# Constrained Regular Expressions in SPARQL

Faisal Alkhateeb (contact author)
INRIA Rhône-Alpes and LIG,
France, Faisal.Alkhateeb@inrialpes.fr
Submitted to SWWS'08

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 connection."). 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

RDF (Resource Description Framework, [21]) 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], [14], [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 this kind of entailment. 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). After a review of related work (Section VIII), we conclude in Section IX.

## 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 (*`flight`*,* `ex:from`*,* `C1`*) and (*`flight`*,* `ex:to`*,* `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:*

```
SELECT ?City
WHERE { ?Trip ex:from ex:Roma . ?Trip ex:to ?City .
```

```
        ?City ex:cityIn ex:CanaryIslands . }
```

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

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

*Where "-" is the inverse operator. For example, given the RDF triple (*`ex:Roma`*,* `ex:from`*,* `ex:flight`*), we can deduce (*`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* `const1`*, it is open from left and universal which ensures that all trips are of type plane.*

```
SELECT ?City
WHERE { CONSTRAINT const1 ]ALL ?Trip]:
                           {?Trip rdf:type ex:Plane . }
        ex:Roma (ex:from-%const1%.ex:to)+ ?City .
        ?City ex:cityIn ex:CanaryIslands . }
```

*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.*

```
SELECT ?City
WHERE { CONSTRAINT const1 ]ALL ?Trip]:
                             {?Trip rdf:type ex:Plane .}
        CONSTRAINT const2 ]ALL ?Stop]:
                             {?Stop ex:cityIn ?Country .
                              ?Country ex:partOf ex:Europe .}
        ex:Roma (ex:from-%const1%.ex:to%const2%)+ ?City .
        ?City ex:cityIn ex:CanaryIslands . }
```

*The price of the whole trip is no more than 1000, then we can use the* `SUM` *function in the following query:*

```
SELECT ?City
WHERE { CONSTRAINT const1 SUM(?Sum1,?Price) ]ALL ?Trip]:
                         {?Trip rdf:type ex:Plane .
                          ?Trip ex:price ?Price .
                          FILTER (SUM(?Sum1,?Price) < 1000)}
        CONSTRAINT const2 ]ALL ?Stop]:
                         {?Stop ex:cityIn ?Country .
                          ?Country ex:partOf ex:Europe . }
        ex:Roma (ex:from-%const1%.ex:to%const2%)+ ?City .
        ?City ex:cityIn ex:CanaryIslands . }
```

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) [18]. 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 urirefs, 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 $\mathcal{T}$, is a union of 3 pairwise disjoint infinite sets of *terms*: the set $\mathcal{U}$ 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 `ex:price`).

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

If $G$ is an RDF graph, we use $\mathcal{T}(G)$, $\mathcal{U}(G)$, $\mathcal{L}(G)$, $\mathcal{B}(G)$, $\mathcal{V}(G)$ 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) [18]. The usual notions of validity, satisfiability and consequence are entirely determined by these conditions.

*Definition* **2 (Interpretation of a vocabulary)** *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;*
- $I_{EXT} : I_P \rightarrow 2^{I_R \times I_R}$ *associates to each property a set of pairs of resources called the* extension *of the property;*
- *the interpretation function* $\iota : V \rightarrow I_R$ *associates to each name 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* $\mathcal{V}(G) \subseteq V$*.*

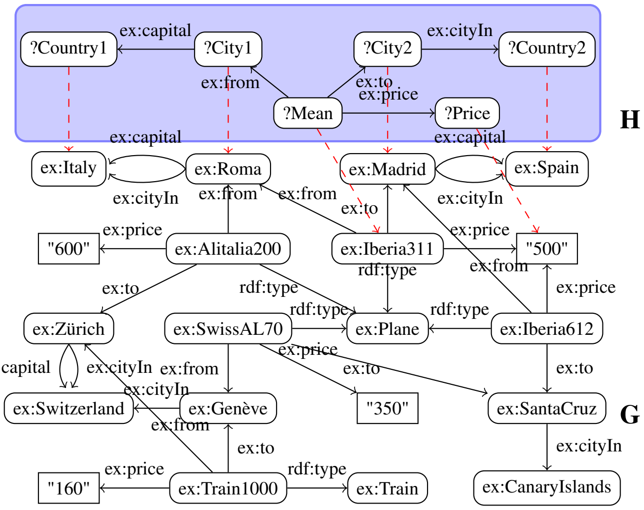

Fig. 1. An RDF homomorphism.

*An interpretation* $I = (I_R, I_P, I_{EXT}, \iota)$ *of* $V$ *is a* model *of* $G$ *iff there exists a mapping* $\iota' : \mathcal{T}(G) \to I_R$ *that extends* $\iota$ *(i.e.,* $t \in V \cap \mathcal{T}(G) \Rightarrow \iota'(t) = \iota(t)$*) 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 $\models_{\text{RDF}}$ (respectively, $\models_{\text{CPRDF}}$) 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* $\pi : \mathcal{T}(G') \to \mathcal{T}(G)$ *that preserves triples, i.e., such that* $\forall(s, p, o) \in G', (\pi(s), \pi(p), \pi(o)) \in G$*.*

**Theorem 1 ([16], [7])** *Let* $G$ *and* $G'$ *be two RDF graphs. Then* $G \models_{RDF} G'$ *iff there exists an RDF homomorphism from* $G'$ *into* $G$*.*

**Example 2** *The map* $\pi$ *defined by* $\{$*(`?Country1`,`ex:Italy`), (`?Country2`,`ex:Spain`), (`?City1`,`ex:Roma`), (`?City2`, `ex:Madrid`), (`?Mean`,`ex:Iberia311`), (`?Price`,`"500"`)*$\}$ *is an RDF homomorphism from the RDF graph* $H$ *into* $G$ *of Fig. 1.*

## IV. CPRDF: SYNTAX

To be able to express properties on nodes that belong to a regular path, we extend PRDF [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* $\dagger_1 Q x \dagger_2 : C$ *where* $C$ *is an RDF graph,* $\dagger_1$ *and* $\dagger_2$ *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 ]`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 $a_1 \cdot \ldots \cdot a_k$. If $A = a_1 \cdot \ldots \cdot a_k$ and $B = b_1 \cdot \ldots \cdot b_q$ are two words over $\Sigma$, then $A \cdot B$ is the word over $\Sigma$ defined by $A \cdot B = a_1 \cdot \ldots \cdot a_k \cdot b_1 \cdot \ldots \cdot b_q$. A constrained regular expression over $(\mathcal{U}, \mathcal{B}, \Phi)$ can be used to define the language over $(\mathcal{U} \cup \mathcal{B})$.

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

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

The inverse operator $^-$ handles only atomic expressions. It specifies the orientation of arcs in the paths retrieved (*i.e.*, it inverses the matching of arcs). Moreover, the constraints are not necessarily grouped together and we can have a constrained regular expression of the form $R\%\psi_1\% \ldots \%\psi_k\%$. This allows us to specify at each grouped block different constraint with(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 $(\mathcal{T} \times \mathcal{R}(\mathcal{U},\mathcal{B},\Phi) \times \mathcal{T})$. A CPRDF[$\Phi$] graph is a set of CPRDF[$\Phi$] triples.*

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

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

*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 [5]. The internal nodes of these trees will be used to define the semantics of constraints.

### A. Generated language

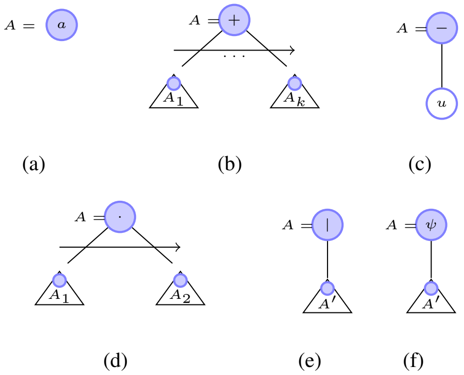

Fig. 2. Constructing a derivation tree of a constrained regular expression.

*Definition* **10 (Derivation tree)** *Let $R \in \mathcal{R}(\mathcal{U},\mathcal{B},\Phi)$ 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 $R = a \in (\mathcal{B} \cup \mathcal{U})$, then $A$ is the tree of Fig. 2(a);*
2) *if $R = (R'^{+})$ and $A_1, \ldots, A_k$ are a set of derivation trees of $\mathcal{DT}(R')$, then $A$ is the tree of Fig. 2(b);*
3) *if $R = (u^{-})$, then $A$ is the tree of Fig. 2(c);*
4) *if $R = (R_1 \cdot R_2)$, $A_1 \in \mathcal{DT}(R_1)$ and $A_2 \in \mathcal{DT}(R_2)$, then $A$ is the tree of Fig. 2(d);*
5) *if $R = (R_1 | R_2)$ and $A' \in \mathcal{DT}(R_1) \cup \mathcal{DT}(R_2)$, then $A$ is the tree of Fig. 2(e);*
6) *if $R = (R'\%\psi\%)$ and $A' \in \mathcal{DT}(R')$, 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 $L^*(R) = \{w \in (\mathcal{U} \cup \mathcal{B})^+ \mid \exists A \in \mathcal{DT}(R),\ w = w(A)\}$. 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 (Proof of a constrained regular expression)** *Let $I = (I_R, I_P, I_{EXT}, \iota)$ 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 $\iota'$ be an extension of $\iota$ to $\mathcal{B}(R)$, and $w(A) = a_1 \cdot \ldots \cdot a_k$ 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 $\iota'$ iff $\forall 1 \leq i \leq k$:*

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

The first item of this definition handles the inverse operator ($^{-}$): if the ancestor of $a_i$ is labeled by $^{-}$ (*i.e.*, it is equivalent to $a_i^{-}$), then we inverse the two resources that belong to the extension of the property of $\iota'(a_i)$. 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 (Constraint satisfaction in an interpretation)** *Let $\psi = \dagger_1 Q x \dagger_2 : C$ be a constraint of $\Phi_{RDF}$, and $I = (I_R, I_P, I_{EXT}, \iota)$ be an interpretation of a vocabulary $V$. A resource $r$ of $I_R$ satisfies $\psi$ iff there exists a proof $\iota' : \mathcal{T} \rightarrow I_R$ of $C$ such that $\iota'(x) = r$.*

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

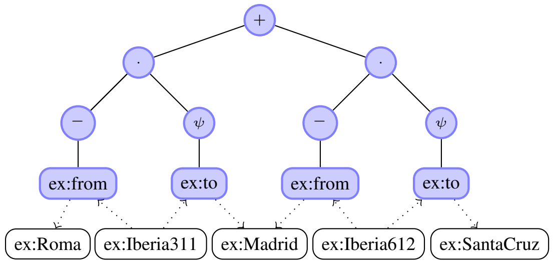

Fig. 3. A derivation tree.

*Definition* **14 (Model of a CPRDF graph)** *Let $I = (I_R, I_P, I_{EXT}, \iota)$ 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 $\iota'$ of $\iota$ such that for each triple $(s, R, o)$ of $G$, there exists a tuple $T = (r_0, \ldots, r_k)$ of resources of $I_R$ ($\iota'(s) = r_0$ and $\iota'(o) = r_k$) and a word $w(A) = a_1 \cdot \ldots \cdot a_k \in L^*(R)$ such that:*

- *$T$ is a proof of $w$ in $I$ according to $\iota'$;*
- *for each node $z$ labeled by a constraint $\psi = \dagger_1 Qx \dagger_2 : C$ in $A$, rooting a subtree $A'$ with $a_p \cdot \ldots \cdot a_{p+q} = w(A')$, then $Q\ r \in \dagger_1 r_{p-1}, \ldots, r_{p+q} \dagger_2$, $r$ satisfies $\psi$.*

## 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[Φ] 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 a path in the knowledge base (Definition 15); and the satisfaction of the corresponding node(s) in the path of the knowledge base to the constraint(s) (Definition 16).

*Definition* **15 (Path word)** *Let $G$ be an RDF graph of vocabulary $V \subseteq \mathcal{V}$, and $R \in \mathcal{R}(\mathcal{U}, \mathcal{B}, \Phi)$ be a constrained regular expression such that $\mathcal{U}(R) \subseteq V$. Let $\mu : \mathcal{B}(R) \rightarrow V$ be a map from the variables of $R$ to $V$, and $w(A) = a_1 \cdot \ldots \cdot a_k$ be a word of $L^*(R)$. A tuple $(n_0, \ldots, n_k)$ of nodes of $G$ is called a* path *of $w$ in $G$ according to $\mu$ iff $\forall 1 \leq i \leq k$:*

- *$(n_i, \mu(a_i), n_{i-1}) \in G$ if $\rho(A, i)$ has an ancestor labeled by $^-$;*
- *$(n_{i-1}, \mu(a_i), n_i) \in G$, otherwise.*

As done for the interpretation (Definition 12), the first item handles the inverse operator: if the ancestor of $a_i$ is labeled by $^-$, then we inverse the orientation of the arc.

**Example 4** *Fig. 3 shows a possible derivation tree of the constrained regular expression $R =$(`ex:from`$^-$·`ex:to`%$\psi$%)$^+$ of the graph $H$ in Example 3, where $\psi$ =]`ALL ?Stop`]: {(`?Stop, ex:cityIn, ?Country`), (`?Country, ex:partOf, ex:Europe`)}. The nodes in white color, which correspond to the path of nodes in the RDF graph $G$ of Fig. 1, together with the path labels are used to quantify the elements of the tree. The tuple $T$=(`ex:Roma, ex:Iberia311, ex:Madrid, ex:Iberia612, ex:SantaCruz`) of nodes in the RDF graph $G$ of Fig. 1 is a path of the word $w$=(`ex:from`$^-$· `ex:to`·`ex:from`$^-$ ·`ex:to`) $\in L^*(R)$ according to the empty map.*

The following definition gives the condition(s) when a constraint of $\Phi_{\text{RDF}}$ is satisfied. This definition can be extended based on the constraints (see notes in Section VII).

*Definition* **16 (Constraint satisfaction in RDF graphs)** *Let $G$ be an RDF graph, $\psi = \dagger_1 Qx \dagger_2 : C$ be a constraint of $\Phi_{RDF}$, and $s$ a term of $G$. Then $s$ satisfies $\psi$ in $G$ if there exists an RDF homomorphism $\pi$ from $C$ into $G$ such that $\pi(x) = s$.*

Intuitively, in CPRDF[Φ] homomorphisms, each internal node labeled by a constraint $\psi$ of a derivation tree determines the subtree (not necessary the whole tree, since a constraint $\psi$ may be applied to a partial part of a constrained regular expression, Definition 8) whose corresponding nodes in the knowledge base graph must satisfy $\psi$ (see the second item of the following definition).

*Definition* **17 (CPRDF homomorphism)** *Let $G$ be an RDF graph and $H$ be a CPRDF[Φ] graph. A* CPRDF[Φ] homomorphism *from $P$ into $G$ is a map $\pi : \mathcal{T}(H) \rightarrow \mathcal{T}(G)$ 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 $\psi = \dagger_1 Qx \dagger_2 : C$ in $A$, rooting a subtree $A'$ with $a_p \cdot \ldots \cdot a_{p+q} = w(A')$, then $Q\ n \in \dagger_1 n_{p-1}, \ldots, n_{p+q} \dagger_2$, $n$ satisfies $\psi$.*

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

**Theorem 2 (CPRDF-RDF entailment [4])** *Let $G$ be a RDF graph and $H$ be a CPRDF[$\Phi_{RDF}$] graph, then $G \models_{CPRDF} H$ iff there is a CPRDF[$\Phi_{RDF}$] homomorphism from $H$ into $G$.*

**Example 5** *Consider the CPRDF[$\Phi_{RDF}$] graph $H$ of Example 3, the RDF graph $G$ of Fig. 1, and the map $\pi$ defined by {(`?City1,ex:Roma`), (`?City2,ex:SantaCruz`), (`ex:from,ex:from`), (`ex:to,ex:to`), (`ex:cityIn,ex:cityIn`), (`ex:Italy,ex:Italy`), (`?Country,ex:CanaryIslands`)}. 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. 3). So, $\pi$ is a CPRDF[$\Phi_{RDF}$] 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[Φ] graphs.

Analogously, the set of answers to a CPSPARQL query is defined inductively from the set of maps of the CPRDF[$\Phi$] 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 `SUM` for summation of values along paths, `AVG` for the average, `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 and that our proposal is based upon extending these graph patterns, we illustrate our extension using the SELECT ... FROM ... WHERE ... query form[1]. 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 patterns)** *A* CPSPARQL [$\Phi$] graph pattern *is defined inductively by:*

- *every CPRDF[$\Phi$] graph is a CPSPARQL[$\Phi$] graph pattern;*
- *if $P_1$ and $P_2$ are two CPSPARQL[$\Phi$] 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 CPSPARQL[$\Phi$] graph patterns.*

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

**CPSPARQL query.** A CPSPARQL[$\Phi$] query for the select form `SELECT` $\vec{B}$ `FROM` $u$ `WHERE` $P$ such that $P$ is a CPSPARQL[$\Phi$] graph pattern.

### B. Answers to CPSPARQL queries

We first need to introduce some notations and operations in maps. If $\mu$ is a map, then the domain of $\mu$, denoted by $dom(\mu)$, is the subset of $\mathcal{T}$ where $\mu$ is defined. If $P$ is a graph pattern, then $\mu(P)$ is the graph pattern obtained by the substitution of $\mu(b)$ to each variable $b \in \mathcal{B}(P)$. Two maps $\mu_1$ and $\mu_2$ are *compatible* when $\forall x \in dom(\mu_1) \cap dom(\mu_2)$, $\mu_1(x) = \mu_2(x)$. If $\mu_1 : T_1 \rightarrow \mathcal{T}$ and $\mu_2 : T_2 \rightarrow \mathcal{T}$ are two compatible maps, then we use $\mu = \mu_1 \oplus \mu_2 : T_1 \cup T_2 \rightarrow \mathcal{T}$ to denote the map defined by: $\forall x \in T_1, \mu(x) = \mu_1(x)$ and $\forall x \in T_2, \mu(x) = \mu_2(x)$. Analogously to [22], we define the *join* and *difference* of two sets of maps $\Omega_1$ and $\Omega_2$ as follows:

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

As in the case of SPARQL, the answer to a query reduced to a CPRDF[$\Phi$] 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[$\Phi$] homomorphisms.

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

- *if $P$ is a CPRDF[$\Phi$] graph, $\mathcal{S}(P, G) = \{\mu \mid \mu$ is a CPRDF[$\Phi$] 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` $R)$, $\mathcal{S}(P, G) = \{\mu \in \mathcal{S}(P_1, G) \mid \mu(R) = \top\}$.*

**Note:** *If CPSPARQL graph patterns are constructed over CPRDF[$\Phi_{SPARQL}$] graphs, then we need only to extend Definition 16 in the following way: Let $G$ be a graph, $P$ be a SPARQL graph pattern, $\psi = \dagger_1 Q x \dagger_2 : P$ be a constraint, and $s$ a term of $G$. We say that $s$ satisfies $\psi$ in $G$ if there exists a map $\mu \in \mathcal{S}(P, G)$ such that $\mu(x) = s$. The definition of CPRDF[$\Phi$] homomorphism (Definition 17) and first item of Definition 19 remain unchanged.*

Answers to a CPSPARQL[$\Phi$] 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 a CPSPARQL query)** *Let $Q$ = `SELECT` $\vec{B}$ `FROM` $u$ `WHERE` $P$ be a CPSPARQL[$\Phi$] query. Let $G$ be the RDF graph identified by the URL $u$, and $\Omega$ the set of answers of $P$ in $G$. Then the answers to the query $Q$ are the projections of elements of $\Omega$ to $\vec{B}$, i.e., for each map $\pi$ of $\Omega$, the answer of $Q$ associated to $\pi$ is $\{(x, y) \mid x \in \vec{B}$ and $y = \pi(x)$ if $\pi(x)$ is defined, otherwise `null`$\}$.*

## VIII. Related Work

There are many query languages dealing with paths: G+ [14], GraphLog [11], Lorel [1], UnQL [8], WebSQL [20], 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 recently defined based on our initial proposal [3]: SPARQLeR [19] and SPARQ2L [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

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

by neither SPARQ2L nor SPARQLeR. Several problems are shared by the two extensions when we evaluate such graph patterns. In particular, 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. Since SPARQLeR is not defined with a formal semantics, its use of path variables in the subject position is unclear, in particular, when they are not bound. It seems that the algorithms used in SPARQ2L are not complete with regard to their intuitive semantics, since the set of answers can be infinite in absence of constraints for using shortest or acyclic paths.

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 [10]. Several works attempt to adapt PDL-like or $\mu$-calculus for querying graphs with monadic queries, *cf.* [2]. We conjuncture that CPSPARQL and even SPARQL queries are not reducible to $\mu$-calculus since the evaluation problem of basic SPARQL queries is PSPACE-complete [22] and that for $\mu$-calculus is P.

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).

## IX. 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. Note that the W3C SPARQL Working Group considered our proposal to be one of the extensions in the second version of SPARQL [2].

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[3]. 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 II). The use of regular expressions supports a meaningful and natural use of inverse paths through the use of inverse operator.

As it is shown along the paper, we go far beyond the trivial constraints, *i.e.*, testing simple paths and the existence of a node along the path. Finally, we have implemented a CPSPARQL query engine that is available for both download and online test [4].

[2] http://esw.w3.org/topic/SPARQL/Extensions/Paths

[3] We have recently experimented the implementation and we found an enhancement of $0.3 - 0.4$ of the search time, and we can provide the results if you allow us a little bit place.

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

## References

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

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

[3] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Complex path queries for RDF graphs. In *ISWC, poster paper*, 2005.

[4] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Constrained regular expressions in SPARQL. Research Report 6360, INRIA, 2007.

[5] F. Alkhateeb, J.-F. Baget, and J. Euzenat. RDF with regular expressions. Research report 6191, INRIA, 2007.

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

[7] J.-F. Baget. RDF entailment as a graph homomorphism. In *Proceed. of the 4th ISWC, Galway (IE)*, pages 82–96, 2005.

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

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

[10] J. Clark and S. DeRose. XML Path Language (XPath). W3C Recommendation, 1999.

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

[12] O. Corby, R. Dieng-Kuntz, and C. Faron-Zucker. Querying the semantic web with corese search engine. In *Proceed. of the 16th ECAI'2004, subconference (PAIS'2004), Valencia (Spain)*, pages 705–709, 2004.

[13] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *Proceed. of the ACM SIGMOD*, pages 323–330, 1987.

[14] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G+: Recursive queries without recursion. In *Proceed. of the Expert Database Conference*, pages 355–368, 1988.

[15] P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *Proceed. of PLDI'07*, pages 342–351, 2007.

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

[17] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of RDF query languages. In *Proceed. of 3rd ISWC*, pages 502–517, 2004.

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

[19] K. Kochut and M. Janik. SPARQLeR: Extended sparql for semantic association discovery. In *Proceed. of 4th ESWC*, pages 145–159, 2007.

[20] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the world wide web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.

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

[22] J. Perez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *Proceed. of the 5th ISWC*, pages 30–43, 2006.

[23] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Working draft, 2007.