

An Algebra For Basic Graph Patterns

George H.L. Fletcher

School of Engineering and Computer Science
Washington State University, Vancouver
fletcher@vancouver.wsu.edu

Abstract. Motivated by recent developments in the dataspace, web, and personal information management communities, we outline research directions on query processing for SPARQL, the W3C recommendation language for querying RDF triple stores. The core of each SPARQL query is a basic graph pattern (BGP). BGP is a little logic for extracting subsets of related nodes in an RDF graph. In this paper we undertake a formal study of BGP with an eye towards efficient SPARQL query evaluation. Our main contributions are (1) an algebraization of BGP, and (2) first steps towards a framework for the design of structural indexes to accelerate processing of queries in this algebra.

1 Introduction

The flexibility and fundamental character of ternary relations for data modeling was recognized very early in the development of modern logic [34]. Although there were some initial efforts towards realizing this observation in information systems (e.g., [31]), “triples” have only recently gained traction with the development of the W3C Resource Description Framework (RDF) [24, 29]. As a data model for capturing “things” and their “relationships,” treating each as co-equal first class objects, RDF successfully blurs the somewhat arbitrary distinction one is typically forced to make between “data” and “metadata.”¹

Example 1 Consider the following set of atoms (i.e., RDF terms such as URIs and literals [29]), denoting a small subset of the objects in a particular individual’s dataspace of information:

```
{Yamada, McShea, Herzog, doc1, doc2, doc3, knows, performed, authored,  
type, social action, is a kind of, past action, created on, 14.5.08, PDF, MP3}
```

A sample of some of the (triple) relationships holding between these objects is given in Figure 1(a). For example, the first triple asserts that Yamada authored Document 1; the second triple asserts that Yamada knows McShea; and the third triple asserts that ‘knows’ is a kind of social action. This “graph” of relationships can be visualized as in Figure 1(b).

¹ Note that RDF has affinities to data models previously proposed in the context of graph databases (e.g., [5, 18]) and data integration systems (e.g., [4, 30, 47]).

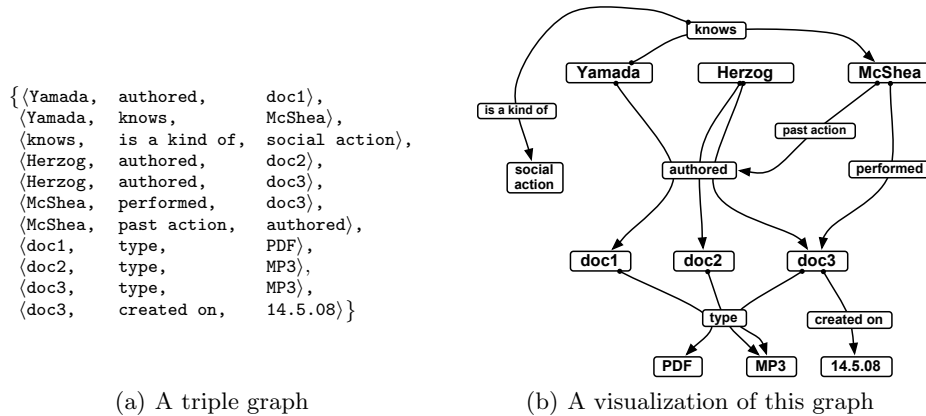


Fig. 1. A small subset of a personal dataspace graph.

The flexibility of the RDF data model has made it a natural choice for technologies being developed in the social and semantic web research communities [2, 40]. In addition to the central role RDF plays in these areas, recent efforts towards the development of dataspace [13, 20] of personal information [26] have also highlighted the natural fit of triples in modeling modern data scenarios. As the use of RDF grows, the need for RDF data management becomes increasingly crucial.

There has been an explosion of proposals for RDF query languages [16]. The W3C recommendation query language SPARQL [36], however, is emerging as the most mature and accepted. SPARQL is a declarative language with a syntax very much like SQL.

Example 2 Consider the query “Retrieve authors and the types of their documents.” In SPARQL, where variables are identified by a preceding *?*, this query can be posed against the graph in Figure 1(a) as follows:

```

SELECT ?a ?t
WHERE { ?a authored ?d . ?d type ?t . }

```

The WHERE clause of a SPARQL query specifies a *basic graph pattern* (BGP). Such patterns, which are at the heart of all SPARQL queries, identify a subset of related atoms to be extracted from the RDF graph, which is then returned as a set of variable mappings.

Example 3 The BGP of Example 2 is $(?a \text{ authored } ?d) ; (?d \text{ type } ?t)$. The semantics of this pattern with respect to the graph of Figure 1(a) is a set of variable mappings, each of which makes the pattern hold in the graph, i.e.,

1	2	3
Yamada	doc1	PDF
Herzog	doc2	MP3
Herzog	doc3	MP3

where column 1 holds the bindings of the first variable occurring in the pattern (namely $?a$), and so forth. The result of the complete SPARQL query is the projection of this mapping set, via the **SELECT** clause, on variables $?a$ and $?t$ (i.e., columns 1 and 3). In essence, a SPARQL query is a result-constructor wrapped around a set of variable bindings generated by a BGP.

As a simple yet expressive conjunctive calculus for RDF, BGP captures a broad class of natural queries (cf., those considered in [9, 13]), hence making it suitable as the underlying language for a variety of casual-user data interaction scenarios.

Recently, there have been formal investigations into the semantics of SPARQL and RDF [10, 11, 17, 35], as well as engineering efforts on SPARQL query optimization (e.g., [23, 42]). In this short paper we develop an algebra for the core of BGP, building on this ongoing work, and study the instance-expressivity of various fragments of this algebra. Unlike efforts towards full triple algebras for SPARQL (and beyond) such as [8, 15, 17, 37], our interest in developing this algebra is for its use as a basic tool in formal investigations into efficient processing of basic graph patterns. In particular, our long-term goal is to develop a framework for the design of index data structures specifically tailored to accelerate BGP evaluation. The present paper describes our first steps in this project.

In the next section, we define BGP and an equivalent algebra BTA. Following this, we consider the expressive power of BTA on graph instances and implications for index structure support for processing BTA queries. We then close with indications for further research.

2 Basic Query Languages for Graphs

2.1 Preliminary Definitions

In this section we develop the basic definitions used in the balance of the paper. In what follows, assume that there is an enumerable set of atoms \mathcal{A} over which triples are formed. Elements of \mathcal{A} correspond to the URI, literal, and blank-node terms of the RDF specification [29].

Definition 4 A triple is an object $t = \langle a_s, a_p, a_o \rangle \in \mathcal{A} \times \mathcal{A} \times \mathcal{A}$. Let $\text{subject}(t) = a_s$, $\text{predicate}(t) = a_p$, and $\text{object}(t) = a_o$.

Definition 5 A graph G is a finite set of triples. Let

$$\begin{aligned} \mathcal{S}(G) &= \{\text{subject}(t) \mid t \in G\}, \\ \mathcal{P}(G) &= \{\text{predicate}(t) \mid t \in G\}, \text{ and} \\ \mathcal{O}(G) &= \{\text{object}(t) \mid t \in G\}. \end{aligned}$$

The active domain of G is the set of atoms occurring in G , denoted as $\mathcal{A}(G) = \mathcal{S}(G) \cup \mathcal{P}(G) \cup \mathcal{O}(G)$.

As discussed in Section 1, basic graph patterns define functions from graphs to mapping sets.

Definition 6 A mapping set μ is a finite subset of $\underbrace{\mathcal{A} \times \dots \times \mathcal{A}}_{3n \text{ times}}$, for some positive integer n . Each $m \in \mu$ is called a mapping.

We next state a few more necessary definitions.

Definition 7 A built-in condition is an expression of the form $i = j$ or $i = a$ for some positive integers i, j and atom $a \in \mathcal{A}$. Such an expression is said to be rank n compatible if, for each integer i occurring in the expression, it is the case that $i \leq 3n$.

Let $m = \langle a_1, \dots, a_{3n} \rangle$ be a mapping and i be a positive integer. If $i \leq 3n$, let $m[i]$ denote the i th component of m (i.e., $m[i] = a_i$), and otherwise let $m[i] = \langle \rangle$.

Definition 8 Let c be a built-in condition. A mapping m is said to satisfy c if $c = "i = j"$ and $m[i] = m[j]$, or $c = "i = a"$ and $m[i] = a$.

2.2 Basic Graph Patterns

Assume that we have an enumerable set of variable names \mathcal{V} , disjoint from \mathcal{A} . Furthermore, fix some bijection $f : \mathcal{V} \rightarrow \mathbb{N}$. We will let v_i denote the element $v \in \mathcal{V}$ for which $f(v) = i - 1$. In this way, we can speak of the “first” and “second” elements, v_1 and v_2 , of \mathcal{V} , and so forth.

We next define the logic BGP, first noting that any basic graph pattern can be “normalized,” both in form and variable choice.

Example 9 The pattern of Example 3 can be rewritten as:

$$(v_1, v_2, v_3); (v_4, v_5, v_6); v_2 = \text{authored}; v_3 = v_4; v_5 = \text{type}$$

Clearly, this pattern has the same semantics as the original pattern. The variables have just been uniformly renamed from an initial segment of \mathcal{V} , and the constraints on the pattern have been captured as a list of built-in conditions. Note that the occurrence of atoms in the original pattern (i.e., `authored` and `type`) induce an early “projecting out” (i.e., quantifying out) of the variables v_2 and v_5 (and similarly for the repeated variable `?d`, i.e., variable v_4). Hence, v_2 , v_4 , and v_5 do not explicitly appear in the resulting mapping set of Example 3.

It is obvious that this normalization can be done for any BGP. Hence, without loss of generality, we define BGP’s following this form.

Definition 10 The set of basic graph patterns (BGP) is defined as follows:

- Let $v_1, \dots, v_{3n} \in \mathcal{V}$ for some positive integer n . Then

$$(v_1, v_2, v_3); \dots; (v_{3n-2}, v_{3n-1}, v_{3n})$$

is a BGP. Such an expression is said to be of rank n .

- If $P \in \text{BGP}$ is of rank n and c is a rank n compatible built-in condition, then $P; c \in \text{BGP}$, also of rank n .²
- No other expressions are in BGP.

Patterns in BGP are evaluated on graphs, as follows.

Definition 11 Given a graph G , the semantics of a pattern $P \in \text{BGP}$ with respect to G , denoted $\llbracket P \rrbracket_G$, is a mapping set defined as follows:

- If $P = (v_1, v_2, v_3); \dots; (v_{3n-2}, v_{3n-1}, v_{3n})$, then $\llbracket P \rrbracket_G = \underbrace{G \times \dots \times G}_n$.
- If $P = Q; c$, then $\llbracket P \rrbracket_G = \{m \in \llbracket Q \rrbracket_G \mid m \text{ satisfies } c\}$.

2.3 Basic Triple Algebra

We next turn to an algebraization of BGP which will be useful in the sequel.

Definition 12 The set of basic triple algebra (BTA) expressions is defined as follows:

- The primitive \rightarrow is a BTA. Such an expression is said to be of rank 1.
- If $E_1, E_2 \in \text{BTA}$ with ranks n_1 and n_2 , respectively, then $E_1 \cdot E_2 \in \text{BTA}$, and is of rank $n_1 + n_2$.
- If $E \in \text{BTA}$, with rank n , and c is a rank n compatible built-in condition, then $\sigma_c(E) \in \text{BTA}$, also of rank n .
- No other expressions are in BTA.

Expressions in BTA are evaluated on graphs, as follows.

Definition 13 Given a graph G , the semantics of an expression $E \in \text{BTA}$ with respect to G , denoted $\llbracket E \rrbracket_G$, is a mapping set defined as follows:

- If $E = \rightarrow$, then $\llbracket E \rrbracket_G = G$.
- If $E = E_1 \cdot E_2$, then $\llbracket E \rrbracket_G = \llbracket E_1 \rrbracket_G \times \llbracket E_2 \rrbracket_G$.
- If $E = \sigma_c(E_1)$, then $\llbracket E \rrbracket_G = \{m \in \llbracket E_1 \rrbracket_G \mid m \text{ satisfies } c\}$.

Example 14 Note that the BGP of Example 9 can be expressed in BTA as

$$\sigma_{3=4}(\sigma_{2=\text{authored}}(\rightarrow) \cdot \sigma_{2=\text{type}}(\rightarrow)).$$

It is also possible to rewrite BTA expressions as BGP patterns. In fact, such rewrites are always possible in both directions.

Theorem 15 For every pattern $P \in \text{BGP}$ there is a semantically equivalent expression $E \in \text{BTA}$, and vice versa.

² Here we use the fact that an integer i occurring in a built-in condition identifies with variable v_i .

In other words, BGP and BTA are equivalent as graph query languages. This equivalence is established in the Appendix by straightforward structural induction on expressions (Lemma 21) and patterns (Lemma 22).

Over three decades of research on conjunctive query optimization and evaluation (e.g., [3, 38]) can be fruitfully applied towards BTA processing. For example, many of the algebraic transformations for the relational algebra [3] can be applied during compilation of BTA expressions. Indeed, research efforts towards a systematic understanding of the use and extension of such standard techniques for SPARQL query optimization are currently under way (e.g., [23, 39, 42]). Complementing these ongoing efforts, we are interested in using BTA as a vehicle to study core issues in efficient BGP evaluation. We next consider in what ways the special characteristics of graphs and BTA expressions may be profitably leveraged in the design of index data structures specifically tailored for accelerated BTA evaluation.

3 Towards Structural Indexes for BTA Query Processing

The need for native RDF indexes has been highlighted in many recent investigations, e.g., [1, 12, 13, 22, 23, 25, 42]. The development of such indexes is crucial to the success of RDF triple stores, just as they are critical for the practical success of any data management system [6]. State of the art RDF stores use either variations of standard *value*-based indexing schemes based on hashing or search trees (e.g., [22, 46]), or materializations of specific substructures in the data graph (e.g., [1, 41, 43, 45]).

A next step in this line of research is to develop a clear understanding of the design space of *structural* indexes which generalize these results to reflect the native *structure* of graphs and *expressivity* of graph query languages. Such indexes have been developed and successfully used in the context of relational (e.g., [44]), complex object (e.g., [6, 21, 28]), and semistructured (e.g., [27, 32]) databases. However, as graphs are neither typed, hierarchical, nor rooted, a direct adaptation of the index structures developed in these domains is not feasible.

Recently, Gyssens et al. have developed a methodology for tightly coupling index structures to query languages, focusing on the XML query language XPath [14, 19]. In this methodology, one considers the expressive power of a given query language to distinguish objects in a data instance. If a subset of objects in the instance can not be distinguished by the language, then its members are essentially equivalent and should be handled together during query processing. In this way, language-indistinguishability induces a partition on the objects of the instance. This partition forms the (ideal) basis for structural indexes which support query processing in the language. The outcome of the methodology is a language-independent characterization of the induced partition which can then be applied towards construction of such indexes.

We now turn our attention to a structural characterization of the semantics of BTA, following this methodology, with a goal of utilizing such a characterization

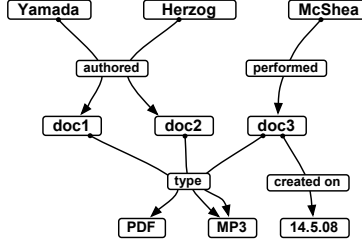


Fig. 2. Visualization of a subset of the personal dataspace graph of Figure 1(a).

in the design of native RDF structural indexes. In particular, we next consider the ability of various fragments of BTA to distinguish atoms in a given graph instance G . On first consideration, it might seem reasonable to consider indistinguishability of atoms in $\mathcal{A}(G)$, the complete active domain of G . However, such a choice ignores the “roles” which atoms play in G , by virtue of their appearance as **subjects**, **predicates**, or **objects** in triples. Indeed, if we inspect Definition 13, we note that each “variable” in the evaluation of a BTA expression ranges over only one of these domains.

Example 16 *Let E denote the BTA expression of Example 14 and G denote the graph depicted in Figure 2. For each mapping $m \in \llbracket E \rrbracket_G$, the range of possible bindings for both $m[1]$ and $m[4]$ is the set of atoms appearing as subjects in G , namely $\mathcal{S}(G) = \{\text{Yamada}, \text{Herzog}, \text{McShea}, \text{doc1}, \text{doc2}, \text{doc3}\}$. In general, for any mapping m appearing in a mapping set (of some arity $3n$) resulting from the evaluation of a BTA expression on a graph G , it is the case that $m[3k + 1]$ ranges over the elements of $\mathcal{S}(G)$, for each $0 \leq k \leq n - 1$.*

Hence, it is appropriate to consider language indistinguishability separately on each of $\mathcal{S}(G)$, $\mathcal{P}(G)$, and $\mathcal{O}(G)$.

Intuitively, if a fragment \mathcal{L} of BTA cannot distinguish between some of the elements of $\mathcal{S}(G)$, then they should be handled together as a block during \mathcal{L} query evaluation. We capture this intuition as follows.

Definition 17 *Let G be a graph, $s, s' \in \mathcal{S}(G)$, and \mathcal{L} be a well-defined subset of BTA. We say s and s' are \mathcal{L} -equivalent, denoted $s \equiv_{\mathcal{L}} s'$, if for any $E \in \mathcal{L}$ and $n \in \mathbb{N}$, it is the case that there exists a mapping $m \in \llbracket E \rrbracket_G$ with $m[3n + 1] = s$ if and only if there exists a mapping $m' \in \llbracket E \rrbracket_G$ with $m'[3n + 1] = s'$. The partition induced by $\equiv_{\mathcal{L}}$ on $\mathcal{S}(G)$ is called the \mathcal{L} -partition of $\mathcal{S}(G)$. Similarly, define the \mathcal{L} -partition of $\mathcal{P}(G)$ and $\mathcal{O}(G)$.*

In the balance of this section, we focus on partitions of $\mathcal{S}(G)$. The discussion extends directly to $\mathcal{P}(G)$ and $\mathcal{O}(G)$ partitions.

Consider the full BTA. We argue that the partitions induced by this language are too fine. Indeed, for a graph G , we have that each $s \in \mathcal{S}(G)$ forms its own partition-block, which is clear when we consider the query $\sigma_{1=s}(\rightarrow) \in \text{BTA}$. Hence, it is trivial to give a structural characterization of the semantics of the

full BTA, but the induced partitions give no special help in BTA evaluation. Note, however, that a similar situation holds for any reasonably expressive query language. The key to applying Definition 17 is to consider language fragments which are fundamental to processing arbitrary queries in the full language (or a large class of interesting queries) and which also induce manageable (and useful) partitions.

As a first attempt at such a fragment, relax BTA and consider its fragment \mathcal{L}_1 wherein built-in conditions involving atoms (i.e., of the form “ $i = a$ ” for some atom $a \in \mathcal{A}$ and integer i) are completely disallowed. In this case, we have a language which can only discern the length of “paths” in the graph. In particular, for each $k \geq 0$, atoms with a maximal path of length k emanating from them will be grouped together (modulo the route the paths take through the triples).

Example 18 *For the graph G depicted in Figure 2, the \mathcal{L}_1 -partition of $\mathcal{S}(G)$ consists of two blocks: $\{\text{[Yamada, Herzog, McShea]}, \text{[doc1, doc2, doc3]}\}$. The blocks are distinguished by expression $\sigma_{3=4}(\rightarrow \cdot \rightarrow) \in \mathcal{L}_1$.*

We argue that the partitions induced by \mathcal{L}_1 are, in general, too coarse. Indeed, in Example 18 the atoms Yamada, Herzog, and McShea are grouped together, thereby dropping basic information such as McShea has never been an author.

To address this shortcoming, consider fragment \mathcal{L}_2 of BTA wherein built-in conditions involving atoms are only allowed on predicates (i.e., of the form “ $i = a$ ” where $i = 3n + 2$ for some $n \geq 0$).³ In this case, we have a language for querying the structure of predicate-paths in the graph, which captures a very large class of common queries (cf. [9, 13]). We argue that the partitions induced by \mathcal{L}_2 are too fine, however. Indeed, relationships which are very distant from an atom have direct impact on the atom’s partition-block membership, thus leading to a very large number of partition blocks. For example, for the graph G depicted in Figure 2, the \mathcal{L}_2 -partition and BTA-partition of $\mathcal{S}(G)$ are identical.

Taking inspiration from work on “localized” path-indexes for semi-structured data [14, 27], consider, for each $k \geq 0$, fragments \mathcal{L}_3^k of \mathcal{L}_2 , wherein expressions are restricted to “paths” of length at most k . Here, a path is in the expression itself, and path length is essentially equal to the number of joins in the expression. We illustrate the notion of expression paths with a few examples.

Example 19 *The expression $\sigma_{3=4}(\sigma_{2=\text{authored}}(\rightarrow) \cdot \sigma_{2=\text{type}}(\rightarrow))$ of Example 14 has a path length of 1, as illustrated in Figure 3(a). For a slightly larger example, the expression $\sigma_{1=4}(\sigma_{2=3}(\sigma_{6=7}(\sigma_{9=11}(\rightarrow \cdot \rightarrow \cdot \rightarrow \cdot \rightarrow)))$) has a path length of 3, as illustrated in Figure 3(b).*

The \mathcal{L}_3^k family represents a balance between the laxity of \mathcal{L}_1 and the strictness of \mathcal{L}_2 . This is particularly true for small k values.

³ One could also consider fragments wherein built-in conditions are allowed only on subjects or objects, etc.

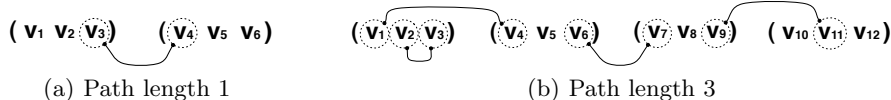


Fig. 3. Illustrations of path length for the BTA expressions of Example 19.

Example 20 For the graph G depicted in Figure 2, the \mathcal{L}_3^1 -partition of $\mathcal{S}(G)$ consists of four blocks: $\{\text{[Yamada, Herzog]}, \text{[McShea]}, \text{[doc1, doc2]}, \text{[doc3]}\}$, capturing the local (predicate) relationships of the subjects of G . For example, the first and second blocks are now distinguished by expression $\sigma_{2=\text{authored}}(\rightarrow) \in \mathcal{L}_3^1$.

The expressions in “little” fragments, such as \mathcal{L}_3^1 , can be viewed as the building blocks of larger expressions in \mathcal{L}_2 , thereby supporting a broad class of commonly used queries.⁴ Hence, we argue that partitions induced by the \mathcal{L}_3^k algebras (as well as other localized fragments of BTA) are an appropriate basis for the design and construction of structural indexes for BTA, a topic which we are actively investigating.

4 Summary and Research Directions

In this paper, we described initial steps in a systematic investigation into the design of structural indexes for SPARQL query processing. In particular, we presented BTA, an algebraization of the pattern logic which is at the core of SPARQL, and investigated the structure of the partitions on graphs induced by various fragments of this algebra. These partitions are the basis of native RDF index data structures to support acceleration of BTA evaluation.

The back-and-forth of Section 3 between relaxation and refinement of BTA fragments highlights a new space which this research opens for RDF indexing. Currently, we are studying the use of the partitions induced by localized BTA fragments in the design of index data structures. As a next step, an investigation of query processing techniques and algorithms using these indexes will be necessary. We close by mentioning two independent topics which further arise out of these investigations. First, the structural characterizations of Section 3 lead to BP-style instance-expressivity results [7, 19] for various fragments of BTA. An exposition of such results would give further insight into SPARQL. Second, the partitioning results of Section 3 can also be viewed as a form of “schema” or “ontology” discovery [33, 41] for schema-less RDF data. An investigation into this perspective on language-induced partitions may prove fruitful.

Acknowledgments. The author would like to express gratitude to Dirk Van Gucht his for encouragement and helpful feedback, to Jan Van den Bussche for an encouraging discussion on databases and RDF, to Ed Robertson for pointing out pioneering work on triple calculi, and to the reviewers for their helpful comments.

⁴ Note that \mathcal{L}_3^k recovers the expressive power of \mathcal{L}_2 when k is greater than or equal to the length of the longest path in the graph. We anticipate that fine-tuning the choice of k will be an interesting engineering issue.

Appendix

Theorem 15, that BTA and BGP are equivalent as query languages, is due to the following two facts.

Lemma 21 *Let $E \in \text{BTA}$ be of rank $n \geq 1$. There exists a pattern $P \in \text{BGP}$, also of rank n , such that for any graph G it is the case that $\llbracket P \rrbracket_G = \llbracket E \rrbracket_G$.*

Proof. By structural induction on BTA expressions. For the base case, $E = \rightarrow$ of rank 1, we have $P = (v_1, v_2, v_3)$, also of rank 1. Assume the statement holds for $E_1, E_2 \in \text{BTA}$ and consider:

- $E = \sigma_c(E_1)$. By hypothesis, there exists $P_1 \in \text{BGP}$ semantically equivalent to E_1 (and of the same rank, say n). E is also of rank n , and hence c is rank n compatible. Therefore, we have $P = P_1; c$, also of rank n .
- $E = E_1 \cdot E_2$. By hypothesis, there exist patterns

$$P_1 = (v_1, v_2, v_3); \dots; (v_{3n_1-2}, v_{3n_1-1}, v_{3n_1}); c_1^1; \dots; c_{m_1}^1, \text{ and}$$

$$P_2 = (v_1, v_2, v_3); \dots; (v_{3n_2-2}, v_{3n_2-1}, v_{3n_2}); c_1^2, \dots; c_{m_2}^2$$

in BGP of ranks n_1 and n_2 , resp., with m_1 and m_2 conditions, resp., and which are semantically equivalent to (and of the same rank as) E_1 and E_2 , resp. Then

$$P = (v_1, v_2, v_3); \dots; (v_{3(n_1+n_2)-2}, v_{3(n_1+n_2)-1}, v_{3(n_1+n_2)});$$

$$c_1^1; \dots; c_{m_1}^1; c_1^{2'}; \dots; c_{m_2}^{2'}$$

where, for $1 \leq i \leq m_2$, condition $c_i^{2'}$ of P is identical to condition c_i^2 of P_2 , except that each integer k occurring in c_i^2 is replaced by $3n_1 + k$ in $c_i^{2'}$. Clearly, P has rank $n_1 + n_2$, as does E .

In both cases, we have that the statement holds, thus completing the induction. \square

Lemma 22 *Let $P \in \text{BGP}$ be of rank $n \geq 1$. There exists an expression $E \in \text{BTA}$, also of rank n , such that for any graph G it is the case that $\llbracket E \rrbracket_G = \llbracket P \rrbracket_G$.*

Proof. By structural induction on BGP expressions. For the base case $P = (v_1, v_2, v_3); \dots; (v_{3n-2}, v_{3n-1}, v_{3n})$, of rank n , and we have $E = \underbrace{\rightarrow \dots \rightarrow}_{n \text{ times}}$, also

of rank n . Assume the statement holds for $P_1 \in \text{BGP}$, of some rank n_1 . For the inductive step, consider $P = P_1; c$, for some rank n_1 compatible built-in condition c . By hypothesis there exists $E_1 \in \text{BTA}$ semantically equivalent to (and of the same rank as) P_1 . Thus, we have $E = \sigma_c(E_1)$, also of rank n_1 , completing the induction. \square

References

- [1] D. J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. In *VLDB*, pages 411–422, Vienna, 2007.
- [2] K. Aberer. Data Management in the Social Web. In *EDBT*, pages 1203–1204, Munich, 2006.
- [3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading, MA, USA, 1995.
- [4] R. Agrawal, A. Somani, and Y. Xu. Storage and Querying of E-Commerce Data. In *VLDB*, pages 149–158, Rome, 2001.
- [5] R. Angles and C. Gutiérrez. Survey of Graph Database Models. *ACM Comput. Surv.*, 40(1):1–39, 2008.
- [6] E. Bertino and B. C. Ooi. The Indispensability of Dispensable Indexes. *IEEE Trans. Knowl. Data Eng.*, 11(1):17–27, 1999.
- [7] A. K. Chandra and D. Harel. Computable Queries for Relational Data Bases. *J. Comput. Syst. Sci.*, 21(2):156–178, 1980.
- [8] R. Cyganiak. A Relational Algebra for SPARQL. Technical Report HPL-2005-170, HP Labs, 2005.
- [9] N. Dalvi and D. Suciu. Indexing Heterogeneous Data. Technical Report 04-01-01, University of Washington, CSE, 2004.
- [10] J. de Bruijn, E. Franconi, and S. Tessaris. Logical Reconstruction of Normative RDF. In *OWLED*, Galway, Ireland, 2005.
- [11] J. de Bruijn and S. Heymans. Logical Foundations of (e)RDF(S): Complexity and Reasoning. In *ISWC*, Busan, Korea, 2007.
- [12] M. del Mar Roldán García and J. F. A. Montes. A Survey on Disk Oriented Querying and Reasoning on the Semantic Web. In *SWDB*, Atlanta, 2006.
- [13] X. Dong and A. Y. Halevy. Indexing Dataspaces. In *ACM SIGMOD*, pages 43–54, Beijing, 2007.
- [14] G. H. L. Fletcher, D. Van Gucht, Y. Wu, M. Gyssens, S. Brenes, and J. Paredaens. A Methodology for Coupling Fragments of XPath with Structural Indexes for XML Documents. In *DBPL*, pages 48–65, Vienna, 2007.
- [15] F. Frasincar, G.-J. Houben, R. Vdovjak, and P. Barna. RAL: An Algebra for Querying RDF. *World Wide Web J.*, 7(1):83–109, 2004.
- [16] T. Furche, B. Linse, F. Bry, D. Plexousakis, and G. Gottlob. RDF Querying: Language Constructs and Evaluation Methods Compared. In *Reasoning Web*, pages 1–52, Lisbon, Portugal, 2006.
- [17] C. Gutiérrez, C. A. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web Databases. In *ACM PODS*, pages 95–106, Paris, 2004.
- [18] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht. A Graph-Oriented Object Database Model. *IEEE TKDE*, 6(4):572–586, 1994.
- [19] M. Gyssens, J. Paredaens, D. Van Gucht, and G. H. L. Fletcher. Structural Characterizations of the Semantics of XPath as Navigation Tool on a Document. In *ACM PODS*, pages 318–327, Chicago, 2006.
- [20] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of Dataspace Systems. In *ACM PODS*, pages 1–9, Chicago, 2006.
- [21] J. Han et al. Join Index Hierarchy: An Indexing Structure for Efficient Navigation in Object-Oriented Databases. *IEEE TKDE*, 11(2):321–337, 1999.
- [22] A. Harth, J. Umbrich, A. Hogan, and S. Decker. YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In *ISWC*, Busan, 2007.

- [23] O. Hartig and R. Heese. The SPARQL Query Graph Model for Query Optimization. In *ESWC*, pages 564–578, Innsbruck, Austria, 2007.
- [24] P. Hayes. RDF Semantics. W3C Recommendation, 2004.
- [25] R. Heese, U. Leser, B. Quilitz, and C. Rothe. Index Support for SPARQL. In *ESWC*, Innsbruck, Austria, 2007.
- [26] W. Jones and J. Teevan, editors. *Personal Information Management*. University of Washington Press, Seattle, 2007.
- [27] R. Kaushik et al. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In *IEEE ICDE*, pages 129–140, San Jose, CA, 2002.
- [28] A. Kemper and G. Moerkotte. Access Support Relations: An Indexing Method for Object Bases. *Inf. Syst.*, 17(2):117–145, 1992.
- [29] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 2004.
- [30] W. Litwin, M. A. Ketabchi, and R. Krishnamurthy. First Order Normal Form for Relational Databases and Multidatabases. *SIGMOD Record*, 20(4):74–76, 1991.
- [31] C. R. Longyear. Further Towards a Triadic Calculus, Part 1, 2 & 3. *J. of Cybernetics*, 2(1, 2, & 3):50–65, 7–25, & 51–78, 1972.
- [32] T. Milo and D. Suciu. Index Structures for Path Expressions. In *ICDT*, pages 277–295, Jerusalem, 1999.
- [33] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. In *ACM SIGMOD*, pages 295–306, Seattle, 1998.
- [34] C. S. Peirce. Description of a Notation for the Logic of Relatives. *Memoirs of the American Academy of Arts and Sciences*, 9:317–378, 1870.
- [35] J. Pérez, M. Arenas, and C. Gutiérrez. Semantics and Complexity of SPARQL. In *ISWC*, pages 30–43, Athens, GA, 2006.
- [36] E. Prud’hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008.
- [37] E. L. Robertson. Triadic Relations: An Algebra for the Semantic Web. In *SWDB*, pages 91–108, Toronto, 2004.
- [38] F. Scarcello. Query Answering Exploiting Structural Properties. *SIGMOD Record*, 34(3):91–99, 2005.
- [39] G. Serfiotis et al. Containment and Minimization of RDF/S Query Patterns. In *ISWC*, pages 607–623, Galway, Ireland, 2005.
- [40] N. Shadbolt, W. Hall, and T. Berners-Lee. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.
- [41] M. Sintek and M. Kiesel. RDFBroker: A Signature-Based High-Performance RDF Store. In *ESWC*, pages 363–377, Budva, Montenegro, 2006.
- [42] M. Stocker et al. SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In *ACM WWW*, Beijing, 2008.
- [43] Y. Theoharis et al. Benchmarking Database Representations of RDF/S Stores. In *ISWC*, pages 685–701, Galway, Ireland, 2005.
- [44] P. Valduriez. Join Indices. *ACM Trans. Database Syst.*, 12(2):218–246, 1987.
- [45] K. Wilkinson. Jena Property Table Implementation. In *SSWS*, pages 35–46, Athens, Georgia, USA, 2006.
- [46] D. Wood, P. Gearon, and T. Adams. Kowari: A Platform for Semantic Web Storage and Analysis. In *XTech*, Amsterdam, 2005.
- [47] C. M. Wyss and F. I. Wyss. Extending Relational Query Optimization to Dynamic Schemas for Information Integration in Multidatabases. In *ACM SIGMOD*, pages 473–484, Beijing, 2007.