Reconstructing Graph Pattern Matches Using SPARQL

Stephan Mennicke[1](✉), Denis Nagel[2], Jan-Christoph Kalo[2], Niklas Aumann[2], and Wolf-Tilo Balke[2]

[1] Institut für Programmierung und Reaktive Systeme, TU Braunschweig, Germany
mennicke@ips.cs.tu-bs.de

[2] Institut für Informationssysteme, TU Braunschweig, Germany
{denis.nagel,n.aumann}@tu-bs.de,{kalo,balke}@ifis.cs.tu-bs.de

Abstract. Pattern matching is the foundation for handling complex queries to graph databases. Commonly used algorithms stem from the realm of graph isomorphism and simulations, being well understood theoretical frameworks. On the practical side, there are established graph query languages that often allow for a wide variety of query tasks, often even beyond pattern matching. However, very little is known how graph queries from common query languages relate to graph pattern matching relations. In this paper, we propose a study in this respect for SPARQL, the W3C recommendation for querying RDF data. The homomorphic nature of the SPARQL semantics allows for a straight-forward formulation of graph-isomorphic matching. However, the somewhat artificial nature of these queries motivates the study of sole basic graph patterns, the foundational concept of SPARQL. For basic graph patterns, we show a correspondence to strong simulation, an efficient graph pattern matching relation appreciated for its polynomial bound matches. In consequence, graph query languages are capable of serving as generating frameworks for established graph pattern matching relations.

1 Introduction

Graph databases have gained lots of attention due to their popularity in emerging applications like the Semantic Web, social network analysis, or bio-technology. These graphs usually provide *entity-centric data*, in which nodes represent entities, while the edges model relations between entities. Several graph database query languages were developed, enabling users to query graph-structured data in an SQL-like fashion. Most notably, SPARQL is the W3C standard for querying Semantic Web data, and is also used for a wide range of applications [13]. On the foundational side of graph querying, graph pattern matching in terms of special homomorphisms forms the main influence. However, to the best of our knowledge, only little is known on the relationship between commonly used graph query languages — SPARQL in this paper — and other graph pattern matching relations that have been researched for decades in conceptual frameworks. Research in this area led to several complexity results, the development of

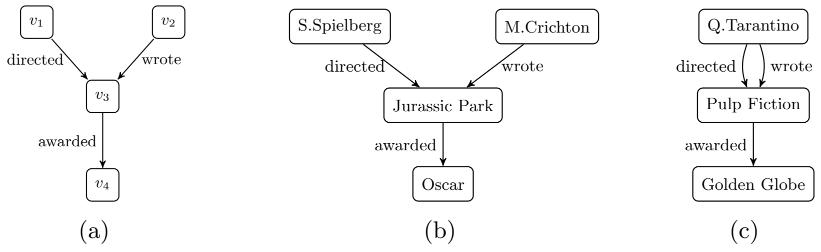

Fig. 1: (a) An example graph pattern P and (b) an isomorphic match of P and (c) a (dual-)simulating match of P.

fast algorithms, and insights on the semantics of the respective relations, whose potential is, in our view, not yet fully exploited in the area of graph database querying. Lately, some effort were expended to use graph pattern matching algorithms for matching relations different from classical subgraph isomorphism, often appreciated due to their advantages in performance over traditional graph querying languages [9, 11].

In this paper, we study the relationship between graph pattern relations and graph query languages, here exemplified by SPARQL. Our first result is best explained by an example. Imagine a user writing a query to search for the directors and writers of movies that won an award, see Fig. 1(a). An isomorphic match to this query is, for instance, the movie Jurassic Park which won an Academy Award, was written by M. Crichton, and directed by S. Spielberg (cf. Fig. 1(b)). The same result is achieved by the SPARQL query, depicted in Fig. 2. The first part, also called *basic graph pattern*, comprises variables for the graph pattern nodes and is arranged as triples representing the edges of the graph pattern. The filter condition at the end of the pattern ensures that each assignment to the variables is a bijective one, a necessary condition for graph-isomorphic matching. Being able to formulate such a query is not a coincidence. We prove, in Sect. 4, for every graph pattern there is a query returning every subgraph from a database that is isomorphic to the given pattern. A closer look at the filter condition raises the question, whether a user would be using such an artificial formulation. Removing the filter allows for (possibly unintended) variable assignments, and may produce answers as depicted in Fig. 1(c). Solely relying on basic graph patterns yields answers showing a (dual-)simulating character upon the original graph pattern, being the result of Sect. 5.

We provide partial and complete characterizations of graph pattern matching by SPARQL, based on basic graph patterns. We observe that dual simulation cannot be fully characterized, since the matching relation allows for arbitrary additions to matches, being also matches of the pattern. Strong simulation, an extension of dual simulation, removes this arbitrariness and is a matching relation renown for its efficient evaluation and its polynomial bound number of matches [9]. We find that SPARQL results may be used as building blocks to obtain all strong simulating matches. In return, strong simulation may also serve as a pruning method for SPARQL query engines. Sect. 3 provides basic notions.

```
SELECT *
WHERE { ?v_{v_1} directed ?v_{v_3} . ?v_{v_2} wrote ?v_{v_3} . ?v_{v_3} awarded ?v_{v_4} .
   FILTER ( ?v_{v_1} != ?v_{v_2} && ?v_{v_1} != ?v_{v_3} && ?v_{v_1} != ?v_{v_4} &&
            ?v_{v_2} != ?v_{v_3} && ?v_{v_2} != ?v_{v_4} && ?v_{v_3} != ?v_{v_4} ) }
```

Fig. 2: The *query for graph isomorphism* of the graph pattern Fig. 1(a)

Related work and conclusions are given in Sect. 2 and Sect. 6. Due to space limitations, full proofs of the theorems are included in the appendix of this paper.

2 Related Work

Graph Pattern Matching is an extensively studied topic in various domains of computer science [5]. Its applications range from social network analysis, over structural analysis of chemical entities to various applications in the database domain, particularly in graph databases. Recently, emerging applications led to the trend of graph pattern matching relations, different from the canonical though costly candidate of graph isomorphism, with the goal of reducing structural requirements of the answer graphs. For example, the idea of simulation for graph pattern matching has been implemented for different graph database tasks [1, 3, 2]. Indeed, experiments have shown advantages of simulation-based matching relations when analyzing social network patterns, as they offer the possibility to collapse several nodes into one node and vice versa. Another recent case study in this respect are so called *Exemplar Queries* [11], representing an attempt to enable an easy access to databases without the need of knowing the formal requirements of a query language. Based on an example graph pattern from the database (the *exemplar*), the query process of Exemplar Queries checks for similarity, e. g., up to *strong simulation*, between the exemplar and other database structures, retrieving and ranking them for presentation to the user.

Graph Query Languages basically all are founded on the idea of graph pattern matching (with suitable substitutions) [14]. A matching mechanism common to most of these query languages is (sub-)graph isomorphism [4, 8]. Mainly due to the advances in the field of Semantic Web, SPARQL has become the W3C recommendation for querying Semantic Web data, i. e., RDF. A more detailed introduction to SPARQL follows in the next section. In general, graph query languages differ greatly with respect to their area of operation. Therefore, many different graph database operations, e. g., subgraph matching or adding new nodes, are considered, particularly when comparing the expressive power of different graph query languages [14]. Most languages solely rely on homomorphic, more specifically, isomorphic pattern matching, but their connection to other matching relations is not yet studied extensively. Therefore, here we give insights into this aspect of graph query languages with respect to SPARQL. The basic idea however, could also be applied to other graph querying languages, also relying on the idea of basic graph patterns (e. g., Cypher or Gremlin). Regarding expressiveness of graph querying languages, in [7], the authors describe the graph query language *GraphQL*. It is based on a modified relational algebra that uses graph pattern matching for querying. They prove that their graph query algebra is relationally complete and therefore as expressive as relational algebra.

While we focus on graph isomorphism and simulations, the key question behind this work is not restricted to those relations. Many more comparison relations are discussed in the literature which may correlate very well with the semantics of graph query languages. For instance, the *linear-time branching-time spectrum* [6] provides several matching relations, under the term *comparative semantics*, used w. r. t. different aspects of system correctness. It contains comparative system relations for processes modeled as labeled transition systems, i. e., edge-labeled directed graphs with a distinct initial state. Most of the semantics come with a logical characterization in terms of a modal logic equipped with explicit quantification over edges to be traversed. Finding such a characterization is a common task, as it allows for expressing distinguishing characteristics of system behaviors in a precise manner. Similar to our subject, these characterizations express that two systems are equivalent whenever they satisfy the same logical formulas. In this paper, we try to adjust to the circumstances as imposed by SPARQL semantics.

Variants of graph homomorphism are also studied in the context of graph databases [4]. While the relations in our work always match an edge of a graph to other edges, as of preserving structure of a graph pattern to a certain extent, p-homomorphism takes each edge and maps it to paths in the match graph. This way, a p-homomorphic match graph may show a very different structure, being only loosely coupled with the graph pattern. Instead, p-homomorphic matching relies on a metric of node similarity.

3 Preliminaries

In this section, we define graphs, graph databases complemented by the general concept of graph pattern matching. Furthermore, we introduce an algebra of SPARQL.

A *(Σ-)labeled directed graph* is a triple $G = (V, \Sigma, E)$, where V is a finite set of nodes, Σ a finite alphabet, and $E \subseteq V \times \Sigma \times V$ a labeled edge relation. We represent an edge $(v, a, v') \in E$ by $v \xrightarrow{a} v'$. Labeled directed graphs range over by G, G_1, G_2, P with node sets V, V_1, V_2, V_P, a fixed alphabet Σ common to all graphs, and edge relations E, E_1, E_2, E_P with respective notations $\longrightarrow, \longrightarrow_1, \longrightarrow_2, \longrightarrow_P$. A graph G_1 is a subgraph of a graph G_2, denoted $G_1 \sqsubseteq G_2$, iff $V_1 \subseteq V_2$ and $E_1 \subseteq E_2 \cap (V_1 \times \Sigma \times V_1)$. Two graphs G_1 and G_2 are *isomorphic*, written $G_1 \cong G_2$, iff there is a bijective function $\kappa : V_1 \to V_2$ such that $v \xrightarrow{a}_1 v'$ if and only if $\kappa(v) \xrightarrow{a}_2 \kappa(v')$. κ is called an *isomorphism between* G_1 *and* G_2. For two nodes v, v' of a graph G, we define the distance $dist_G(v, v')$ to be the length of the shortest undirected path from v to v'. If there is no path between v and v', $dist_G(v, v') = \infty$. However, we are interested in *connected graphs* throughout the rest of the paper, i. e., for every two nodes v, v', $dist_G(v, v') \neq \infty$. The *diameter of a graph* G with node set V, denoted $dia(G)$, is the greatest distance between nodes in this graph, i. e., $dia(G) := \max\{dist_G(v, v') \mid v, v' \in V\}$.

Graph databases store objects from a countable universe $\mathcal{O}$ together with attributes over the objects as either relations between objects or properties of objects. As an example, consider an *isFriendOf* relation between objects referring to persons who are friends in *social networks*. Properties of objects are expressed as assignments of concrete data values, also called *literals*, from a usually infinite domain $\mathcal{L}$, to an object, e. g., the age of a person as a positive integer. Inspired by the treatment of literals in RDF, objects as well as literals are represented as nodes in a graph database. Relation symbols and property symbols stem from a finite set, here Σ. A *graph database* is a directed labeled graph $DB = (V, \Sigma, E)$ with a finite set of database objects $V \subseteq \mathcal{O} \cup \mathcal{L}$.

A *graph pattern* is a connected graph $P = (V_P, \Sigma, E_P)$. A subgraph G of a graph database DB is an *isomorphic match of* P *(in DB)* iff $P \cong G$. By $[\![P]\!]^{\cong}_{DB}$ we denote the set of all isomorphic matches of P.

Since SPARQL aims at querying RDF-stored data, the basic building blocks of the query language are triples of the form (s, p, o). Subjects (s) refer to objects in $\mathcal{O}$ or variables being assigned by actual database objects during the querying process. Objects (o) may further be associated with literals from $\mathcal{L}$. Predicates (p) are thought of as the relation and property symbols in Σ gluing together subjects with objects. Variables are place-holders for actual objects or literals as present in concrete databases. The result of a SPARQL query process is an assignment of objects and literals to the variables mentioned in a query expression. We denote the set of all variables by $\mathcal{V}$. Notation and semantics are based on [12].

Sets of such (s, p, o)-triples are called *basic graph patterns* (BGP), which we will assume to be graphs. For every BGP $B = \{(s_1, p_1, o_1), \ldots, (s_k, p_k, o_k)\}$ $(k \geq 0)$ where $s_i \in \mathcal{O} \cup \mathcal{V}$, $p_i \in \Sigma$, and $o_i \in \mathcal{O} \cup \mathcal{L} \cup \mathcal{V}$ $(i = 1, \ldots, k)$, the associated graph is (V_B, Σ, B) such that $V_B = \{s_1, \ldots, s_k, o_1, \ldots, o_k\}$. Notice that the nodes in the graph may also be variables. In fact, from the next section on, we employ BGPs in which all nodes are variables. In this paper, BGP B and its graph representation are used interchangeably. By $vars(B)$ we denote the set of all variables occurring in B.

The semantics of SPARQL BGPs B, and henceforth of SPARQL queries Q, is given in terms of *assignments* of objects and literals to variables in B (Q, respectively). An *assignment* is a partial function $\mu : \mathcal{V} \to (\mathcal{O} \cup \mathcal{L})$. By $\mu(B)$ we reference the graph where each variable node $v \in vars(B)$ is replaced by $\mu(v)$. We define $dom(\mu) := \{v \in \mathcal{V} \mid \mu(v) \textit{ is defined}\}$. An assignment μ is *valid w. r. t.* B *and a graph database* DB iff (a) $dom(\mu) = vars(B)$ and (b) $\mu(B)$ is a subgraph of DB. Thus, μ is a graph homomorphism. The set of all valid assignments w. r. t. B and a graph database DB forms the foundation of the SPARQL query semantics. We denote this set by $[\![B]\!]_{DB}$.

The second concept of SPARQL we use is that of *filter conditions*, also called *built-in conditions*. Filters are used to further restrict the set of (valid) assignments of a SPARQL query. Thereby, we may check for equality ($=$) or inequality ($<, \leq, \geq, >$) of variable assignments, objects, and literals. The usual propositional connectives ($\wedge, \vee, \neg$) are used to build complex constraints. For a full list of features we refer to the W3C recommendation report [13]. We denote by $\mu \models \varphi$ that assignment μ satisfies filter condition φ. Let $\mathbf{Q}$ be any SPARQL query, e. g., $\mathbf{Q} = B$ for a BGP B, and φ a filter condition. Then $\mathbf{Q}$ FILTER φ is a SPARQL query. The semantics is given recursively in terms of the assignments from $[\![\mathbf{Q}]\!]_{DB}$ such that every assignment conforms to φ. Thus, $[\![\mathbf{Q} \text{ FILTER } \varphi]\!]_{DB} := \{\mu \in [\![\mathbf{Q}]\!]_{DB} \mid \mu \models \varphi\}$.

Throughout the paper, we make use of BGPs adjoint with filter conditions. The last SPARQL concept we need throughout Sect. 5 is that of a join of two queries. $\mathbf{Q}_1$ AND $\mathbf{Q}_2$ represents the *join of* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$. Given two assignments $\mu_i \in [\![Q_i]\!]_{DB}$ ($i = 1, 2$). Then they are *compatible* if for every $v \in dom(\mu_1) \cap dom(\mu_2)$, $\mu_1(v) = \mu_2(v)$. Compatible assignments may be joined, thus, $[\![\mathbf{Q}_1 \text{ AND } \mathbf{Q}_2]\!]_{DB} := \{\mu_1 \cup \mu_2 \mid \mu_i \in [\![\mathbf{Q}_i]\!]_{DB} \text{ are compatible}\}$. The remaining operations of *union* and *optional* queries are not needed in this paper.

Next, we show that for every graph pattern P, there is a SPARQL query $\mathbf{Q}_P$ that uses a BGP and a specific filter condition to obtain all graph-isomorphic matches of P from a database DB. Please note that we assume graphs to be loop-free, throughout the paper, i. e., for each edge $v_1 \xrightarrow{a} v_2$, $v_1 \neq v_2$.

4 Querying like Graph Isomorphism

A graph pattern P gives rise to a canonical BGP. In order to characterize isomorphic matches of P from some database DB, we need to adapt the nodes of P to obtain the possibility of arbitrary assignments of database objects/literals to the nodes of P. In SPARQL terms, this adaptation is performed by exchanging each node by a variable. Fig. 2 gives an example conversion in the `where`-clause, excluding the filter condition.

Definition 1. *Let $P = (V_P, \Sigma, E_P)$ be a graph pattern. Define $\nu : V_P \to \mathcal{V}$ such that $\nu(v) := \mathsf{v}_v$. The BGP of P is defined as the graph $\hat{P} := (\hat{V}_P, \Sigma, \hat{E}_P)$ such that $\hat{V}_P = \{\nu(v) \mid v \in V_P\}$ and $(\nu(v), a, \nu(v')) \in \hat{E}_P$ iff $(v, a, v') \in E_P$.*

From a graph-theoretic perspective, it directly follows that each graph pattern P is isomorphic to its BGP $\hat{P}$ by isomorphism ν. Every assignment $\mu \in [\![\hat{P}]\!]_{DB}$ is a homomorphism, but it is not guaranteed that each graph $\mu(\hat{P})$ is isomorphic to P. We enforce bijectivity of μ by adding a filter condition checking that for each two distinct nodes $v, v' \in \hat{V}_P$, μ assigns different objects from the database.

Definition 2. *Let P be a graph pattern with set of nodes $V_P = \{v_1, v_2, \ldots, v_n\}$. Define a filter condition for P alongside ν (Def. 1) as $\varphi_P = \bigwedge_{i<j} \nu(v_i) \neq \nu(v_j)$. The P-*query for isomorphism *is $\mathbf{Q}_{\cong}(P) := \hat{P}$ FILTER φ_P.*

Reconsider the query given in Fig. 2 as an example query for isomorphism. We now show that the assignments of a P-query for isomorphism are equivalent to the set of all isomorphic matches w. r. t. P.

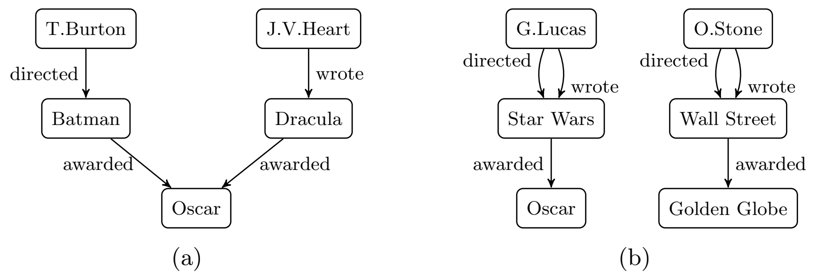

Fig. 3: Matches to the graph pattern graph depicted in Fig. 1(a): (a) a simulating but not dual-simulating match thus not strong simulating, and (b) a dual simulating but not strong simulating match because of the locality requirement.

Theorem 1. *Let DB be a graph database and P a graph pattern. Then* $G \in [\![P]\!]^{\cong}_{DB}$ *if and only if there is an assignment* $\mu \in [\![\mathbf{Q}_{\cong}(P)]\!]_{DB}$ *such that* $\mu(\hat{P}) = G$.

The proof exploits that assignments μ from respective queries already are isomorphisms. This means that by using BGPs and a specific filter condition, we reach the same expressive power as isomorphic graph pattern matching. In a way, Theorem 1 states that SPARQL is complete w. r. t. graph-isomorphic matching. Graph isomorphism is among the strongest similarity-based matching relations between graphs. Reducing the constructed SPARQL query in this section to only the BGP yields a similar result between SPARQL and graph homomorphism, since a valid assignment amounts to a graph homomorphism. However, as we will see throughout the next section, BGPs themselves show an interesting correspondence to graph pattern matching by similarity, or more specifically, strong simulation. Strong simulation is renown for its efficient evaluation, compared to isomorphic matching, and its polynomial bound on the number of matches [9].

5 Basic Graph Patterns Query for Simulations

In this section, we first show that every valid assignment of a BGP $\hat{P}$ corresponds to a so-called *dual simulating match* of the underlying graph pattern P. Thereupon, we argue that there is no reasonable way to capture all dual simulating matches by a single SPARQL query, since a match may be extended arbitrarily, even by database relations that are not mentioned by pattern P. The notion of *strong simulation* extends dual simulation by restricting (1) the size of the matches by the diameter of the pattern and (2) all occurring nodes and edges to match nodes and edges in the pattern. We show this restriction to be sufficient to prove the existence of a SPARQL query that captures all strong simulating matches. We introduce the necessary notions as needed.

5.1 Dual Simulation

Simulations stem from studies of (concurrent) system behavior [10, 6]. Intuitively, system 2 simulates system 1 if whatever action system 1 performs, system 2 is capable of mimicking this behavior. When we look at graphs, the notion carries over in the sense that we assume the nodes of the graphs to play the role of states and the labels on edges to represent the actions. We continue the introductory example (cf. Fig. 1). Considering the graph pattern P of Fig. 1(a), the graph in Fig. 3(a) is a simulating match of P. Starting by node v_1, P may only perform the actions, as represented by labeled edges 'directed' and 'awarded', in this order. Alternatively, the graph may also perform the sequence 'wrote' and 'awarded' when starting in v_2 or just 'awarded' when starting in v_3. Identical actions in the same order may be performed in the match graph. Therefore, it is indeed a simulating match of P. Formally, a simulation is a binary relation over the nodes of the respective graphs such that each node of the simulated graph is actually simulated in the above-mentioned sense. A dual simulation extends the notion of simulation in such a way that it also looks at actions going backwards from a simulated node. While Fig. 3(a) represents a simulating match for P, it is not a dual-simulating match. This is because the pattern may go backwards from v_4 via 'awarded' and then face the choice to go for 'directed' or 'wrote', which is not possible in Fig. 3(a). Fig. 3(b), on the other hand, is a valid dual-simulating match. In the study of concurrent systems, dual simulation does not have a feasible interpretation, since we are usually not able to let a system revert its actions. For graphs, in general, and graph database objects, this makes sense, since an object may be part of a relation, either as subject or object, which is equally important w. r. t. the represented relation.

Definition 3. *Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2)$ be two graphs. A* dual simulation *between* G_1 and G_2 *is a relation $S \subseteq V_1 \times V_2$ such that (a) for each node $v_1 \in V_1$, there is $v_2 \in V_2$ such that $(v_1, v_2) \in S$ and (b) for each $(v_1, v_2) \in S$,*

1. *$v_1 \xrightarrow{a}_1 v_1'$ implies that there is a $v_2' \in V_2$ with $v_2 \xrightarrow{a}_2 v_2'$ and $(v_1', v_2') \in S$,*
2. *$v_1' \xrightarrow{a}_1 v_1$ implies that there is a $v_2' \in V_2$ with $v_2' \xrightarrow{a}_2 v_2$ and $(v_1', v_2') \in S$.*

Let DB be a graph database and Q be a graph pattern. A subgraph G of DB is a dual simulating match of Q in DB *iff $Q \preceq_D G$. The set of all dual simulating matches of Q in DB is denoted by $[\![Q]\!]_{DB}^{\preceq_D}$.*

It is easy to show that the union of two dual simulations again yields a dual simulation. Since an assignment to a BGP is a homomorphism, we obtain only matches respecting at least the edge structure of a pattern. Therefore, each assignment corresponds also to a dual simulating match.

Proposition 1. *Let DB be a graph database and P a graph pattern. Then for all $\mu \in [\![\hat{P}]\!]_{DB}$, it holds that $\mu(\hat{P}) \in [\![P]\!]_{DB}^{\preceq_D}$.*

For every $\mu \in [\![\hat{P}]\!]_{DB}$, $S_\mu = \{(v, \mu(\nu(v))) \mid v \in V_P\}$ is the desired dual simulation. The converse does not hold, in general. This is because given a dual simulating match of a graph pattern P, every graph that contains this match as a subgraph is also a match. In theory, if the size of the database as well as the maximum in- and out-degrees of the database nodes are given, one could try to iteratively build BGPs extending the given graph pattern by structures allowed for dual simulation. Such a methodology is rather costly, since we may assume the database to be very large. In contrast, a query is often smaller, easily rendering the answers produced by the enlarged query useless. In fact, if there is at least one dual-simulating match in the graph database, then the graph database itself is also a match. *Strong simulation* overcomes these issues by limiting the size of matches by the diameter of the pattern. Matches to the graph pattern are locally bounded. Furthermore, irrelevant edges in the match are filtered out, letting a characterization of the respective SPARQL queries come in reach.

In order to exclude irrelevant edges, and thus, also nodes from a matching, the notion of *match graph of a dual simulation* S is introduced. In a match graph of S, each node and each edge play a role in the simulation. Formally, a graph G is a *match graph w. r. t. dual simulation* S iff (a) for each node v of G, there is a pair $(x, v) \in S$ for some node x of the pattern, and (b) for each edge $v_1 \xrightarrow{a} v_2$, there is an edge $u_1 \xrightarrow{a} u_2$ in the pattern such that $(u_i, v_i) \in S$ $(i = 1, 2)$. While strong simulation considers match graphs of special dual simulations, as we will explain in the next subsection, also dual simulation may benefit from this notion. By $|G|$ we denote the size of G, defined as the number of nodes in G. Considering all graphs $G \in [\![P]\!]_{DB}^{\preceq_D}$ for some graph pattern P, if G contains at most as many nodes as P, i. e., $|G| \leq |P|$, then an assignment constructing the match G exists.

Lemma 1. *Let DB be a graph database and P a graph pattern. For all matches* $G \in [\![P]\!]_{DB}^{\preceq_D}$ *with dual simulation* S *such that* $|G| \leq |P|$ *and* G *is a match graph w. r. t.* S*, there is a* $\mu \in [\![\hat{P}]\!]_{DB}$ *such that* $\mu(\hat{P}) = G$.

Proof. Let G be a dual simulating match of P under dual simulation S, as required. Then S is a homomorphism between P and G. Each node of the pattern has exactly one simulating node in G, since G is a match graph w. r. t. S (cond. (a)) and $|G| \leq |P|$. As G dual simulates P, an edge in P is reflected by an edge in G, homomorphically. From match graph cond. (b), it follows that each edge in G is reflected by some edge in the pattern. By taking $\mu = S$, we get $\mu(\hat{P}) = G$. □

5.2 Strong Simulating Matches

The last Lemma has given an exact characterization of valid assignments of SPARQL BGPs in terms of matching by dual simulation. In this section, we look at further restrictions on dual simulations, culminating to the notion of *strong simulation*. Strong simulation is an extension of dual simulation, i. e., a match still needs to dual simulate the pattern (e. g., reconsider Fig. 3(a) as not strong simulating), but this time, a simulation S certifying for dual simulation must be maximal, i. e., any other dual simulation is already contained in S. Uniqueness of the maximal dual simulation is easy to show and may be found, e. g., in [9]. Furthermore, a prospective graph needs to be a match graph w. r. t. maximal dual simulation S.

Strong simulation aims at keeping possible matches locally constrained such that the size of a match is bounded and also the overall number of matches is non-exponential. In order to meet this locality requirement, matches are bounded in the diameter d of the pattern in such a way that for every match, there is a node having distance at most d to any other node, e. g., Fig. 3(b) is not strong simulating to the graph pattern in Fig. 1(a), because the match graph is disconnected. Therefore, the locality requirement is violated. Formally, we require a match to be a subgraph of a so-called *ball* of the database *DB*. Let v be a node of *DB* and $r \in \mathbb{N}$ a radius. Then the subgraph of *DB* containing node v and all nodes and edges reached from v in at most r steps (backwards and forwards along the edges) is called a *ball of DB*, denoted $\widehat{DB}[v, r]$. For strong simulation, the radius is chosen to be the diameter of the graph pattern.

Definition 4. *Let DB be a graph database and P a graph pattern ($dia(P) = d$). Subgraph G of DB is a* strong simulating match of P in *DB* *iff there is a node v such that G is a subgraph of $\widehat{DB}[v, d]$ containing v with the following properties: (1) $P \preceq_D G$ by maximal dual simulation S and (2) G is the match graph of S. By $[\![P]\!]_{DB}^{\preceq_S}$ we denote the set of all strong simulating matches of P.*

Both, Fig. 1(b) and Fig. 1(c) are strong simulating matches of the pattern in Fig. 1(a). Ma et al. [9] call a strong simulating match *perfect subgraph*. In the spirit of Proposition 1, one can show that each assignment also corresponds to a strong simulating match. Again, not all matches can be recovered by a simple transformation of a graph pattern into a BGP. However, since the size of the matches is bounded by the diameter of the given graph pattern, an iterative method, like the one explained at the end of Sect. 5.1, may be feasible.

The rest of this section is devoted to proving the existence of a SPARQL query for graph pattern P, i. e., a *P-query for strong simulation*. Therefore, we first look at the SPARQL answers, already giving us strong simulating matches, and join them to bigger answers, similar to the SPARQL AND-operation (cf. Sect. 3). We then prove that for every strong simulating match there is a set of SPARQL assignments that amounts to the graph by joining the components. The resemblance of our join operator allows us to conclude that there is a SPARQL query reflecting the strong simulating matches of a graph pattern.

The *join of two subgraphs of DB* is the union of both graphs if they are *compatible*. Two graphs are *compatible* iff they share at least one node. Remember that we assume only connected graphs as graph patterns, justifying this stronger condition compared to the AND-operator of SPARQL.

Definition 5. *Let DB be a graph database. Two subgraphs $G_1, G_2 \sqsubseteq DB$ are* compatible *iff $V_1 \cap V_2 \neq \emptyset$. The* join of compatible graphs G_1 and G_2 *is defined as the graph $G_1 \blacktriangleright\!\blacktriangleleft G_2 = (V_1 \cup V_2, \Sigma, E_1 \cup E_2)$.*

Joining arbitrary strong simulating matches still yields dual simulating matches, but the locality requirement may be violated. Therefore, a second form of compatibility is necessary which restricts the possible combinations of strong simulating matches to be joined. For a graph G and radius $r \in \mathbb{N}$, $\mathcal{C}(G, r)$ denotes *the* *set of center nodes of* G *w. r. t.* r, i. e., all other nodes may be reached in at most r steps (ignoring the directions of the edges). Restricting $\bowtie$ to only join center nodes w. r. t. the diameter of the pattern maintains the locality requirement of strong simulation.

Definition 6. *Let* DB *be a graph database and* $r \in \mathbb{N}$ *a radius. Compatible subgraphs* $G_1, G_2 \sqsubseteq DB$ *are* r-compatible *iff* $V_1 \cap V_2 \subseteq \mathcal{C}(G_1, r) \cap \mathcal{C}(G_2, r)$.

While $\bowtie$ alone is commutative and associative, we loose associativity requiring r-compatibility. Therefore, we assume left-associativity of $\bowtie$ throughout the rest of the paper. The join of two strong simulating matches of a graph pattern P is again a strong simulating match of P if the matches are d-compatible with $d = dia(P)$. This is because the union of two (maximal) dual simulations yields a (maximal) dual simulation and, as long as only center nodes are joined, the distance requirement of strong simulation remain unaffected. This insight paves the way for the main theorem of this section. Every strong simulating match of pattern P may be reconstructed out of assignments of query $\hat{P}$. Since empty matches are trivially constructible, from zero BGP assignments, we rule out this case in the next theorem.

Theorem 2. *Let* DB *be a graph database,* P *a graph pattern with diameter* d*, and* $G \in [\![P]\!]_{DB}^{\preceq_S}$ *be non-empty. There are assignments* $\mu_1, \mu_2, \ldots, \mu_k \in [\![\hat{P}]\!]_{DB}$ *such that for each* $0 < j < k$*,* $\mu_1(\hat{P}) \bowtie \ldots \bowtie \mu_j(\hat{P})$ *and* $\mu_{j+1}(\hat{P})$ *are* d*-compatible and* $\mu_1(\hat{P}) \bowtie \mu_2(\hat{P}) \bowtie \ldots \bowtie \mu_k(\hat{P}) = G$.

Corollary 1. *Let* DB *be a graph database and* P *a graph pattern. Then there exists a* P*-query for strong simulation* $\mathbf{Q}_{\preceq_S}(P)$.

6 Conclusion

We provided novel insights into the relation between the widespread graph query languages, exemplified by SPARQL, and graph pattern matching. Thereby, we obtained a fresh look at the expressive power of graph query languages w. r. t. well-understood graph pattern matching relations. To the best of our knowledge, this is the first attempt that characterizes graph pattern matching relations by state-of-the-art graph query languages. Our findings are general in that every query language complete w. r. t. the SPARQL semantics we used (cf. Sect. 3), may reproduce our theorems. By the homomorphic nature of the SPARQL semantics, it was possible to formulate simple queries consisting only of a BGP and a filter condition that constructs all isomorphic matches of a given graph pattern (cf. Theorem 1). Removing the somewhat artificial filter condition from P-queries for graph isomorphism, thus only considering BGPs, yielded queries returning dual simulating matches. From a computational point of view, the construction of a query capturing all dual simulating matches is rather costly. However, limiting matches to relevant (w. r. t. the pattern) nodes and edges allows to fully characterize dual simulation in this spirit (cf. Lemma 1). Ultimately, we showed the existence of SPARQL queries for matching by strong simulation, an extension of dual simulation that locally restricts the size of the matches (cf. Theorem 2 and Corollary 1). As our first point for future work, we would like to give a constructive proof of Theorem 2 in order to make our findings applicable.

From a practical perspective, our results may be beneficial to query processing performance in very large graph databases. It is known that the evaluation of SPARQL queries itself is coNP-complete [12]. In contrast, graph pattern matching based on strong simulation only needs cubic time [9]. Query processing heuristics built on existing strong simulation algorithms could therefore lead to improvements with regard to general graph database query processing. Thereby, strong simulating matches may serve as over-approximations of BGPs, possibly reducing the number of *relevant candidate assignments*. Whether or not preprocessing of graph database queries by strong simulation leads to a significant reduction of computational time is left to an empirical evaluation. Further investigations on the interplay of BGPs and other graph query operations need to be performed.

References

1. Brynielsson, J., Högberg, J., Kaati, L., Mårtenson, C., Svenson, P.: Detecting Social Positions Using Simulation. In: ASONAM 2010. pp. 48–55 (2010)
2. Fan, W.: Graph Pattern Matching Revised for Social Network Analysis. In: ICDT 2012. pp. 8–21. ACM, New York, NY, USA (2012)
3. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y.: Graph pattern matching: From intractable to polynomial time. PVLDB Endow. 3(1-2), 264–275 (Sep 2010)
4. Fan, W., Li, J., Ma, S., Wang, H., Wu, Y.: Graph Homomorphism Revisited for Graph Matching. In: Proc. of VLDB '10. vol. 3, pp. 1161–1172 (2010)
5. Gallagher, B.: Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. In: Papers from the AAAI FS '06. pp. 45–53 (2006)
6. van Glabbeek, R.J.: The linear time - branching time spectrum. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. pp. 278–297. Springer, Berlin, Heidelberg (1990)
7. He, H., Singh, A.K.: Graphs-at-a-time: query language and access methods for graph databases. In: Proc. of SIGMOD'08. p. 405. ACM Press, New York, New York, USA (2008)
8. Lee, J., Han, W.S., Kasperovics, R., Lee, J.H.: An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. PVLDB Endow. 6(2), 133–144 (Dec 2012)
9. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Strong Simulation: Capturing Topology in Graph Pattern Matching. ACM Trans. Database Syst. 39(1), 4:1–4:46 (Jan 2014)
10. Milner, R.: An algebraic definition of simulation between programs. In: Proc. of IJCAI'71. pp. 481–489. Morgan Kaufmann Publishers Inc. (1971)
11. Mottin, D., Lissandrini, M., Velegrakis, Y., Palpanas, T.: Exemplar Queries: A New Way of Searching. The VLDB Journal 25(6), 741–765 (Dec 2016)
12. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Transactions on Database Systems 34(3), 1–45 (Aug 2009)
13. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (2008), http://www.w3.org/TR/rdf-sparql-query/
14. Wood, P.T.: Query languages for graph databases. SIGMOD Rec. 41(1), 50–60 (Apr 2012)

A Proof of Theorem 1

We show the two directions, separately.

if: Let $\mu \in [\![\mathbf{Q}_{\cong}(P)]\!]_{DB}$ be an assignment. We need to show that $\mu(\hat{P})$ is an isomorphic match of P, i. e., $P \cong \mu(\hat{P})$. Therefore, we prove that $\mu \circ \nu$ is an isomorphism. By our observation that ν is an isomorphism and isomorphisms are preserved by function composition, it is sufficient to show that μ is an isomorphism. μ is injective due to the filter condition φ_P. μ is surjective by the definition of a valid assignment. Thus, μ is an isomorphism.

only if: Let G be an isomorphic match of P. Thus, there is an isomorphism κ between P and G. Isomorphisms are closed under reversal, i. e., κ^{-1} is also an isomorphism. Furthermore, ν^{-1} is an isomorphism for the same reason. We construct an assignment by composing these two isomorphisms as $\mu = \nu^{-1} \circ \kappa^{-1}$. μ is an isomorphism, thus a bijective homomorphism and a valid assignment for $\hat{P}$, resulting in $\mu(\hat{P}) = G$. □

B Proof of Theorem 2

We proceed by induction on the size of the graph G, estimated by the number k of assignments needed to construct G. In the base case, $k = 1$, we look at graphs $G \in [\![P]\!]_{DB}^{\preceq_S}$ with $|G| \leq |P|$. The statement follows directly from Lemma 1.

Assume for some i, the claim holds in case $k = i$ and any smaller number. We need to show that the claim also holds in case $k = i + 1$. The induction hypothesis implies that for every match G with $|G| \leq i \cdot (|P| - 1) + 1$, there are at most i assignments constructing the match. This follows from the base case constructing graphs of size at most $|P|$, while any further addition contributes at most $|P| - 1$ nodes to the graph, since by assumption at least one center node is involved in the join.

For case $k = i + 1$, we establish the following bound of the size of G,

$$|G| \leq (i+1) \cdot (|P| - 1) + 1 = i \cdot |P| - i + |P| = \underbrace{i \cdot (|P| - 1) + 1}_{\text{induction hypothesis}} + (|P| - 1).$$

Towards a contradiction, suppose G is not constructible from the assignments of the BGP $\hat{P}$. Then there is a largest constructible subgraph $G' \sqsubseteq G$, which needs at most i assignments, and they exist by the induction hypothesis. The subgraph of G that has no shared nodes with G' is bounded by $|P| - 1$ due to the above equation. There is at least one edge $v' \xrightarrow{a} v$ (or $v \xrightarrow{a} v'$, resp.) with v' is in G' and v is in G but not in G'. Since G is a match of P, there is a smallest subgraph G'' of the database with $G'' \sqsubseteq G$, containing $v' \xrightarrow{a} v$ ($v \xrightarrow{a} v'$, resp.) and dual simulating P. If G'' is not a sugraph of G, then there is an edge outside of G necessary to dual simulate pattern P; establishing a contradiction to the assumption that G is a strong simulating match of P. Since the size of G'' is also bounded due to the equation above, G'' adds no more than $|P| - 1$ nodes to G'. By Lemma 1, there is an assignment of $\hat{P}$ that amounts to G''. Since this step may be repeated until no more edges remain, constructibility of G is implied. □