

An Algebraic Graph Transformation Approach for RDF and SPARQL

Dominique Duval

LJK, CNRS and Univ. Grenoble Alpes, Grenoble, France

Dominique.Duval@univ-grenoble-alpes.fr

Rachid Echahed

LIG, CNRS and Univ. Grenoble Alpes, Grenoble, France

Rachid.Echahed@univ-grenoble-alpes.fr

Frédéric Prost

LIG, CNRS and Univ. Grenoble Alpes, Grenoble, France

Frederic.Prost@univ-grenoble-alpes.fr

We consider the recommendations of the World Wide Web Consortium (W3C) about RDF framework and its 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 define 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 and verification techniques, to quote a few. Nowadays, graph databases are becoming a very influential technology in our society. Mastering programming languages involved in the encoding of such graph data is a necessity to elaborate robust modern data management systems. Relational algebra [6] was the main mathematical foundation underlying oldy SQL-like formalisms for databases. However, with the advent of new graph oriented formalisms such as the most recent recommendations of the World Wide Web Consortium (W3C) about the Resource Description Framework (RDF) [19] and the associated query language SPARQL [18], there is a clear need of an alternative to relational algebra which copes with this change in data encodings, see e.g., [4, 15, 12]. In this paper, we consider RDF and SPARQL languages and propose a new mathematical semantics of a kernel of these formalisms within algebraic graph transformations setting.

RDF graphs are the key data structure in RDF. In [19, Section 3], an RDF graph is defined as a set of RDF triples, where an RDF triple has the 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, let us consider a toy database, T_{data} , consisting of the following four triples $T_{data} = \{(Alice, knows, Bob), (Tom, knows, Dave), (Bob, knows, blank_1), (blank_1, knows, Alice)\}$. The two first triples say that Alice knows Bob and Tom knows Dave whereas the last two triples say that Bob knows someone, represented by the blank node $blank_1$, who knows Alice. Notice that a predicate in

an RDF triple cannot be a blank. For example, a triple such as $(Paul, blank_2, Henry)$ standing for “there is some relationship between Paul and Henry” is not allowed in RDF, but only in generalized RDF [19, Section 7]. Following the theoretical point of view we propose 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 [18, Section 2]. In this paper, we consider *query graphs* which generalize basic graph patterns by allowing blanks to be predicates. The SPARQL query processor searches for triples within a given RDF database which match the triple patterns in the given basic graph pattern, and returns a multiset of solutions or an RDF graph. Considering basic graph patterns, one may wonder what is the difference between variables and blank nodes. SPARQL specifications in [18, Section 4.1.4] suggest similarities between them, whereas the opposite is made in [18, Section 16.2]. In the formalization of SPARQL we propose, blank nodes and variables are clearly distinguished by their respective roles in the definition of morphisms.

In the SPARQL recommendation [18], the SELECT query form is described lengthily. This query form can be compared to the SELECT query form of SQL, which returns a multiset of solutions. In contrast, the CONSTRUCT query form returns an RDF graph. Let us consider again the previous toy database T_{data} and assume we formulate a CONSTRUCT query that constructs triples of the form $R = (x, acquaintedWith, z)$ every time there exists a third party y such that the following condition, which we call L , is satisfied : $(x, knows, y)$ and $(y, knows, z)$. Then, L and R are query graphs with variables x, y and z . They intuitively stand for the left-hand and right-hand sides of a rule representing the considered CONSTRUCT query. To perform such query, one should consider all matches, m , of the condition L against the database T_{data} and for each match m , create a new triple $(m(x), acquaintedWith, m(z))$. Starting from database, T_{data} , this process yields the following result

$$H = \{(Alice, acquaintedWith, blank_1), (blank_1, acquaintedWith, Bob), (Bob, acquaintedWith, Alice)\}$$

From graph transformation point of view, T_{data} is a host graph to be transformed by a rule whose left-hand and right-hand sides are respectively L and R . Notice that the resulting graph H does not contain the non-matched triple $(Tom, knows, Dave)$. This means that the considered transformations are not necessarily local. In addition, the graph H gathers all possible results triggered by the matches of L against the host graph T_{data} . This toy example is considered in Example 28.

Following our formalization, the CONSTRUCT query form, which is described very shortly in [18, Section 16.2], 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. This new POIM approach represents a CONSTRUCT query as a rule of the form $L \rightarrow K \leftarrow R$ where L, K and R are basic graph patterns, and a rewrite step is made of a pushout followed by an image factorization. The result of a CONSTRUCT query is the outcome of the transformation one obtains when running the above rule against an RDF database. It happens that such rules and rewrite techniques can be used also to encode the solutions computed by SELECT query forms. As said earlier, the involved graph transformations are not local in the sense that only query answers should be computed out of the graph database. All parts which are not matched are deleted. Classical graph transformation techniques such as Double Pushout [8] and Single Pushout [9] or even more RDF oriented transformations like MPOC-PO [5] are not best recommended in this case (cf. Section 6 for a comparison with related work).

The paper is organized as follows. Section 2 defines the objects and the morphisms of the categories of data graphs and query graphs. Section 3 introduces the POIM algebraic transformation. In Section 4, we define two different operational semantics for CONSTRUCT queries and prove their equivalence. We first define a *high-level calculus* as a mere application of the POIM transformation. Then we propose a *low-level calculus* which is defined by means of several applications of the POIM transformation followed by a “merging” process. Both calculi implement faithfully the SPARQL semantics for CONSTRUCT queries (Theorem 21). In Section 5, we show how the POIM transformation can be used to define a novel operational semantics of the SELECT queries. This semantics, which is faithful to SPARQL definitions (Theorem 40), is obtained by an original translation of each SELECT query into a CONSTRUCT query. Concluding remarks and related work are discussed in Section 6.

2 Graphs of Triples

The set of IRIs, denoted Iri , and the set of literals, denoted Lit , with its usual operations, are defined in [19]. Essentially, an IRI (Internationalized Resource Identifier) is an internet address and a literal denotes a value such as a string, a number or a date. The sets Iri and Lit are disjoint. In addition, let B be a countably infinite set, disjoint from Iri and Lit . The elements of B are called *blanks*. According to [19, Section 3.1], an *RDF graph* is a set of *RDF triples*, where 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 $Tr = (Iri \cup B) \times Iri \times (Iri \cup Lit \cup B)$, then an RDF triple is an element of Tr and an RDF graph is a subset of Tr . Let us also consider the following extension of RDF [19, Section 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. Let $I = Iri \cup Lit$, so that a generalized RDF triple is an element of $(I \cup B)^3$ and a generalized RDF graph is a subset of $(I \cup B)^3$.

Let V be a countably infinite set disjoint from Iri , Lit and B . The elements of V are called *variables*. According to [18, Section 2] a set of triple patterns is called a *basic graph pattern*, where triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable. Let $Tr_V = (Iri \cup B \cup V) \times (Iri \cup V) \times (Iri \cup Lit \cup B \cup V)$, 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 $(I \cup B \cup V)^3$, each basic graph pattern is a subset of $(I \cup B \cup V)^3$.

RDF graphs and basic graph patterns are generalized in Definition 2 as *data graphs* and *query graphs* respectively, are both relying on Definition 1.

Definition 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 . 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 $|T|^3$. Let T and T' be two graphs on A . A *morphism $a : T \rightarrow T'$* is a map such that there is a map $M : |T| \rightarrow |T'|$ such that a is the restriction of M^3 to T . Then M is uniquely determined by a and will be denoted by $|a|$. This yields the *category of graphs on A* , denoted $\mathbf{G}(A)$. We say that a morphism $a : T \rightarrow T'$ of graphs on A *fixes* a subset C of A if $|a|(x) = x$ for each x in $|T| \cap C$. For each subset C of A , the subcategory of $\mathbf{G}(A)$ made of the graphs on A with the morphisms fixing C is denoted $\mathbf{G}_C(A)$.

Thus, by mapping a to $|a|$ we get a one-to-one correspondence between the morphisms $a : T \rightarrow T'$ of graphs on A and the maps $M : |T| \rightarrow |T'|$ such that $M^3(T) \subseteq T'$.

An isomorphism (i.e., an invertible morphism) in $\mathbf{G}(A)$ is a morphism $a : T \rightarrow T'$ of graphs on A such that $|a| : |T| \rightarrow |T'|$ is a bijection and $a(T) = T'$. A morphism a fixing C is determined by the restriction of the map $|a|$ to $|T| \cap \bar{C}$, where $\bar{C} = A \setminus C$. An isomorphism a in $\mathbf{G}_C(A)$ is a morphism $a : T \rightarrow T'$ of graphs on A such that $|a|$ is the identity on $|T| \cap C$ and a bijection between $|T| \cap \bar{C}$ and $|T'| \cap \bar{C}$ and $a(T) = T'$. 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 .

Definition 2. Let I , B and V be three pairwise distinct countably infinite sets, called respectively the sets of *resource identifiers*, *blanks* and *variables*. Let $IB = I \cup B$, $IV = I \cup V$ and $IBV = I \cup B \cup V$. The *category of data graphs* is $\mathbf{D} = \mathbf{G}(IB)$ and for each subset C of IB the category of data graphs *fixing* C is the subcategory $\mathbf{D}_C = \mathbf{G}_C(IB)$ of \mathbf{D} . The *category of query graphs* is $\mathbf{Q} = \mathbf{G}(IBV)$ and for each subset C of IBV the category of query graphs *fixing* C is the subcategory $\mathbf{Q}_C = \mathbf{G}_C(IBV)$ of \mathbf{Q} .

Thus, since $I = Iri \cup Lit$, 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. Then the isomorphisms of RDF graphs, as defined in [19, Section 3.6.], are the isomorphisms in the category \mathbf{D}_I of data graphs fixing I : indeed, two data graphs G_1 and G_2 are isomorphic in \mathbf{D}_I if and only if they differ only by the names of their blanks. For each data graph T , let $|T|_I = |T| \cap I$ and $|T|_B = |T| \cap B$, so that $|T|$ is the disjoint union of $|T|_I$ and $|T|_B$. 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. For each query graph T , let $|T|_I = |T| \cap I$, $|T|_B = |T| \cap B$ and $|T|_V = |T| \cap V$, so that $|T|$ is the disjoint union of $|T|_I$, $|T|_B$ and $|T|_V$.

Morphisms of graphs can be used, for instance, for substituting the variables of a query graph (Definition 3) or for interpreting a data graph in a universe of discourse (Definition 4).

Definition 3. A *match* from a query graph L to a data graph G is a morphism of query graphs from L to G which fixes I . The set of matches from L to G is denoted $Match(L, G)$ and the set of all matches from L to any data graph is denoted $Match(L)$.

Thus, a match fixes each resource identifier and it maps a variable or a blank to a resource identifier or a blank.

The interpretations of an RDF graph are also kinds of morphisms, see Definition 4. Note that this will not be used later in this paper. We define an interpretation of a data graph G in a universe of discourse U by generalizing the definition of a morphism, according to [19, Section 1.2.]: *Any IRI or literal denotes something in the world (the “universe of discourse”). These things are called resources. The predicate itself is an IRI and denotes a property, that is, a resource that can be thought of as a binary relation.* Recall that the binary relations on a set R are the subsets of R^2 . It may happen that a binary relation on R is itself an element of R , this is important for understanding Definition 4 and the semantics of RDF in general.

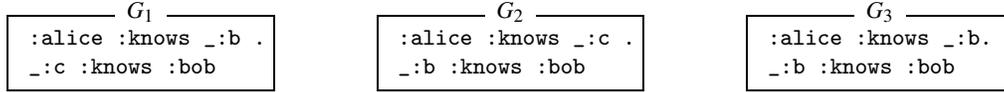
Definition 4. Given a set R and a subset P of R^2 made of binary relations on R , let U be the set of triples (s, p, o) in R^3 such that $p \in P$ and $(s, o) \in p$. The *universe of discourse* with R as set of *resources* and P as set of *properties* is the graph U on R . Given a universe of discourse U on a set R and a map $M_I : I \rightarrow R$, an *interpretation* of a data graph G is a map $i : G \rightarrow U$ such that $i = M^3$ for a map $M : |G| \rightarrow |U|$ which extends M_I .

In this paper, we consider categories \mathbf{D}_C and \mathbf{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 resource identifiers 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

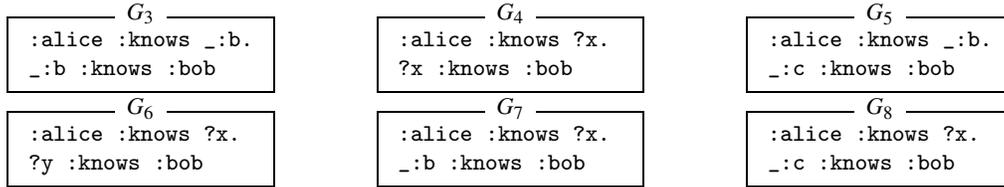
blank node identifiers in [19, Section 3.4] distinguishes two kinds of syntaxes for RDF: an abstract syntax where blank nodes do not have identifiers and concrete syntaxes where blank nodes have identifiers. In our approach a blank *is* an attribute, which corresponds to a concrete syntax, and the abstract syntax is obtained by considering data graphs as objects of the category \mathbf{D}_I up to isomorphism, so that any blank node can be changed for a fresh blank node if needed.

Notation 5. In the examples variables are denoted as $?x, ?y, \dots$ and blanks as $_:b, _:c, \dots$. The IRIs are written in an abbreviated way as $:alice$ (the address of Alice’s web page), $:knows$ (the address where the “knows” relation is described), \dots . Each triple (s, p, o) is written as $s\ p\ o$ and a dot is used for separating triples inside an RDF graph or a basic graph pattern.

Example 6. Consider three RDF graphs G_1, G_2 and G_3 as follows. They are pairwise distinct, thus pairwise non-isomorphic in \mathbf{D}_{IB} . Graphs G_1 and G_2 are isomorphic in \mathbf{D}_I . In the RDF semantics the name of blanks does not matter, so that both G_1 and G_2 mean that “Alice knows someone” and “someone knows Bob”. Graph G_3 is not isomorphic to G_1 (thus nor to G_2) in \mathbf{D}_I , it means more precisely that “Alice knows someone who knows Bob”.



Now consider basic graph patterns G_3 to G_8 (each RDF graph can be seen as a basic graph pattern without variables, like G_3 and G_5). They are pairwise non-isomorphic in \mathbf{Q}_{IBV} because they are pairwise distinct. In \mathbf{Q}_{IV} only G_7 and G_8 are isomorphic. In \mathbf{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.



Assumption 7. From now on A is a countably infinite set, C a subset of A , $\bar{C} = A \setminus C$ the complement of C in A , and we assume that both C and \bar{C} are countably infinite.

Remark 8. Since \bar{C} is countably infinite, when dealing with a finite number of finite graphs on A it is always possible to find a *fresh attribute outside C* , i.e., an element of \bar{C} that is not an attribute of any of the given graphs. We will use repeatedly the following consequence of this fact:

Given a graph T on A , if any attribute of T in \bar{C} is replaced by any fresh attribute outside C the result is a graph T' on A that is isomorphic to T in $\mathbf{G}_C(A)$. Such a T' exists when T is finite.

Now let us focus on some kinds of colimits of graphs on A : coproducts in Proposition 9 and pushouts in Proposition 10. Recall that colimits in any category are defined up to isomorphism in this category. Remember that a morphism $a : T \rightarrow T'$ in $\mathbf{G}(A)$ and a map $M : |T| \rightarrow |T'|$ are such that $M(x) = |a|(x)$ for each attribute $x \in |T|$ if and only if $M^3(t) = a(t)$ for each triple $t \in T$.

Proposition 9. *Given graphs T_1, \dots, T_k on A such that $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$, the union $T_1 \cup \dots \cup T_k$ is a coproduct of T_1, \dots, T_k in $\mathbf{G}_C(A)$. Given any finite graphs T_1, \dots, T_k on A there are graphs T'_1, \dots, T'_k on A such that T'_i is isomorphic to T_i in $\mathbf{G}_C(A)$ for each i and $|T'_i| \cap |T'_j| \subseteq C$ for each $i \neq j$, and then the union $T'_1 \cup \dots \cup T'_k$ is a coproduct of T_1, \dots, T_k in $\mathbf{G}_C(A)$.*

Proof. First, assume that $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$. Consider morphisms $a_i : T_i \rightarrow T$ in $\mathbf{G}_C(A)$ for $i = 1, \dots, k$ and the maps $|a_i| : |T_i| \rightarrow |T|$. Note that $|T_1 \cup \dots \cup T_k| = |T_1| \cup \dots \cup |T_k|$ and that $|T_1| \cup \dots \cup |T_k|$

is the disjoint union of the sets $|T_i| \setminus C$ for $i = 1, \dots, k$ and $(|T_1| \cup \dots \cup |T_k|) \cap C$, because of the assumption $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$. Thus we can define a map $M : |T_1| \cup \dots \cup |T_k| \rightarrow |T|$ by: $M(x) = |a_i|(x)$ for each i and each $x \in |T_i| \setminus C$ and $M(x) = x$ for each $x \in (|T_1| \cup \dots \cup |T_k|) \cap C$. Then M coincides with $|a_i|$ on $|T_i|$ for each i . Thus for each $t \in T_i$ we have $M^3(t) = a_i(t)$, which proves that the image of $T_1 \cup \dots \cup T_k$ by M^3 is in T and that the restriction of M^3 defines a morphism $a : T_1 \cup \dots \cup T_k \rightarrow T$ in $\mathbf{G}_C(A)$ which coincides with a_i on T_i for each i . Unicity is clear. Now, the last statement, about any finite graphs T_1, \dots, T_k on A , is a consequence of Remark 8. \square

Proposition 10. *Let $l : L \rightarrow K$ and $m : L \rightarrow G$ be morphisms of graphs on A such that L and K are finite, 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 $\mathbf{G}_C(A)$, by Remark 8). Let $N : |K| \rightarrow |G| \cup |K \setminus L|$ be such that $N(x) = |m|(x)$ for $x \in |L|$ and $N(x) = x$ otherwise. Let $D = G \cup N^3(K)$, let $n : K \rightarrow D$ be the restriction of N^3 and $g : G \rightarrow D$ the inclusion. Then $|D| = |G| \cup |K \setminus L|$ and the square (l, m, n, g) is a pushout square in $\mathbf{G}_C(A)$.*

This means that D is a kind of “union of G and K over L ”, however it is *not* the case that D is the union of G and $K \setminus L$, in general. It is always the case that $D = G \cup D_2$ where $D_2 = N^3(K \setminus L)$ but in general N^3 is not the identity on $K \setminus L$ and moreover G and D_2 are not disjoint.

Proof. From $D = G \cup N^3(K)$ we get $|D| = |G| \cup |N^3(K)|$, and since $|N^3(K)| = N(|K|) = N(|L| \cup |K \setminus L|) = N(|L|) \cup N(|K \setminus L|) = |m|(|L|) \cup |K \setminus L|$ with $|m|(|L|) \subseteq |G|$ we get $|D| = |G| \cup |K \setminus L|$. The definition of n implies that $g \circ m = n \circ l$. Now let $a : G \rightarrow T$ and $b : K \rightarrow T$ be any morphisms in $\mathbf{G}_C(A)$ such that $a \circ m = b \circ l$. First, let us focus on attributes. We have $|g| \circ |m| = |n| \circ |l|$ and $|a| \circ |m| = |b| \circ |l|$. Since $|G| \cap |K \setminus L| \subseteq C$ we have $|a|(x) = |b|(x) = x$ for each $x \in |G| \cap |K \setminus L|$. Since $|D| = |G| \cup |K \setminus L|$ there is a unique map $F : |D| \rightarrow |T|$ such that $F(x) = |a|(x)$ for $x \in |G|$ and $F(x) = |b|(x)$ for $x \in |K \setminus L|$. Thus on the one hand $F(|g|(x)) = F(x) = |a|(x)$ for each $x \in |G|$, so that $F \circ |g| = |a|$. And on the other hand for each $x \in |K|$, if $x \in |L|$ then $F(|n|(x)) = F(|m|(x)) = |a|(|m|(x)) = |b|(|l|(x)) = |b|(x)$, otherwise $F(|n|(x)) = F(x) = |b|(x)$, so that $F \circ |n| = |b|$. Second, let us consider triples. Since $D = G \cup N^3(K)$ and $F^3(G) = a(G)$ and $F^3(N^3(K)) = F^3(n(K)) = b(K)$ we get $F^3(D) \subseteq T$, which means that there is a morphism $f : D \rightarrow T$ of graphs on A such that $|f| = F$, $f \circ g = a$ and $f \circ n = b$. Unicity is clear. \square

3 The POIM Transformation

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 [18, Section 4.1.4] holds for L , whereas blank nodes in R give rise to fresh blank nodes in the result of the query as in Examples 18 and 23. 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 fresh blank.

We generalize this situation in Definition 11 by allowing any data graphs for L and R up to isomorphism in \mathbf{Q}_{IV} : the resource identifiers and the variables in L and R are fixed but each blank can be replaced by a fresh blank. Thus, without loss of generality, we can assume that $|L|_B \cap |R|_B = \emptyset$. Under this assumption, the set of triples $K = L \cup R$ with the inclusions of L and R in K is a coproduct of L and R in the category \mathbf{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 with respect to SPARQL queries is discussed in Section 4.1. Note that this assumption $|R|_V \subseteq |L|_V$ is equivalent to $|K|_V = |L|_V$.

Definition 11. A *basic construct query* is a pair of finite query graphs (L, R) such that $|L|_B \cap |R|_B = \emptyset$ and $|R|_V \subseteq |L|_V$, up to isomorphism in the category \mathbf{Q}_I . The *transformation rule* of a basic construct query (L, R) is the cospan $P_{L,R} = (L \xrightarrow{l} K \xleftarrow{r} R)$ where $K = L \cup R$ and l and r are the inclusions. Its *left-hand side* is L and its *right-hand side* is R .

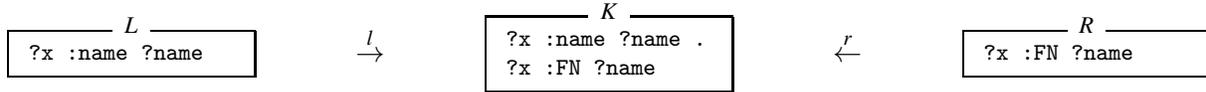
$$P_{L,R} = L \xrightarrow[\subseteq]{l} K = L \cup R \xleftarrow[\supseteq]{r} R$$

Example 12. Consider the following SPARQL CONSTRUCT query, based on the examples given in the “CONSTRUCT” section (Section 16.2) of [18]. This query builds a triple $?x :FN ?name$ for each triple $?x :name ?name$:

Query

CONSTRUCT { ?x :FN ?name } WHERE { ?x :name ?name }

In the corresponding transformation rule $L \xrightarrow{l} K \xleftarrow{r} R$ there are no blanks in L nor in R , thus $K = L \cup R$ and l and r are the inclusions.

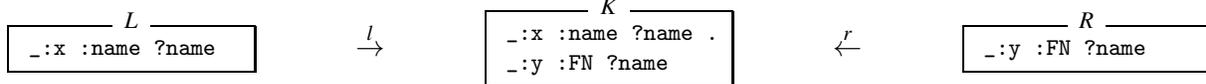


Example 13. Now the SPARQL CONSTRUCT query from Example 12 is modified by replacing both occurrences of the variable $?x$ by the blank node $_:x$:

Query

CONSTRUCT { _:x :FN ?name } WHERE { _:x :name ?name }

In the corresponding transformation rule one blank has been modified so as to ensure that $|L|_B \cap |R|_B$ is empty, so that it is still the case that $K = L \cup R$.



When a basic SPARQL query “CONSTRUCT $\{R\}$ WHERE $\{L\}$ ” is run against an RDF graph G , and when there is precisely one match of L into G , the result of the query is an RDF graph H obtained by substituting 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-step process involving an intermediate match of K in an RDF graph D . This claim will be proved, more generally, in Section 4. The definition of this two-step process relies on an algebraic construction that we call the *POIM transformation*: PO for *pushout* and IM for *image* (Definition 14). The POIM transformation is related to a large family of algebraic graph transformations based on pushouts, like the SPO (Simple Pushout) [9], DPO (Double Pushout) [8] or SqPO (Sesqui-Pushout) [7]. In the POIM transformation, the PO step creates fresh blank nodes and instantiates the variables of K , while the IM step deletes everything that is not in the image of R , as explained now.

Given a basic construct query (L, R) and its transformation rule $L \xrightarrow{l} K \xleftarrow{r} R$, the POIM transformation is defined as a map from the matches of L to the matches of R , in two steps: first from the matches of L to the matches of K , then from the matches of K to the matches of R . Given an inclusion $l : L \rightarrow K$ in \mathbf{Q}_I , the *cobase change along l* is the map $l_* : Match(L) \rightarrow Match(K)$ that maps each $m : L \rightarrow G$ to $l_*(m) : K \rightarrow D$ defined from the pushout of l and m in \mathbf{Q}_I , as described in Proposition 10. Note that D is a data graph because of the assumption $|K|_V = |L|_V$. Given an inclusion $r : R \rightarrow K$ in \mathbf{Q}_I , the *image factorization along r* is the map $r^+ : Match(K) \rightarrow Match(R)$ that maps each $n : K \rightarrow D$ to $r^+(n) : R \rightarrow H$ where H is the image of R in D and $r^+(n)$ is the restriction of n and $h : H \rightarrow D$ is the inclusion. This leads to Definition 14 and Proposition 15.

Definition 14. Let (L, R) be a basic construct query and $L \xrightarrow{l} K \xleftarrow{r} R$ its transformation rule. The *POIM transformation map* of (L, R) is the map

$$PoIm_{L,R} = r^+ \circ l_* : Match(L) \rightarrow Match(R)$$

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

$$\begin{array}{ccccc}
 L & \xrightarrow{l} & K & \xleftarrow{r} & R & (1) \\
 \downarrow m & & \downarrow l_*(m)=n & & \downarrow r^+(n)=p=PoIm_{L,R}(m) \\
 & (PO) & & (IM) & \\
 G & \xrightarrow{g} & D & \xleftarrow{h} & H
 \end{array}$$

Proposition 15 says that H is obtained from R by instantiating all variables as in L and by renaming blanks in an arbitrary way, as long as this renaming is one-to-one. Note that only the image of the match is transformed and the remaining parts of the RDF graph are deleted.

Proposition 15. Let (L, R) be a basic construct query and $m : L \rightarrow G$ a match. Let $P : |R| \rightarrow IB$ be defined by $P(x) = |m|(x)$ for $x \in |R|_V$ and $P(x) = x$ otherwise. Then, up to isomorphism in \mathbf{Q}_I , the result of applying $PoIm_{L,R}$ to m is $p : R \rightarrow H$ where $H = P^3(R)$ and p is the restriction of P^3 .

Proof. We use the notations of Diagram (1). Up to isomorphism in \mathbf{Q}_I we can assume that all blanks in L or in R are distinct from the blanks in G . Then $|G| \cap |K| \subseteq I$, so that by Proposition 10 the data graph D is $D = G \cup n(K)$ where n is such that $|n|(x) = |m|(x)$ for $x \in |L|$ and $|n|(x) = x$ otherwise. It follows that the restriction of n to R is such that $|n|(x) = |m|(x)$ for $x \in |L| \cap |R|$ and $|n|(x) = x$ otherwise. Note that $|L| \cap |R|$ is the disjoint union of $|L|_I \cap |R|_I$, that is fixed by all morphisms in \mathbf{Q}_I , and $|L|_V \cap |R|_V$, with $|L|_V \cap |R|_V = |R|_V$ since $|R|_V \subseteq |L|_V$. Thus the restriction of n to R is such that $|n|(x) = |m|(x)$ for $x \in |R|_V$ and $|n|(x) = x$ otherwise. The result follows. \square

Remark 16. Each set $Match(X)$ can be seen as a coslice category, then the maps r^+ and l_* can be seen as functors: this could be useful when extending this paper to additional features of SPARQL.

Example 17. Consider the SPARQL CONSTRUCT query from Example 12:

```

Query
CONSTRUCT { ?x :FN ?name } WHERE { ?x :name ?name }

```

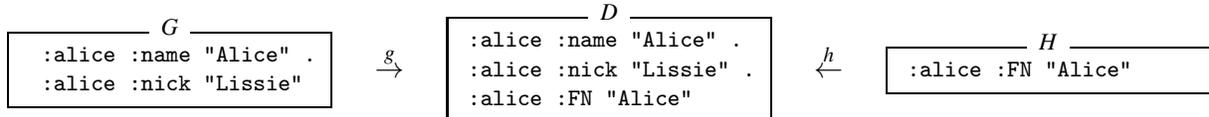
and let us run this query against the RDF graph G :

```

G
:alice :name "Alice" . :alice :nick "Lissie"

```

There is a single match m , it is such that $m(?x) = :alice$ and $m(?name) = "Alice"$. The POIM transformation produces successively the following data graphs D and H , where H is the query result:



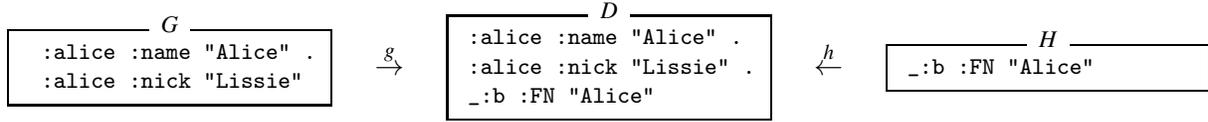
Example 18. Now consider the SPARQL CONSTRUCT query from Example 13:

```

Query
CONSTRUCT { _:x :FN ?name } WHERE { _:x :name ?name }

```

Let us run this query against the RDF graph G from Example 17. There is a single match m , it is such that $m(_:x) = :alice$ and $m(?name) = "Alice"$. The POIM transformation produces successively the following data graphs D and H , where H is the query result:



4 Running Basic Construct Queries

This Section begins with our definition of the *query result* of applying a basic construct query (Definition 11) to a data graph (Definition 2). Theorem 21 in Section 4.1 proves that our query result coincides (up to renaming blanks) with the answer returned by SPARQL when L and R are basic graph patterns and G is an RDF graph. In Section 3, we defined the POIM transformation for running a basic construct query (L,R) against a data graph G , when there is exactly one match from L to G . Now, we define two different calculi for running a basic construct query against a data graph G without any assumption on the number of matches. The *high-level calculus* (Definition 24) is one “large” application of the POIM transformation. The *low-level calculus* (Definition 26) consists of several “small” applications of the POIM transformation followed by a “merging” process. The construction of the high-level calculus is simpler, while the low-level calculus better fits with the description of the running process in SPARQL. Proposition 25 in Section 4.2 and Proposition 29 in Section 4.3 prove that both calculi compute the query result.

Definition 19. Let (L,R) be a basic construct query and G a data graph. Assume (without loss of generality) that $|G|_B \cap |L|_B = \emptyset$ and $|G|_B \cap |R|_B = \emptyset$. Let m_1, \dots, m_k be the matches from L to G . For each $i = 1, \dots, k$ let H_i be the data graph obtained from R by replacing each variable x in R by $m_i(x)$ and each blank in R by a fresh blank (which means: a fresh blank for each blank in R and each i in $\{1, \dots, k\}$). The *query result* of applying the basic construct query (L,R) to the data graph G is the data graph $H = H_1 \cup \dots \cup H_k$.

4.1 Running Basic Construct Queries in SPARQL

The *answer* of a SPARQL CONSTRUCT query over an RDF graph is defined in [14, Section 5], based on the seminal paper [15]. Note that in [14] literals are allowed as subjects or predicates in RDF graphs, however for our purpose this does not matter, so that we stick to the definition of an RDF graph from [19], as reminded at the beginning of Section 2. Thus, as $I = Iri \cup Lit$, a data graph G is an RDF graph if and only if each triple (s, p, o) in G is an RDF triple, which means that $s \in Iri \cup B$ and $p \in Iri$. Note that for each subset T of $(IBV)^3$ and each subset X of $|T|$, each map $f : X \rightarrow IBV$ gives rise to a map $f' : |T| \rightarrow IBV$ such that $f'(x) = f(x)$ when $x \in X$ and $f'(x) = x$ otherwise, then $f' : |T| \rightarrow IBV$ gives rise to $f'' : T \rightarrow (IBV)^3$ which is the restriction of $(f')^3$ to T . There will not be any ambiguity in denoting f not only the given f but also its extensions f' and f'' . For simplicity we consider only the SPARQL queries “CONSTRUCT $\{R\}$ WHERE $\{L\}$ ” such that each variable in R occurs in L . Indeed, variables outside $|L|_V$ cannot be instantiated in the result, and according to [18, Section 16.2], if a triple contains an unbound variable, then that triple is not included in the output RDF graph. Thus, triples involving a variable in $|R|_V \setminus |L|_V$, if any, can be dropped. It is assumed in [14] that there is no blank in L . Indeed, since blank nodes in graph patterns act as variables, each blank in L can be replaced by a fresh variable.

Definition 20 ([14]). A *solution mapping* (or simply a *mapping*) from a basic graph pattern L to an RDF graph G is a map $\mu : |L|_V \rightarrow IB$ such that $\mu(L) \subseteq G$. When L and R are basic graph patterns such that $|R|_V \subseteq |L|_V$, the *answer* of the SPARQL query “CONSTRUCT $\{R\}$ WHERE $\{L\}$ ” over an RDF graph G is the set of all RDF triples $\mu(f_\mu(t))$ for all triples $t \in R$ and all mappings μ from L to G , where for each μ a map $f_\mu : |R|_B \rightarrow B$ is chosen in such a way that the subsets $f_\mu(|R|_B)$ of B are pairwise distinct and all of them are distinct from $|G|_B$.

Theorem 21. Let L and R be basic graph patterns with $|L|_B = \emptyset$ and $|R|_V \subseteq |L|_V$. Then (L, R) is a basic construct query and the set of RDF triples in the query result of applying (L, R) to an RDF graph G is isomorphic in \mathbf{D}_I to the answer of the SPARQL query “CONSTRUCT $\{R\}$ WHERE $\{L\}$ ” over G .

Proof. Clearly (L, R) is a basic construct query and $|G|_B \cap |L|_B = \emptyset$. We can assume without loss of generality that $|G|_B \cap |R|_B = \emptyset$. The query result H of applying (L, R) to G is given by Definition 19, based on the set of matches from L to G . The Theorem now follows from the fact that the maps μ (or μ'') on triples which are associated to the mappings μ are precisely the matches from L to G . \square

Example 22. Consider the SPARQL query from Examples 12 and 17:

```
Query
CONSTRUCT { ?x :FN ?name } WHERE { ?x :name ?name }
```

and let us run this query against the RDF graph G :

```
G
:alice :name "Alice" . :alice :nick "Lissie" .
:bob :name "Bob" . :bob :nick "Bobby"
```

There are two matches and we get the RDF graphs H_1, H_2 and the result H :

```
H1
:alice :FN "Alice"
```

```
H2
:bob :FN "Bob"
```

```
H
:alice :FN "Alice" .
:bob :FN "Bob"
```

Example 23. Consider the SPARQL CONSTRUCT query:

```
Query
CONSTRUCT { _:c :FN ?name } WHERE { ?x :name ?name }
```

Note that this query always returns the same result as the query from Examples 13 and 18. Let us run this query against the RDF graph G from Example 22. There are two matches and we get the RDF graphs H_1, H_2 and the result H :

```
H1
_:c :FN "Alice"
```

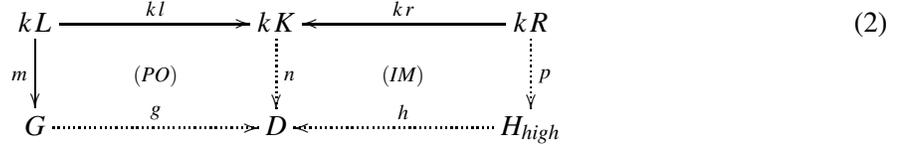
```
H2
_:c :FN "Bob"
```

```
H
_:c1 :FN "Alice" .
_:c2 :FN "Bob"
```

4.2 The High-level Calculus

Let k be a natural number. According to Proposition 9, for each query graph T the query graph kT , coproduct of k copies of T in \mathbf{Q}_I , can be built (up to isomorphism) as follows: for each $i \in \{1, \dots, k\}$ 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 kT is the union $T_1 \cup \dots \cup T_k$. 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 \mathbf{Q}_I , that gives rise to the cospan $kP_{L,R} = (kL \xrightarrow{kl} kK \xleftarrow{kr} kR)$. Since l and r are inclusions, this renaming can be done simultaneously in the copies of L, K and R , so that $kK = kL \cup kR$ and kl and kr are the inclusions. Thus, (kL, kR) is a basic construct query and $P_{kL,kR} = kP_{L,R}$ is its corresponding transformation rule.

Definition 24. Let (L, R) be a basic construct query and G a data graph. Let m_1, \dots, m_k be the matches from L to G . Consider the basic construct query (kL, kR) . Let m be the match from kL to G that coincides with m_i on the i -th component of kL . The *high-level query result* of (L, R) against G is the result H_{high} of applying the POIM transformation map $PoIm_{kL, kR}$ to the match $m : kL \rightarrow G$, as in Diagram (2).



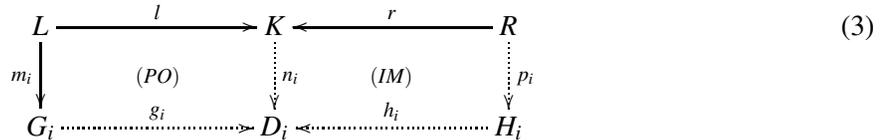
Proposition 25. Let (L, R) be a basic construct query and G a data graph. The high-level query result of (L, R) against G is isomorphic, in the category \mathbf{D}_I , to the query result of (L, R) against G .

Proof. This is a consequence of the description of the result of a POIM transformation from Proposition 15. □

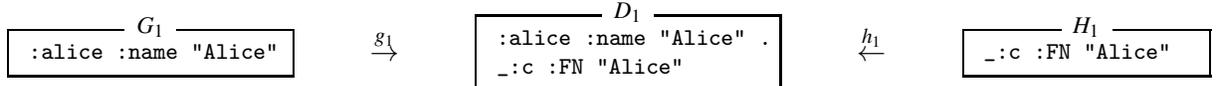
4.3 The Low-level Calculus

The low-level calculus is a two-step process: first one *local result* is obtained for each match, using a POIM transformation, then the local results are glued together in order to form the *low-level query result*.

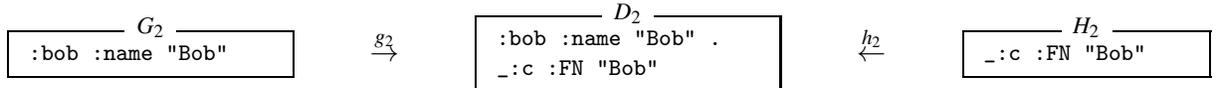
Definition 26. Let (L, R) be a basic construct query and G a data graph. Let m_1, \dots, m_k be the matches from L to G . For each $i = 1, \dots, k$ let G_i be the image of m_i and let us still denote m_i the restriction $m_i : L \rightarrow G_i$. The *local result* H_i of (L, R) against G along m_i is the result of applying the POIM transformation map $PoIm_{L, R}$ to the match $m_i : L \rightarrow G_i$. Let $IB(G) = I \cup |G|_B$. The *low-level query result* H_{low} of (L, R) against G is the coproduct of the H_i 's in the category $\mathbf{D}_{IB(G)}$ of data graphs with morphisms fixing all resource identifiers and the blanks that are in G .



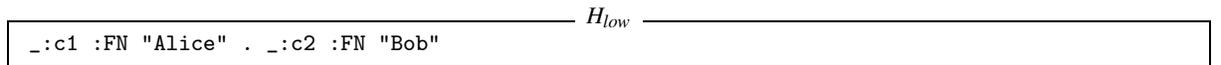
Example 27. Let us apply the low-level calculus to Example 23. The match m_1 produces $G_1 \rightarrow D_1 \leftarrow H_1$:



and similarly the match m_2 produces $G_2 \rightarrow D_2 \leftarrow H_2$:



Finally the query result H_{low} , which is the coproduct of H_1 and H_2 in category $\mathbf{D}_{IB(G)}$, is isomorphic to H from Example 23.



Example 28. This example illustrates how local results are “merged” to compute the result in the low-level calculus. The SPARQL query is the following:

Query

```
CONSTRUCT { ?x :acquaintedWith ?z } WHERE { ?x :knows ?y . ?y :knows ?z }
```

Its corresponding transformation rule is:

$$\begin{array}{ccc}
 \begin{array}{|c|} \hline L \\ \hline ?x :knows ?y . \\ ?y :knows ?z \\ \hline \end{array} & \xrightarrow{l} & \begin{array}{|c|} \hline K \\ \hline ?x :knows ?y . \\ ?y :knows ?z . \\ ?x :acquaintedWith ?z \\ \hline \end{array} & \xleftarrow{r} & \begin{array}{|c|} \hline R \\ \hline ?x :acquaintedWith ?z \\ \hline \end{array}
 \end{array}$$

This query is applied to the following graph G :

G

```
:alice :knows :bob . :bob :knows _:c . _:c :knows :alice . :cathy :knows :david
```

There are three matches m_1, m_2, m_3 , thus three local results H_1, H_2, H_3 :

$$\begin{array}{ccc}
 \begin{array}{|c|} \hline H_1 \\ \hline :alice :acquaintedWith _:c \\ \hline \end{array} & & \begin{array}{|c|} \hline H_2 \\ \hline _:c :acquaintedWith :bob \\ \hline \end{array} & & \begin{array}{|c|} \hline H_3 \\ \hline :bob :acquaintedWith :alice \\ \hline \end{array}
 \end{array}$$

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

H_{low}

```
:alice :acquaintedWith _:c . _:c :acquaintedWith :bob . :bob :acquaintedWith :alice
```

Proposition 29. *Let (L, R) be a basic construct query and G a data graph. The low-level query result of (L, R) against G is isomorphic, in the category \mathbf{D}_I , to the query result of (L, R) against G .*

Proof. This is a consequence of the description of the result of a POIM transformation from Proposition 15 and the description of coproducts in \mathbf{D}_I from Proposition 9. \square

5 Running 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_1, \dots, X_n is a subset of the cartesian product $X_1 \times \dots \times X_n$, while the result of a SELECT query in SQL or SPARQL is a multiset of elements of $X_1 \times \dots \times X_n$. In order to avoid ambiguities, such a multiset is called a *multirelation* on X_1, \dots, X_n . When all X_i 's are the same set X it is called a multirelation of arity n on X .

A SPARQL query such as “SELECT $?s_1, \dots, ?s_n$ WHERE $\{L\}$ ” is called *basic* when L is a basic graph pattern and $?s_1, \dots, ?s_n$ are distinct variables. We generalize this situation by defining a *basic select query* as a pair (L, S) where L is a finite query graph and S is a finite set of variables. Then we associate to each basic select query (L, S) a basic construct query $(L, Gr(S))$. Finally we define the result of running the basic select query (L, S) against a data graph G from the data graph H result of running the basic construct query $(L, Gr(S))$ against G . This process is first described on an example.

Example 30. Consider the following SPARQL SELECT query:

SELECT Query

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

We associate to this SELECT query the following CONSTRUCT query:

CONSTRUCT Query

```
CONSTRUCT { _:r nameX ?nameX . _:r nameY ?nameY }
WHERE { ?x :knows ?y . ?x :name ?nameX . ?y :name ?nameY }
```

Let us run this CONSTRUCT query against the RDF graph G :

G

```
_:alice :name "Alice" . _:bob :name "Bob" . _:bobby :name "Bob" . _:cathy :name "Cathy" .
_:alice :knows _:bob . _:alice :knows _:bobby . _:alice :knows _:cathy
```

The result is the RDF graph H :

H

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

From the data graph H we get the following table, by considering each blank $_:l_i$ in H as the identifier of a line in the table. Note that the set of triples in H becomes a multiset of lines in the table. This table is indeed the answer of the SPARQL SELECT query over G .

nameX	nameY
"Alice"	"Bob"
"Alice"	"Bob"
"Alice"	"Cathy"

In order to generalize Example 30 we have to define a transformation from each SELECT query to a CONSTRUCT query and a transformation from the result of this CONSTRUCT query to the result of the given SELECT query. For this purpose, we first define *relational* data graphs (Definition 31) and *relational* query graphs (Definition 34).

Definition 31. A *relational data graph* on a finite set $\{s_1, \dots, s_n\}$ of resource identifiers is a data graph made of triples $(_:l_i, s_j, y_{i,j})$ where the $_:l_i$'s are pairwise distinct blanks and the $y_{i,j}$'s are in IB , for $j \in \{1, \dots, n\}$ and i in some finite set $\{1, \dots, k\}$.

Proposition 32. Each relational data graph $S = \{(-:l_i, s_j, y_{i,j})\}_{i \in \{1 \dots k\}, j \in \{1 \dots n\}}$ determines a multirelation $Rel(S) = \{(y_{i,1}, \dots, y_{i,n})\}_{i \in \{1 \dots k\}}$ of arity n on IB .

Proof. This result is clear from the definitions of relational data graphs and multirelations. \square

Example 33. In Example 30 the graph H is a relational data graph on the set $\{\text{nameX}, \text{nameY}\}$ and the table result of the SELECT query is its corresponding multirelation.

Assume that each variable in SPARQL is written as “? s ” for some string s .

Definition 34. The *relational query graph* on a finite set of variables $S = \{?s_1, \dots, ?s_n\}$ is the query graph $Gr(S)$ made of the triples $(_:r, s_j, ?s_j)$ where $j \in \{1, \dots, n\}$ and $_:r$ is a blank. Note that $Gr(S)$ is uniquely determined by S up to isomorphism in \mathbf{Q}_{IV} .

Example 35. Here is the relational query graph on $\{?nameX, ?nameY\}$:

$_:r$ nameX ?nameX . $_:r$ nameY ?nameY

Below, we show how a basic select query can be encoded as a basic construct query (Definition 36) and we prove that the result of the given select query is easily recovered from the result of its associated construct query (Theorem 40).

Definition 36. 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 $(L, Gr(S))$ where $Gr(S)$ is the relational query graph on S .

Proposition 37. *Let (L, S) be a basic select query and G a data graph. The query result of $(L, Gr(S))$ against G is a relational data graph H . More precisely, let $S = \{?s_1, \dots, ?s_n\}$ and let m_1, \dots, m_k be the matches from L to G , then H is the set of triples $(- : l_i, s_j, m_i(?s_j))$ where $i \in \{1, \dots, k\}$, $j \in \{1, \dots, n\}$, and the blanks $- : l_1, \dots, - : l_k$ are pairwise distinct.*

Proof. We have $Gr(S) = \{(- : r, s_j, ?s_j)\}_{j \in \{1, \dots, n\}}$, so that according to Definition 19 the query result of $(L, Gr(S))$ against G is $H_1 \cup \dots \cup H_k$ where $H_i = \{(- : l_i, s_j, m_i(?s_j))\}_{j \in \{1, \dots, n\}}$ and the blanks $- : l_1, \dots, - : l_k$ are pairwise distinct. \square

Because of Proposition 37 we can state the following definition.

Definition 38. Let (L, S) be a basic select query and G a data graph. Let H be the query result of $(L, Gr(S))$ against G . The *query result* of (L, S) against G is the multirelation $Rel(H)$ on IB .

The *answer*, or *evaluation*, of a SPARQL SELECT query over an RDF graph is defined in [12, Section 2.3] as follows.

Definition 39 ([12]). Let L be a basic graph pattern of SPARQL, $S = \{?s_1, \dots, ?s_n\}$ a finite set of variables included in $|L|_V$ and let G be an RDF graph. The *answer* of the SPARQL query “SELECT $?s_1, \dots, ?s_n$ WHERE $\{L\}$ ” over G is the multiset with elements the restrictions $\mu|_S$ of the mappings μ from L to G to the subset S of $|L|_V$, each $\mu|_S$ with multiplicity the number of corresponding μ 's.

Theorem 40. *Let L be a basic graph pattern of SPARQL and $S = \{?s_1, \dots, ?s_n\}$ a finite set of variables included in $|L|_V$ and let G be an RDF graph. Then the query result of $(L, Gr(S))$ against G is the answer of the SPARQL query “SELECT $?s_1, \dots, ?s_n$ WHERE $\{L\}$ ” over G .*

Proof. Since the mappings from L to G correspond bijectively to the matches from L to G , the result follows from Proposition 37. \square

6 Conclusion

In this paper, we bet to base our work entirely on algebraic theories behind graphs and their transformations. Suitable categories of data graphs and query graphs are defined and the definition of morphisms of query graphs clarifies the difference between blank nodes and variables. Besides, we propose to encode CONSTRUCT and SELECT queries as graph rewrite rules, of the form $L \rightarrow L \cup R \leftarrow R$, and define their operational semantics following a novel algebraic approach called POIM. From the proposed semantics, blanks in L play the same role as variables and thus can be replaced by variables, whereas blanks in R are used for creating new blanks in the result of a CONSTRUCT query. As in [14], 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. 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 instance, one may expect to define data graph categories for the well-known Edge-labelled graphs or Property graphs [16]. The proposed operational semantics can clearly benefit from all results regarding efficient graph matching implementation, see e.g. [11].

Among related works, a category of RDF graphs as well as their transformations have been proposed in [5]. The authors defined objects of RDF graph categories of the form $(G_{\text{Blank}}, G_{\text{Triple}})$ where G_{Blank} and G_{Triple} denote respectively the set of blank nodes and the set of triples of graph G . In addition, the morphisms of such RDF graphs associate blank nodes to blank nodes. These definitions of object

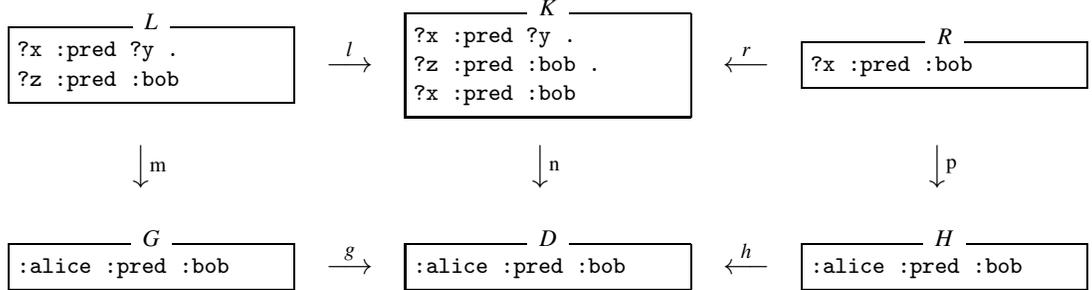
and morphisms are different from ours. Associating a blank node to any element of a triple, as our homomorphisms do, is called instantiation in [5]. The authors did not tackle the problem of answering SPARQL queries but rather proposed an algebraic approach to transform RDF graphs. Their approach, called MPOC-PO, is inspired from DPO where the first square is replaced by a “minimal” pushout complement (MPOC). However, MPOC-PO drastically departs from the POIM transformations we propose. This difference is quite natural since the two approaches have different aims : the POIM approach is dedicated to implement SPARQL queries while the MPOC-PO is intended to transform RDF graphs in general. However, MPOC-PO and DPO approaches are clearly not tailored to implement CONSTRUCT or SELECT queries since the (minimal) pushout complements always include parts of large data graphs which are not matched by the queries while such parts are not involved in the query answers. The same remark applies also to graph transformations where rules are cospans as in [10].

The image factorization part of POIM steps does not yield, in general, the same result as a pushout complement or a pullback complement constructs. For example, let us consider the following query

```

CONSTRUCT {?x :pred :bob} WHERE {?x :pred ?y . ?z :pred :bob}
    
```

The POIM rule $L \rightarrow K \leftarrow R$ corresponding to this query and its application to a graph G consisting of one triple $(:alice, :pred, :bob)$ are depicted below.



The reader can easily check that the right square is neither a pushout nor a pullback.

In [1], even if the authors use a categorical setting, their objectives and results depart from ours as they mainly encode every ontology as a category. However, Graph Transformations have already been used in modeling relational databases, see e.g. [3] where a visual and textual hybrid query language has been proposed. In [13], the main features of a data management system based on graphs have been proposed where the underlying typed attributed data graphs are different from those of RDF and SPARQL. In [2], triple graph grammars (TGG) have also been used for data modelling and model transformation rules to be compiled into Graph Data Bases code for execution.

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 study to other features of RDF and SPARQL in order to encompass general SPARQL queries. We also consider investigating RDF Schema [20] and ontologies from this point of view.

References

- [1] S. Aliyu, S.B. Junaidu & A. F. Donfack Kana (2015): *A Category Theoretic Model of RDF Ontology*. *International Journal of Web & Semantic Technology* 6(3), pp. 41–51, doi:10.5121/ijwest.2015.6304.
- [2] Abdullah Alqahtani & Reiko Heckel (2018): *Model Based Development of Data Integration in Graph Databases Using Triple Graph Grammars*. In: *STAF 2018, LNCS 11176*, Springer, pp. 399–414, doi:10.1007/978-3-030-04771-9_29.

- [3] Marc Andries & Gregor Engels (1996): *A Hybrid Query Language for an Extended Entity-Relationship Model*. *J. Vis. Lang. Comput.* 7(3), pp. 321–352, doi:10.1006/jvlc.1996.0017.
- [4] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter & Domagoj Vrgoc (2017): *Foundations of Modern Query Languages for Graph Databases*. *ACM Comput. Surv.* 50(5), pp. 68:1–68:40, doi:10.1145/3104031.
- [5] Benjamin Braatz & Christoph Brandt (2008): *Graph Transformations for the Resource Description Framework*. *ECEASST 10*, doi:10.14279/tuj.eceasst.10.158.
- [6] Edgar F. Codd (1990): *The relational Model for Database Management (Version 2 ed.)*. Addison-Wesley.
- [7] Andrea Corradini, Tobias Heindel, Frank Hermann & Barbara König (2006): *Sesqui-Pushout Rewriting*. In: *ICGT 2006, LNCS 4178*, Springer, pp. 30–45, doi:10.1007/11841883_4.
- [8] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel & Michael Löwe (1997): *Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach*. In Rozenberg [17], pp. 163–246, doi:10.1142/3303.
- [9] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner & Andrea Corradini (1997): *Algebraic Approaches to Graph Transformation - Part II: Single Pushout Approach and Comparison with Double Pushout Approach*. In Rozenberg [17], pp. 247–312, doi:10.1142/3303.
- [10] Hartmut Ehrig, Frank Hermann & Ulrike Prange (2009): *Cospan DPO Approach: An Alternative for DPO Graph Transformations*. *Bulletin of the EATCS* 98, pp. 139–149. Available at <https://eatcs.org/images/bulletin/beatcs98.pdf>.
- [11] Wenfei Fan, Jianzhong Li, Shuai Ma, Hongzhi Wang & Yinghui Wu (2010): *Graph Homomorphism Revisited for Graph Matching*. *PVLDB* 3(1), pp. 1161–1172, doi:10.14778/1920841.1920986.
- [12] Mark Kaminski, Egor V. Kostylev & Bernardo Cuenca Grau (2017): *Query Nesting, Assignment, and Aggregation in SPARQL 1.1*. *ACM Trans. Database Syst.* 42(3), pp. 17:1–17:46, doi:10.1145/3083898.
- [13] Norbert Kiesel, Andy Schürr & Bernhard Westfechtel (1995): *GRAS, a Graph-Oriented (Software) Engineering Database System*. *Inf. Syst.* 20(1), pp. 21–51, doi:10.1016/0306-4379(95)00002-L.
- [14] Egor V. Kostylev, Juan L. Reutter & Martín Ugarte (2015): *CONSTRUCT Queries in SPARQL*. In: *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, pp. 212–229, doi:10.4230/LIPIcs.ICDT.2015.212.
- [15] Jorge Pérez, Marcelo Arenas & Claudio Gutiérrez (2009): *Semantics and complexity of SPARQL*. *ACM Trans. Database Syst.* 34(3), pp. 16:1–16:45, doi:10.1145/1567274.1567278.
- [16] Ian Robinson, Jim Webber & Emil Eifrem (2013): *Graph Databases*. O’Reilly Media, Inc.
- [17] Grzegorz Rozenberg, editor (1997): *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, doi:10.1142/3303.
- [18] (2013): *SPARQL 1.1 Query Language*. W3C Recommendation. Available at <https://www.w3.org/TR/sparql11-query/>.
- [19] (2014): *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. Available at <https://www.w3.org/TR/rdf11-concepts/>.
- [20] (2014): *RDF Schema 1.1*. W3C Recommendation. Available at <https://www.w3.org/TR/2014/REC-rdf-schema-20140225/>.