Constrained Regular Expressions in SPARQL

Faisal Alkhateeb
INRIA Rhône-Alpes and LIG, France, Faisal.Alkhateeb@inrialpes.fr
Jean-François Baget
INRIA Sophia-Antipolis and LIRMM, France, Baget@lirmm.fr
Jérôme Euzenat
INRIA Rhône-Alpes and LIG, France, Jerome.Euzenat@inrialpes.fr

Abstract—We have proposed an extension of SPARQL, called PSPARQL, to characterize paths of variable lengths in an RDF knowledge base (e.g. "Does there exists a trip from town A to town B?"). However, PSPARQL queries do not allow expressing constraints on internal nodes (e.g. "Moreover, one of the stops must provide a wireless access."). This paper proposes an extension of PSPARQL, called CPSPARQL, that allows expressing constraints on paths. For this extension, we provide an abstract syntax, semantics as well as a sound and complete inference mechanism for answering CPSPARQL queries.

Keywords: RDF, SPARQL, constrained regular expressions, graph homomorphisms, query languages.

I. INTRODUCTION

The Resource Description Framework (RDF) [20] is a knowledge representation language dedicated to the annotation of resources within the Semantic Web. In its abstract syntax, an RDF document is a set of triples (subject, predicate, object), that can be represented by a directed labeled graph (hence the name RDF graph). The language is provided with a model-theoretic semantics [18], that defines the notion of consequence between two RDF graphs. Answers to an RDF query (the knowledge base and the query are RDF graphs) are determined by consequence, and can be computed using a particular map, a graph homomorphism [16], [7].

SPARQL [23] is a W3C recommendation language developed in order to query an RDF knowledge base (see a survey [17]). The heart of a SPARQL query, the graph pattern, is an RDF graph with variables. The maps that are used to compute answers to a query in an RDF knowledge base are exploited by [22] to define answers to the more expressive SPARQL queries (using, for example, disjunctions or functional constraints).

For added expressivity, we have proposed an extension of SPARQL, called PSPARQL (for Path SPARQL), that allows using regular expressions as predicates in an RDF triple [3], [5]. As done before in databases [13], [4], [1], [8], each regular expression can encode regular paths in an RDF graph such that the concatenation of arcs labels in each path form a word that belongs to the language generated by the regular expression. Using PSPARQL queries, we can ask questions of the form: "does there exists a trip from town A to town B?".

However, PSPARQL do not allow specifying properties on the nodes that belong to a path defined by a regular expression. It is thus impossible, for example, to enrich the previous query by "One of the stops must provide a wireless access."

This paper proposes an extension of PSPARQL, called CPSPARQL. Our definition to CPSPARQL relying on two main principles: the need to extend PSPARQL and thus SPARQL to allow expressing constraints on nodes of traversed paths, and the need to enhance the search process for finding paths that satisfy graph patterns involving path expressions.

In order to achieve these goals, we first define a kind of constrained regular expressions that extends the usual ones with constraints (constraints allowing to reduce the search space by selecting while doing the matching process nodes satisfying constraints). Then, we use constrained regular expressions to extend RDF graphs (i.e. the basic graph patterns of SPARQL) to have CPRDF graphs (for Constrained Paths RDF). Finally, we use CPRDF graphs to generalize SPARQL graph patterns, defining the CPSPARQL language.

Paper outline. The remainder of the paper is structured as follows. Section II presents some motivating examples which cannot be expressed by SPARQL and require to constrain paths. Section III is devoted to the presentation of the RDF language. The presentation framework of the CPRDF language is as follows: we first define the abstract syntax of the language in Section IV then its semantics by extending the standard RDF model-theoretic semantics in Section V. This is necessary to define answers to CPRDF graphs: there exists a solution S to a CPRDF graph P in an RDF graph G if G entails S(P) with respect to the extended semantics. This leads us to define a kind of graph homomorphism for finding answers to CPRDF graphs over RDF graphs in Section VI. CPRDF graphs (respectively, the maps) are used in Section VII to extend the SPARQL
query language to CPSPARQL (respectively, to answer CPSPARQL queries). Section VIII shows the effectiveness of our evaluation strategy which is demonstrated by a performance test on synthetic data. After a review of related work (Section IX), we conclude in Section X.

II. MOTIVATING EXAMPLES – INTRODUCING CPSPARQL

The following example queries attempt to give an insight of CPSPARQL.

Example 1 Consider the RDF graph $G$ of Fig. 1 that represents the transportation means between cities, the type of the transportation mean, and the price of tickets. For example, the existence of two triples like $(\text{flight}, \text{ex:from}, \text{C1})$ and $(\text{flight}, \text{ex:to}, \text{C2})$ means that C2 is directly reachable from C1 using flight.

Suppose someone wants to go from Roma to a city in one of the Canary Islands. The following SPARQL query finds the name of such city with only direct trips:

```sparql
SELECT ?City
WHERE { ex:Roma (ex:from-.ex:to)+ ?City . }
```

Nonetheless, SPARQL cannot express indirect trips with variable length paths. We can express that using regular expressions with the following (C)PSPARQL query:

```sparql
SELECT ?City
WHERE { ex:Roma (ex:from-.ex:to)+ ?City . }
```

$\neg$ is the inverse operator. For example, given the RDF triple $(\text{ex:Roma, ex:from, ex:flight})$, we can deduce $(\text{ex:flight, ex:from-, ex:Roma})$.

Suppose that he want to use only planes. To do that, we first define a constraint that consists of a name, interval delimiters to include or exclude path node extremities, a quantifier, and a variable is used to be substituted by nodes, and a graph to be matched. For example, the name of the constraint in the following query is cst1, it is open from left and universal which ensures that all trips are of type plane.

```sparql
SELECT ?City
WHERE { CONSTRAINT cst1 [ALL ?Trip]:
    FILTER (?Price < 500) }
```

The price of each direct trip is no more than 500:

```sparql
SELECT ?City
WHERE { CONSTRAINT cst1 [ALL ?Trip]:
    FILTER (?Price < 500) }
```

Moreover, if the user cannot go out the European union, e.g. for the visa problem, then we will require all intermediate stops to be cities in Europe.

```sparql
SELECT ?City
WHERE { CONSTRAINT cst1 [ALL ?Trip]:
    FILTER (?Price < 500) }
```

As we can see, CPSPARQL is definitely a more expressive language than SPARQL. We will now present it in details.

III. RDF

This section is dedicated to the presentation of the simple RDF language. The decision to use simple RDF as the basic building block for our extensions (and not RDF or RDFS) is justified by the fact that RDF and RDFS entailments are obtained from simple RDF entailments by applying rules to the knowledge base (a polynomial procedure) [13]. The same framework could easily be applied to CPRDF to extend, for example, our languages to CPRDFS. For the sake of clarity and brevity, we do not discuss these extensions in this paper. Moreover, to simplify notations, and without loss of generality, we do not distinguish here between simple and typed literals.

A. RDF syntax

RDF graphs are usually constructed over the set of uris, blanks, and literals [9]. "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. Terminology. An RDF terminology, noted $T$, is a union of 3 pairwise disjoint infinite sets of terms: the set $U$ of
urirefs, the set \( \mathcal{L} \) of literals and the set \( \mathcal{B} \) of variables. We call vocabulary and use \( \mathcal{V} = \mathcal{U} \cup \mathcal{L} \) to denote the set of names. From now on, we use the following notations for the elements of these sets: a variable will be prefixed by \(?\) (like \(?x1\)), a literal will be between quotation marks (like "27"), remaining elements will be urirefs (like \(\text{ex:price}\)).

**Definition 1 (RDF graph)** An RDF graph is a set of triples of \((\mathcal{U} \cup \mathcal{L}) \times \mathcal{B} \times \mathcal{T}\).

If \( G \) is an RDF graph, we use \( T(G), \mathcal{U}(G), \mathcal{L}(G), \mathcal{B}(G), \mathcal{V}(G) \) to denote the set of terms, urirefs, literals, variables or names that appeared at least in a triple of \( G \) (in Section VI these notations take into account the terms appearing in the constrained regular expressions). In a triple \((s, p, o)\), \(s\) is called the subject, \(p\) the predicate and \(o\) the object. It is possible to associate to a set of triples \( G \) a labeled directed graph, where the set of nodes is the set of terms appearing as a subject or object at least in a triple of \( G \), the set of arcs is the set of triples of \( G \), (i.e. if \((s, p, o)\) is a triple, then \( s \xrightarrow{p} o \) is an arc). By drawing these graphs, the nodes resulting from literals are represented by rectangles while the others are represented by rectangles with rounded corners. In what follows, we conflate the two views of RDF syntax (as sets of triples or labeled directed graphs). We will then speak interchangeably about its nodes, its arcs, or the triples which make them up.

### B. RDF semantics

By providing RDF with formal semantics, we express the conditions under which an RDF graph truly describes a particular world (i.e. an interpretation is a model for the graph) [13]. The usual notions of validity, satisfiability and consequence are entirely determined by these conditions.

**Definition 2 (Interpretation)** Let \( V \subseteq (\mathcal{U} \cup \mathcal{L}) \) be a vocabulary. An interpretation of \( V \) is a tuple \( I = (I_R, I_P, I_{EXT}, \iota) \) where:

- \( I_R \) is a set of resources that contains \( V \cap \mathcal{L} \);
- \( I_P \subseteq I_R \) is a set of properties;
- \( I_{EXT} : I_P \to 2^{I_R \times I_R} \) associates to each property a set of pairs of resources called the extension of the property;
- the interpretation function \( \iota : V \to I_R \) associates to each name \( v \) in \( V \) a resource of \( I_R \), if \( v \in \mathcal{L} \), then \( \iota(v) = v \).

**Definition 3 (Model of an RDF graph)** Let \( V \subseteq \mathcal{V} \) be a vocabulary, and \( G \) be an RDF graph such that \( \mathcal{V}(G) \subseteq V \). An interpretation \( I = (I_R, I_P, I_{EXT}, \iota) \) of \( G \) is a model of \( G \) if there exists a mapping \( \iota' : T(G) \to I_R \) that extends \( \iota \) (i.e. \( t \in V \cap T(G) \Rightarrow \iota'(t) = \iota(t) \)) such that for each triple \((s, p, o)\) of \( G \), \( \iota'(p) \in I_P \) and \((\iota'(s), \iota'(o)) \in I_{EXT}(\iota'(p)) \). The mapping \( \iota' \) is called a proof of \( G \) in \( I \).

The following definition is the standard model-theoretic definition of consequence.

**Definition 4 (Consequence)** A graph \( G' \) is a consequence of a graph \( G \), denoted by \( G \models G' \), iff every model of \( G \) is also a model of \( G' \).

In what follows, we use \( \models_{RDF} \) (respectively, \( \models_{CPRDF} \)) to denote RDF (respectively, CPRDF) consequences.

### C. Inference mechanism for RDF

The consequence in RDF is of utmost importance, since it is the basis for query answering. As done in [16], we use homomorphisms to prove consequence and answer queries.

**Definition 5 (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 \( \mu : V_1 \to V_2 \) such that \( \forall x \in (V_1 \cap \mathcal{V}) \), \( \mu(x) = x \) (i.e. that preserves names).

A map \( \mu \) and an extension \( \iota' \) of an interpretation function \( \iota \) are two different mappings, i.e. \( \mu \) is a mapping from terms to terms that preserves urirefs and literals while \( \iota' \) is a mapping from terms to resources that preserves the values of \( \iota \).

**Definition 6 (RDF homomorphism)** Let \( G \) and \( G' \) be two RDF graphs. An RDF homomorphism from \( G' \) into \( G \) is a map \( \pi : T(G') \to T(G) \) that preserves triples, i.e. such that \( \forall (s, p, o) \in G' \), \( \pi(s), \pi(p), \pi(o) \in G \).

**Theorem 1** [16, 27] Let \( G \) and \( G' \) be two RDF graphs. Then \( G \models_{RDF} G' \) iff there exists an RDF homomorphism from \( G' \) into \( G \).
homomorphism from $G'$ into $G$.

Example 2 The map $\tau$ defined by \{(?City1, ex:Roma), (?City2, ex:Madrid), (?Mean, ex:Iberia3), (?Price, "500"), (?Country, ex:Italy), (?Country, ex:Spain)\} is an RDF homomorphism from the RDF graph $H$ into $G$ of Fig. 7.

IV. CPRDF: SYNTAX

To be able to express properties on nodes that belong to a regular path, we extend PRDF \cite{5} by adding constraints to a regular expression. For the sake of simplicity and without loss of generality, we restrict the constraints in this section to be RDF graphs. Then, we parametrize the CPRDF language in the way that allows us to naturally extend it to include more general constraints.

Definition 7 (RDF constraint) An RDF constraint is written $\uparrow_1 Q \downarrow_2 C$ where $C$ is an RDF graph, $\uparrow_1$ and $\downarrow_2$ are one of the interval delimiters ( and ), $Q$ is a quantifier either ALL or EXISTS, and $x$ is a variable that labels a node of $C$.

A constraint consists of interval delimiters which are used to include or exclude the extremities of a path, a quantifier either ALL or EXISTS, a variable, and an RDF graph that must be satisfied by the internal nodes. For example, the constraint defined by $\text{[ALL ?Stop]}$ \{(?Stop, ex:cityIn, ?Country), (?Country, ex:partOf, ex:Europe)\} when applied to a regular expression $R$ ensures that all nodes except the source extremity in a path satisfying $R$ are cities in Europe. Intuitively, a path $p$ satisfies a regular expression $R$ if the word formed by concatenating the labels of the arcs along the path belongs to the language generated by $R$.

In what follows, we use $\Phi_{\text{RDF}}$ to denote the set of RDF constraints. When this restriction is not necessary, we use $\Phi$ to denote a constraint language.

Let $\Sigma$ be an alphabet. A language over $\Sigma$ is a subset of $\Sigma^*$: its elements are sequences of elements of $\Sigma$ called words. A word (non empty) $(a_1, \ldots, a_k)$ is denoted $a_1 \ldots a_k$. If $A = a_1 \ldots a_k$ and $B = b_1 \ldots b_q$ are two words over $\Sigma$, then $A \cdot B$ is the word over $\Sigma$ defined by $A \cdot B = a_1 \ldots a_k b_1 \ldots b_q$. A constrained regular expression over $(U, B, \Phi)$ can be used to define the language over $(U \cup B)$.

Definition 8 (Constrained regular expression) An infinite set of constrained regular expressions over $(U, B, \Phi)$ (denoted by $\mathcal{R}(U, B, \Phi)$) is defined inductively by:

- if $u \in U$, then $u$ and $u^{-} \in \mathcal{R}(U, B, \Phi)$;
- if $b \in B$, then $b \in \mathcal{R}(U, B, \Phi)$;
- if $R \in \mathcal{R}(U, B, \Phi)$, then $(R^\ast) \in \mathcal{R}(U, B, \Phi)$;
- if $R_1, R_2 \in \mathcal{R}(U, B, \Phi)$, then $(R_1 \cdot R_2)$ and $(R_1 | R_2)$ are elements of $\mathcal{R}(U, B, \Phi)$.
- if $R \in \mathcal{R}(U, B, \Phi)$ and $\psi \in \Phi$ is a constraint, then $R \psi \in \mathcal{R}(U, B, \Phi)$.

The inverse operator $\sim$ handles only atomic expressions. It specifies the orientation of arcs in the paths retrieved (i.e. it inverts the matching of arcs). Moreover, the constraints are not necessarily grouped together and we can have a constrained regular expression of the form $R \psi_1 \ldots \psi_k \in \mathcal{R}(U, B, \Phi)$. This allows us to specify at each grouped block different constraint (out) different variable(s), which is more flexible and general than grouping all constraints in one block.

Informally, a CPRDF[\Phi] graph is a graph whose arcs are labeled with constrained regular expressions whose constraints are elements of $\Phi$.

Definition 9 (CPRDF graph) A CPRDF[\Phi] triple is an element of $(T \times \mathcal{R}(U, B, \Phi) \times T)$. A CPRDF[\Phi] is a set of CPRDF[\Phi] triples.

Example 3 The CPRDF[\Phi_{\text{RDF}}] graph $H$ represented by the following triples:

\{(?City1, ex:cityIn, ex:Italy), (?City2, ex:cityIn, ex:CanaryIslands), (?City1, ex:from-, ex:to%, ALL ?Stop): (?City1, ex:partOf, ex:Europe) ?Country.
\}

when used as a query, finds pairs of cities (?City1, ?City2), one in Italy and the other in the Canary Islands, such that ?City2 is reachable from ?City1 by passing through only cities in Europe.

V. CPRDF: SEMANTICS

To be able to express the semantics of CPRDF[\Phi] graphs, we have first to define the language generated by a regular expression. The derivation trees used here are just a visual representation of the more usual inductive definition of derivation \cite{5}. The internal nodes of these trees will be used to define the semantics of constraints.

A. Generated language

Constraints of a given constrained regular expression has no effect on the generated regular language.

Definition 10 (Derivation tree) Let $R \in \mathcal{R}(U, B, \Phi)$ be a constrained regular expression. A rooted labeled tree with ordered subtrees $A$ is called a derivation tree of $R$ (denoted $A \in \mathcal{DT}(R)$) iff $A$ can be constructed inductively in the following way:

1) if $R = a \in (B \cup U)$, then $A$ is the tree of $R$.
2) if $R = (R^\ast)$ and $A_1, \ldots, A_k$ are a set of derivation trees of $\mathcal{DT}(R')$, then $A$ is the tree of $R$.

In the CPSPARQL implementation, we have used a prefix notation for expression operators like $+ R$ and $\sim R$. In the CPRDF implementation, we have used a prefix notation for expression operators like $+ R$ and $\sim R$. In the CPRDF implementation, we have used a prefix notation for expression operators like $+ R$ and $\sim R$.
3) if $R = (u^−)$, then $A$ is the tree of Fig. 2(e).

4) if $R = (R_1 \cdot R_2)$, $A_1 \in DT(R_1)$ and $A_2 \in DT(R_2)$, then $A$ is the tree of Fig. 2(d).

5) if $R = (R_1 | R_2)$ and $A' \in DT(R_1) \cup DT(R_2)$, then $A$ is the tree of Fig. 2(e).

6) if $R = (R \%\%\cdot\%\%)$ and $A' \in DT(R')$, then $A$ is the tree of Fig. 2(f).

The elements of a derivation tree are quantified using path labels in a given graph (see an example in the sequel).

**Definition 11 (Word)** To a derivation tree $A$ we associate a unique word $w(A)$, obtained by concatenating the labels of the leaves of $A$, totally ordered by the depth-first exploration of $A$ determined by the order of its subtrees. We use $\rho(A, i)$ to denote the $i^{th}$ leaf of $A$, according to that order.

The word associated to a derivation tree $A$ of a constrained regular expression $R$ belongs to the language generated by $R$, as usually defined by $L^*(R) = \{ w \in (\mathcal{U} \cup \mathcal{B})^+ \mid \exists A \in DT(R), w = w(A) \}$. Note that our definition ranges over $(\mathcal{U} \cup \mathcal{B})$, which is necessary when extending our work to RDF with variables as predicates (see [4]).

**B. Interpretations and models of CPRDF graphs**

A CPRDF interpretation of a vocabulary $V \subseteq \mathcal{V}$, is an RDF interpretation of $V$. However, an RDF interpretation must meet specific conditions to be a model for a CPRDF[\Phi] graph (Definition 14). These conditions are the transposition of the classical path semantics within the RDF semantics (Definition 12); and the satisfaction of the constraints by the resources of RDF interpretations (Definition 13).

**Definition 12 (Constrained regular expression proof)** Let $I = (I_R, I_P, I_{EXT}, i)$ be an interpretation of a vocabulary $V$, and $R \in \mathcal{R}(\mathcal{U}, \mathcal{B}, \Phi)$ be a constrained regular expression such that $\mathcal{U}(R) \subseteq V$. Let $i'$ be an extension of $i$ to $\mathcal{B}(R)$, and $w(A) = a_1, \ldots, a_k$ be a word of $L^*(R)$. A tuple $(r_0, \ldots, r_k)$ of resources of $I_R$ is called a proof of $w$ in $I$ according to $i'$ if $\forall 1 \leq i \leq k$:

- $\langle r_i, r_{i-1} \rangle \in I_{EXT}(i'(a_i))$ if $\rho(A, i)$ has an ancestor labeled by $\neg$;
- $\langle r_{i-1}, r_i \rangle \in I_{EXT}(i'(a_i))$, otherwise.

The first item of this definition handles the inverse operator ($\neg$): if the ancestor of $a_i$ is labeled by $\neg$ (i.e. it is equivalent to $a_i^\neg$), then we inverse the two resources that belong to the extension of the property of $i'(a_i)$. This definition is used for defining CPRDF models in which it replaces the direct correspondence that exists in RDF between a relation and its interpretation (see first item of Definition 14), by a correspondence between a constrained regular expression and a sequence of relation interpretations. This allows to match constrained regular expressions with variable length paths.

**Definition 13 (Proof of a constraint)** Let $\psi = \check{1}_1 Q x \check{1}_2 C$ be a constraint of $\Phi_{RDF}$, and $I = (I_R, I_P, I_{EXT}, i)$ be an interpretation of a vocabulary $V$. A resource $r$ of $I_R$ satisfies $\psi$ iff there exists a proof $i' : T \rightarrow I_R$ of $C$ such that $i'(x) = r$.

Now we are ready to define when an interpretation is a model of a CPRDF[\Phi_{RDF}] graph.

**Definition 14 (Model of a CPRDF graph)** Let $I = (I_R, I_P, I_{EXT}, i)$ be an interpretation of a vocabulary $V$, and $G$ be a CPRDF[\Phi_{RDF}] graph such that $\mathcal{U}(G) \subseteq V$. We say that $I$ is a model of $G$ iff there exists an extension $i'$ of $i$ such that for each triple $(s, R, o)$ of $G$, there exists a tuple $T = (r_0, \ldots, r_k)$ of resources of $I_R$ ($i'(s) = r_0$ and $i'(o) = r_k$) and a word $w(A) = a_1, \ldots, a_k \in L^*(R)$ such that:

- $T$ is a proof of $w$ in $I$ according to $i'$;
- for each node $v$ labeled by a constraint $\psi = \check{1}_1 Q x \check{1}_2 C$ in $A$, rooting a subtree $A'$ with $a_p, \ldots, a_{p+q} = w(A')$, then $Q r \in \check{1}_1 r_{p-1}, \ldots, r_{p+q} \check{1}_2$, $r$ satisfies $\psi$.

It is shown in the second item of this definition that adding constraints to a CPRDF[\Phi] graph reduces the number of models by selecting those ones whose resources satisfy constraints.

**VI. INFERENCE MECHANISM FOR CPRDF**

Two conditions must be satisfied for the notion of homomorphism to be able to find the answers to a CPRDF[\Phi] query in an RDF knowledge base (Definition 17): instead of proving an arc (a triple) of the query by an arc in the knowledge base, we prove it by
If there exists an RDF homomorphism $\pi : T(H) \rightarrow T(G)$ such that $\forall(s, R, o) \in H$, there exists a tuple $T = (n_0, \ldots, n_k)$ of nodes of $G$ ($\pi(s) = n_0$ and $\pi(o) = n_k$) and a word $w(A) = a_1 \cdot \ldots \cdot a_k \in L^*(R)$ such that:

- $T$ is a path of $w$ in $G$ according to $\pi$;
- for each node $z$ labeled by a constraint $\psi = \uparrow_1 Q x \uparrow_2 : C$ in $A$, rooting a subtree $A'$ with $a_p \ldots a_{p+q} = w(A')$, then $Q n \in \uparrow_1 n_{p-1} \ldots n_{p+q} \uparrow_2$. $n$ satisfies $\psi$.

The existence of a CPRDF[$\Phi$] homomorphism is exactly what is needed for deciding entailment between RDF and CPRDF[$\Phi$] graphs.

**Theorem 2 (CPRDF-RDF entailment)** Let $G$ be a RDF graph and $H$ be a CPRDF[$\Phi$] graph. A CPRDF[$\Phi$] homomorphism from $P$ into $G$ is a map $\pi : T(H)$ such that $\forall(s, R, o) \in H$, there exists a tuple $T = (n_0, \ldots, n_k)$ of nodes of $G$ ($\pi(s) = n_0$ and $\pi(o) = n_k$) and a word $w(A) = a_1 \cdot \ldots \cdot a_k \in L^*(R)$ such that:

- $T$ is a path of $w$ in $G$ according to $\pi$;
- for each node $z$ labeled by a constraint $\psi = \uparrow_1 Q x \uparrow_2 : C$ in $A$, rooting a subtree $A'$ with $a_p \ldots a_{p+q} = w(A')$, then $Q n \in \uparrow_1 n_{p-1} \ldots n_{p+q} \uparrow_2$. $n$ satisfies $\psi$.

The existence of a CPRDF[$\Phi$] homomorphism is exactly what is needed for deciding entailment between RDF and CPRDF[$\Phi$] graphs.

**Example 5** Consider the CPRDF[$\Phi$] RDF graph $H$ of Example 3 the RDF graph $G$ of Fig. 7 and the map $\pi$ defined by \{(?City1, ex:Roma), (?City2, ex:SantaCruz), (ex:from, ex:to), (ex:excityIn, ex:CityInland), (ex:Italy, ex:Italy)\}. According to Definition 17 the tuple of nodes of Example 4 (such that the first node ex:Roma and the last node ex:SantaCruz are the images of ?City1 and ?City2, respectively) is a path of a word of the regular expression of $H$ according to $\pi$ in $G$, and the stops along the path are all cities in Europe (see Fig. 7). So, $\pi$ is a CPRDF[$\Phi$] homomorphism from $H$ into $G$.

**VII. CPSPARQL**

[22] presents an alternate characterization of query answering with the SPARQL query language that relies upon operations on maps from the graph patterns of a query into an RDF knowledge base. We use this framework to extend SPARQL to CPSPARQL, by defining graph patterns as CPRDF[$\Phi$] graphs. Analogously, the set of answers to a CPSPARQL query is defined...
inductively from the set of maps of the CPRDF[Φ] graphs of the query into the RDF knowledge base.

A. Syntax

In CPSPARQL there are several functions that can be used for capturing the values along the paths like \textsc{sum} for summation of values along paths, \textsc{avg} for the average, \textsc{count} for counting nodes satisfying constraints. For the sake of simplicity, we have not introduced these function, and illustrate them with examples (cf. Section II). Moreover, since the graph patterns in the SPARQL query language are shared by all SPARQL query forms, we illustrate our extension using the \textsc{select …from …where …query form}[	extsuperscript{2}]. Our extension can then be applied to other query forms.

CPSPARQL graph patterns are built on top of CPRDF in the same way that SPARQL is built on top of RDF.

Definition 18 (CPSPARQL graph pattern) A CPSPARQL[Φ] graph pattern is defined inductively by:

1. every CPRDF[Φ] graph is a CPSPARQL[Φ] graph pattern;
2. if \(P_1\) and \(P_2\) are two CPSPARQL[Φ] graph patterns and \(R\) is a SPARQL constraint, then \((P_1\ \text{and}\ P_2), (P_1\ \text{union}\ P_2), (P_1\ \text{opt}\ P_2),\) and \((P_1\ \text{filter}\ R)\) are CPSPARQL[Φ] graph patterns.

Note: The parametrization of CPSPARQL[Φ] by \(Φ\) allows us to extend naturally its graph patterns to more general constraints. If \(Φ_{SPARQL}\) denotes the set of all possible SPARQL graph patterns, then a CPRDF[Φ\_{SPARQL}] graph could be a CPSPARQL[Φ\_{SPARQL}] graph pattern.

CPSPARQL query. A CPSPARQL[Φ] query for the select form \textsc{select} \(\tilde{B}\) \textsc{from} \(u\) \textsc{where} \(P\) such that \(P\) is a CPSPARQL[Φ] graph pattern.

B. Answers to CPSPARQL queries

We first need to introduce some notations and operations in maps. If \(µ\) is a map, then the domain of \(µ\), denoted by \(\text{dom}(µ)\), is the subset of \(T\) where \(µ\) is defined. If \(P\) is a graph pattern, then \(µ(P)\) is the graph pattern obtained by the substitution of \(µ(b)\) to each variable \(b \in B(P)\). Two maps \(µ_1\) and \(µ_2\) are compatible when \(∀x \in \text{dom}(µ_1) \cap \text{dom}(µ_2), \ µ_1(x) = µ_2(x)\). If \(µ_1 : T_1 \rightarrow T\) and \(µ_2 : T_2 \rightarrow T\) are two compatible maps, then we use \(µ = µ_1 \oplus µ_2 : T_1 \cup T_2 \rightarrow T\) to denote the map defined by: \(∀x \in T_1, µ(x) = µ_1(x)\) and \(∀x \in T_2, µ(x) = µ_2(x)\). Analogously to [22], we define the \textit{join} and \textit{difference} of two sets of maps \(Ω_1\) and \(Ω_2\) as follows:

1. (join) \(Ω_1 \times Ω_2 = \{µ_1 \oplus µ_2 \mid µ_1 \in Ω_1, µ_2 \in Ω_2\)
2. (difference) \(Ω_1 \setminus Ω_2 = \{µ_1 \in Ω_1 \mid ∀µ_2 \in Ω_2, µ_1 \oplus µ_2 \text{ is compatible}\}.

As in the case of SPARQL, the answer to a query reduced to a CPRDF[Φ] graph is also given by a map. The definition of an answer to a CPSPARQL query will be thus identical to the one given for SPARQL [22], but it will use CPRDF[Φ] homomorphisms.

Definition 19 (Answer to a graph pattern) Let \(G\) be an RDF graph and \(P\) be a CPSPARQL[Φ] graph pattern, then the set \(S(P,G)\) of answers of \(P\) in \(G\) is defined inductively by:

1. if \(P\) is a CPRDF[Φ] graph, \(S(P,G) = \{µ \mid µ\text{ is a CPRDF[Φ] homomorphism from }P\text{ into }G\};\)
2. if \(P = (P_1\ \text{and}\ P_2)\), \(S(P,G) = S(P_1,G) \times S(P_2,G)\);
3. if \(P = (P_1\ \text{union}\ P_2)\), \(S(P,G) = S(P_1,G) \cup S(P_2,G)\);
4. if \(P = (P_1\ \text{opt}\ P_2)\), \(S(P,G) = (S(P_1,G) \times S(P_2,G)) \cup (S(P_1,G) \setminus S(P_2,G))\);
5. if \(P = (P_1\ \text{filter}\ R)\), \(S(P,G) = \{µ \in S(P_1,G) \mid µ(R) = \top\}\).

Note: If CPSPARQL graph patterns are constructed over CPRDF[Φ\_{SPARQL}] graphs, then we need only to extend Definition 16 in the following way. Let \(G\) be a graph, \(P\) be a SPARQL graph pattern, \(ψ = \tilde{Q}_1\tilde{x}_1\tilde{Q}_2 : P\) be a constraint, and \(s\) a term of \(G\). We say that \(s\) satisfies \(ψ\) in \(G\) if there exists a map \(µ \in S(P,G)\) such that \(µ(x) = s\). The definition of CPRDF[Φ] homomorphism (Definition 27) and first item of Definition 79 remain unchanged.

Answers to a CPSPARQL[Φ] query are the instantiations of the set of maps from its graph patterns into the graph representing the knowledge base(s).

Definition 20 (Answer to CPSPARQL query) Let \(Q = \text{select} \tilde{B}\ \text{from} \ u\ \text{where} \ P\ \text{be} \ \text{a CPSPARQL[Φ] query,}\) Let \(G\) be the RDF graph identified by the URL \(ν\) and \(Ω\) the set of answers of \(P\) in \(G\). Then the answers to the query \(Q\) are the projections of elements of \(Ω\) to \(\tilde{B}\), i.e. for each map \(π\) of \(Ω\), the answer of \(Q\) associated to \(π\) is \(\{(x,y) \mid x \in \tilde{B}\ \text{and} \ y = π(x) \text{ if } π(x) \text{ is defined, otherwise } null\}\).

VIII. PERFORMANCE TEST

In this section, we provide a selected test of the CPSPARQL performance. Its main goal is to show
the usefulness of constraints in regular expressions for enhancing the search time (See the thesis for more tests).

We have tested the performance of the CPSPARQL prototype on a Dell machine with Bi-processor Xeon 5050 3GHz and 4GB of RAM. Java 1.5.0_07 has been used, and assigned 976 MB of RAM. We have run the test using several queries against different RDF graph sizes from 5, 10, ..., up to 10,000 triples. We have repeated the execution 50 times for each graph size, and the average time is taken.

**RDF graphs.** The RDF graphs are constructed randomly with different sizes using a random graph generator. To have a connected graph and to test queries containing path expressions, nodes of the graphs are selected from 800 distinct nodes representing cities and edges are selected from 2 distinct edge labels namely \{from,to\}. The average in and out degrees \((in-d\) and \(out-d\)) are calculated in function of the graph size, \(in-d = out-d = \sqrt{n}\), where \(n\) is the required graph size in edges. These settings increase the opportunity of having paths between cities with the same label, and also cycles. More precisely, we have constructed randomly an RDF graph similar to the one in Fig. 1 i.e. a graph containing only the following three kinds of triples \(\{\text{?blank, ex:from, C1}\}, \{\text{?blank, rdf:type, transport}\}, \text{and}\ \{\text{?blank, ex:to, C2}\}\), where \(\text{transport}\) is selected randomly from one of the following transportation means \{train, plane, bus, taxi\}.

**Test.** The goal of this test is to observe the effects of constraints in the performance. To achieve this goal, we have executed the following two CPSPARQL queries, \(Q_1\) containing a constrained regular expression and \(Q_2\) with a regular expression (without a constraint):

\[
\text{SELECT * WHERE } \{ \text{CONSTRAINT const1 }\} \text{ALL ?Trip]: } \{ ?s (ex:from=const1.ex:to)+ ?o . \}
\]

\[
\text{SELECT * WHERE } \{ ?s (ex:from- . ex:to)+ ?o . \}
\]

As shown in Fig. 4 the time for the query with constrained regular expression is better than that of the query without it. This shows that our query evaluator takes advantage of the constraints for cutting the search space during evaluation as it does not explore paths that cannot lead to a solution. Table I shows some of the average number of answers of each query (i.e. \(Q_1\) and \(Q_2\)) for selected graph sizes. There exists a large difference between the two expected answers.

**IX. Related Work**

There are many query languages dealing with paths: G+ [14], GraphLog [11], Lorel [1], UnQL [8], WebSQL [21], Corese [12] including our own extension to SPARQL, PSPARQL, [3], [5]. None of them deal with constraints.

Two extensions of SPARQL, which are closely similar to PSPARQL, have been defined:SPARQLeR [19] and SPARQL2L [6]. Both languages extend SPARQL by allowing query graph patterns involving path variables. Each path variable is used to capture paths in RDF graphs, and is matched against any arbitrary composition of RDF triples between two given nodes. The constraints in these extensions are simple, i.e. restricted to testing the length of paths and testing if a given node is in the resulting path. The queries in CPSPARQL are examples that can be emulated by neither SPARQL2L nor SPARQLeR. In addition, the strategy of obtaining paths and then filtering them is inefficient since it can generate a large number of paths due to the use of path variables. Multiple uses of same path variable is not fully defined: it is not specified which path is to be returned or if is it enforced to be the same.

A kind of constrained regular expressions has been proposed for XPath [15]. However, XPath operates on trees (not on graphs), and only defines monadic queries

\[\text{http://www.inrialpes.fr/exmo/people/alkhateeb/Thesis.pdf}\]
Several works attempt to adapt PDL-like queries for querying graphs with only monadic queries, cf. [2]. To our knowledge no other language for querying graphs supports constraints on paths. The originality of our proposal, CPSPARQL, lies in our adaptation of the RDF model-theoretic semantics to take into account constrained regular expressions, providing a wide range of querying paradigms. Moreover, CPSPARQL allows filtering constraints on the fly (during path search) and not a posteriori, and is not restricted to simple paths. This relaxation is not only useful for many applications (cf. [6] for some examples), but also provides polynomial classes for the regular expression satisfiability problem (i.e. when they do not contain variables).

X. Conclusion

Our initial proposal, PSPARQL, extends SPARQL to allow expressing variable length paths. Since PSPARQL and SPARQL do not allow specifying characteristics of the nodes traversed by a regular path, we have extended the PSPARQL language syntax and semantics to handle constraints to have the CPSPARQL language. We have also characterized answers to a CPSPARQL query in an RDF knowledge base as maps. This property was sufficient to extend the SPARQL query language to have a sound and complete inference mechanism for answering CPSPARQL queries over RDF graphs.

The proposed language, CPSPARQL has several advantages. First of all, it allows expressing variable length paths which can be qualified through the use of constraints. It enhances efficiency since the use of predefined constraints inside regular expressions prunes irrelevant paths during the evaluation process and not a posteriori. The constraints in CPSPARQL are extensible (i.e. it can be extended to include constraints that can be more general, as shown in Section VII), and partial (i.e. can be applied to a part of a regular expression, see examples in Section IV). The use of regular expressions supports a meaningful and natural use of inverse paths through the use of inverse operator.

Finally, we have implemented a CPSPARQL query engine that is available for both download and online test. This evaluator can be used to query RDF graphs written in N Triples or Turtle language as well as RDF data written within (X)HTML documents.

References


