On foundational aspects of RDF and SPARQL

Dominique Duval, Rachid Echahed, Frederic Prost

ON FOUNDATIONAL ASPECTS OF RDF AND SPARQL

DOMINIQUE DUVAL [1], RACHID ECHAHED [2], AND FRÉDÉRIC PROST [2]

Abstract. We consider the recommendations of the World Wide Web Consortium (W3C) about the Resource Description Framework (RDF) and the associated query language SPARQL. We propose a new formal framework based on category theory which provides clear and concise formal definitions of the main basic features of RDF and SPARQL. We propose to define the notions of RDF graphs as well as SPARQL basic graph patterns as objects of some nested categories. This allows one to clarify, in particular, the role of blank nodes. Furthermore, we consider basic SPARQL CONSTRUCT and SELECT queries and formalize their operational semantics following a novel algebraic graph transformation approach called POIM.

1. Introduction

Mathematical semantics of computer science languages has been advocated since early 1970's. It allows one to give precise meaning of syntactical objects and paves the way for involved reasoning methods such as modularity, compositionality, security as well as advanced verification techniques, to quote a few. Nowadays, data science represents one of the most influential technology in our society. Mastering the languages involved in the encoding of data or the formulation of queries is a necessity to elaborate robust data management systems. In this paper, we consider the most recent recommendations of the World Wide Web Consortium (W3C) about the Resource Description Framework (RDF) [W3C14a] and the associated query language SPARQL [W3C13] and propose a mathematical semantics of these formalisms.

The key data structure in RDF is the structure of RDF graph. In [W3C14a, Section 3], an RDF graph is defined as a set of RDF triples, where an RDF triple is of form $(subject, predicate, object)$. The subject is either an IRI (Internationalized Resource Identifier) or a blank node, the predicate is an IRI and the object is either an IRI, a literal (denoting a value such as a string, a number or a date) or a blank node. Blank nodes are arbitrary elements as long as they differ from IRIs and literals and they do not have any internal structure: they are used for indicating the existence of a thing and the blank node identifiers are locally scoped. For instance, the triples $(Paul, knows, blank1)$ and $(blank2, knows, Henry)$ mean that Paul knows someone and someone knows Henry. Surprisingly, a triple such as $(Paul, blank3, Henry)$ standing for "there is some relationship between Paul and Henry" is not allowed in RDF, but only in generalized RDF [W3C14a, Section 7]. Following the theoretical point of view we investigate in this paper, there is no harm to consider blank predicates within RDF triples. We thus consider *data graphs* in a more general setting including RDF graphs.

The query language SPARQL for RDF databases is based on basic graph patterns, which are kinds of RDF graphs with variables [W3C13, Section 2]. In this paper, we consider

Date: [1] Laboratoire Jean Kuntzmann, Univ. Grenoble Alpes, CNRS, France
[2] Laboratoire d'Informatique de Grenoble, Univ. Grenoble Alpes, CNRS, France
October 15, 2019.

query graphs which generalize basic graph patterns by allowing blanks to be predicates. The SPARQL query processor searches for triples within the given RDF database which match the triple patterns in the given basic graph pattern, and returns a result set or an RDF graph. Considering basic graph patterns, one may wonder what is the difference between variables and blank nodes. SPARQL specifications in [W3C13, Section 4.1.4] suggest similarities between them, whereas in [W3C13, Section 16.2], they make a clear difference between them. In the formalization of SPARQL we propose, blank nodes and variables are clearly distinguished by their respective roles in the definition of homomorphisms.

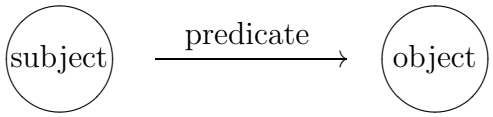
In SPARQL, the SELECT query form is described lengthily. This query form can be compared to the SELECT query form of SQL, which returns a set of results. In contrast, the CONSTRUCT query form returns a graph of results. The latter is described very shortly in [W3C13, Section 16.2]. Following our formalization, the CONSTRUCT query form is more fundamental than the SELECT query form. Actually, we start by proposing an operational semantics for CONSTRUCT queries based on a new approach of algebraic graph transformations which we call POIM and we show afterward how SELECT queries can be easily encoded as CONSTRUCT queries.

The paper is organized as follows. Section 2 defines the objects and the morphisms of the categories of data graphs and query graphs. We use basic notions of category theory, mainly colimits (such as coproducts and pushouts). These notions are defined in all textbooks on category theory, and Wikipedia [Wik] provides an elementary presentation. All colimits used in this paper are kinds of "unions" of specific sets. We describe them also in a set-theoretic way, so that it should be possible to get an idea of the content of the paper without knowing the formal definition of colimits. Section 3 introduces the PO-IM algebraic transformation where rewrite rules are of the following shape $L \to K \leftarrow R$ with L, K and R being basic graph patterns. Afterward, we define, in Section 4, two different calculi for running a CONSTRUCT query against a data graph G. We first define a *high-level calculus* as a mere application of the PO-IM transformation. Then we propose a *low-level calculus* which is defined by means of several applications of the PO-IM transformation followed by a "merging" process. The low-level calculus is close to the description of the running process in SPARQL. Both calculi are shown to return the same result. In Section 5, we show how the PO-IM transformation can be adapted to define the operational semantics of the SELECT queries. Concluding remarks are given in Section 6. The missing proofs can be found in the Appendix.

2. Graphs of triples

2.1. **Graphs of triples.**

The set of IRIs (Internationalized Resource Identifiers), denoted *Iri*, is defined in [W3C14a]. The set of literals (numbers, strings, booleans or dates), denoted *Lit*, with its usual operations, is defined in [W3C14a]. The sets *Iri* and *Lit* are disjoint. In addition let B be some countably infinite set, disjoint from *Iri* and *Lit*. The elements of B are called the *blanks*. According to [W3C14a, 3.1], *an RDF graph is a set of RDF triples and an RDF triple consists of three components: the subject, which is an IRI or a blank node; the predicate, which is an IRI; and the object, which is an IRI, a literal or a blank node. The set of nodes of an RDF graph is the set of subjects and objects of triples in the graph.*



Using set-theoretic notations, this can be expressed as follows: let $Tr = (Iri+B) \times Iri \times (Iri+Lit+B)$, then an RDF triple is an element of Tr and an RDF graph is a subset of Tr. Let us also consider the following non-standard extension of RDF [W3C14a, 7]: *A generalized RDF triple is a triple having a subject, a predicate, and object, where each can be an IRI, a blank node or a literal. A generalized RDF graph is a set of generalized RDF triples.* This means that a generalized RDF triple is an element of $(Iri+Lit+B)^3$ and that a generalized RDF graph is a subset of $(Iri+Lit+B)^3$.

In addition to the sets Iri for IRIs, Lit for litterals and B for blanks, let V be some countably infinite set disjoint from Iri, Lit and B. The elements of V are called the *variables*. According to [W3C13, 2] *a set of triple patterns is called a basic graph pattern. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable.* Let $Tr_V = (Iri+B+V) \times (Iri+V) \times (Iri+Lit+B+V)$, then a triple pattern is an element of Tr_V and a basic graph pattern is a subset of Tr_V. Since Tr_V is a subset of $(Iri+Lit+B+V)^3$, each basic graph pattern is a subset of $(Iri+Lit+B+V)^3$.

More generally let I denotes any countably infinite set called the set of *immutable attributes*, generalizing the disjoint union $Iri + Lit$. Let $IB = I+B$, $IV = I+V$ and $IBV = I+B+V$.

Definition 2.1. For each set A, the *triples on* A are the elements of A^3. For each triple $t = (s, p, o)$ on A the elements s, p and o of A are called respectively the *subject*, the *predicate* and the *object* of t, and they are denoted $subj(t)$, $pred(t)$ and $obj(t)$. A *graph on* A is a set of triples on A, i.e. a subset of A^3. For each graph T on A, the subset of A made of the subjects, predicates and objects of T is called the set of *attributes* of T and is denoted $|T|$; it follows that T is a subset of $|T|^3$. A *data graph* is a graph on IB and a *query graph* is a graph on IBV.

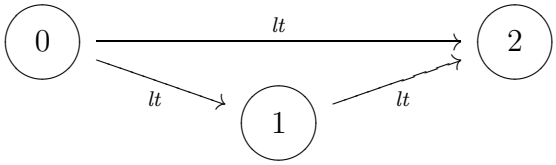
Thus, the RDF graphs are the data graphs where only nodes can be blanks and only nodes that are not subjects can be literals, and the RDF terms of an RDF graph are its attributes when it is seen as a data graph. Similarly, the basic graph patterns of SPARQL are the query graphs where only nodes can be blanks and only nodes that are not subjects can be literals.

The notions of *inclusion*, *subgraph*, *image* and *union* for graphs on A are defined as inclusion, subset, image and union for subsets of A^3.

For each data graph T, let $I(T) = I \cap T$ and $B(T) = B \cap T$, so that $|T|$ is the disjoint union of $I(T)$ and $B(T)$. Similarly, for each query graph T, let $I(T) = I \cap T$, $B(T) = B \cap T$ and $V(T) = V \cap T$, so that $|T|$ is the disjoint union of $I(T)$, $B(T)$ and $V(T)$.

Remark 2.1. Each graph T on A determines a graph in the usual sense of "directed multigraph", with nodes the elements of A which are subjects or objects of T and with edges from s to o the triples (s, p, o) in T. This graph has no isolated node. This point of view can be used for illustrating graphs on A by drawing each triple (s, p, o) as an edge $s \xrightarrow{p} o$. Note that the name of the edge $s \xrightarrow{p} o$ is the whole triple (s, p, o), since the predicate p may be shared by several triples: see Example 2.1

Example 2.1. Let $N = \{0, 1, 2\}$ and let lt (or $<$) be the relation "less than" on N, i.e. $lt = \{(0, 1), (0, 2), (1, 2)\}$. Let $P = \{lt\} = \{\{(0, 1), (0, 2), (1, 2)\}\}$ and $T = \{(0, lt, 1), (0, lt, 2), (1, lt, 2)\}$. Then T is the set of $(s, p, o) \in N \times P \times N$ such that $(s, o) \in p$. Let $A = \{0, 1, 2, lt\}$, then both N and P are subsets of A and T is the set of triples (s, p, o) on A such that $(s, o) \in p$. The graph T on A can be illustrated as follows:



2.2. **Morphisms for graphs of triples.**

The isomorphisms of RDF graphs are defined in [W3C14a, 3.6.] as follows. *Two RDF graphs G and G' are isomorphic if there is a bijection M between the sets of nodes of the two graphs, such that: – M maps blank nodes to blank nodes. – $M(lit) = lit$ for all RDF literals lit which are nodes of G. – $M(iri) = iri$ for all IRIs iri which are nodes of G. – The triple (s, p, o) is in G if and only if the triple $(M(s), p, M(o))$ is in G'.*
With our notations, this means that two RDF graphs G and G' are isomorphic if there is a bijection $M : |G| \to |G'|$ such that $(s, p, o) \in G$ if and only if $(M(s), M(p), M(o)) \in G'$ and $M(x) = x$ for each $x \in I(G)$ (which includes all predicates of G) and $M(x) \in B(G')$ for each $x \in B(G)$.

Definition 2.2. For each set A a *morphism* $a : T \to T'$ from a graph T on A to a graph T' on A is a map $a : T \to T'$ (subsets of A^3) such that $a(s, p, o) = (M(s), M(p), M(o))$ for a map $M : |T| \to |T'|$ (subsets of A). This map M will be denoted $|a|$, this is not ambiguous since M is determined by a. A morphism $a : T \to T'$ of graphs on A *fixes* a subset C of A if $|a|(x) = x$ for each x in $|T| \cap C$.

Definition 2.2 provides a one-to-one correspondence between the morphisms $a : T \to T'$ of graphs on A and the maps $|a| : |T| \to |T'|$ such that $|a|^3(T) \subseteq T'$. A morphism a fixing C is determined by the restriction of the map $|a|$ to $A \backslash C$. Definition 2.2 is such that when $A = IB$, the invertible morphisms fixing I between RDF graphs are the isomorphisms of RDF graphs.

Definition 2.3. A *matching morphism* is a morphism of query graphs fixing I from a query graph to a data graph. A query graph L *matches* a subgraph G' of a data graph G if there is a matching morphism m from L to G such that G' is the image of m.

Thus, a matching morphism fixes each immutable attribute and it maps a variable or a blank to any immutable attribute or blank. A match in the sense of SPARQL is a matching morphism from a basic graph pattern to an RDF graph. Indeed, quoting [W3C13, 4.1.4]: *Blank nodes in graph patterns act as variables* and [W3C13, 2]: *a basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph.* The expression "RDF graph equivalent" in the quotation above is the old terminology for "RDF graph isomorphic", which means that each blank node can be replaced with a new blank node (compare [W3C04, 6.3] and [W3C14a, 3.6]).

The interpretations of a RDF graph are also kinds of morphisms. In Definition 2.4 we define an interpretation of a data graph T in a universe of discourse U by generalising Definition 2.2 according to [W3C14a, 1.2.]: *Any IRI or literal denotes something in the world (the "universe of discourse"). These things are called resources. The resource denoted by an IRI is called its referent, and the resource denoted by a literal is called its literal value. Asserting an RDF triple says that some relationship, indicated by the predicate, holds between the resources denoted by the subject and object. This statement corresponding to an RDF triple is known as an RDF statement. The predicate itself is an IRI and denotes a property, that is, a resource that can be thought of as a binary relation.* Note that the binary relations on a set R are the subsets of R^2 and that a binary relation on R may be an element of R: see Example 2.1.

Definition 2.4. Given a set R and a subset P of R made of binary relations on R, let U be the set of triples (s, p, o) in R^3 such that $p \in P$ and $(s, o) \in p$. The *universe of discourse* with R as set of *resources* and P as set of *properties* is the graph U on R. Given a universe of discourse U on a set R and a map $M_I : I \to R$, an *interpretation* of a data graph T is a map $i : T \to U$ such that $i = M^3$ for a map $M : |T| \to |U|$ which extends M_I.

2.3. **Categories for graphs of triples.**

Definition 2.5. For each set A the category made of the graphs on A (Definition 2.1) with the morphisms between them (Definition 2.2) is called the *category of graphs on* A and denoted $\mathcal{G}(A)$. For each subset C of A the subcategory of $\mathcal{G}(A)$ made of the graphs on A with the morphisms fixing C is denoted $\mathcal{G}_C(A)$. The *category of data graphs* is $\mathcal{D} = \mathcal{G}(IB)$ and for each subset C of IB the category of data graphs fixing C is the subcategory $\mathcal{D}_C = \mathcal{G}_C(IB)$ of $\mathcal{D}$. The *category of query graphs* is $\mathcal{Q} = \mathcal{G}(IBV)$ and for each subset C of IBV the category of query graphs fixing C is the subcategory $\mathcal{Q}_C = \mathcal{G}_C(IBV)$ of $\mathcal{Q}$.

In this paper we consider categories $\mathcal{D}_C$ and $\mathcal{Q}_C$ for various subsets C of IB and IBV respectively. It will always be the case that C contains I, so that we can say that immutable attributes have a "global scope". In contrast, blanks have a "local scope": in the basic part of RDF and SPARQL considered in this paper, the scope of a blank node is restricted to one data graph or one query graph. The note about *blank node identifiers* in [W3C14a, 3.4] distinguishes two kinds of syntaxes for RDF: an abstract syntax where blank nodes do not have identifiers and concrete syntaxes where blank nodes have identifiers. In our approach this distinction is formalized as follows: a blank *is* an attribute, which corresponds to a concrete syntax, and the abstract syntax is obtained by considering data graphs as objects of the category $\mathcal{D}_I$ up to isomorphism, so that any blank node can be changed for a new blank node if needed. The flexibility of this point of view is important: in order to formalize the SPARQL evaluation process for the CONSTRUCT queries we have to consider a category $\mathcal{D}_{IB_0}$ where $IB_0 = I + B_0$ for a well-chosen set B_0 of blanks: in this category only the blanks outside B_0 have a local scope.

Example 2.2. In all examples we use the following prefixes (`@prefix` for data and `PREFIX` for queries):

Prefixes

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/>.
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
```

Consider two RDF graphs G_1, G_2 as follows. They are isomorphic in $\mathcal{D}_I$ but not in $\mathcal{D}_{IB}$ because blanks are swapped.

G_1

```
<http://example.org/Al> foaf:knows _:b. _:c foaf:knows <http://example.org/Bob>.
```

G_2

```
<http://example.org/Al> foaf:knows _:c. _:b foaf:knows <http://example.org/Bob>.
```

Now consider basic graph patterns G_3 to G_8. They are pairwise non-isomorphic in $\mathcal{Q}_{IBV}$ because they are pairwise distinct. In $\mathcal{Q}_{IV}$ only G_7 and G_8 are isomorphic. In $\mathcal{Q}_I$ these query graphs belong to two different isomorphism classes: on one side G_3 and G_4 are isomorphic and on the other side G_5, G_6, G_7 and G_8 are isomorphic.

G_3

```
<http://example.org/Al> foaf:knows _:b. _:b foaf:knows <http://example.org/Bob>.
```

G_4

```
<http://example.org/Al> foaf:knows ?x. ?x foaf:knows <http://example.org/Bob>.
```

G_5

```
<http://example.org/Al> foaf:knows _:b. _:c foaf:knows <http://example.org/Bob>.
```

G_6

```
<http://example.org/Al> foaf:knows ?x. ?y foaf:knows <http://example.org/Bob>.
```

G_7

```
<http://example.org/Al> foaf:knows ?x. _:b foaf:knows <http://example.org/Bob>.
```

G_8

```
<http://example.org/Al> foaf:knows ?x. _:c foaf:knows <http://example.org/Bob>.
```

Assumption 2.1. **From now on A is a set, C is a subset of A, $\overline{C} = A \setminus C$ is the complement of C in A, and it is assumed that both C and $\overline{C}$ are countably infinite.**

Remark 2.2. Since $\overline{C}$ is countably infinite, when dealing with a finite number of finite graphs on A it is always possible to find a *new attribute outside* C, i.e., an element of $\overline{C}$ that is not an attribute of any of the given graphs. It follows from Definition 2.2 that given a graph T on A, if any attribute of T in $\overline{C}$ is replaced by any new element of $\overline{C}$ the result is a graph T' on A that is isomorphic to T in $\mathcal{G}_C(A)$. The fact that C is countably infinite will be used in Section 5.

Proposition 2.1. *The coproduct of graphs $T_1, ..., T_k$ in $\mathcal{G}_C(A)$ is the union $T'_1 \cup ... \cup T'_k$ where T'_i is isomorphic to T_i in $\mathcal{G}_C(A)$ for each i and $|T'_i| \cap |T'_j| \subseteq C$ for each $i \neq j$.*

Proof. By Remark 2.2 there are graphs T'_i's on A such that $T'_i \cong T_i$ in $\mathcal{G}_C(A)$ and $|T'_i| \cap |T'_j| \subseteq C$ when $i \neq j$. Thus, up to isomorphism in $\mathcal{G}_C(A)$ we can assume that $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$, and the required result is that the coproduct $T_1 + ... + T_k$ in $\mathcal{G}_C(A)$ is the union $T_1 \cup ... \cup T_k$ in A^3. Consider morphisms $a_i : T_i \to T$ in $\mathcal{G}_C(A)$ for $i = 1, ..., k$ and the maps $|a_i| : |T_i| \to |T|$. Note that $|T_1 \cup ... \cup T_k| = |T_1| \cup ... \cup |T_k|$ and that $|T_1| \cup ... \cup |T_k|$ is the disjoint union of the sets $|T_i| \setminus C$ for $i = 1, ..., k$ and $(|T_1| \cup ... \cup |T_k|) \cap C$, because of the assumption $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$. Thus we can define a map $M : |T_1 \cup ... \cup T_k| \to |T|$ by: $M(x) = |a_i|(x)$ for each i and each $x \in |T_i| \setminus C$ and $M(x) = x$ for each $x \in (|T_1| \cup ... \cup |T_k|) \cap C$. Then M coincides with $|a_i|$ on $|T_i|$ for each i. Thus for each $t \in T_i$ we have $M^3(t) = |a_i|^3(t)$, which proves that the image of $T_1 \cup ... \cup T_k$ by M^3 is in T and that the restriction of M^3 defines a morphism $a : T_1 \cup ... \cup T_k \to T$ in $\mathcal{G}_C(A)$ which coincides with a_i on T_i for each i. □

Proposition 2.2. *Let $l : L \to K$ and $m : L \to G$ be morphisms of graphs on A such that l is an inclusion and m fixes C. Let us assume that $|K| \cap |G| \subseteq C$ (this is always*

possible up to isomorphism in $\mathcal{G}_C(A)$, by Remark 2.2). Let $N : |K| \to A$ be such that $N(x) = |m|(x)$ for $x \in |L|$ and $N(x) = x$ otherwise. Let $D_1 = N^3(K)$ and $D = G \cup D_1$. Let $n : K \to D$ be the restriction of N^3 and $g : G \to D$ the inclusion. Then the square (l, m, n, g) is a pushout in $\mathcal{G}_C(A)$.

Proof. The square (l, m, n, g) is a commutative square in $\mathcal{G}_C(A)$. Consider morphisms $a : G \to T$ and $b : K \to T$ in $\mathcal{G}_C(A)$ such that $a \circ m = b \circ l$. Then the maps $|a| : |G| \to |T|$ and $|b| : |K| \to |T|$ are such that $|a| \circ |m| = |b| \circ |l|$. Since $D = G \cup D_1$ and $D_1 = N^3(K)$, it follows from the definition of N that $|D| = |G| + (|K| \setminus |L|)$. Let $M : |D| \to |T|$ be the map such that $M(x) = |a|(x)$ for $x \in |G|$ and $M(x) = |b|(x)$ for $x \in |K| \setminus |L|$, then $M \circ |g| = |a|$ and $M \circ N = |b|$. Thus $M^3(t) = |a|^3(t)$ for each $t \in G$ and $M^3(n(t)) = |b|^3(t)$ for each $t \in K$, which proves that $M^3(D)$ is in T and that the restriction of M^3 defines a morphism $c : D \to T$ in $\mathcal{G}_C(A)$ such that $c \circ g = a$ and $c \circ n = b$. □

Remark 2.3. The construction in Proposition 2.2 does not depend on C: it provides a pushout of l and m in $\mathcal{G}_C(A)$ for any C fixed by m and satisfying Assumption 2.1.

3. The PO-IM transformation

3.1. **Basic construct queries.** A SPARQL query like "CONSTRUCT $\{R\}$ WHERE $\{L\}$" is called *basic* when both R and L are basic graph patterns. In such a query, variables with the same name in L and R denote the same RDF term, whereas it is not the case for blank nodes. The statement "*blank nodes in graph patterns act as variables*" in [W3C13, 4.1.4] holds for L, whereas blank nodes in R give rise to new blank nodes in the result of the query as in Example 3.4. Thus, the meaning of blank nodes in L is unrelated to the meaning of blank nodes in R, and in both L and R each blank can be replaced by a new blank.

We generalise this situation in Definition 3.1 by allowing any data graphs for L and R up to isomorphism in $\mathcal{Q}_{IV}$: the immutable attributes and the variables in L and R are fixed but each blank can be replaced by a new blank. Thus, without loss of generality, we can assume that $B(L) \cap B(R) = \emptyset$. Under this assumption, the set of triples $K = L \cup R$ with the inclusions of L and R in K is the coproduct of L and R in the category $\mathcal{Q}_{IV}$. We also assume that each variable in R occurs in L, so that every substitution for the variables in L provides a substitution for the variables in R; the relevance of this assumption is discussed in Remark 3.2. This assumption $V(R) \subseteq V(L)$ is equivalent to $V(K) = V(L)$.

Definition 3.1. A *basic construct query* is a pair of finite query graphs (L, R) such that $B(L) \cap B(R) = \emptyset$ and $V(R) \subseteq V(L)$, up to isomorphism in the category $\mathcal{Q}_{IV}$. The *transformation rule* of a basic construct query (L, R) is the coproduct $P_{L,R} = (L \xrightarrow{l} K \xleftarrow{r} R)$ of L and R in $\mathcal{Q}_{IV}$, where $K = L \cup R$ with l and r the inclusions. Its *left-hand side* is L and its *right-hand side* is R.

$$P_{L,R} \;=\; L \xrightarrow[\subseteq]{l} K = L \cup R \xleftarrow[\supseteq]{r} R$$

3.2. **Basic construct queries with one match.**

Example 3.1. Consider the following SPARQL CONSTRUCT" query and its corresponding basic CONSTRUCT query which is the pair (L, R) where:

```
CONSTRUCT {?x vcard:FN ?name}
WHERE     {?x foaf:name ?name}
```

L

```
?x foaf:name ?name.
```

R

```
?x vcard:FN ?name.
```

There are no blanks in L nor in R, thus the transformation rule is $L \xrightarrow{l} K \xleftarrow{r} R$ where l and r are the inclusions of L and R in $K = L \cup R$:

K

```
?x foaf:name ?name ; vcard:FN ?name.
```

Example 3.2. Now the SPARQL CONSTRUCT query from Example 3.1 is modified by replacing the variable `?x` by the blank node `_:x` giving:

```
CONSTRUCT   { _:x vcard:FN ?name }
WHERE       { _:x foaf:name ?name }
```

The corresponding basic construct query is the pair (L, R) where one blank has been modified so as to ensure that $B(L) \cap B(R)$ is empty:

L

```
_:x foaf:name ?name.
```

R

```
_:b vcard:FN ?name.
```

There is no blank common to L and R, thus the transformation rule is $L \xrightarrow{l} K \xleftarrow{r} R$ where l and r are the inclusions of L and R in $K = L \cup R$:

K

```
_:x foaf:name ?name. _:b vcard:FN ?name.
```

3.3. **The PO-IM transformation.** When a basic SPARQL query "CONSTRUCT $\{R\}$ WHERE $\{L\}$" is runned against an RDF graph G, and when there is precisely one match of L into G, the result is an RDF graph H which is obtained by substituting for the variables in R. This substitution can be seen as a match of R into H. We claim that the process of building H with this match of R into H from the match of L into G can be seen as a two-steps process involving an intermediate match of K in some RDF graph D. The definition of this process relies on an algebraic construction that we call the *PO-IM transformation*: PO for *pushout* and IM for *image* (Definition 3.2). The PO-IM transformation is related to a large family of algebraic graph transformations based on pushouts, like the SPO (Simple Pushout), DPO (Double Pushout) or SqPO (Sesqui Pushout) for instance [CHHK06, CMR+97].

The PO-IM transformation is defined as a functor between *categories of matches*.

Given a query graph X, the *category of matches of* X is the category $\mathcal{M}atch(X)$ where an object is a matching morphism from X to any data graph and an arrow from $m : X \to Y$ to $m' : X \to Y'$ is a morphism $g : Y \to Y'$ in $\mathcal{D}_I$ such that $g \circ m = m'$. Given a morphism $r : R \to K$ in $\mathcal{Q}_I$, the *image factorisation functor along* r is the functor $r^+ : \mathcal{M}atch(K) \to \mathcal{M}atch(R)$ that maps each $n : K \to D$ to $r^+(n) : R \to H$ where H is the image of $n \circ r : R \to D$ and $r^+(n)$ is the restriction of n, as in the right part of Diagram (1) where $h : H \to D$ is the inclusion. Given a morphism $l : L \to K$ in $\mathcal{Q}_I$, the *cobase change functor along* l is the functor $l_* : \mathcal{M}atch(L) \to \mathcal{M}atch(K)$ that maps each $m : L \to G$ to $l_*(m) : K \to D$ defined from the pushout of l and m in $\mathcal{Q}_I$ as in the left part of Diagram (1). Note that "the" functor l_* is defined only up to isomorphism, so that we can assume that $|K| \cap |G| \subseteq I$, at the cost of changing some blanks. The pushout $(l, m, l_*(m), g)$ is described in Proposition 2.2: the data graph D is $G \cup D_1$ where $D_1 = N^3(K)$ and $N : |K| \to A$ is such that $N(x) = |m|(x)$ for $x \in |L|$ and $N(x) = x$ otherwise, the morphism $l_*(m) : K \to D$ is the restriction of N^3 and $g : G \to D$ is the

inclusion. Note that D_1 is a data graph because of the assumption $V(K) = V(L)$, so that $D = G \cup D_1$ is also a data graph.

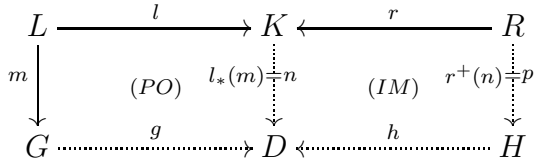
The PO-IM transformation is defined as a functor between *categories of matches*. Given a query graph X, the *category of matches of* X is the category $\mathcal{M}atch(X)$ where an object is a matching morphism from X to any data graph and an arrow from $m : X \to Y$ to $m' : X \to Y'$ is a morphism $g : Y \to Y'$ in $\mathcal{D}_I$ such that $g \circ m = m'$. Given an inclusion $r : R \to K$ in $\mathcal{Q}_I$, the *image factorisation functor along* r is the functor $r^+ : \mathcal{M}atch(K) \to \mathcal{M}atch(R)$ that maps each $n : K \to D$ to $r^+(n) : R \to H$ where H is the image of of R in D and $r^+(n)$ is the restriction of n, as in the right part of Diagram (1), and $h : H \to D$ is the inclusion. Given an inclusion $l : L \to K$ in $\mathcal{Q}_I$, the *cobase change functor along* l is the functor $l_* : \mathcal{M}atch(L) \to \mathcal{M}atch(K)$ that maps each $m : L \to G$ to $l_*(m) : K \to D$ defined from the pushout of l and m in $\mathcal{Q}_I$ as in the left part of Diagram (1), so that D is a kind of "union of G and K over L", as follows. Note that the functor l_* is defined only up to isomorphism, so that we can assume that $|K| \cap |G| \subseteq I$, at the cost of changing some blanks. The pushout $(l, m, l_*(m), g)$ is described in Proposition 2.2: the data graph D is $G \cup D_1$ where $D_1 = N^3(K)$ and $N : |K| \to A$ is such that $N(x) = |m|(x)$ for $x \in |L|$ and $N(x) = x$ otherwise, the morphism $l_*(m) : K \to D$ is the restriction of N^3 and $g : G \to D$ is the inclusion. Note that D_1 is a data graph because of the assumption $V(K) = V(L)$, so that $D = G \cup D_1$ is also a data graph. Now the PO-IM transformation is defined categorically (Definition 3.2) then it is described in a set-theoretic way (Proposition 3.1).

Definition 3.2. Let (L, R) be a basic construct query and $L \xrightarrow{l} K \xleftarrow{r} R$ its transformation rule. The *PO-IM transformation functor* of (L, R) is the functor

$$PoIm_{L,R} = r^+ \circ l_* : \mathcal{M}atch(L) \to \mathcal{M}atch(R)$$

composed of the cobase change functor l_* and the image factorisation functor r^+. The *result* of applying $PoIm_{L,R}$ to a matching morphism $m : L \to G$ is the morphism $PoIm_{L,R}(m) : R \to H$ or simply the query graph H.

(1)



Proposition 3.1. *Let (L, R) be a basic construct query and $m : L \to G$ a matching morphism. Let $P : |R| \to A$ be defined by $P(x) = |m|(x)$ for $x \in V(R)$ and $P(x) = x$ otherwise. Then, up to isomorphism in $\mathcal{Q}_I$, the result of applying $PoIm_{L,R}$ to m is $p : R \to H$ where $H = P^3(R)$ and p is the restriction of P^3.*

Proof. With the notations of Diagram (1), we know that $H = |p|^3(R)$ where $|p|(x) = |n|(x)$ for every $x \in |R|$, and we know that $|n|(x) = |m|(x)$ for $x \in |L|$ and $|n|(x) = x$ otherwise. The proposition follows since $|L| \cap |R|$ is the disjoint union of $I(L) \cap I(R)$, that is fixed by all morphisms in $\mathcal{Q}_I$, and $V(L) \cap V(R)$, with $V(L) \cap V(R) = V(R)$ since $V(R) \subseteq V(L)$. □

Remark 3.1. Note that the result of applying $PoIm_{L,R}$ to m is unchanged if G is replaced by any subgraph that contains the image of m. For instance G can be replaced by the

image of m without changing the result. This remark can be used for controling the size of the graphs involved. All graphs L, R and G are finite, so that K, D and H are finite as well. But typically G is "large" whereas L and R are "small", so that K and R are "small" but D is "large". When G is replaced by the image of m then all graphs are "small".

Remark 3.2. When L and R are basic graph patterns and G is an RDF graph, Proposition 3.1 shows that the result returned by SPARQL when the query "CONSTRUCT $\{R\}$ WHERE $\{L\}$" is runned against G is the same as the result H of applying $PoIm_{L,R}$ and then dropping all the triples that are not RDF triples; note that there is no triple containing an unbound variable in H because $V(R) \subseteq V(L)$. Indeed here is the description from [W3C13, 16.2], simplified in order to fit with the assumption that there is exactly one match: *The CONSTRUCT query form returns a single RDF graph specified by a graph template. The result is an RDF graph formed by [...] substituting for the variables in the graph template [...]. If [...] [this] instantiation produces a triple containing an unbound variable or an illegal RDF construct, such as a literal in subject or predicate position, then that triple is not included in the output RDF graph. The graph template can contain triples with no variables (known as ground or explicit triples), and these also appear in the output RDF graph returned by the CONSTRUCT query form.*

Example 3.3. Consider the SPARQL CONSTRUCT query from Example 3.1, and let us run this query against the RDF graph G:

```
CONSTRUCT {?x vcard:FN ?name}
WHERE     {?x foaf:name ?name}
```

G

```
_:a foaf:name "Alice";
    foaf:mbox <mailto:alice@example.org>.
```

There is a single matching morphism m, it is such that $m(?\texttt{x}) = \texttt{\_:a}$ and $m(\texttt{?name}) = \texttt{"Alice"}$. The PO-IM transformation produces successively the following data graphs D and H, where H is the query result:

D

```
_:a foaf:name "Alice" ;
    foaf:mbox <mailto:alice@example.org> ;
    vcard:FN "Alice".
```

H

```
_:a vcard:FN "Alice".
```

Example 3.4. Now consider the SPARQL CONSTRUCT query from Example 3.2:

```
CONSTRUCT   { _:x vcard:FN ?name }
WHERE       { _:x foaf:name ?name }
```

Let us run this query against the RDF graph G from Example 3.3. There is a single matching morphism m, it is such that $m(\texttt{\_:x}) = \texttt{\_:a}$ and $m(\texttt{?name}) = \texttt{"Alice"}$. The PO-IM transformation produces successively the following data graphs D and H, where H is the query result:

D

```
_:a foaf:name "Alice" ;
    foaf:mbox <mailto:alice@example.org>.
_:b vcard:FN "Alice".
```

H

```
_:b vcard:FN "Alice".
```

4. Basic construct queries

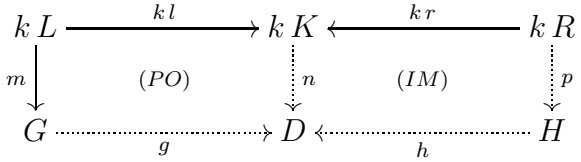
In Section 3 we defined the PO-IM transformation and we applied it to run a CONSTRUCT query against a data graph G, under the strong assumption that there is exactly one matching morphism from L to G. Now we define two different calculi for

running a CONSTRUCT query against a data graph G without any assumption on the number of matching morphisms. The *high-level calculus* (Definition 4.1) is defined as a simple application of the PO-IM transformation. The *low-level calculus* (Definition 4.2) is defined from several applications of the PO-IM transformation followed by a "merging" process, it is less simple but it fits with the description of the running process in SPARQL. In Theorem 4.1 we prove that both calculi return the same result.

4.1. **The high-level calculus.** Let k be a natural number. According to Proposition 2.1, for each query graph T the query graph $k\,T$, coproduct of k copies of T in $\mathcal{Q}_I$, can be built (up to isomorphism) as follows: for each $i \in \{1, ..., k\}$ let T_i be a copy of T where each blank and variable has been renamed in such a way that there is no blank or variable common to two of the T_i's, then the query graph $k\,T$ is the union $T_1 \cup ... \cup T_k$. Now let (L, R) be a basic construct query. The transformation rule $P_{L,R} = (L \xrightarrow{l} K \xleftarrow{r} R)$ is a cospan in $\mathcal{Q}_I$, that gives rise to the cospan $k\,P_{L,R} = (k\,L \xrightarrow{k\,l} k\,K \xleftarrow{k\,r} k\,R)$. Since l and r are inclusions, this renaming can be done simultaneously in the copies of L, K and R, so that $k\,K = k\,L \cup k\,R$ and $k\,l$ and $k\,r$ are the inclusions. Thus, $(k\,L, k\,R)$ is a basic construct query and $P_{k\,L,k\,R} = k\,P_{L,R}$ is its transformation rule. As before, we can assume that $B(k\,L) \cap B(G) = \emptyset$ and $B(k\,R) \cap B(G) = \emptyset$ without loss of generality.

Definition 4.1. Let (L, R) be a basic construct query and let G be a data graph such that $B(L) \cap B(G) = \emptyset$ and $B(R) \cap B(G) = \emptyset$. Let $m_1, ..., m_k$ be the matching morphisms from L to G. Build the basic construct query $(k\,L, k\,R)$ in such a way that $B(k\,L) \cap B(G) = \emptyset$ and $B(k\,R) \cap B(G) = \emptyset$. Let m be the matching morphism from $k\,L$ to G that coincides with m_i on the i-th component of $k\,L$. The *high-level query result* (or simply the *query result*) *of* (L, R) *against* G is the result H of applying the PO-IM transformation functor $PoIm_{k\,L,k\,R}$ to the match $m : k\,L \to G$, as in Diagram (2).

(2)



Example 4.1. Let us run the query from Example 3.1 against the following RDF graph G in the high-level calculus:

G

```
_:a foaf:name "Alice" ; foaf:mbox <mailto:alice@example.org>.
_:b foaf:name "Bob"   ; foaf:mbox <mailto:bob@example.org>.
```

There are two matching morphisms m_1 and m_2, such that $m_1(?\texttt{x}) = \texttt{\_:a}$, $m_1(\texttt{?name}) = \texttt{"Alice"}$ and $m_2(?\texttt{x}) = \texttt{\_:b}$, $m_2(\texttt{?name}) = \texttt{"Bob"}$.

The high-level calculus builds the transformation rule $2\,L \xrightarrow{2\,l} 2\,K \xleftarrow{2\,r} 2\,R$ where:

$2\,L$

```
?x1 foaf:name ?name1.
?x2 foaf:name ?name2.
```

$2\,K$

```
?x1 foaf:name ?name1 ;
    vcard:FN ?name1.
?x2 foaf:name ?name2 ;
    vcard:FN ?name2.
```

$2\,R$

```
?x1 vcard:FN ?name1.
?x2 vcard:FN ?name2.
```

The matching morphism $m : 2\,L \to G$ is such that $m(?\texttt{x1}) = \texttt{\_:a}$, $m(\texttt{?name1}) = \texttt{"Alice"}$ and $m(?\texttt{x2}) = \texttt{\_:b}$, $m(\texttt{?name2}) = \texttt{"Bob"}$. The PO-IM transformation produces first D then the query result H:

D

```
_:a foaf:name "Alice"; vcard:FN "Alice"; foaf:mbox <mailto:alice@example.org>.
_:b foaf:name "Bob"; vcard:FN "Bob"; foaf:mbox <mailto:bob@example.org>.
```

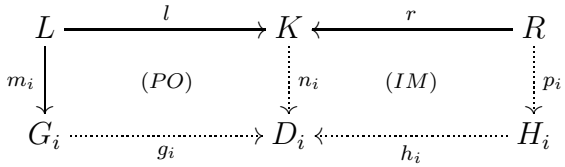
H

```
_:a vcard:FN "Alice". _:b vcard:FN "Bob".
```

4.2. **The low-level calculus.** Our low-level calculus is a two-step process: first one *local result* is obtained for each match, using a PO-IM transformation, then the local results are glued together in order to form the *low-level query result*. In order to simplify the notations we assume that $B(L) \cap B(G) = \emptyset$ and $B(R) \cap B(G) = \emptyset$; since blanks are not fixed by the morphisms in $\mathcal{Q}_I$ this can be done without loss of generality.

Definition 4.2. Let (L, R) be a basic construct query and let G be a data graph such that $B(L) \cap B(G) = \emptyset$ and $B(R) \cap B(G) = \emptyset$. Let $m_1, ..., m_k$ be the matching morphisms from L to G. For each $i = 1, ..., k$ let G_i be the image of m_i and let us still denote m_i the restriction $m_i : L \to G_i$. The *local result* H_i of (L, R) against G along m_i is the result of applying the PO-IM transformation functor $PoIm_{L,R}$ to the match $m_i : L \to G_i$. Let $IB(G) = I + B(G)$. The *low-level query result* H_{low} of (L, R) against G is the coproduct of the H_i's in the category $\mathcal{D}_{IB(G)}$ of data graphs with morphisms fixing all immutable attributes and the blanks that are in G.

(3)



Example 4.2. Let us run the same query against the same RDF graph as in Example 4.1, but now in the low-level calculus.

The matching morphism m_1 produces G_1, D_1 and H_1:

G_1

```
_:a foaf:name "Alice".
```

D_1

```
_:a foaf:name "Alice" ;
    vcard:FN "Alice".
```

H_1

```
_:a vcard:FN "Alice".
```

Similarly, the matching morphism m_2 produces G_2, D_2 and H_2:

G_2

```
_:b foaf:name "Bob".
```

D_2

```
_:b foaf:name "Bob"
    vcard:FN "Bob"
```

;

.

H_2

```
_:b vcard:FN "Bob".
```

Finally we get the query result H_{low}, which coincides with H from Example 4.1:

H_{low}

```
_:a vcard:FN "Alice". _:b vcard:FN "Bob".
```

Example 4.3. This example illustrates how local results are "merged" to compute the result in the low-level calculus. The SPARQL query is the following:

```
CONSTRUCT {?x rel:acquaintanceof ?z .}
WHERE     {?x foaf:knows ?y.
           ?y foaf:knows ?z. }
```

Its transformation rule is $L \stackrel{l}{\to} K \stackrel{r}{\leftarrow} R$ where:

L

```
?x foaf:knows ?y.
?y foaf:knows ?z.
```

K

```
?x foaf:knows ?y ;
   rel:acquaintanceof ?z.
?y foaf:knows ?z.
```

R

```
?x rel:acquaintanceof ?z.
```

This query is runned against the following graph G:

G

```
<http://example.org/Alice> foaf:knows <http://example.org/Bob> .
<http://example.org/Bob> foaf:knows _:c .
_:c foaf:knows <http://example.org/Alice>
```

There are three matching morphisms m_1, m_2, m_3, defined as follows:
$m_1(?x) = <http://example.org/Alice>$, $m_1(?y) = <http://example.org/Bob>$, $m_1(?z) = _:c$,
$m_2(?x) = _:c$, $m_2(?y) = <http://example.org/Alice>$, $m_2(?z) = <http://example.org/Bob>$,
$m_3(?x) = <http://example.org/Bob>$, $m_3(?y) = _:c$, $m_3(?z) = <http://example.org/Alice>$.

The three local results H_1, H_2, H_3 are respectively:

H_1

```
<http://example.org/Alice>  rel:acquaintanceof _:c.
```

H_2

```
 _:c rel:acquaintanceof <http://example.org/Bob>.
```

H_3

```
<http://example.org/Bob>  rel:acquaintanceof <http://example.org/Alice>.
```

The blank `_:c` in H_1 and H_2 is not duplicated in the coproduct because it comes from G. Thus the result is the data graph H_{low}:

H_{low}

```
<http://example.org/Alice>  rel:acquaintanceof _:c.
_:c rel:acquaintanceof <http://example.org/Bob>.
<http://example.org/Bob>  rel:acquaintanceof <http://example.org/Alice>.
```

4.3. **Running a basic construct query.** Now we prove that both calculi return the same result.

Theorem 4.1. *Let (L, R) be a basic construct query and let G be a data graph. The low-level query result of (L, R) against G is the same as the high-level query result of (L, R) against G. Assume (without loss of generality) that $B(G) \cap B(L) = \emptyset$ and $B(G) \cap B(R) = \emptyset$. Let $m_1, ..., m_k$ be the matching morphisms from L to G. Let H_0 be the data graph $H_0 = H_1 \cup ... \cup H_k$ where each H_i is obtained from R by replacing each variable x in R by $m_i(x)$ and each blank in R by a new blank. Then both H and H_{low} coincide with H_0.*

Proof. For the high-level calculus, we get as a straightforward consequence of Proposition 3.1 that $H = |p|^3(k\,R)$ where $k\,R = R_1 \cup ... \cup R_k$ and $|p| : |k\,R| \to A$ satisfies $|p|(x) = |m_i|(x)$ for each i and each $x \in V(R_i)$ and $|p|(x) = x$ otherwise. In particular $|p|(x) = x$ for each blank x in $k\,R$. Since each blank y in R gives rise to one new blank y_i in R_i for each i we get $H = H_0$. For the low-level calculus, Proposition 3.1 says that for each $i = 1, ..., k$ the local result is $H_i = |p_i|^3(R)$ where $|p_i| : |R| \to A$ satisfies $|p_i|(x) = |m_i|(x)$ when $x \in V(R)$ and $|p_i|(x) = x$ otherwise. Thus H_i is obtained from R by replacing each variable x by $m_i(x)$. It follows that each blank in H_i is either in G or in R. Then Proposition 2.1 proves that the query result is the union $H'_1 \cup ... \cup H'_k$ where H'_i is obtained from H_i by replacing each blank that is not in G by a new blank, so that $H_{low} = H_0$. □

Remark 4.1. Let us consider the description of the SPARQL result in [W3C13, 16.2]. *The CONSTRUCT query form returns a single RDF graph specified by a graph template. The result is an RDF graph formed by taking each query solution in the solution sequence, substituting for the variables in the graph template, and combining the triples into a single RDF graph by set union. If any such instantiation produces a triple containing an unbound variable or an illegal RDF construct, such as a literal in subject or predicate position, then that triple is not included in the output RDF graph. The graph template can contain triples with no variables (known as ground or explicit triples), and these also appear in the output RDF graph returned by the CONSTRUCT query form.* This description relies on the notion of "solution" and "solution sequence", or rather "solution set", that has been introduced previously in [W3C13, 2.2] for dealing with the SELECT queries. Here we consider the CONSTRUCT queries as the fundamental SPARQL queries, so we never refer to the solutions of any SELECT query. However Theorem 4.1 applied to a SPARQL CONSTRUCT query and an RDF graph G gives an explicit description of the result that fits with the description in [W3C13, 16.2], if one takes into account two facts: the assumption $V(R) \subseteq V(L)$ ensures that there cannot be any unbound variable (see Section 6); and the non-RDF triples in the low-level result must be dropped in order to ensure that illegal RDF constructs are not included.

5. Basic select queries

The CONSTRUCT query form of SPARQL returns a data graph whereas the SELECT query form returns a table, like the SELECT query form of SQL. Both in SQL and in SPARQL, it is well-known that such tables are not exactly relations in the mathematical sense: in mathematics a relation on X of arity k is a subset of X^k while the result of a SELECT query in SQL or SPARQL is a multiset of elements of X^k. In order to avoid ambiguities, a multiset of elements of X^k is called a *multirelation* on X of arity k.

A string x can be used for identifying an immutable attribute or a variable or a blank. In order to avoid confusion we write "$?x$" when x is a variable and "$\_ : x$" when x is a blank.

A SPARQL query like "SELECT $?s_1, ..., ?s_k$ WHERE $\{L\}$" is called *basic* when L is a basic graph pattern and $?s_1, ..., ?s_k$ are distinct variables. We generalise this situation by defining a *basic select query* as a pair (L, S) where L is a query graph and S is a set of variables. Then we associate to each basic select query (L, S) a basic construct query $(L, R(S))$ and we define the result of running the basic select query (L, S) against some data graph G from the data graph H result of running the basic construct query $(L, R(S))$ against G.

5.1. **Relational data graphs and query graphs.**

Definition 5.1. A *relational data graph* on a finite set $\{s_1, ..., s_k\}$ of immutable attributes is a data graph made of triples $(\_ : l_j, s_i, y_{i,j})$ where the $\_ : l_j$'s are pairwise distinct blanks and the $y_{i,j}$'s are in I, for $i \in \{1, ..., k\}$ and j is in some finite set J, up to isomorphism in $\mathcal{D}_I$.

The fact that a relational data graph is defined up to isomorphism in $\mathcal{D}_I$ means that the blanks $\_ : l_j$ can be modified, as long as they remain pairwise distinct.

Proposition 5.1. *Each relational data graph* $S = \{(\_ : l_j, s_i, y_{i,j})\}_{i \in \{1,...,k\}, j \in J}$ *determines a multirelation* $Rel(S) = \{(y_{1,j}, ..., y_{k,j})\}_{j \in J}$ *on* IB *of arity* k.

Definition 5.2. The *relational query graph* on a set of variables $S = \{?s_1, ..., ?s_k\}$ is the query graph $R(S)$ made of the triples $(\_ : r, s_i, ?s_i)_{i\in\{1,...,k\}}$ where $\_ : r$ is a blank and the s_i's are pairwise distinct immutable attributes, up to isomorphism in the category $\mathcal{Q}_{IV}$.

The fact that a relational query graph is defined up to isomorphism in $\mathcal{Q}_{IV}$ means that the blank $\_ : r$ can be modified.

Example 5.1. The following query graph is relational:

```
_:r nameX ?x ; nameY ?y.
```

and the following data graph is relational on $\{\texttt{nameX}, \texttt{nameY}\}$ with corresponding multirelation:

```
_:l1 nameX "Alice" ; nameY "Bob".
_:l2 nameX "Alice" ; nameY "Caty".
_:l3 nameX "Alice" ; nameY "Caty".
```

nameX	nameY
"Alice"	"Bob"
"Alice"	"Caty"
"Alice"	"Caty"

5.2. **Basic select queries.**

Definition 5.3. A *basic select query* is a pair (L, S) where L is a finite query graph and S is a finite set of variables such that each variable in S occurs in L. The *basic construct query associated to* a basic select query (L, S) is $(L, R(S))$ where $R(S)$ is the relational query graph on S.

Proposition 5.2. *Let (L, S) be a basic select query and let G be a data graph. Let $m_1, ..., m_k$ be the matching morphisms from L to G. The query result of $(L, R(S))$ against G is $H_1 \cup ... \cup H_k$ where each H_i is obtained from R by replacing each variable x by $m_i(x)$ and each blank by a new blank. It is a relational data graph.*

Proof. Since there is no blank in $R(S)$, this is Theorem 4.1 applied to the construct query $(L, R(S))$. □

Definition 5.4. The *query result* of (L, S) against G is the multirelation $Rel(H)$ on IB where H the query result of $(L, R(S))$ against G.

Theorem 5.1. *Let L be a basic graph pattern and $S = \{?s_1, ..., ?s_k\}$ a set of variables included in $V(L)$. Let G be an RDF graph. The result of running the SPARQL query "SELECT $?s_1, ..., ?s_k$ WHERE $\{L\}$" against G is the query result of (L, S) against G.*

Proof. The SPARQL result is described in [W3C13, 2.2] as follows: *Each solution gives one way in which the selected variables can be bound to RDF terms so that the query pattern matches the data. The result set gives all the possible solutions.* Proposition 5.2 applied to this situation provides the same result. Note that all triples in the query result are RDF triples. □

Example 5.2. Consider the following simple SPARQL SELECT query, that we run against the RDF graph G defined as follows:

```
SELECT ?nameX ?nameY
WHERE{ ?x foaf:knows ?y ;
          foaf:name ?nameX.
       ?y foaf:name ?nameY.}
```

G

```
_:a foaf:name "Alice" ;
    foaf:knows _:b ; foaf:knows _:c.
_:b foaf:name "Bob".
_:c foaf:name "Caty".
```

On the one hand, the SPARQL result is:

nameX	nameY
"Alice"	"Bob"
"Alice"	"Bob"
"Alice"	"Caty"

On the other hand, the associated basic construct query is:

```
CONSTRUCT { _:r <http://example.org/nameX> ?nameX.
            _:r <http://example.org/nameY> ?nameY.}
WHERE     { ?x foaf:knows ?y ; foaf:name ?nameX.
            ?y foaf:name ?nameY.}
```

with result H:

H

```
_:l1 <http://example.org/nameX> "Alice"; <http://example.org/nameY> "Bob".
_:l2 <http://example.org/nameX> "Alice"; <http://example.org/nameY> "Bob".
_:l3 <http://example.org/nameX> "Alice"; <http://example.org/nameY> "Caty".
```

Thus H is a relational data graph and its associated multirelation $Rel(H)$ is indeed the result of the SELECT query.

6. Conclusion

Relational algebra [Cod90] is the main mathematical foundation underlying SQL-like formalism for databases. However new frameworks such as RDF and SPARQL, where data structures are represented as graphs, are better adapted to the needs of big data and web applications. So, new mathematical foundations are needed to cope with this change in data encodings, see e.g., [AAB+17].

In this paper, we make the bet to base our work entirely on algebraic theories behind graphs and their transformations. We define suitable categories of data graphs and query graphs. Our definition of homomorphisms of query graphs make clear the differences between blank nodes and variables. Besides, we propose to encode CONSTRUCT and SELECT queries as graph rewrite rules, of the form $L \to L \cup R \leftarrow R$, and define their operational semantics following a novel algebraic approach called POIM. From the proposed semantics, one may easily notice that blanks in L play the same role as variables, so that blanks within L are not mandatory and can be replaced by variables, whereas blanks in R are used for creating new blanks in the result of a CONSTRUCT query. One of the benefits of using category theory is that coding of data graphs as sets of triples is not that important. The results we propose hold for all data models which define a category with enough colimits. For intance, one may expect to define data graph categories for the well-known Edge-labelled graphs or Property graphs [RWE13].

We are not aware of any existing work close to ours. In [AJK15], even if the authors use a categorical setting, their objectives and results depart from ours as they mainly encode every ontology as a category. The operational semantics we propose can clearly benefit from all proposals regarding efficient graph matching implementation, see e.g. [FLM+10]. As in [KRU15], we focus on the CONSTRUCT query form as the fundamental query form. In addition we propose a translation of the SELECT queries as CONSTRUCT queries compatible with their operational semantics.

In this paper we consider basic graphs and queries, which form a significant kernel of RDF and SPARQL. Future work includes the generalization of the present work to other

features of RDF and SPARQL in order to encompass general SPARQL queries. We also consider studying RDF Schema [W3C14b] and ontologies from this point of view.

References

[AAB+17] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017.

[AJK15] S. Aliyu, S.B. Junaidu, and A. F. Donfack Kana. A category theoretic model of RDF ontology. *International Journal of Web & Semantic Technology (IJWesT)*, 2015.

[CHHK06] Andrea Corradini, Tobias Heindel, Frank Hermann, and Barbara König. Sesqui-pushout rewriting. In *ICGT 2006*, volume 4178 of *LNCS*, pages 30–45. Springer, 2006.

[CMR+97] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation - part I: basic concepts and double pushout approach. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars*, pages 163–246. World Scientific, 1997.

[Cod90] Edgar F. Codd. *The relational Model for Database Management (Version 2 ed.)*. Addison-Wesley, 1990.

[FLM+10] Wenfei Fan, Jianzhong Li, Shuai Ma, Hongzhi Wang, and Yinghui Wu. Graph homomorphism revisited for graph matching. *PVLDB*, 3(1):1161–1172, 2010.

[KRU15] Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, pages 212–229, 2015.

[RWE13] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, Inc., 2013.

[W3C04] Ressource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, February 2004. .

[W3C13] SPARQL 1.1 Query Language. W3C Recommendation, march 2013. `https://www.w3.org/TR/sparql11-query/`.

[W3C14a] RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, February 2014. .

[W3C14b] RDF Schema 1.1. W3C Recommendation, February 2014. `www.w3.org/TR/2014/REC-rdf-schema-20140225/`.

[Wik] Limit (category theory). Wikipedia.