*)*

where P is the RDF graph of Fig. 1, and the RDF graph by the URL of the FROM clause is the graph G of Fig. 3. We build the answer to the query starting from the joint of RDF homomorphisms from Q into G and RDF homomorphisms from the optional triple into G; i.e., of RDF homomorphisms from Q into G (there is one of them, it is the RDF homomorphism π_1 of Example 4, and of RDF homomorphisms from P into G which do not extend to the optional triple (there is one of them, it is the RDF homomorphisms π_2 of Example 4. There are thus two answers to the query:

`_:name`	`_:mbox`
"Deema"	null
"Natasha"	"natasha@yahoo.com"

3.2 Path queries

Path queries are usually expressed using regular expressions [16, 17, 12, 1, 18, 26]. However, we prefer to present in this section a more general framework for path queries to query an RDF knowledge base, since they can be also expressed by other means such as grammars or automates. 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 expressions

An alphabet is a set of tokens. A word over an alphabet Σ is an element of Σ^*, *i.e.*, a tuple of Σ^k, for some integer k. A language L over Σ is a possibly infinite subset of Σ^*, *i.e.*, a set of words.

Notations A word $(a_1, \ldots, a_k)$ will be noted $a_1 \cdot \ldots \cdot a_k$, and the empty word $()$ will be noted ϵ.

A *concatenation* of two words $w_1 = a_1 \cdot \ldots \cdot a_p$ and $w_2 = b_1 \cdot \ldots \cdot b_k$, denoted by $w_1 \cdot w_2$, is defined by $w_1 \cdot w_2 = a_1 \cdot \ldots \cdot a_p \cdot b_1 \cdot \ldots \cdot b_k$.

Definition 10 (Regular Expressions) *Let Σ be an alphabet. The set $\mathcal{RE}(\Sigma)$ of regular expressions is inductively defined by:*

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

Here $A|B$ denotes the disjunction of A and B, $A \cdot B$ the concatenation of A and B, A^* the Kleene closure, A^+ the positive closure, and $!A$ the negation.

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

Definition 11 *Let Σ be an alphabet, and $R \in \mathcal{RE}(\Sigma)$ be a regular expression. $L^*(R)$ is the set of words of Σ^* defined by:*

- *if $R = \epsilon$, $L^*(R) = \emptyset$;*
- *if $R = a \in \Sigma$, $L^*(R) = \{a\}$;*
- *if $R = \Sigma$, $L^*(R) = \Sigma$;*
- *if $R = R_1 \mid R_2$, then $L^*(R) = \{w \mid w \in L^*(R_1) \cup L^*(R_2)\}$;*
- *if $R = R_1 \cdot R_2$, then $L^*(R) = \{w_1 \cdot w_2 \mid w_1 \in L^*(R_1)$ and $w_2 \in L^*(R_2)\}$;*
- *if $R = (R_1^+)$, then $L^*(R) = \{w_1 \cdot \ldots \cdot w_k \mid w_i \in L^*(R_i), 1 \leq i \leq k, k \in \mathbb{N}\backslash\{0\}\}$;*
- *if $R = (R_1^*)$, then $L^*(R) = \{\epsilon\} \cup L^*(R_1^+)$;*
- *if $R =!R_1$, then $L^*(R) = \Sigma^* \setminus L^*(R_1)$.*

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 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.

A path in a directed multigraph Let $G = (V, E, \gamma)$ be a directed multigraph. Let x and y Be two nodes of V. A *path* from x to y is a non-empty list of arcs $(a_1, \ldots, a_k)$ from 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 12 (Word associated to a path) *Let $G = (V, E, \gamma, \lambda)$ be a labeled directed multigraph, whose arcs are labeled over an alphabet Σ. A* word associated *to a path $P = (a_1, \ldots, a_k)$ of G be the word noted $\lambda(P)$ over Σ^* defined by $\lambda(P) = \lambda(a_1) \cdot \ldots \cdot \lambda(a_k)$.*

Definition 13 *Let $G = (V, E, \gamma, \lambda)$ be a directed labeled multigraph where the arcs are labeled by elements of an alphabet Σ. A pair $\langle x, y\rangle$ of nodes of G* satisfies *a language L over Σ if one of the following conditions is satisfied:*

- *$\epsilon \in L$ and $x = y$; or*
- *there exists a word $w \in L$ and a path P from x to y in G such that $\lambda(P) = w$.*

Introduction of variables More general forms of regular expressions are the ones that include blanks. Regular expression with blanks have several names, here we call them *regular expression patterns*. Their combined power and simplicity contribute to their wide use in different fields. For example, in [18], in which they are called *universal regular expressions*, they are used for compiler optimizations. In [26], they are called *parametric regular expressions*, and are used for program analysis and model checking.

Definition 14 *Consider an alphabet Σ which are the union of two disjoint sets, a set of* constant *(in RDF, think of urirefs and literals) and a set of* variables *(in RDF, think of blanks). We call* substitution *over Σ an application $\sigma : \Sigma \rightarrow \Sigma$ which preserves the constants (if x is a constant of Σ, $\sigma(x) = x$). Let us note that such a substitution corresponds to a map in RDF. Let L be a language over Σ and $G = (V, E, \gamma, \lambda)$ be a directed multigraph, whose arcs are labeled over Σ. Then a pair (x, y) of nodes of V* S-satisfies *the language L if and only if one of the following conditions holds:*

- *$\epsilon \in L$ and $x = y$; or*
- *there is a word w of L, a path P from x to y, and a substitution σ over Σ such that $\lambda(P) = \sigma(w)$ (where $\sigma(a_1 \cdot \ldots \cdot a_k) = \sigma(a_1) \cdot \ldots \cdot \sigma(a_k)$). We also say that the path P S-satisfies L.*

3.2.3 Regular expressions as queries

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

Example **6** *Let us consider the RDF graph G of Fig. 5 as knowledge base, and the following regular path expression $R = (\texttt{ex:son}|\texttt{ex:daughter})^+ \cdot \texttt{\_:b5}$. Intuitively, this query use paths from an entity x to an entity y such that y is neighbor, by a predicate, to a descendant of x. The set of answers of R is:*
{(_:c1, _:c3, {(_:b5, ex:son)})
(_:c1, ex:Person1, {(_:b5, ex:friend)})
(_:c1, ex:Person2, {(_:b5, ex:friend)})
(_:c1, ex:Person3, {(_:b5, ex:friend)})}

3.2.4 Path queries and language generators

Generally, the set of words composing a language is not given in extension, but is rather defined intensionally by a grammar. Languages defined by type 1 grammars can be also defined by *regular expressions* over an alphabet Σ.

Whatever the means used to define a language, we call a *generator* over Σ any object that can be used to specify a language over Σ. If R is such a generator, we also note $L^*(R)$ the language specified by R (named language generated by R). This will allow us to define a general framework for the generalization of RDF (cf. Section 4) without restricting ourself to a specified language generators (e.g. regular expressions).

The following definition is very useful for defining the language denoted by a generator over Σ that contains blanks, *i.e.*, $B \subseteq \mathcal{B}$ and $B \subseteq \Sigma$.

Definition 15 *Let R be a generator over Σ, and σ be a map from Σ to Σ. If $m = a_1 \cdot \ldots \cdot a_k \in \Sigma^*$, we note $\sigma(m) = \sigma(a_1) \cdot \ldots \cdot \sigma(a_k)$, and $\sigma(R)$ is the generator such that $L^*(\sigma(R))$= $\{\sigma(m) \mid m \in L^*(R)\}$.*

When the generator R contains blanks (cf. regular expressions with blanks), then we search a map from the blanks appearing in R to the arc labels along the path, *i.e.*, an assignment to blanks. In such case, we will be interested in the following problem.
$\mathcal{R}$-PATH SATISFIABILITY
Instance: a directed labeled multigraph G, two nodes x, y of G, and a generator $R \in \mathcal{R}(\Sigma)$, where $\Sigma \supseteq \mathcal{V}(G)$.
Question: Is there a map μ from Σ to $term(G)$ such that the pair $\langle x, y\rangle$ satisfies $L^*(\mu(R))$?

3.3 Discussion

We have presented in this section the SPARQL query language, which is based on RDF entailment. We have also presented a general framework for path expressions. Though path expressions, e.g. regular expressions, 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 requested information. Furthermore, both approaches are orthogonal, *i.e.*, there are some queries that can be expressed by one approach and cannot be expressed by the other (cf. Example 1).

So we will extend SPARQL with path expressions, in particular, regular expression patterns. Moreover, we will extend the graph patterns of SPARQL queries with regular expression patterns, and call these patterns 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

This section presents PRDF, a general extension of RDF with path expressions. Though we use regular expressions in the demonstration examples and in the extension of SPARQL (Section 5), we prefer to present our definitions with abstract generators, since the soundness and completeness result (Theorem 2) does not depend upon the language used for path queries. The PRDF language extends GRDF naturally to allow using generated path expressions as labels for the arcs of GRDF graphs. We present in Section 4.1 its abstract syntax, and its semantics in Section 4.2. In Section 4.3 we present an inference mechanism for querying GRDF with PRDF graphs.

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 generators over $\Sigma = \mathcal{U} \cup \mathcal{B}$. In what follows, we consider an abstract set $\mathcal{R}(\mathcal{U} \cup \mathcal{B})$ of such generators. A particular case for this set is the set of regular expressions over $\mathcal{U} \cup \mathcal{B}$.

We note PRDF$[\mathcal{R}]$ for the extension to the GRDF language with $\mathcal{R}$, where $\mathcal{R}$ is the set of language generators allowed to be used in the predicate position of PRDF$[\mathcal{R}]$ triples.

Definition 16 (PRDF graph) *A* PRDF$[\mathcal{R}]$ triple *is an element of* $\mathcal{T} \times \mathcal{R}(\mathcal{U} \cup \mathcal{B}) \times \mathcal{T}$. *A PRDF*$[\mathcal{R}]$ *graph is a set of PRDF*$[\mathcal{R}]$ *triples.*

Note that all PRDF$[\mathcal{R}]$ graphs with atomic predicates are not necessarily RDF graphs. They can be a generalization of RDF graphs with blanks as predicates, as called *generalized RDF graphs* [25]. However, A PRDF$[\mathcal{R}]$ graph with atomic predicates is a GRDF graph.

A PRDF$[\mathcal{R}]$ graph can be represented graphically in the same way as done in GRDF (but in that case, the arcs can be labeled by elements of $\mathcal{R}(\mathcal{U} \cup \mathcal{B})$).

Notations Let R be a generator, $u \in \mathcal{U}(R)$ if $u \in U$ and U is the smallest set such that $R \in \mathcal{R}(U \cup \mathcal{B})$ (*i.e.*, $\mathcal{U}(R)$ is the set of urirefs appearing in R). In the same way, $b \in \mathcal{B}(R)$ if $b \in B$ and B is the smallest set such that $R \in \mathcal{R}(\mathcal{U} \cup B)$ (*i.e.*, $\mathcal{B}(R)$ is the set of blanks appearing in R). Let G be a PRDF$[\mathcal{R}]$ graph, $pred(G)$ is the set of generators appeared 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 $term(G) = subj(G) \cup \mathcal{UB}(\mathcal{R}) \cup obj(G)$.

Example **7** *Let us suppose that we wish to find among, the entities related to Faisal or the descendants of Faisal, those which know Faisal. Let us suppose that we want to know their names and their address email. This query can be expressed in PRDF*$[\mathcal{RE}]$ *by the graph* P *as shown in Fig. 2.*

4.2 PRDF Semantics: interpretations and models

Interpretations in the PRDF$[\mathcal{R}]$ language are defined in the same way as in Definition 2. However, an interpretation has specific conditions to be a model for a PRDF$[\mathcal{R}]$ graph. These conditions are the transposition of the classical path semantics within RDF semantics.

Definition 17 (Support of a generator) *Let* $I = \langle IR, IP, I_{EXT}, I_S, I_L \rangle$ *be an interpretation of a vocabulary* V, I' *be an extension of* I *to* $B \subseteq \mathcal{B}$, *and* $R \in \mathcal{R}(\mathcal{V} \cup B)$. *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 $length \geq 1$ $w = w_1 \cdot \ldots \cdot w_n$ *where* $w \in L^*(R)$ *and* $w_i \in \mathcal{V} \cup B$ $(1 \leq i \leq n)$, *and a sequence of resources of* IR $x = r_0, \ldots, r_n = y$ *such that* $\langle r_{i-1}, r_i \rangle \in I_{EXT}(I'(w_i))$, $1 \leq i \leq n$.

This definition is similar to the satisfiability of a generator by two nodes (Definition 13). In such case, we verify if the pair of nodes $\langle x, y \rangle$ satisfies a generator R in a GRDF graph G by searching a path $x = x_0 \xrightarrow{p_1} x_1 \xrightarrow{p_2} \ldots \xrightarrow{p_n} x_n = y$ from x to y, and a substitution σ such that

$w = p_1 \cdot \ldots \cdot p_n \in L^*(\sigma(R))$. The same case is applied for the definition of support, we verify if the pair of resources $\langle x, y\rangle$ support a generator R in an extension of an interpretation I' by searching a path $x = r_0 \xrightarrow{p_1} r_1 \xrightarrow{p_2} \ldots \xrightarrow{p_n} r_n = y$ from x to y such that $w = w_1 \cdot \ldots \cdot w_n \in L^*(R)$, $I'(w_i) = p_i \in IP$, and $\langle r_{i-1}, r_i\rangle \in I_{EXT}(p_i)$. In particular, if $R = u \in U$, $L^*(R) = \{u\}$ and the condition (ii) becomes $\langle x, y\rangle \in I_{EXT}(I'(u))$, which corresponds to the usual RDF semantic condition.

We can see in Fig. 4 a path of resources starting with the resource associated to x and ending with the resource associated to y such that each pair of resources $\langle r_{i-1}, r_i\rangle$ belongs to the extension of a property $I'(w_i)$ and these properties form a word $w_1 \cdot \cdots \cdot w_n$ that belongs to the language generated by R.

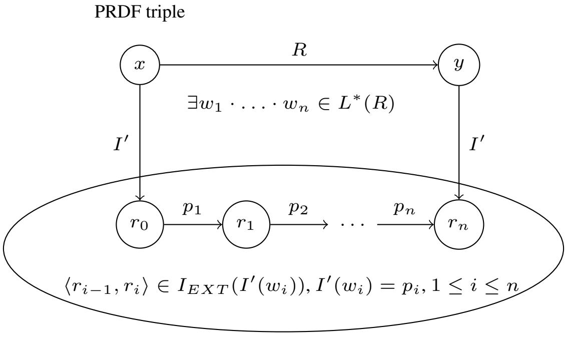
PRDF triple
x
R
y
$\exists w_1 \cdot \ldots \cdot w_n \in L^*(R)$
I'
I'
r_0
p_1
r_1
p_2
$\ldots$
p_n
r_n
$\langle r_{i-1}, r_i\rangle \in I_{EXT}(I'(w_i)), I'(w_i) = p_i, 1 \leq i \leq n$

Figure 4: Support of a generator.

This way, an interpretation is a model for a triple $\langle s, R, o\rangle$, if the two resources corresponding to s and o, denoted by $\langle I'(s), I'(o)\rangle$ supports $I'(R)$.

Definition 18 (Model) *Let G be a PRDF$[\mathcal{R}]$ 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$[\mathcal{R}]$ 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'.*

GRDF graphs are PRDF$[\mathcal{R}]$ graphs since the generators used to label the arcs of PRDF$[\mathcal{R}]$ graphs can be reduced to atomic generators, *i.e.*, urirefs and blanks.

Property 1 *If G is a PRDF$[\mathcal{R}]$ 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 4) iff I is a PRDF$[\mathcal{R}]$ model of G (Definition 18).*

Proof. We prove both directions of the property.

($\Rightarrow$) Suppose that I is a 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 4). Since $pred(G) \subseteq \mathcal{U} \cup \mathcal{B}$, $\langle I'(s), I'(o)\rangle$ supports p in I' (Definition 17)(with a word $w = p$), *i.e.*, I is also a PRDF$[\mathcal{R}]$ model (Definition 18).

($\Leftarrow$) Suppose that I is a PRDF[$\mathcal{R}$] 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 18). Since $pred(G) \subseteq \mathcal{U} \cup \mathcal{B}$, $\epsilon \notin L^*(p)$. So there exists a word of $length = 1$ where $w \in L^*(p)$, *i.e.*, $w = p$, 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 17). 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 4).

4.3 PRDF homomorphism

We want to find an inference mechanism that takes a PRDF[$\mathcal{R}$] graph as a query and a GRDF graph as a data graph. When using RDF homomorphism from a GRDF graph H into a GRDF graph G, which depends deeply in the neighborhood of nodes of G, two nodes of H that are linked by a predicate p must be mapped into nodes also linked by p in G (if p is not a blank, otherwise it can be mapped to any predicate). However, this classical definition of the homomorphism defined for GRDF graphs does not work with PRDF[$\mathcal{R}$] graphs.

An inference mechanism that maps each two neighbor nodes linked by a generator R into nodes linked by a path whose concatenation $e_1 \cdot \ldots \cdot e_k$ of labels belongs to the language generated by R, *i.e.*, $e_1 \cdot \ldots \cdot e_k \in L^*(R)$ is defined below.

Definition 19 (PRDF homomorphism) *Let G be a GRDF graph, and H be a PRDF[$\mathcal{R}$] graph. A PRDF[$\mathcal{R}$] 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 19 is equivalent to $\forall\langle s, R, o\rangle \in H$, $\langle\pi(s), \pi(o)\rangle$ satisfies $L^*(\pi(R))$ in G (Definition 13). This means that we can reformulate the definition using Definition 13. If R is a regular expression, then $\pi(R)$ is the regular expression obtained by substituting $\pi(x)$ to each atom x in R (cf. Definition 15). Also (thanks to Definition 7), $\pi(x) = x$ where $x \in \mathcal{U}$: no mapping is needed in that case.

Example **8** *Fig. 5 represents a PRDF[$\mathcal{RE}$] homomorphism from the PRDF[$\mathcal{RE}$] graph P into the RDF graph G. Let us note that the path satisfying the regular expression of P is one of those given in Example 6.*

What remains is to prove that the PRDF[$\mathcal{R}$] homomorphism is sound and complete w.r.t. our extension of RDF semantics. We have proven Theorem 2 via a transformation to hypergraphs following the proof framework in [7]. Since this requires a long introduction to hypergraphs, we prefer here to give a simple direct proof to Theorem 2.

Theorem 2 *Let G be a GRDF graph, and H be a PRDF[$\mathcal{R}$] graph. Then there is a PRDF[$\mathcal{R}$] homomorphism from H into G iff $G \models_{PRDF[\mathcal{R}]} H$.*

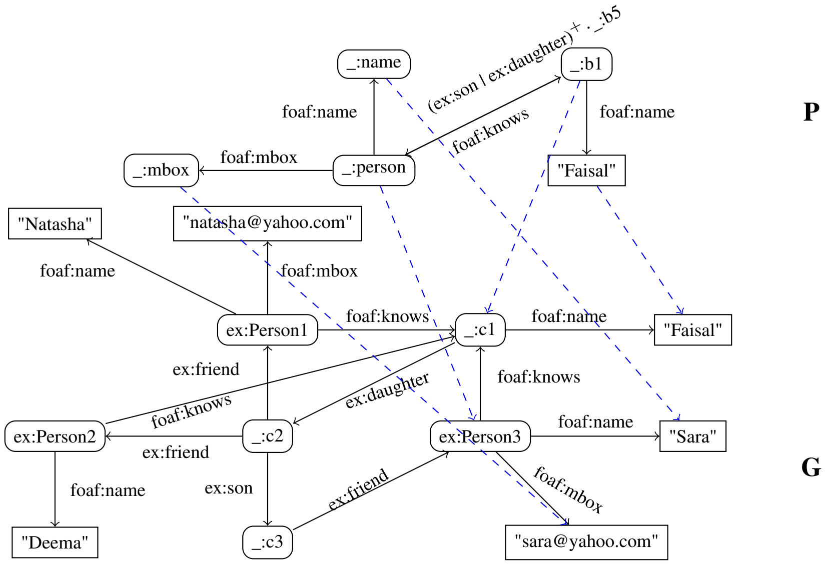

Figure 5: A PRDF[$\mathcal{R}$] homomorphism from a PRDF[$\mathcal{R}$] graph to a GRDF graph.

Proof. We prove both directions of the theorem.

($\Rightarrow$) Suppose that there exists a PRDF[$\mathcal{R}$] homomorphism from H into G, $\pi : term(H) \rightarrow term(G)$. We want to prove that $G \models_{PRDF[\mathcal{R}]} H$, *i.e.*, every model of G is a model of H. Let us consider the interpretation I of a vocabulary $\mathcal{V}$.

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 4). 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 us define the map $I'' = (I' \circ \pi)$, and show that I'' verifies the following properties:

1. I is an interpretation of $\mathcal{V}(H)$.
2. I'' is an extension to blanks of H, *i.e.*, $\forall x \in \mathcal{V}(H)$, $I''(x) = I(x)$ (Definition 3).
3. I'' satisfies the conditions of PRDF[$\mathcal{R}$] models (Definition 18), *i.e.*, for every triple $\langle s, R, o\rangle$ $\in H$, the two resources $\langle I''(s), I''(o)\rangle$ supports R in I''.

1. Since each term $x \in \mathcal{V}(H)$ is mapped by π to a term $x \in \mathcal{V}(G)$ and I interprets all $x \in \mathcal{V}(G)$, I interprets all $x \in \mathcal{V}(H)$.
2. $\forall x \in \mathcal{V}(H)$, $I''(x) = (I' \circ \pi)(x)$ (definition of I''). $I''(x) = I'(x)$ (since $\pi(x) = x$ by Definition 19). Hence, $I''(x) = I(x)$ (Definition 3).

3. It remains to prove that for every triple $\langle s, R, o\rangle \in H$, the two resources $\langle I''(\pi(s)), I''(\pi(o))\rangle$ supports R in I'' (by Definition 17):

   (i) If the empty word $\epsilon \in L^*(R)$ and $\pi(s) = \pi(o) = y$ ($y \in term(G)$, Definition 19), 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 18).

   (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 19). It follows that $\langle I'(\pi(s)), I'(n_1)\rangle \in I_{EXT}(I'(p_1))$, $\ldots$, $\langle I'(n_{k-1}), I'(\pi(o))\rangle \in I_{EXT}(I'\ (p_k))$ (Definition 4). So the two resources given by $\langle I'(\pi(s))$, $I'(\pi(o))\rangle$ 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 blank 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 $w \in \pi(R), w \in R$).

($\Leftarrow$) Suppose that $G \models_{PRDF[\mathcal{R}]} H$. We want prove that there is a PRDF[$\mathcal{R}$] 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. Lemma 1). 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 4). 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 18). Let us consider the function $\pi = (\iota^{-1} \circ I')$. To prove that π is a PRDF[$\mathcal{R}$] homomorphism from H into G, we must prove that:

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 3 and Lemma 1). $\forall x \in \mathcal{V}(H)$, $(\iota^{-1} \circ I')(x) =$ $(\iota^{-1} \circ \iota)(x) = x$.

(3i) If $\epsilon \in L^*(R)$ and $I'(s) = I'(o) = r \in IR$ (Definition 17), 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 $length \geq 1$ $w = a_1\cdot\ldots\cdot a_n$ where $w \in L^*(R)$ and $a_i \in \mathcal{V} \cup \mathcal{B}(G)$ ($1 \leq i \leq k$), 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 17). 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 $ISO(G)$, Lemma 1). So $(\iota^{-1} \circ I')(s)$ $= \iota^{-1}(r_0) = n_0$, $(\iota^{-1} \circ I')(o) = \iota^{-1}(r_k) = n_k$, and $p_1 \cdot \ldots \cdot p_k \in L^*((\iota^{-1} \circ I')(R))$.

This result shows that, as for RDF, there is an equivalence between PRDF[$\mathcal{R}$] homomorphisms and entailment of a PRDF[$\mathcal{R}$] graph by a GRDF graph. So, testing the entailment of between PRDF[$\mathcal{R}$] graphs, can be reduced to the PRDF[$\mathcal{R}$] HOMOMORPHISM problem:
PRDF[$\mathcal{R}$] HOMOMORPHISM
Instance: a PRDF[$\mathcal{R}$] graph H and a GRDF graph G.
Question: Does G entails H, *i.e.*, is there a PRDF[$\mathcal{R}$] homomorphism from H into G?

This definition is parameterized by the language generator $\mathcal{R}$ and subject to its satisfaction checking. The problem is at least NP-hard, since it contains RDF HOMOMORPHISM which is equivalent to SIMPLE RDF ENTAILMENT, an NP-complete problem. Moreover, any solution can be checked by checking as many times as there is edges in the query an instance of the $\mathcal{R}$-PATH SATISFIABILITY problem. Hence, if $\mathcal{R}$-PATH SATISFIABILITY is in NP then PRDF[$\mathcal{R}$] HOMOMORPHISM is NP-complete.

Section 5 presents an instantiation of this framework used to define an extension of the SPARQL query language with regular expression paths. In Section 6 we will present algorithms enumerating the answers of such queries, *i.e.*, for enumerating all PRDF[$\mathcal{R}$] homomorphisms from a given PRDF[$\mathcal{RE}$] graph H into a GRDF graph G, where $\mathcal{RE}$ is the set of regular expressions defined in Section 3.2.

5 The PSPARQL query language

We have defined, in the previous section, the syntax and the semantics of PRDF[$\mathcal{R}$], where $\mathcal{R}$ is the set of generators that can be used in the predicate position of PRDF graphs. The PSPARQL query language is built ontop of PRDF[$\mathcal{RE}$], *i.e.*, PRDF with regular expression patterns 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 based on the definition of [31], as well as an evaluation algorithm. Finally, Section 5.3 presents the complexity study of evaluating PSPARQL graph patterns.

5.1 PSPARQL syntax

SPARQL considers GRDF graphs as basic graph patterns for querying RDF graphs (cf. Section 3.1). In PSPARQL, we will use PRDF[$\mathcal{RE}$] graphs as basic graph patterns, where $\mathcal{RE}$ denotes a set of regular expression patterns, *i.e.*, regular expressions with blanks constructed over the set of urirefs and the set of blanks, *i.e.*, $\mathcal{RE}(\mathcal{U} \cup \mathcal{B})$.

Definition 20 (PSPARQL graph patterns) *A* PSPARQL graph pattern *is defined inductively in the following way:*

- *every PRDF[$\mathcal{RE}$] graph is a PSPARQL graph pattern;*

- *if P_1, P_2 are PSPARQL graph patterns and R is a SPARQL constraint, then (P_1* `AND` *P_2), (P_1* `UNION` *P_2), (P_1* `OPT` *P_2), and (P_1* `FILTER` *R) are PSPARQL graph patterns.*

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.

The use of variables (*i.e.*, blanks) in the regular expressions is intended to be a generalization of the use of variables (*i.e.*, blanks) as predicates in the basic graph patterns of SPARQL.

Our use of variables (*i.e.*, blanks) in regular expressions is different from the use of variables in Unix ("regular expressions with back referencing" in [2]). Here a blank appearing in a regular expression matches any symbol of the alphabet, while a blank in regular expressions with back referencing can match strings. Matching strings with regular expression with back referencing is shown to be NP-complete [2].

If the regular expression includes a blank with closure operator (e.g. $?X^+$), then the blank matches the repeated occurrences of the same symbol. The language denoted by a regular expression pattern R is defined using a map from the blanks of R to some set of alphabet (cf. Definition 15).

As PSPARQL introduces PRDF[$\mathcal{RE}$] graph, we give in Table 1 the necessary modifications to SPARQL grammar [33] in the EBNF specification. Where the production rule [22'] replaces [22] in SPARQL, and all other rules are added to SPARQL grammar to have a complete grammar for PSPARQL.

[22']	$\langle BlockOfTriples\rangle$	::=	$\langle PathTriples1\rangle$
		|	('.' $\langle PathTriples1\rangle$?)*
[30.1]	$\langle PathTriples1\rangle$	::=	$\langle VarOrTerm\rangle$ $\langle PathPropLNE\rangle$
		|	$\langle PathTripleNode\rangle$ă$\langle PathPropL\rangle$
[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]	$\langle PathTripleNode\rangle$	::=	$\langle PathCollection\rangle$
		|	$\langle PathBNodePropL\rangle$
[36.1]	$\langle PathBNodePropL\rangle$	::=	'[' $\langle PathPropLNE\rangle$ ']'
[37.1]	$\langle PathCollection\rangle$	::=	'(' $\langle PathGraphNode\rangle$+ ')'
[38.1]	$\langle PathGraphNode\rangle$	::=	$\langle VarOrTerm\rangle$
		|	$\langle PathTripleNode\rangle$
[39.1]	$\langle RegularExp\rangle$	::=	$\langle Rexp\rangle$ (('|' | '·') $\langle Rexp\rangle$)*
[39.2]	$\langle Rexp\rangle$	::=	('+' | '*')? $\langle Atom\rangle$
[39.3]	$\langle Atom\rangle$	::=	'!' $\langle IRIref\rangle$
		|	$\langle VarOrIRIref\rangle$
		|	'(' $\langle RegularExp\rangle$ ')'

Table 1: PSPARQL graph pattern grammar.

5.2 Evaluating PSPARQL queries

As in the case of RDF/GRDF, the answer to a query reduced to a PRDF graph is also given by a map. The definition of a answer to a PSPARQL query will be thus identical to that given for SPARQL (but will use PRDF homomorphisms).

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

- if P is a PRDF[$\mathcal{RE}$] graph, $\mathcal{S}(P, G) = \{\mu \mid \mu$ is a PRDF homomorphism from P into $G\}$;
- if $P = (P_1 \texttt{ AND } P_2)$, $\mathcal{S}(P, G) = \mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G)$;
- if $P = (P_1 \texttt{ UNION } P_2)$, $\mathcal{S}(P, G) = \mathcal{S}(P_1, G) \cup \mathcal{S}(P_2, G)$;
- if $P = (P_1 \texttt{ OPT } P_2)$, $\mathcal{S}(P, G) = (\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G)) \cup (\mathcal{S}(P_1, G) \setminus \mathcal{S}(P_2, G))$.
- if $P = (P_1 \texttt{ FILTER } R)$, $\mathcal{S}(P, G) = \{\mu \in \mathcal{S}(P_1, G) \mid \mu(R) = \top\}$.

Let $Q =$SELECT $\vec{B}$ FROM u WHERE P be a PSPARQL query. Let G be the RDF graph appearing in the URL u, and Ω the set of answers of P in G. Then the answers of the query Q are the homomorphisms of elements of Ω to $\vec{B}$, *i.e.*, for each PRDF$_c$ homomorphism π of Ω, the answer of Q associated to π is $\{(x, y) \mid x \in \vec{B}$ and $y = \pi(x)$ if $\pi(x)$ is defined, otherwise `null`$\}$.

5.3 Complexity of evaluating PSPARQL graph patterns

PSPARQL queries can be evaluated in the same way as SPARQL queries (cf. Section 3.1). However, in PSPARQL we calculate the set of PRDF[$\mathcal{RE}$] homomorphisms instead of computing RDF homomorphisms. As mentioned in Section 4.3, the complexity of the PRDF[$\mathcal{RE}$] HOMOMORPHISM problem depends on the complexity of $\mathcal{RE}$-PATH SATISFIABILITY. So, we study its complexity.

Lemma 2 $\mathcal{RE}$-PATH SATISFIABILITY*, in which* $\Sigma \subseteq \mathcal{U}$ *(*$R \in \mathcal{RE}$ *is a regular expression that does not contain blanks) is in* NLOGSPACE *in* G *and* R*.*

Proof. We can view G as a non-deterministic finite automaton, NDFA, with initial state x and final state y (G can be transformed to an equivalent NDFA in NLOGSPACE). Constructing M, a NDFA accepting $L^*(R)$ (the language generated by R) can be done in NLOGSPACE. Constructing the product automaton $\mathcal{A}$, that is, the intersection of G and M, can be done in NLOGSPACE. Now, checking if there is a directed path from x to y is equivalent to checking whether $L^*(\mathcal{A})$ is not empty, and the latter can be done in NLOGSPACE in $\mathcal{A}$ [4, 27] (with the fact that the class of LOGSPACE transformations is closed under compositions [9]).

Lemma 3 $\mathcal{RE}$-PATH SATISFIABILITY*, in which* $\Sigma \subseteq (\mathcal{U} \cup \mathcal{B})$*, is in* NP*.*

Proof. $\mathcal{RE}$-PATH SATISFIABILITY is in NP, since each blank in the regular expression R can be mapped (assigned) to p terms, where p denotes the number of terms appearing as predicates in G. If the number of blanks in R is x, then there are (p^x) possible assignments (mappings) in all.

Once an assignment of terms to blanks is fixed, the problem is reduced to $\mathcal{RE}$-PATH SATISFIABILITY ($\Sigma \subseteq \mathcal{U}$), which is in NLOGSPACE.

It follows that a non-deterministic algorithm needs to guess a map μ and a path $p = (u_1, \ldots, u_k)$ from the node x to the node y, and check in polynomial time if $label(u_1) \cdot \ldots \cdot label(u_k) \in L^*(\mu(R))$, that is, the pair $\langle x, y \rangle$ satisfies $L^*(\mu(R))$.

From the above results, we can conclude that adding blanks to regular expressions may affect the complexity of query evaluation. However, in the case of PSPARQL (or SPARQL), the size of the graph patterns of a given query is considerably smaller than the size of the data graph. In particular, the number of blanks in each regular expression of the graph pattern is very small and can be assumed to be fixed. With this assumption, the data-complexity [35], which is defined as the complexity of query evaluation for a fixed query, is considered.

Corollary 1 PRDF$[\mathcal{RE}]$ HOMOMORPHISM *is* NP-*complete.*

If the entailed graph, *i.e.*, the query, is *ground* then PRDF$[\mathcal{RE}]$ HOMOMORPHISM problem is in NLOGSPACE. In consequence, PRDF$[\mathcal{RE}]$ entailment can be decided in polynomial time.

Theorem 3 *Let G be a GRDF graph and H be a ground PRDF$[\mathcal{RE}]$ graph. Deciding whether $G \models_{PRDF} H$ can be done in* NLOGSPACE.

Proof. 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 $\langle \pi(s), \pi(o) \rangle$ satisfies $\pi(R) = R$. Since each of these operations corresponds to the case of $\mathcal{RE}$-PATH SATISFIABILITY, in which $\Sigma \subseteq \mathcal{U}$, the complexity of each of them is NLOGSPACE (see Lemma 2) (Since H is ground, R does not contain blanks). Hence, the total time is also NLOGSPACE.

6 Answering PSPARQL queries: algorithms for PRDF homomorphism

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

1. Is there a PRDF$[\mathcal{RE}]$ homomorphism from H into G?
2. Exhibit, if it exists, a PRDF$[\mathcal{RE}]$ homomorphism from H into G (PRDF$[\mathcal{RE}]$ HOMOMORPHISM).
3. Enumerate all PRDF$[\mathcal{RE}]$ homomorphisms from H into G (PRDF$[\mathcal{RE}]$ HOMOMORPHISMS).

In Section 6.1, we first present an algorithm, called $reach$ [26], that calculates the set of all pairs of nodes in an RDF graph satisfying a regular expression pattern. In Section 6.2, we present an algorithm that uses the $reach$ algorithm for calculating all PRDF$[\mathcal{RE}]$ homomorphisms from a PRDF$[\mathcal{RE}]$ graph H into an RDF graph G.

6.1 Matching regular expression patterns

We are interested here in an algorithm which, given an RDF graph and a regular expression pattern R, calculates the set of triples $\langle s, o, \mu\rangle$ such that the pair of nodes $\langle s, o\rangle$ of G satisfying $\mu(R)$ in G under each possible map μ. We call this set the satisfiability set, noted $\textsc{sat}(R, G)$.

Algorithm 1: $reach(G, R, v_0, s_0)$

```
Data: An RDF graph $G$, a regular expression, a start node $v_0$ in $G$, and a partial map $\mu_p$.
Result: The set of triples $\langle v_0, o, \mu\rangle$ s.t. the pair $\langle v_0, o\rangle$ satisfies $E$ with a map $\mu'$ in $G$,
        $\mu(y) = o$, $\mu'$, $\mu_p$ are compatible, and $\mu' \leftarrow (\mu \oplus \mu_p)$.
begin
    Let $A_\phi = \langle S, s_0, \delta, F, C\rangle$ be the NDFA$_\phi$ of $E$;
    $R \leftarrow \{\}$;
    $W \leftarrow \{\}$;
    $S(G) \leftarrow \{\}$;
    for $\langle s_0, tl, s\rangle \in A_\phi$ do
        for $\langle v_0, el, v\rangle \in G$ do
            if $match(tl, el, \mu_p)$ then
                $\mu \leftarrow \{\langle tl, el\rangle\}$;
                $\mu' = (\mu \oplus \mu_p)$;
                $W \leftarrow W \cup \{\langle v, s, \mu'\rangle\}$;
    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 $\langle s, tl, s_1\rangle \in A_\phi$ do
            for $\langle v, el, v_1\rangle \in G$ do
                if $match(tl, el, \mu)$ then
                    $\mu_1 \leftarrow \{\langle tl, el\rangle\}$; $\mu_2 \leftarrow (\mu \oplus \mu_1)$;
                    if ($\langle v_1, s_1, \mu_2\rangle \notin R$) then
                        $W \leftarrow W \cup \{\langle v_1, s_1, \mu_2\rangle\}$;
        if $s \in F$ then
            $S(G) \leftarrow S(G) \cup \{\langle v_0, v, \mu\rangle\}$;
    return $S(G)$;
end
```

Consider the *existential query problem* [18, 26]: given a directed labeled multigraph G, a node v_0, a regular expression pattern R, the initial state s_0 of the NDFA (non-deterministic finite automaton) accepting $L^*(R)$, and a set of final states F of NDFA, compute all triples $\langle v_0, v, \mu\rangle$ such that there is some path from v_0 to node v that matches some path from s_0 to some state in F under map μ. Where a path $P_1 = (u_1, \ldots, u_k)$ matches a path $P_2 = (v_1, \ldots, v_k)$ under a map μ if $label(u_i)$ matches $label(v_i)$ under map μ, $\forall 1 \leq i \leq k$.

We reuse the definition of matching two regular expression found in [26].

Matching. Let R_1 be a ground regular expression, *i.e.*, R_1 does not contain blanks, and R_2 be a regular expression pattern. Then we say that R_2 *matches* R_1 under the mapping μ, noted $match(R_2, R_1, \mu)$, if one of the following conditions hold: (1) $R_1 = \mu(R_2)$; (2) $R_2 = \#$; (3) $R_2 = !R_3$, and recursively, R_1 does not match R_3; (4) $R_1(e_1, \ldots, e_k)$, $R_2(a_1, \ldots, a_k)$, 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 $(?Z \cdot ?Y)$ matches the ground regular expression (ex:train·ex:plane) with the mapping $\{\langle ?Z, ex : train\rangle, \langle ?Y, ex : plane\rangle\}$.

Let $reach(G, R, v_0, s_0)$, called the *reach* set, be the set of triples $\langle v, s, \mu\rangle$ such that some path from v_0 to node v matches some path from s_0 to some state in F under mapping μ. We present Algorithm 1 [26] that computes matching information for reachable nodes.

$\{\langle v_0, v, \mu\rangle \mid \exists s \in F : \langle v, s, \mu\rangle \in reach(G, R, v_0, s_0)\}$

Property 2 *Let G be a directed labeled multigraph, R be a regular expression pattern, A be a NDFA accepting $L^*(R)$ with the set of final state F, and v_0, v be two nodes of G. Then $\langle v_0, v\rangle$ satisfies $L^*(\mu(R))$ in G under a map μ iff $\exists s \in F$ such that $\langle v, s, \mu\rangle \in reach(G, R, v_0, s_0)\}$.*

In Algorithm 1, we use a non deterministic finite automate, denoted by NDFA. To construct a NDFA that generates an equivalent language to a given regular expression, we use the same way described in [3].

6.2 Calculating PRDF homomorphisms

The $reach(G, R, s, \mu)$ algorithm is used by the algorithm $Evaluate$ (Algorithm 2), which, given an RDF graph G and a PRDF$_c$ triple (x, R, y), 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 satisfy (x, R, y) in G).

The result of the algorithm $Evaluate$ are called to calculate the PRDF$_c$ homomorphisms of a PRDF$_c$ graph P into an RDF graph G by successive joints in the algorithm $Eval$ (Algorithm 3), whose initial call will be $Eval(P, G, \{\mu_\emptyset\})$, where $\mu_\emptyset$ is the map with the empty domain.

The algorithms that we currently use to calculate the answers to a PSPARQL query (or SPARQL) are not optimized yet.

- Semantic consequence RDF is calculated by an homomorphism of graphs, using a basic version of backtrack. We intend to use optimizations resulting from the constraints networks, following the work conducted in [8, 7].

  Semantic consequence RDF/PRDF is not yet really related to the preceding backtrack (for example, it is a breadth exploration and not a depth of the backtrack tree). It makes call for the search for paths using algorithms which are not will adapted to our problem. A total rewriting of these algorithms is currently studied, for better integration of the search for paths in the backtrack.

- Regarding the negation in the regular expressions, we have implemented only atomic negation.

Algorithm 2: $Evaluate(t, G)$.

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

Algorithm 3: $Eval(P, G, \Omega)$.

```
Data: An RDF graph $G$, a set of maps, a PRDF$_c$ graph $P$.
Result: The set $\{\mu \mid \mu$ is a PRDF$_c$ homomorphism from $P$ into $G\}$.
begin
    if $P = \{t\}$ then
        return $\Omega \bowtie Evaluate(t, G)$;
    else
        if $P = (t \cup P')$ then
            return $Eval(\{t\}, G, Eval(P', G, \Omega))$;
end
```

7 Implementation and experiments

We have implemented in Java a PSPARQL query evaluator. It is provided with two parsers: one for parsing PSPARQL queries based upon the syntax of PSPARQL[4], and the second one for parsing RDF graphs (documents) written in the Turtle language [10].

The algorithm follows a backtrack technique optimizing the algorithm presented before and its evaluation of regular expression patterns generalizes those of [26].

This evaluator passed successfully all tests suggested by the W3C Data Access Working Group for the specification of the SPARQL query language[5].

In addition, the evaluator can parse PRDF[$\mathcal{RE}$] graphs and evaluate PSPARQL queries. It is currently being thoroughly tested for performances and practical hard problem detection.

8 Related work: query languages for graphs

This work generalizes, and adapts to RDF, the work that had been carried out in the databases, structured or semi-structured:

- G and its extension G+ for querying databases and limited to finding cycle-free paths (*i.e.*, simple paths) [16, 17];
- Graphlog, a visual query language which is proven to be equivalent to linear Datalog in [14], extends G+ by combining it with the Datalog notation.
- Lorel [1] and UnQL [12] for querying semi-structured data, and use regular expressions with variables to find cycle-free paths;

The originality of our approach lies in the generalization of the languages mentioned above, and its adaptation to RDF language the basic language of the Semantic Web.

We can find several algorithms for solving path queries starting from solving simpler path queries: path queries without variables [17, 14], path queries involving uncorrelated paths [36], parametric regular path queries [26], and universal regular path queries [18]. The author of [36, 34] discusses variants of path problems, algorithms for path queries, and their time complexities.

9 Conclusion and futur work

Two approaches can be used for querying RDF graphs. The one used by SPARQL [33] uses the semantic consequence. The second approach is based on the structure of the graph [27, 17]. Both approaches are orthogonal, *i.e.*, there are some queries that can be expressed in one approach and cannot be expressed in the other.

[4] <http://psparql.inrialpes.fr/>

[5] <http://www.w3.org/2001/sw/DataAccess/tests/>

The goal of this report is to add regular expression patterns to SPARQL queries. In order to achieve this goal, we have provided an extension of generalized RDF graphs, called PRDF, in which predicates are replaced by terms in a language identified by its generator. We provided the syntax and semantics of PRDF and proved the validity of the homomorphism approach in this context.

We have defined PSPARQL as an extension of SPARQL which uses PRDF graphs as graph patterns. We have provided a sound and complete inference mechanism for querying RDF graphs with PSPARQL queries. Answering PSPARQL basic graph pattern query, *i.e.*, evaluating PRDF graphs over RDF graphs, is shown to be NP-complete. We provided two algorithms for PSPARQL query evaluation. A PSPARQL query engine has been implemented and experimented.

As it extends SPARQL with path expressions, PSPARQL provides more expressive power than SPARQL as it can capture information between nodes connected with complex paths and relate them. Our use of blanks in regular expressions was indeed intended to be a generalization of the use of blanks as predicates in SPARQL. Extension of RDF to RDFS (RDF Schema), which allows, for example, to order the classes by inclusion, does not change the computational properties of the PSPARQL query language: the semantic consequence in RDFS is reduced polynomially to the semantic consequence in RDF [23].

We plan to apply some optimization techniques to PSPARQL as Forward checking and Backjump, and compare the PSPARQL with other path languages (*e.g.*, G, G+ [17], and GraphLog [14] used in database community).

References

[1] ABITEBOUL, S., QUASS, D., MCHUGH, J., WIDOM, J., AND L.WIENER, J. The Lorel query language for semistructured data. *Journal on Digital Libraries 1*, 1 (1997), 68–88.

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

[3] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading (MA US), 1974.

[4] ALECHINA, N., DEMRI, S., AND DE RIJKE, M. A modal perspective on path constraints. *Journal of Logic and Computation 13* (2003), 1–18.

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

[6] BAGET, J.-F. Homomorphismes d'hypergraphes pour la subsomption en RDF/RDFS. In *10e conférence sur langages et modèles à objets* (Mars 2004), vol. 10, pp. 203–216.

[7] BAGET, J.-F. RDF entailment as graph homomorphism. In *Proceedings of the 4th ISWC* (2005).

[8] BAGET, J.-F., DE MOOR, A., LEX, W., AND GANTER, B. Simple conceptual graphs revisited: Hypergraphs and conjunctive types for efficient projection algorithms. In *Proceedings of the 11th International Conference on Conceptual Structures (ICCS'03)* (2003), pp. 229–242.

[9] BALCAZAR, J. L., DIAZ, J., AND GABARRO, J. *Structural complexity 1*. Springer-Verlag, Inc., New York, NY, USA, 1988.

[10] BECKETT, D. Turtle - terse RDF triple language. Tech. rep., Hewlett-Packard, Bristol (UK), 2006.

[11] BRICKLEY, D., AND GUHA, R. V. RDF vocabulary description language 1.0: RDF schema, 23 January 2003.

[12] BUNEMAN, P., DAVIDSON, S., HILLEBRAND, G., AND SUCIU, D. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (1996), pp. 505–516.

[13] CARROLL, J. J., AND KLYNE, G. RDF concepts and abstract syntax. Recommendation, W3C, February 2004.

[14] CONSENS, M. P., AND MENDELZON, A. O. Graphlog: a visual formalism for real life recursion. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (1990), pp. 404–416.

[15] CORBY, O., DIENG, R., AND HÉBERT, C. A conceptual graph model for W3C resource description framework. In *Proceedings of the International Conference on Conceptual Structures* (2000), pp. 468–482.

[16] CRUZ, I. F., MENDELZON, A. O., AND WOOD, P. T. A graphical query language supporting recursion. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1987), ACM Press, pp. 323–330.

[17] CRUZ, I. F., MENDELZON, A. O., AND WOOD, P. T. G^+: Recursive queries without recursion. In *Proceedings of Second International Conference on Expert Database Systems* (1988), pp. 355–368.

[18] DE MOOR, O., AND DAVID, E. Universal regular path queries. *Higher-Order and Symbolic Computation 16*, 1-2 (2003), 15–35.

[19] FREUDER, E. C. A sufficient condition for backtrack-free search. *Journal of the ACM 29*, 1 (1982), 24–32.

[20] GUTIERREZ, C., HURTADO, C., AND MENDELZON, A. O. Foundations of semantic web databases. In *ACM Symposium on Principles of Database Systems (PODS)* (2004), pp. 95–106.

[21] HAASE, P., BROEKSTRA, J., EBERHART, A., AND VOLZ, R. A comparison of RDF query languages. In *Proceedings of the 3rd International Semantic Web Conference (ICWC)* (Hiroshima (JP), 2004), pp. 502–517.

[22] HAYES, P. RDF model theory. working draft, W3C, April 2002.

[23] HAYES, P. RDF semantics. Recommendation, W3C, February 2004.

[24] HORST, H. J. Extending the $RDFS$ entailment lemma. In *Proceedings of the Third International Semantic web Conference (ISWC)* (2004), pp. 77–91.

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

[26] LIU, Y. A., ROTHAMEL, T., YU, F., STOLLER, S., AND HU, N. Parametric regular path queries. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation* (2004), pp. 219–230.

[27] MENDELZON, A. O., AND WOOD, P. T. Finding regular simple paths in graph databases. *SIAM Journal on Computing 24*, 6 (1995), 1235–1258.

[28] MILLER, E., SWICK, R., AND BRICKLEY, D. Resource description framework (RDF). Recommendation, W3C, 2004.

[29] MUGNIER, M.-L., AND CHEIN, M. Conceptual graphs: Fundamental notions. *Revue d'intelligence artificielle 6*, 4 (1992), 365–406.

[30] MUGNIER, M.-L., AND CHEIN, M. Polynomial algorithms for projection and matching. In *Proceedings of the 7th Annual Workshop on Conceptual Structures: Theory and Implementation* (London, UK, 1993), Springer-Verlag, pp. 239–251.

[31] PEREZ, J., ARENAS, M., AND GUTIERREZ, C. Semantics and complexity of SPARQL. In *Proceedings 5th International Semantic Web Conference* (Athens (GA US), 2006), pp. 30–43.

[32] POLLERES, A. From SPARQL to rules (and back). In *Proceedings of the 16th World Wide Web Conference (WWW)* (2007).

[33] PRUD'HOMMEAUX, E., AND SEABORNE, A. SPARQL query language for RDF. Working draft, W3C, 2006.

[34] TARJAN, R. E. Fast algorithms for solving path problems. *J. ACM 28*, 3 (1981), 594–614.

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

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

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399