Designing and Comparing RPQ Semantics

Victor Marsault
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Antoine Meyer
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Abstract

Modern Property graph database query languages such as Cypher, PGQL, GSQL, and the standard GQL draw inspiration from the formalism of regular path queries (RPQs). In order to output walks explicitly, they depart from the classical and well-studied *homomorphism* semantics. However, it then becomes difficult to present results to users because RPQs may match infinitely many walks. The aforementioned languages use ad-hoc criteria to select a finite subset of those matches. For instance, Cypher uses *trail* semantics, discarding walks with repeated edges; PGQL and GSQL use *shortest walk* semantics, retaining only the walks of minimal length among all matched walks; and GQL allows users to choose from several semantics. Even though there is academic research on these semantics, it focuses almost exclusively on evaluation efficiency.

In an attempt to better understand, choose and design RPQ semantics, we present a framework to categorize and compare them according to other criteria. We formalize several possible properties, pertaining to the study of RPQ semantics seen as mathematical functions mapping a database and a query to a *finite* set of walks. We show that some properties are mutually exclusive, or cannot be met. We also give several new RPQ semantics as examples. Some of them may provide ideas for the design of new semantics for future graph database query languages.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Information systems → Query languages for non-relational engines; Information systems → Graph-based database models; Theory of computation → Regular languages

Keywords and phrases Regular Path Queries, RPQ, Semantics, Graph Databases, Pattern matching, Regular Expression

Digital Object Identifier 10.4230/LIPIcs.ICDT.2026.6

1 Introduction

In the past decades, there has been increasing interest and use for graph database management systems (DBMS). The typical task when exploiting such databases consists in navigating the graph. To achieve this, most modern query languages contain features inspired by Regular Path Queries (RPQs) [5]. RPQs and their generalizations were thoroughly studied in academic literature. Essentially, an RPQ is a regular expression used to query an edge-labeled graph. A walk (that is, an alternating sequence of vertices and edges) matches an RPQ R if its label (that is, the concatenation of its edge labels) conforms to R.

In industry, RPQs were added to the standard language SPARQL [25] for RDF. They were also the basis of the `MATCH` clause in Cypher [22, 9], the language introduced with the successful property graph DBMS Neo4j. RPQs were an inspiration for other query languages for graph DBMS [24, 23]. Due in part to the success of Cypher and Neo4j, a standard query language for property graphs was designed [14]. Its navigational part is heavily influenced by RPQs, and was made compatible with SQL [13].

RPQs are usually evaluated under *homomorphism semantics* [2]. In this setting, the answer to a query R is the set of all pairs (s, t) such that there exists a walk matching R from s to t. In other words, the answer consists of the set of *endpoints* of all matching walks. However, endpoint semantics turns out to be inadequate in some practical scenarios: users sometimes need to have access to whole matched walks, and not only to their endpoints.

RPQs typically match infinitely many walks, hence if queries are to return whole walks then their presentation to the user in full is problematic. To address this issue, many real-life query languages for property graphs choose to return only a finite subset of matching walks to the user, selected in various ways. Cypher uses *trail semantics*: a matching walk is only returned if it contains no edge repetition. Other languages like GSQL or PGQL use *shortest semantics*: a matching walk is returned only if there is no shorter match with the same source and target. Another well-known example is *acyclic semantics*: a matching walk is returned only if it contains no vertex repetition. GQL allows the user to choose one of these semantics, among others. We follow this modern approach in the current article: what we call an RPQ semantics is any specification of a way to select, among a potentially infinite set of matching walks, a *finite* subset of walks to return to the user.

Recent work on RPQs under various semantics focus on the evaluation process. Computational problems related to evaluation are untractable under acyclic [4] or trail [18] semantics. On the other hand, there exist efficient algorithms for evaluation under shortest semantics [21, 7]. One of the challenges of query evaluation is the sheer number of results, and there has been work on better, more succinct representations of the set of results [19, 20]. This draws a very one-sided picture: shortest semantics always yields algorithms that are much more efficient. However, the most popular system as of today (Neo4j) uses trail semantics, and our hypothesis is that this can be ascribed to other desirable properties.

Contributions. To start this investigation, we propose a formal framework to define and categorize RPQ semantics. We call semantics like trail or acyclic *filter-based*, because they act as post-filters: each walk is individually considered good (e.g., with no repetition) or bad (e.g. with repetitions), independently of other matches. We call semantics like shortest *order-based*, because they select the best matched walks according to some global criterion. More precisely, such semantics are defined using (partial) ordering relations on walks, and return all walks that are minimal (among matches) with respect to this ordering.

We then define many properties that RPQ semantics may or may not possess, and discuss why these properties may be desirable. To state a few, we introduce properties related to monotonicity, continuity, composability using rational operators, and coverage. Since we define RPQ semantics in the abstract, one may easily come up with unpractical or even nonsensical semantics. Hence, we propose a list of properties that any reasonable semantics should possess. We also show several impossibility theorems, in the form of sets of mutually exclusive properties. The goal of these results is to facilitate the design of new and interesting RPQ semantics. We often illustrate our definitions and statements with examples of new RPQ semantics (which are compiled in the index, page 19). Some of them appear to be reasonable candidates for possible practical use.

Outline. Section 2 contains preliminary vocabulary and definitions, in particular the models we use for data (labeled graph) and queries (regular expressions). Then, Section 3 presents our framework: what RPQ semantics are, how to define one, and which ones we are interested in. Section 4 introduces several properties that RPQ semantics might possess. We summarize the properties of several usual semantics, and provide a few general statement (e.g., which properties are mutually exclusive). We also discuss the desirability of those properties.

In this article, we introduce many new semantics, each with an acronym. An index is provided on page 19. Due to space constraints, all proofs are omitted.

Related work. As far as we know, few RPQ semantics were studied in academic work other than the endpoint, acyclic, trail and shortest semantics. The fragment of the RDF query language SPARQL which is related to RPQs is called *property paths* [25], and in the process leading to their definition, one of the candidate semantics indeed computed sets of walks. In order to only present a finite number of results, it was decided that in this semantics the same walk could not be iterated using a Kleene star. This candidate semantics was discarded due to the fact that it was returning too many results [3], a phenomenon sometimes referred to as the *Yotabyte problem*. It was finally decided that property paths in SPARQL would be evaluated under endpoint semantics, with a variation best explained in terms of rewriting: property paths are automatically rewritten using other SPARQL constructs in order to obtain a query in which all property paths have a Kleene star as outermost operator (see Section 9.3, *Property Paths and Equivalent Patterns*, of [25]). This seemingly strange decision stems from the will to handle multiplicity consistently.

In GQL[14] and SQL/GPQ[13], the keyword `SIMPLE` switches on a variant of acyclic semantics that returns not only simple walks but also simple cycles. It is usually not considered in theoretical articles, since it is computationally equivalent to acyclic semantics.

In [6], the authors present two new semantics (simple run and binding trail) that filter walks based on the underlying matching process and not only on their topology. Such semantics are outside the scope of this article (section 3.4), but the introduction of [6] is particularly relevant to our work. Indeed, their semantics are computationally better-behaved than trail, but worse than shortest. In order to explain that shortest is not always the best semantics, they use informal criteria that are not based on theoretical algorithmic efficiency. Our aim in the current text is to provide a more systematic argument.

2 Preliminaries

We assume the existence of $\mathbb{V}$, $\mathbb{E}$, $\mathbb{L}$, three disjoint countably infinite sets of *vertex identifiers*, *edge identifiers* and *labels/letters*, respectively. We will later define all databases and regular expressions over these sets. This will allow us to define several notions independently of the underlying database. We call *identifier* any element in $\mathbb{V} \cup \mathbb{E}$, *renaming* any permutation ν of $\mathbb{V} \cup \mathbb{E}$ such that $\nu(\mathbb{V}) = \mathbb{V}$ and $\nu(\mathbb{E}) = \mathbb{E}$. We call *relabeling* any permutation of $\mathbb{L}$.

2.1 Words, languages, expressions

An *alphabet* is a finite subset Σ of $\mathbb{L}$. A *word* over Σ is a finite sequence of elements of Σ; the empty word is denoted by ε. The set of all words over Σ is denoted by Σ^*. Subsets of Σ^* are called *languages* over Σ. A word $u = (a_1, \dots, a_k)$ is usually written $a_1 \cdots a_k$, and k is called the *length* of u. The *concatenation* of two words $u = a_1 \cdots a_k$ and $v = b_1 \cdots b_\ell$ is $u \cdot v = a_1 \cdots a_k b_1 \cdots b_\ell$, usually written simply uv. Concatenation is lifted to languages as follows: $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1,\ w_2 \in L_2\}$.

A *regular expression* R over Σ is any formula over the set of letters, the unary function * and the binary functions $+$ and $\cdot$ consistent with the following grammar.

$$R := \varepsilon \mid a \mid R^* \mid R \cdot R \mid R + R \qquad \text{with } a \in \Sigma \tag{1}$$

Any such R *denotes* a language $L(R)$ over Σ, defined inductively as usual. Any word in $L(R)$ is said to *conform* to R.

2.2 Walks and databases

A *walk* is a finite nonempty alternating sequence of vertices in $\mathbb{V}$ and edges in $\mathbb{E}$ that starts and ends with a vertex. We usually write such a walk $w = (v_0, e_1, v_1, \ldots, e_k, v_k)$ as $v_0 \underset{e_1}{\longrightarrow} v_1 \underset{e_2}{\longrightarrow} \cdots \underset{e_k}{\longrightarrow} v_k$. We call k the *length* of w and denote it by $\mathsf{Len}(w)$; we respectively call v_0 and v_k the *source* and *target* of w and denote them by $\mathsf{Src}(w)$ and $\mathsf{Tgt}(w)$. The pair (v_0, v_k) is the pair of *endpoints* of w, written $\mathsf{EndP}(w)$. Walks of length 0 are called *trivial* and consist of a single vertex. We denote by $w[i]$ the edge e_i, with $i \in \{1, \ldots, k\}$ and by $w\langle i \rangle$ the vertex v_i, with $i \in \{0, \ldots, k\}$. Note that walks have no particular topology. Thus, it is possible for a walk w to be inconsistent with a database $\mathcal{D}$ or even inconsistent in itself (see below).

We let Walks denote the set of all walks (independently of any database). Two walks w and w' *concatenate* if $\mathsf{Tgt}(w) = \mathsf{Src}(w')$, and their *concatenation*, written $w \cdot w'$, is the walk

$$w \cdot w' = \mathsf{Src}(w) \underset{w[1]}{\longrightarrow} \cdots \underset{w[k]}{\longrightarrow} \mathsf{Tgt}(w) \underset{w'[1]}{\longrightarrow} \cdots \underset{w'[k']}{\longrightarrow} \mathsf{Tgt}(w')$$

where $k = \mathsf{Len}(w)$ and $k' = \mathsf{Len}(w')$. Note that $w \cdot w'$ is of length $k + k'$: it has $k + k'$ edges and $k + k' + 1$ vertices (vertex $\mathsf{Tgt}(w) = \mathsf{Src}(w')$ is not repeated). We lift $\cdot$ to sets of walks as usual: $W \cdot W'$ contains all walks $w \cdot w'$ with $w \in W$ and $w \in W'$ such that w and w' concatenate. We lift the $*$ operator similarly: W^* contains all walks $w_1 \cdot w_2 \cdots w_k$ where every w_i belongs to W and every pair w_i, w_{i+1} of consecutive walks concatenate.

In this document, we model graph databases as directed, labeled, multi-edge graphs, and simply refer to them as *databases*. A *database* $\mathcal{D}$ is a tuple $(L_\mathcal{D}, V_\mathcal{D}, E_\mathcal{D}, \mathsf{Src}_\mathcal{D}, \mathsf{Tgt}_\mathcal{D}, \mathsf{Lbl}_\mathcal{D})$ where $L_\mathcal{D} \subset_{\text{fin}} \mathbb{L}$, $V_\mathcal{D} \subset_{\text{fin}} \mathbb{V}$, $E_\mathcal{D} \subset_{\text{fin}} \mathbb{E}$ and $\mathsf{Src}_\mathcal{D}$, $\mathsf{Tgt}_\mathcal{D}$ and $\mathsf{Lbl}_\mathcal{D}$ are functions respectively mapping each edge in $E_\mathcal{D}$ to its source, target and label.

A walk w is *consistent with* $\mathcal{D}$ if all elements in w are in $V_\mathcal{D} \cup E_\mathcal{D}$ and for every $0 < i \leq \mathsf{Len}(w)$, $\mathsf{Src}_\mathcal{D}(w[i]) = w\langle i-1 \rangle$ and $\mathsf{Tgt}_\mathcal{D}(w[i]) = w\langle i \rangle$. The set of all walks consistent with $\mathcal{D}$, or for short the set of *walks in* $\mathcal{D}$, is denoted by $\mathsf{Walks}(\mathcal{D})$. We say that n walks $w_1, \ldots, w_n$ are *consistent* if there is a database $\mathcal{D}$ such that $\{w_1, \ldots, w_n\} \subseteq \mathsf{Walks}(\mathcal{D})$. Whenever $n = 1$, we say w_1 is *self-consistent*. We extend $\mathsf{Lbl}_\mathcal{D}$ to $\mathsf{Walks}(\mathcal{D})$ as expected: $\mathsf{Lbl}_\mathcal{D}(w)$ is the concatenation of successive edge labels in w. When the database $\mathcal{D}$ is clear from the context, we abuse terminology and notation by using each edge $e \in E_\mathcal{D}$ as the walk $(\mathsf{Src}_\mathcal{D}(e), e, \mathsf{Tgt}_\mathcal{D}(e))$. In particular we can ask whether a walk w and an edge e concatenate, and when they do, use the notation $w \cdot e$.

Any renaming function ν is extended to walks by applying ν to all identifiers in it. It can also be extended to whole databases as follows: $\nu(\mathcal{D}) = \big(L_\mathcal{D}, \nu(V_\mathcal{D}), \nu(E_\mathcal{D}), \mathsf{Src}'_\mathcal{D}, \mathsf{Tgt}'_\mathcal{D}, \mathsf{Lbl}'_\mathcal{D}\big)$, where $\mathsf{Src}'_\mathcal{D}(x) = \nu(\mathsf{Src}_\mathcal{D}(\nu^{-1}(x)))$, $\mathsf{Tgt}'_\mathcal{D} = \nu(\mathsf{Tgt}_\mathcal{D}(\nu^{-1}(x)))$ and $\mathsf{Lbl}'_\mathcal{D}(x) = \mathsf{Lbl}_\mathcal{D}(\nu^{-1}(x))$. Similarly, any relabeling function $\lambda\colon \mathbb{L} \to \mathbb{L}$ is extended to regular expressions by applying λ to all atoms, and to databases as follows: $\lambda(\mathcal{D}) = \big(\lambda(L_\mathcal{D}), V_\mathcal{D}, E_\mathcal{D}, \mathsf{Src}_\mathcal{D}, \mathsf{Tgt}_\mathcal{D}, (\lambda \circ \mathsf{Lbl}_\mathcal{D})\big)$.

3 Regular Path Queries (RPQ) and their semantics

The concept of Regular Path Query (RPQ) consists in querying the contents of a graph database $\mathcal{D}$ thanks to regular expressions over its edge labels.

▶ **Definition 1** (Matches). *Given a regular expression R and a database $\mathcal{D}$, we denote by* $\mathsf{Matches}(\mathcal{D}, R)$ *the set of* matches of R in $\mathcal{D}$*, which is the set of all walks in $\mathcal{D}$ whose label conforms to R:* $\mathsf{Matches}(\mathcal{D}, R) = \{\, w \in \mathsf{Walks}(\mathcal{D}) \mid \mathsf{Lbl}_\mathcal{D}(w) \in L(R) \,\}$*. Given $v, v' \in V_\mathcal{D}$, we will also use the notation* $\mathsf{Matches}(\mathcal{D}, R, v, v')$ *for the set of the walks w in* $\mathsf{Matches}(\mathcal{D}, R)$ *such that* $\mathsf{EndP}(w) = (v, v')$*.*

3.1 Arbitrary semantics

▶ **Definition 2.** *A* semantics S *is a function mapping a database* $\mathcal{D}$ *and a query* R *to a finite subset* $S(\mathcal{D}, R)$ *of* $\mathsf{Matches}(\mathcal{D}, R)$.

Aside from homomorphism semantics, which does not fit our framework because it does not return walks, the two most popular RPQ semantics are shortest-walk semantics and trail semantics. We define them below.

▶ **Example 3** (Trail semantics Tr)**.** Trail semantics is the semantics of Cypher [10] and may be used in GQL [8, 14] thanks to the keyword `TRAIL`. It is defined as follows. A walk w is a *trail* if it has no duplicate edge, i.e. for all $i \neq j$, $w[i] \neq w[j]$. Trail semantics is defined by: $\mathsf{Tr}(\mathcal{D}, R) = \{\, w \in \mathsf{Matches}(\mathcal{D}, R) \mid w \text{ is a trail} \,\}$.

▶ **Example 4** (Shortest-walk semantics Sh)**.** Shortest semantics is the semantics of GSQL [24] and PGQL [23]; it may be used in GQL [8, 14] thanks to the keyword `SHORTEST`. Let $\mathcal{D}$ be a database and R a query. For every $s, t \in V_{\mathcal{D}}$, we write $\ell_{(s,t)}$ for the length of the shortest walk in $\mathsf{Matches}(\mathcal{D}, R, s, t)$. Then, shortest semantics is defined by: $\mathsf{Sh}(\mathcal{D}, R) = \bigcup_{s,t \in V_{\mathcal{D}}} \left\{\, w \in \mathsf{Matches}(\mathcal{D}, R, s, t) \;\middle|\; \mathsf{Len}(w) = \ell_{(s,t)} \,\right\}$.

We focus on semantics that are *walk-based*, as defined below. Intuitively, such semantics act as post-filters on matches: they have no access to the database, the regular expression, or on the why and how some walk was matched by the query, but only to the walks themselves.

▶ **Definition 5.** *A semantics* S *is* walk-based *if there is a function* $r_S \colon 2^{\mathsf{Walks}} \to 2^{\mathsf{Walks}}$ *such that* $S(D, R) = r_S(\mathsf{Matches}(D, R))$.

Even though the previous definition seems very general, walk-based semantics still have to output walks which match the query due to Definition 2. Instances of walk-based semantics are Tr and Sh given above, as well as shortest-trail semantics defined below. On the other hand, we will give in Section 3.4 a few examples of semantics that are not walk-based, such as binding-trail semantics and cheapest-walk semantics.

▶ **Example 6** (Shortest-trail semantics ShT)**.** Shortest-trail semantics may be used in GQL [8, 14] thanks to the keyphrase `SHORTEST TRAIL`. For every $W \subseteq 2^{\mathsf{Walks}}$, and every vertices $s, t \in \mathbb{V}$, we let $\ell_{(s,t)}(W)$ denote the length of the shortest trail w in W such that $\mathsf{EndP}(w) = (s, t)$, if it exists. Then, shortest-trail semantics is defined by: $\mathsf{ShT}(\mathcal{D}, R) = r_{\mathsf{ShT}}(\mathsf{Matches}(D, R))$, where $r_{\mathsf{ShT}}(M)$ contains all trails w in M such that $\mathsf{Len}(w) = \ell_{\mathsf{EndP}(w)}(M)$.

For every $\mathcal{D}$ and R, $\mathsf{ShT}(\mathcal{D}, R)$ is clearly a subset of $\mathsf{Tr}(\mathcal{D}, R)$, since it only returns trails matching R in $\mathcal{D}$. However, it may not be a subset of $\mathsf{Sh}(\mathcal{D}, R)$ since the shortest walk matching R between two vertices of $\mathcal{D}$ may or may not be a trail.

3.2 Filter-based semantics

▶ **Definition 7.** *A semantics* S *is called* filter-based *if there exists a function* $f_S \colon \mathsf{Walks} \to \{\top, \bot\}$ *such that* $S(\mathcal{D}, R) = \{\, w \in \mathsf{Matches}(\mathcal{D}, R) \mid f_S(w) = \top \,\}$.

Note that every filter-based semantics is also walk-based. Trail semantics (Ex. 3) is the paramount example of a filter-based semantics. It is defined by the function f_{Tr}, that maps a walk w to $\bot$ if there are $i < j$ such that $w[i] = w[j]$, or to $\top$ otherwise. However, it is quite easy to see that neither shortest-walk semantics (Ex. 4) nor shortest-trail semantics (Ex. 6) are filter-based. We give below three examples of filter-based semantics.

▶ **Example 8** (Acyclic semantics Ac)**.** Acyclic semantics (corresponding to the `ACYCLIC` keyword in GQL) is the semantics defined by the filter f_{Ac} that maps a walk w to $\bot$ if there are $i < j$ such that $w\langle i\rangle = w\langle j\rangle$, and to $\top$ otherwise.

▶ **Example 9** (Simple-cycle-or-walk semantics SWC)**.** Simple-cycle-or-walk semantics is a variant of Ac that corresponds to keyword `SIMPLE` in GQL. The difference is that simple cycles are also returned. It is defined by the filter f_{SWC} that maps a walk w to $\bot$ if there are $i < j$ such that$(i, j) \neq (0, \mathsf{Len}(w))$ and $w\langle i\rangle = w\langle j\rangle$, and to $\top$ otherwise.

▶ **Example 10** (Two-acyclic-sem 2Ac)**.** Two-acyclic semantics is a variant of Ac where each vertex is allowed to appear once or twice, but not more. It is defined by the filter below.

$$f_{\mathsf{2Ac}} \colon w \mapsto \begin{cases} \bot & \text{if there are } i < j < k \text{ such that } w\langle i\rangle = w\langle j\rangle = w\langle k\rangle \\ \top & \text{otherwise} \end{cases}$$

3.3 Order-based semantics

First, let us recall a few definitions about partial orders. A *partial order* is a binary, transitive, antisymmetric and reflexive relation. For any partial order $\preceq$ we will denote by $\prec$ the associated *strict partial order*. A *well-partial-order* on a set X, or *wpo*, is a partial order $\preceq$ such that in every infinite sequence $x_1, x_2, \ldots, x_k, \ldots$ of elements from X there are two positions $i < j$ such that $x_i \preceq x_j$. In other words, there exists no infinite strictly decreasing sequence nor any infinite sequence in which all elements are incomparable w.r.t. $\preceq$. It is well-known that any subset of X admits finitely many minimal elements w.r.t. a wpo over X, and that the restriction of any wpo over X to a subset of X is also a wpo.

▶ **Definition 11** (Order-based semantics)**.** *Let $\preceq$ be a partial order on* Walks *such that for every database $\mathcal{D}$, $\preceq$ is a wpo on* $\mathsf{Walks}(\mathcal{D})$*. Such an order is said to be* suitable*. The* semantics based on $\preceq$*, denoted by $S_\preceq$ is defined as follows for every database $\mathcal{D}$ and regular expression R : $S_\preceq(\mathcal{D}, R) = \bigcup_{s,t \in V_\mathcal{D}} \min_\preceq(\mathsf{Matches}(\mathcal{D}, R, s, t))$*

Note that a suitable order $\preceq$ is a wpo on $\mathsf{Matches}(\mathcal{D}, R, s, t)$, since $\mathsf{Matches}(\mathcal{D}, R, s, t)$ is a subset of $\mathsf{Walks}(\mathcal{D})$. It follows that $S_\preceq(\mathcal{D}, R)$ is always finite. Note also that the ordering of two walks w, w' w.r.t. $\preceq$ is irrelevant whenever w and w' have different endpoints. The same is true for walks w and w' that are inconsistent. It is sometimes useful to trim such irrelevant pairs from the orders we consider.

▶ **Definition 12.** *For every suitable order $\preceq$ we let $\preceq_t$ denote the order defined by: $w \preceq_t w'$ iff w and w' are consistent walks with the same endpoints such that $w \preceq w'$. We say that $\preceq$ is* trimmed *if it is equal to $\preceq_t$*

▶ **Lemma 13.** *Each order-based semantics $S_\preceq$ is identical to $S_{\preceq_t}$.*

A simple example of order-based semantics is the shortest-walk semantics Sh (Ex. 4). It is based on the partial order $\preceq_{\mathsf{Sh}}$, the reflexive closure of the strict partial order defined by $w \prec_{\mathsf{Sh}} w'$ if $\mathsf{Len}(w) < \mathsf{Len}(w')$. It is *not* a wpo on Walks, since there are infinitely many walks of a given length, which are all incomparable w.r.t. $\prec_{\mathsf{Sh}}$. However, one may verify that it is indeed a wpo on $\mathsf{Walks}(\mathcal{D})$ for every database $\mathcal{D}$, and that it is therefore suitable. Let us now define another order-based semantics.

▶ **Definition 14** (Subwalk order)**.** *Let F be the following factor-inserting binary relation: $w\ F\ w'$ if there are three walks w_1, w_2, w_3 such that $w = w_1 \cdot w_3$ and $w' = w_1 \cdot w_2 \cdot w_3$. We define the* subwalk order $\sqsubseteq$ *as the reflexive and transitive closure of F.*

It is straightforward to check that $\sqsubseteq$ is a partial order on Walks. Moreover, one may show that $\sqsubseteq$ is a suitable order using Higman's Lemma [11] (see also [15, Sect. 2.2]). We call *subwalk-minimal* and denote by SM the semantics based on $\sqsubseteq$.

▶ **Lemma 15.** *For every database $\mathcal{D}$, $\sqsubseteq$ is a wpo on* Walks($\mathcal{D}$)

The reason for considering the subwalk order $\sqsubseteq$ and the associated semantics SM stems from the property of subwalk coverage guarantee, which we define later. Preliminary results about the complexity of query evaluation under SM can be found in [16]. To conclude this section, let us mention another interesting order-based semantics.

▶ **Example 16** (Shortlex Semantics ShL)**.** Let $\preceq_{\text{lex}}$ be the lexicographic ordering of walks based on some arbitrary order over $\mathbb{V}\cup\mathbb{E}$. Let $\preceq_{\text{shortlex}}$ be the the corresponding length-lexicographic order: $w \preceq_{\text{shortlex}} w'$ if either $|w| < |w'|$ or ($|w| = |w'|$ and $w \preceq_{\text{lex}} w'$). Even though $\preceq_{\text{lex}}$ is not suitable, it may be verified that $\preceq_{\text{shortlex}}$ is. The corresponding shortlex-minimal semantics ShL is a possible implementation of the `ANY SHORTEST` keyphrase in GQL [8, 14].

3.4 Examples of some out-of-scope semantics

This document focuses on semantics that output walks (Def. 3) and are walk-based (Def. 5), but it is of course not the only possible approach to defining meaningful RPQ semantics. The most studied semantics for RPQs in scientific literature is *homomorphism semantics* [2], which outputs the endpoints of matches only, hence does *not* fit our notion of semantics. On the other hand, [6] propose *run-based* semantics, i.e. semantics S which require information on the way a walk was matched by the query in order to decide whether it belongs to $S(\mathcal{D}, R)$. We present one such run-based semantics below. Afterwards, we define another semantics that is not walk-based. It is loosely inspired by the keyword `CHEAPEST`, discussed during the elaboration of GQL in internal documents, but was not retained for version 1 (see language opportunity 052 in [13, 14]).

▶ **Example 17** (Binding-trail semantics BT [6])**.** Let $\mathcal{D}$ be a database, R a query, and k is the number of atoms in R. A *binding walk (in $\mathcal{D}$ matching R)* is a walk $w \in \mathsf{Matches}(\mathcal{D}, R)$ together with a function $\{1, \dots, \mathsf{Len}(w)\} \to \{1, \dots, k\}$ that maps each edge-position in w to the position of the atom in R that matched that edge. A *binding trail (in $\mathcal{D}$ matching R)* is a binding-walk (w, α) such that there are no distinct i, j such that $\alpha(i) = \alpha(j)$ and $w[i] = w[j]$. Binding-trail semantics, denoted by BT, is defined as follows.

$$\mathsf{BT}\colon (\mathcal{D}, R) \mapsto \{\, w \mid \exists \alpha \ \text{ s.t. } \ (w, f) \text{ is a binding trail in } \mathcal{D} \text{ matching } R \,\} \tag{2}$$

See [6] for a more formal definition. A similar semantics called *simple-run semantics*, based on the states of a finite automaton representing the query, is defined in the same paper.

▶ **Example 18** (Cheapest-Walk Semantics ChW_c)**.** Let $c\colon \mathbb{L} \to (\mathbb{N} - \{0\})$ be a fixed cost function, which we lift to words by: $c(a_1 \cdots a_k) = \sum_{i=1}^{k} c(a_i)$. Then, the cost of a walk in a database $\mathcal{D}$ and query R is the cost of its label in $\mathcal{D}$. Cheapest-walk semantics, ChW_c, is defined like shortest-walk semantics, using the cost of matches instead of their length.

Note that ChW_c is not walk-based (unless the cost function c is constant), because a walk provides no information about the label its edges bear.

4 Properties

In this section we define and investigate several possible properties of RPQ semantics, in order to help compare and classify them. After listing a few basic properties, we distinguish properties relative to some forms of monotonicity, continuity, compatibility with rational operators, and coverage. Then, we systematically investigate the inclusions between sets of result for the same query according to various semantics. We conclude this section with a general discussion regarding the set of properties any "reasonable" semantics should possess.

4.1 Basics

We first list a few basic properties, which are almost always required for a semantics to be considered useful in our model.

▶ **Definition 19.** *A semantics S is called* identifier-independent *if $\nu(S(\mathcal{D}, R)) = S(\nu(\mathcal{D}), R)$, for every renaming ν, database $\mathcal{D}$ and regular expression R. Similarly, it is* label-independent *if $S(\mathcal{D}, R) = S(\lambda(\mathcal{D}), \lambda(R))$ for every relabeling λ, database $\mathcal{D}$ and regular expression R.*

For instance, a semantics selecting results based on some arbitrary ordering on vertex and edge identifiers (e.g., ShL in Ex. 16), or based on the presence of similar arbitrary criteria, would *not* be identifier-independent. Identifier-independence is related to *genericity* in the classical database setting (see [1] for instance). Indeed, if we consider a query q consisting of a regular expression and a semantics S to interpret it, then S is identifier-independent if and only if q is $\emptyset$-generic.

Similarly, label-dependent semantics select their results based on arbitrary information regarding labels. Moreover, no label-dependent semantics is walk-based, as stated below.

▶ **Lemma 20.** *Every walk-based semantics is label-independent.*

In this work, we will always be considering identifier- and label-independent semantics, unless explicitly stated otherwise.

▶ **Definition 21.** *A semantics S is* expression-independent *if, for every database $\mathcal{D}$ and every expressions R_1, R_2 such that $L(R_1) = L(R_2)$, $S(\mathcal{D}, R_1) = S(\mathcal{D}, R_2)$.*

Expression-independent semantics select their results on the sole basis of the language of the regular expression constituting the RPQ. In some settings, it may make sense to instead let the selection of results depend on the structure of the expression itself. Binding-trail semantics (Ex. 17) is one such example. However, this is out of the scope of this work, since all walk-based semantics are expression-independent, as stated below.

▶ **Lemma 22.** *Every walk-based semantics is expression-independent.*

▶ **Definition 23.** *A semantics S is called* unbounded *if, for every integer n, there is a database D and a regular expression R such that $S(D, R)$ contains a walk of length n or more. Otherwise S is called* bounded*.*

Of course, we expect any reasonable semantics to be unbounded since a bounded semantics never returns a walk longer than some constant.

4.2 Monotony

▶ **Definition 24** (Monotony, co-monotony and restrictibility). *A semantics S is said to be* monotonous *if $S(\mathcal{D}, R) \subseteq S(\mathcal{D}', R)$, for every regular expression R and databases $\mathcal{D}, \mathcal{D}'$ such that $\mathcal{D} \subseteq \mathcal{D}'$. It is* co-monotonous *if $S(\mathcal{D}, R) \supseteq \big(S(\mathcal{D}', R) \cap \mathsf{Walks}(D)\big)$, for every regular expression R and databases $\mathcal{D}, \mathcal{D}'$ such that $\mathcal{D} \subseteq \mathcal{D}'$. It is* restrictible *if both hold.*

These notions are about the behavior of the semantics when a database is enriched with new vertices and edges. A monotonous semantics does not remove results when the database is enriched. A co-monotonous semantics does not add results that are unrelated to the new elements. Finally, a restrictible semantics means that one is able to compute part of the result, given part of the database. A simple verification shows that all semantics defined above are co-monotonous. This is in fact true of all filter-based and order-based semantics. Additionally, filter-based semantics are also monotonous, and therefore restrictible.

▶ **Lemma 25.** *Every filter-based semantics is restrictible, and every order-based semantics is co-monotonous.*

We expect all "reasonable" semantics to be co-monotonous. In fact, semantics that are not co-monotonous are usually quite strange, as illustrated below.

▶ **Example 26** (Log-length semantics). The *log-length* semantics LL returns all matches that are shorter than $\log_2(|\mathcal{D}|)$. This semantics has the property that the size of $\mathsf{LL}(\mathcal{D}, R)$ is always polynomial in the size of $\mathcal{D}$. It is not co-monotonous since doubling the size of $\mathcal{D}$ (for instance by adding enough isolated vertices) may return longer walks which do not necessarily use the new isolated vertices. Note however that LL is monotonous, since no previous result may disappear when enriching the database.

Although LL is not walk-based, one could consider a variant that is. Indeed, it could return the walks shorter than $\log_2(|X(\mathcal{D}, R)|)$, where $X(\mathcal{D}, R)$ is the set of all edges appearing in some walk in $\mathsf{Matches}(\mathcal{D}, R)$.

As for monotony, the landscape is more interesting. We already saw that all filter-based semantics are monotonous (Lemma 25). However, it is notable that shortest semantics is not.

▶ **Lemma 27.** Sh *is not monotonous.*

At first glance, it may seem that all order-based semantics $S_{\preceq}$ are bound to be non-monotonous, since it is enough to make a shortcut (w.r.t $\preceq$) using new vertices and edges. However, a simple verification shows that the (order-based) subwalk-minimal semantics SM (Def. 14) is indeed monotonous.

We now characterize monotonous order-based semantics using a notion of element-inclusion (Proposition 28). In the following, we denote by $\mathsf{VertSet}(w)$ (resp. $\mathsf{EdgeSet}(w)$) the set of vertices (resp. of edges) occurring in w. We also use $\mathsf{ElemSet}(w)$ (resp. $\mathsf{ElemBag}(w)$) to denote the set (resp. the bag) of vertices and edges appearing in w. We say that an order $\preceq$ on Walks *respects element inclusion* if $w \preceq w'$ implies $\mathsf{ElemSet}(w) \subseteq \mathsf{ElemSet}(w')$.

▶ **Proposition 28.** *Let $S_{\preceq}$ be a semantics based on a trimmed order $\preceq$. Then $\preceq$ respects element inclusion iff $S_{\preceq}$ is monotonous.*

The proof of the forward direction of Proposition 28 is straightforward. The backward direction requires some extra machinery, namely the *characteristic database* and *characteristic expression* of sets of walks, which are also useful in the proofs of several subsequent results. Even though all proofs are omitted due to space constraints, we include these particular notions in the appendix, page 18, to help interested readers to reconstructs proofs.

Consider the relation $\preceq_1$ defined as: $w \preceq_1 w'$ if $\mathsf{ElemSet}(w) \subseteq \mathsf{ElemSet}(w')$. It is not antisymmetric, hence not a partial order. Now consider the relation $\preceq_2$ defined as: $w \prec_2 w'$ if $\mathsf{ElemSet}(w) \subsetneq \mathsf{ElemSet}(w')$. Although $\preceq_2$ is a partial order, it is **not** a well partial order on $\mathsf{Walks}(D)$ whenever $\mathcal{D}$ contains a cycle. We give below two examples of semantics based on an order that preserves element-inclusion. Example 29 takes multiplicity into account. Example 30 refines $\preceq_2$ by adding a tiebreaker for walks with identical element sets.

▶ **Example 29** (Minimal-multiset semantics MM)**.** Let MM be the semantics based on the order $\preceq$ defined by: $w \prec w'$ if $\mathsf{ElemBag}(w) \subsetneq \mathsf{ElemBag}(w')$. Recall that $\sqsubseteq$ denotes the subwalk order on walks (Def. 14). A simple verification yields that $w \sqsubseteq w' \implies w \preceq w'$. Then, $\preceq$ is a wpo on $\mathsf{Walks}(\mathcal{D})$ for each database $\mathcal{D}$, because so is $\sqsubseteq$ (Lemma 15). Indeed, let $(w_1, w_2, \ldots)$ be an infinite sequence of walks in $\mathcal{D}$. Since $\sqsubseteq$ is a wpo on $\mathcal{D}$, there are $i < j$ such that $w_i \sqsubseteq w_j$, which implies that $w_i \preceq w_j$.

▶ **Example 30** (Shortest-minimal-set semantics ShMS)**.** Let ShMS be the semantics based on the order $\preceq$ defined by:

$$w \prec w' \iff \begin{cases} \text{either } \mathsf{ElemSet}(w) \subsetneq \mathsf{ElemSet}(w') \\ \text{or } \mathsf{ElemSet}(w) = \mathsf{ElemSet}(w') \text{ and } \mathsf{Len}(w) < \mathsf{Len}(w') \end{cases}$$

The semantics MM and ShMS are finite, identifier-independent and restrictible. The complexity of query evaluation under them is explored in [17].

4.3 Continuity

In this section, we attempt to capture the property for a semantics S to behave "reasonably" at the limit. Essentially, the sequence $\big(S(\mathcal{D}, a + a^2 + \cdots + a^k)\big)_{k\in\mathbb{N}}$ should to tend to $S(\mathcal{D}, a^*)$ when k tends to ∞. More precisely, results of non-decreasing sequences of RPQs (with respect to language inclusion) should eventually stabilize to some fixed set W, and furthermore whenever the whole sequence is captured by a single RPQ, then the set of results over that RPQ should be W itself. These conditions prevent S to change gears and arbitrarily alter the set of results when moving to the limit. As we show below, it turns out that all filter-based or order-based semantics have these properties.

▶ **Definition 31.** *A semantics S is said to be* stabilizing *if, for every database $\mathcal{D}$ and every sequence of regular expressions $\big(R_i\big)_{i\in\mathbb{N}}$ such that $\big(L(R_i)\big)_{i\in\mathbb{N}}$ is non-decreasing, $\big(S(\mathcal{D}, R_i)\big)_{i\in\mathbb{N}}$ is ultimately equal to some set W, and we say that $\big(S(\mathcal{D}, R_i)\big)_{i\in\mathbb{N}}$ stabilizes to W.*

Moreover, when S is stabilizing, it is said to be continuous *if, whenever there exists a sequence of regular expressions $\big(R_i\big)_{i\in\mathbb{N}}$ as above and a regular expression R_∞ such that $L(R_\infty) = \bigcup_{i\in\mathbb{N}} L(R_i)$, it holds that $W = S(\mathcal{D}, R_\infty)$.*

▶ **Proposition 32.** *All order-based and filter-based semantics are continuous.*

We expect any "reasonable" (walk-based) semantics to be continuous. Indeed, a non-continuous walk-based semantics must treat finite sets and infinite sets differently. To give the reader a flavor of what non-continuous semantics might look like, we give below toy examples of non-stabilizing semantics (Ex. 33) and of stabilizing but non-continuous semantics (Ex. 34).

▶ **Example 33** (Giving-Up Semantics)**.** Let $\mathsf{GU}(\mathcal{D}, R)$ be the semantics which returns the whole set $\mathsf{Matches}(\mathcal{D}, R)$ if this set is finite, and the empty set of walks otherwise. This implies that $\mathsf{GU}(\mathcal{D}, R)$ is always a finite set.

Given expressions $\left(R_i = a^{\leq i}\right)_{i\in\mathbb{N}}$ (each matching all walks labeled by some word a^j with $j \leq i$), if $\mathcal{D}$ contains an a-labeled loop e from some vertex v to itself, then each $\mathsf{GU}(\mathcal{D}, R_i)$ contains the new walk $(ve)^i v$. Hence GU is clearly not stabilizing (and thus not continuous: note that in the above case $\mathsf{GU}(\mathcal{D}, a^*) = \emptyset$).

Semantics that are stabilizing but non-continuous are even stranger. The typical example is a semantics that changes behaviour in the presence of an infinite set.

▶ **Example 34** (Stabilizing non-continuous semantics)**.** Let S be the semantics defined by:

$$S(\mathcal{D}, R) = \begin{cases} \mathsf{Sh}(\mathcal{D}, R) & \text{if } \mathsf{Matches}(\mathcal{D}, R) \text{ is finite,} \\ \mathsf{Tr}(\mathcal{D}, R) & \text{if } \mathsf{Matches}(\mathcal{D}, R) \text{ is infinite.} \end{cases} \tag{3}$$

It is stabilizing because, for every sequence of regular expressions $\left(R_i\right)_{i\in\mathbb{N}}$ such that $\left(L(R_i)\right)_{i\in\mathbb{N}}$ is non-decreasing, the sequence $(\mathsf{Matches}(\mathcal{D}, R_i))_{i\in\mathbb{N}}$ either contains only finite sets, or there is a bound b such that $\mathsf{Matches}(\mathcal{D}, R_i)$ is finite only if $i \leq b$. In either cases, $(S(\mathcal{D}, R_i))_{i\in\mathbb{N}}$ stabilizes because Sh and Tr do. It is not continuous, as can be seen by considering the sequence $(a^{\leq i})_{i\in\mathbb{N}}$ and the limit expression a^*.

Note that a semantics that is not expression-independent, such as binding-trail semantics (Ex. 17), may be discontinuous but still be considered meaningful.

4.4 Compatibility with rational operators

In this section we study interactions between the structure of an RPQ as a regular expression and the set of results. We define in particular the notions of composability and decomposability of a semantics with respect to a given RPQ operator. The intuition behind composability, stated in the case of concatenation, is as follows. A semantics is $\cdot$-decomposable if any result for $R \cdot R'$, that is a walk w in $S(D, R \cdot R')$, can be *decomposed* into a result for R and a result for R'. This means that there exist $w' \in S(D, R)$ and $w'' \in S(D, R')$ such that $w = w' \cdot w''$. Conversely, a semantics S is $\cdot$-composable if the result sets for R and R' can be *composed* into (a subset of) results for $R \cdot R'$. These notions are defined in more details below.

▶ **Definition 35.** *We say that S is* $\cdot$-composable *(resp.* $\cdot$-decomposable*) if Equation* (4) *(resp. Equation* (5)*), below, holds for every database $\mathcal{D}$ and RPQs R, R'.*

$$S(\mathcal{D}, R \cdot R') \supseteq S(\mathcal{D}, R) \cdot S(\mathcal{D}, R') \tag{4}$$

$$S(\mathcal{D}, R \cdot R') \subseteq S(\mathcal{D}, R) \cdot S(\mathcal{D}, R') \tag{5}$$

If both (4) *and* (5) *hold, we say that S is* compatible with $\cdot$*, or* $\cdot$-compatible *for short. Similarly, we define* $+$-composability*,* $+$-decomposability*,* $+$-compatibility *from* (6)*, below.*

$$S(\mathcal{D}, R + R') \supseteq S(\mathcal{D}, R) \cup S(\mathcal{D}, R') \quad \textit{and/or} \quad S(\mathcal{D}, R + R') \subseteq S(\mathcal{D}, R) \cup S(\mathcal{D}, R') \tag{6}$$

and $*$-composability*,* $*$-decomposability*,* $*$-compatibility *from* (7) *below.*

$$S(\mathcal{D}, R^*) \supseteq (S(\mathcal{D}, R))^* \quad \textit{and/or} \quad S(\mathcal{D}, R^*) \subseteq (S(\mathcal{D}, R))^* \tag{7}$$

Finally, we say that a semantics is decomposable *if it is decomposable w.r.t. $\cdot$, $+$ and $*$.*

▶ **Definition 36.** *We say that a semantics S is* compatible with atoms*, or a-compatible, if $S(\mathcal{D}, a) = \mathsf{Matches}(\mathcal{D}, a)$, for every $a \in \mathbb{L}$ and database $\mathcal{D}$.*

We give below several statements regarding compatibility with rational operators. First, Lemma 37 states that all filter-based and order-based semantics are +-decomposable, and that +-composability is essentially equivalent to being filter-based.

▶ **Lemma 37** (Compatibility with $+$)**.**

1. *Every filter-based semantics is $+$-compatible.*
2. *Every order-based semantics is $+$-decomposable but not $+$-composable.*
3. *For every $+$-compatible semantics S, there exists a filter-based semantics S' such that for every database $\mathcal{D}$, regular expression R and walk w of positive length: $w \in S(\mathcal{D}, R)$ iff $w \in S'(\mathcal{D}, R)$.*

▶ Remark 38. The equivalence in the statement of Item 3 is not true for walks of length 0. Indeed, consider a semantics S defined as follows. We denote by $\mathsf{Walks}_0(\mathcal{D})$ the walks in $\mathsf{Walks}_0(\mathcal{D})$ of length 0. Then, we define $S(\mathcal{D}, R) = \mathsf{Walks}_0(\mathcal{D})$ if $\varepsilon \in L(R)$ and $\mathcal{D}$ has an even number of vertices; or $S(\mathcal{D}, R) = \emptyset$ otherwise. It may be checked that S is walk-based and $+$-compatible, although it is clearly not filter-based.

As hinted previously, the two families of filter-based and order-based semantics are disjoint. It is indeed a corollary of Lemma 37.

▶ **Corollary 39.** *There is no semantics S that is both order-based and filter-based.*

Next, we state in Lemmas 40 and 41 that no order-based or filter-based semantics is $\cdot$-composable or $*$-composable, under very weak requirements.

▶ **Lemma 40** (Composability with $\cdot$)**.**

1. *No order-based semantics is $\cdot$-composable.*
2. *No identifier-independent, unbounded, filter-based semantics is $\cdot$-composable.*

▶ **Lemma 41** (Composability with $*$)**.**

1. *No order-based semantics is $*$-composable.*
2. *No identifier-independent, unbounded, filter-based semantics is $*$-composable.*

We conjecture that no "reasonable" semantics is composable w.r.t. $\cdot$ nor $*$. However, it is not easy to define the minimal set of "reasonable" properties that implies $*$-non-composability or $\cdot$-non-composability.

To conclude this section, Proposition 42 below summarizes properties regarding the compatibilities of some of the semantics defined in the previous sections.

▶ **Proposition 42.** *Among the different composability and decomposability properties, only the following hold for the mentioned semantics:*

1. Ac *is only $+$-compatible and $(\cdot, *)$-decomposable.*
2. Tr *and* SWC *are only $(a, +)$-compatible and $(\cdot, *)$-decomposable.*
3. Sh, SM, MM *are only a-compatible and $(\cdot, +, *)$-decomposable.*
4. BT *is only $(a, \cdot, +)$-compatible and $*$-decomposable.*

Recall that BT is not walk-based (Ex. 17), hence although it is $\cdot$-composable (item 4), it is not a counterexample to our conjecture that any "reasonable" walk-based semantics cannot be $\cdot$-composable.

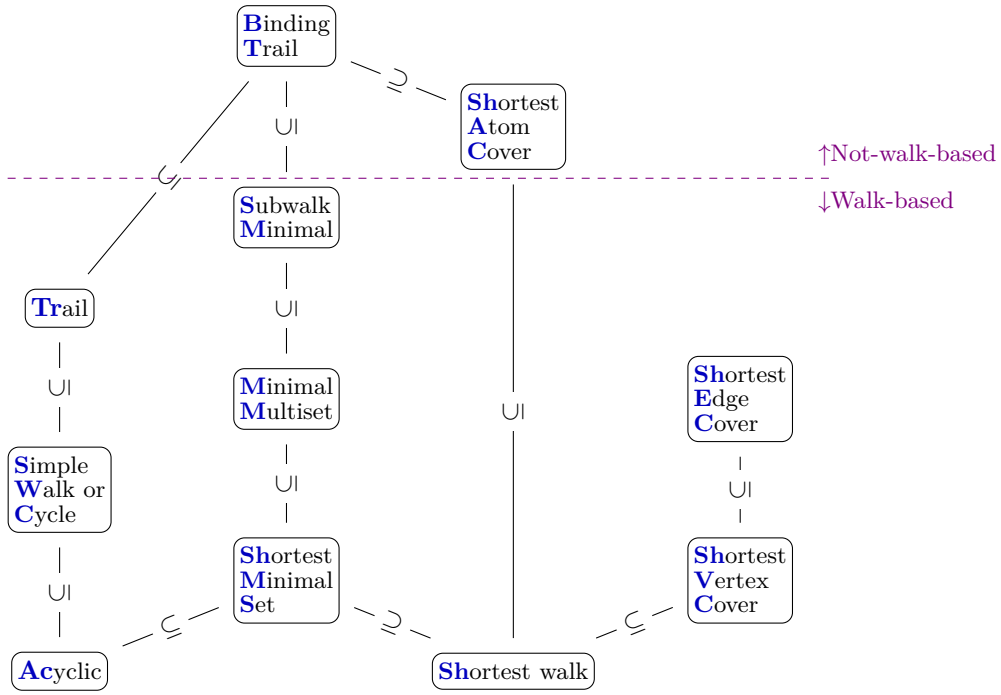

Figure 1 Inclusions of interesting semantics.

4.5 Inclusion of semantics

▶ **Definition 43.** *Given two semantics S_1, S_2, we say that S_1* is included *in S_2, and denote by $S_1 \subseteq S_2$, if it holds $S_1(\mathcal{D}, R) \subseteq S_2(\mathcal{D}, R)$ for every database $\mathcal{D}$ and query R. We say that S_1 and S_2 are* incomparable *if neither $S_1 \subseteq S_2$ nor $S_1 \supseteq S_2$ hold.*

Figure 1 shows the inclusions of the main semantics defined in this document. All inclusions are strict and the absence of a path means that semantics are incomparable. Due to space constraints we do not include proofs of these claims. All of them are easy, and some may be found in [6] and [16].

Note that it is not easy to interpret the fact that two semantics S_1, S_2 satisfy $S_1 \subseteq S_2$. It is not always clear whether the results in the difference are mostly meaningful or redundant. Moreover, there are cases where the difference seems very small while query evaluation complexity differ (e.g., MM [17] and SM[16]).

4.6 Coverage

Recall that a semantics S returns a finite subset of results ($S(\mathcal{D}, R)$) among the possibly infinite set of matches ($\mathsf{Matches}(\mathcal{D}, R)$). Some information is bound to be lost in the process. Then, an interesting question is to characterize the *coverage* of a given semantics S, by which we mean *how well* $S(\mathcal{D}, R)$ *represents* $\mathsf{Matches}(\mathcal{D}, R)$. In this section, we present a few possible definitions in an attempt to capture this intuition. On the one hand, coverage can be expressed in terms of vertices (resp. edges) of the database: all vertices (resp. edges) in some match must also appear in some result. On the other hand, coverage may be expressed in terms of subwalks: for every walk w in $\mathsf{Matches}(\mathcal{D}, R)$, the set $S(\mathcal{D}, R)$ must contain at least one subwalk of w. Finally, one may be interested in covering the expression instead: if some atom of R is useful to match any walk, then some walk using it must be in $S(\mathcal{D}, R)$.

4.6.1 Vertex and edge coverage

▶ **Definition 44.** *A walk w* covers *a vertex v if v occurs in w. A set of walks W* covers *v if some walk in W covers v. We say that a semantics S is* vertex-covering *if for every $\mathcal{D}, R$ all vertices covered by $\mathsf{Matches}(\mathcal{D}, R)$ are also covered by $S(\mathcal{D}, R)$.*

It can be shown that no filter-based semantics is vertex-covering, and any order-based semantics can be made vertex-covering, at the price of an exponential blow-up in the size of the set of results.

▶ **Lemma 45.** *No filter-based semantics is vertex-covering.*

▶ **Lemma 46.** *For any order-based semantics S based on order $\preceq$, there exists a vertex-covering order-based semantics S' based on some order $\preceq'$ that is a refinement of $\preceq$.*

The idea behind Lemma 46 is to define $\preceq'$ as: $w \preceq' w'$ if both $w \preceq w'$ and the sets of vertices appearing in w and w' are equal. This construction induces an exponential blow-up in the size of the result set on some databases, since all subsets of vertices over which there exists a match have to be considered. For instance, over the a-labeled n-clique with query a^*, where the order-based semantics Sh returns a single result per pair of endpoints, the semantics obtained via this construction would return at least one result per non-empty set of vertices.

We now define a vertex-covering semantics which is neither filter-based nor (strictly speaking) order-based, while avoiding the previous blow-up in the size of results.

▶ **Example 47** (Shortest-vertex-cover Semantics)**.** Let $\preceq_{\mathsf{Sh}}$ be the order used to define Sh.

$$\mathsf{ShVC}(\mathcal{D}, R) = \bigcup_{(s,t,v)\in(V_{\mathcal{D}})^3} \min_{\preceq_{\mathsf{Sh}}}\{w \in \mathsf{Matches}(\mathcal{D}, R) \mid w \text{ starts in } s, \text{ ends in } t \text{ and covers } v\}$$

Note that for each vertex, this ShVC returns *all* minimal-length matches covering this vertex. The complexity of query evaluation under ShVC is studied in [16]. The following lemma summarizes some of its other properties.

▶ **Lemma 48.** ShVC *is*

1. *identifier-independent and unbounded;*
2. *neither filter-based nor order-based;*
3. *co-monotonous, but not monotonous;*
4. *continuous;*
5. *θ-decomposable and not θ-composable for all $\theta \in \{\cdot, +, *\}$.*

Note that one may replace $\preceq_{\mathsf{Sh}}$ by any suitable order in Example 47. This opens up a whole new way of defining semantics that we leave for future work.

The situation with respect to edge coverage is much like the one for vertex coverage. It may be defined similarly, and no semantics defined in this document is edge-covering. A shortest covering semantics, the shortest-edge-cover semantics ShEC, may be defined by adapting Example 47.

4.6.2 Subwalk guarantee

▶ **Definition 49.** *A semantics S possesses the* subwalk guarantee *if for every database $\mathcal{D}$, query R, and walk w such that $w \in \mathsf{Matches}(\mathcal{D}, R)$ there exists a subwalk of w that belongs to $S(\mathcal{D}, R)$.*

This property was introduced in Lemma 13 of [6], which states that BT possesses it. This led to the definition of SM, which possesses it by definition. Other semantics discussed in this document do not possess it.

▶ **Lemma 50.**

1. SM *and* BT *possess the subwalk coverage guarantee.*
2. ShVC, ShMS, MM *and* Sh *do not possess the subwalk coverage guarantee.*
3. *No filter-based semantics possesses the subwalk guarantee.*

4.6.3 Atom coverage

Another, orthogonal type of coverage is worth mentioning. It covers the regular expression instead of the match set. Intuitively, we say a semantics S is atom-covering if for every $\mathcal{D}$ and R, and for every letter occurrence in R, $S(\mathcal{D}, R)$ contains at least one walk matched using this letter occurrence if any such walk exists. We give below a formal definition.

▶ **Definition 51.** *Let R be a regular expression over Σ. We denote by k the number of atoms in R, and by a_i the i-th atom of R, for $i \in \{1, \dots, k\}$.*

1. *A* linearisation *of R is a copy R' of R in which each atom occurrence in Σ is replaced by a different symbol in a new alphabet Γ. We denote by α the corresponding bijection $\{1, \dots, k\} \to \Gamma$ and by β the function $\Gamma \to \Sigma$ defined by: $\forall i \in \{1, \dots, k\}$ $\beta(\alpha(i)) = a_i$.*
2. *A word u* covers the *i*-th atom *of R if there exists some word $u' \in L(R')$ in which the letter $\alpha(i)$ occurs, and such that $u = \beta(u')$, for some linearisation R' of R. A walk w in $\mathcal{D}$ covers the i-th atom of R if its label does.*

   Note that this notion is independent of the chosen linearisation.
3. *A semantics S is* atom covering *if for every database $\mathcal{D}$, regular expression R and position i in $\{1, \dots, k\}$, if there is a walk covering the i-th atom of R in $\mathsf{Matches}(\mathcal{D}, R)$, there is a walk covering the i-th atom of R in $S(\mathcal{D}, R)$.*

This definition expresses the idea that when designing a query with multiple alternatives, each alternative which yields at least a match should be reflected in the result set. It may be verified that no walk-based semantics can be atom-covering. As was the case for vertex and edge coverage, it is quite easy to define atom-covering semantics (although they are not walk-based in this case), such as the one given next.

▶ **Example 52** (Shortest atom-cover ShAC)**.** Let $\mathcal{D}$ be a database and R be a regular expression. Let $\{1, \dots, k\}$ be the set of positions of atoms in R. Let $\preceq_{\mathsf{Sh}}$ be the *shorter than* order (on which Sh is based). Then:

$$\mathsf{ShAC}(\mathcal{D}, R) = \bigcup_{s,t \in V_{\mathcal{D}}} \bigcup_{i=1}^{k} \min_{\preceq_{\mathsf{Sh}}} \left\{ w \in \mathsf{Matches}(\mathcal{D}, R) \;\middle|\; \begin{array}{l} w \text{ starts in } s \text{ and ends in } t \\ w \text{ covers the } i\text{-th atom of } R \end{array} \right\}$$

It is routine to check that the definition of ShAC implies that it is atom-covering. Since BT includes ShAC semantics (See Section 4.5), it is also atom-covering.

▶ **Proposition 53.** ShAC *and* BT *are atom-covering.*

4.7 Summary: what is a reasonable semantics?

Many of the properties described in this section are desirable. However, we saw that some of them are incompatible. We list below the properties that we think any reasonable semantics should satisfy. As we will see, there are caveats in a lot of cases.

Identifier independence (Def. 19) In database systems, identifiers are not always accessible to the user, nor guaranteed to remain constant over time. Therefore, it seems strange to have a semantics rely on identifiers. Note however that identifier-dependent semantics might still be useful to model cases where the database engine is allowed to make arbitrary choices, like the keyphrase `ANY SHORTEST` in GQL.

Label-independence (Def. 19) Semantics are defined before (hence independently of) database schemas. Since labels are abstractions of relation names in schema, it seems strange to attach a meaning to them. Label-dependence could still be useful to model semantics with a dynamical aspect, like the keyphrase proposal `ANY CHEAPEST` in GQL.

Unboundedness (Def. 23) Bounded semantics only yield walks whose length is bounded by some constant *regardless of the query or the database*, which seems to defeat the whole purpose of using RPQs. Note that we are referring here to a property of the semantics itself, not to possible extensions of the query language allowing users to limit the length of returned walks (which could be useful in certain scenarios).

Co-monotony (Def. 24) A co-monotonous semantics ensures that, for a fixed query, whenever extending a database graph with new vertices or edges, the new results should always use some of the new vertices or edges.

Decomposability (Def. 35) Similarly to co-monotony, semantics that are not decomposable sometimes have new results that materialize out of thin air. In particular, for such semantics, results cannot be computed bottom up from the expression.

Compatibility with atoms (Def. 36) Using a semantics that is not compatible with atoms means that some edges might be missing from replies to atomic queries. If the semantics is moreover decomposable, these missing edges will never appear in the results of any query. Note that Ac, one of the semantics available in GQL, is in fact not compatible with atoms, due to possible self-loops in the graph. This probably means that Ac assumes the graph to have no self-loops.

Continuity (Def. 31) A (walk-based) semantics that is not continuous means that the semantics behaves differently for finite match sets and infinite match sets. However, for semantics that are not walk-based, noncontinuous semantics might still be useful.

5 Conclusion and future work

Only a few RPQ semantics have been considered in industry, and their principled study was generally performed a posteriori. In this work, we took the opposite approach: first define the properties a RPQ semantics could have, in order to design better RPQ semantics in the future. We distinguished two simple classes of semantics: filter-based semantics which select results using a per-walk criterion, and order-based semantics which only retain minimal matches with respect to some ordering relation. We proposed a list of possible properties, some of which may be considered requirements for all RPQ semantics, while others turned out to be incompatible. This process yielded several promising semantics, such as subwalk-minimal and shortest-vertex-cover semantics, which probably deserve further investigations.

Our work is by no means exhaustive. It has several limitations, which constitute interesting leads for future work. First, some of our definitions may be too restrictive. For instance, the class of *filter-based* semantics is tailored to capture several semantics used in practice, but our

definition appears to leave little room for novel designs. Another example is compatibility with rational operators. Most of them are verified by very few reasonable semantics, and weaker notions of compatibility might be worth considering. For instance, a weaker version of $*$-composability could be that $S(\mathcal{D}, R)$ is always included in $S(\mathcal{D}, R^*)$.

Second, we limited the scope of our study to walk-based semantics. A semantics is a way to restrict the potentially infinite set of matches to a finite and hopefully representative set of results for the user. From this perspective, walk-based semantics must make this selection using a very limited amount of information. This prevents us from considering semantics whose results depend on the actual syntax of the regular expression, such as the ones in [3, 6]. Future work could investigate semantics that make use of more information, such as walk labels, the entire database, the syntax of the query, or how walks are matched by the query.

Third, both our data model and query language we consider are rather simplistic. Adding features such as data or node labels should not affect our observations regarding walk-based semantics. However, these changes increase the design space when considering semantics that are not walk-based (as discussed above). Note that the GQL data model [8, 14] allows edges (and nodes) to bear multiple labels. This raises the question of walk multiplicity in the result, a dimension we ignored entirely. It would also be interesting to study *dynamical* semantics: enrich the syntax to give users some control over the semantics, similar to `CHEAPEST` in GQL.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, Reading, Mass, 1995.

2 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5), 2017. doi:10.1145/3104031.

3 Marcelo Arenas, Sebastián Conca, and Jorge Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *World Wide Web (WWW)*, pages 629–638. ACM, 2012. doi:10.1145/2187836.2187922.

4 Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. *J. Comput. Syst. Sci.*, 108:29–48, 2020. doi:10.1016/J.JCSS.2019.08.006.

5 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In Umeshwar Dayal and Irving L. Traiger, editors, *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, CA, USA, May 27-29, 1987*, pages 323–330. ACM Press, 1987. doi:10.1145/38713.38749.

6 Claire David, Nadime Francis, and Victor Marsault. Run-based semantics for rpqs. In *Principles of Knowledge Representation and Reasoning (KR'23)*, pages 178–187, August 2023. doi:10.24963/kr.2023/18.

7 Claire David, Nadime Francis, and Victor Marsault. Distinct shortest walk enumeration for rpqs. *Proc. ACM Manag. Data*, 2(2), 2024. doi:10.1145/3651601.

8 Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Hannes Voigt, Oskar van Rest, Domagoj Vrgoč, Mingxi Wu, and Fred Zemke. Graph pattern matching in GQL and SQL/PGQ. In *SIGMOD'22*. ACM, 2022.

9 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Martin Schuster, Petra Selmer, and Andrés Taylor. Formal semantics of the language cypher, 2018. Arxiv Preprint. arXiv:1802.09984.

10 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *SIGMOD'18*. ACM, 2018. doi:10.1145/3183713.3190657.

11 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of The London Mathematical Society*, pages 326–336, 1952. URL: https://api.semanticscholar.org/CorpusID:123011310.

12 International Organization for Standardization. SQL, 2023. Standard ISO/IEC CD 9075. Edition 6.

13 International Organization for Standardization. SQL — part 16: SQL property graph queries (SQL/PGQ). https://www.iso.org/standard/79473.html, 2023. Standard ISO/IEC CD 9075-16. Part 16 of [12].

14 International Organization for Standardization. GQL. https://www.iso.org/standard/76120.html, 2024. Standard ISO/IEC CD 39075.

15 Prateek Karandikar. *Subwords : automata, embedding problems, and verification*. PhD thesis, École normale supérieure de Cachan - ENS Cachan ; Chennai Mathematical Institute, 2015. URL: https://theses.hal.science/tel-01157502.

16 Sara Khichane. Study of new semantics for regular path queries. Master's thesis, Université Paris-Cité, 2024.

17 Victor Marsault. Matching walks that are minimal with respect to edge inclusion, 2024. ArXiv Preprint. doi:10.48550/arXiv.2408.14048.

18 Wim Martens, Matthias Niewerth, and Tina Popp. A trichotomy for regular trail queries. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:20)2023.

19 Wim Martens, Matthias Niewerth, Tina Popp, Carlos Rojas, Stijn Vansummeren, and Domagoj Vrgoč. Representing paths in graph database pattern matching. *Proc. VLDB Endow.*, 16(7), 2023. doi:10.14778/3587136.3587151.

20 Wim Martens, Matthias Niewerth, Tina Popp, Carlos Rojas, Stijn Vansummeren, and Domagoj Vrgoč. Compact path representations for graph database pattern matching. In *ICDEW'24*, pages 379–380, 2024. doi:10.1109/ICDEW61823.2024.00059.

21 Wim Martens and Tina Trautner. Evaluation and enumeration problems for regular path queries. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, volume 98 of *LIPIcs*, pages 19:1–19:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICDT.2018.19.

22 Neo4j. Cypher manual (version 5). https://neo4j.com/docs/cypher-manual/5, 2024.

23 Oracle. PGQL 2.0 specification. https://pgql-lang.org/spec/2.0/, 2021.

24 TigerGraph. GSQL language reference (version 3.9), 2023. URL: https://docs.tigergraph.com/gsql-ref/3.9/intro/.

25 World Wide Web Consortium. SPARQL 1.1 query language, section 9: Property paths. https://www.w3.org/TR/sparql11-query/#propertypaths, 2013.

A Characteristic Database and Characteristic Expression

The proofs of several statements in this work rely on the notions of characteristic database and of characteristic expression of a walk or set of walks. Although the proofs are omitted due to space constraints, we still provide below definitions of these notions.

▶ **Definition 54** (Characteristic database, characteristic expression)**.** *For simplicity, we will consider in the following that* $\mathbb{E} \subseteq \mathbb{L}$*, i.e. that edge identifiers are possible labels. To be formally correct, we would need to use some arbitrary bijection* $\mathbb{E} \to \mathbb{L}$*, but we prefer to leave it implicit.*

1. *Let* $w_1, \ldots, w_k$ *be consistent walks. Let* $\mathcal{D}$ *be any database such that* $\{w_1, \ldots, w_k\} \subseteq \mathsf{Walks}(\mathcal{D})$*, and*

$$V = \mathsf{VertSet}(w_1) \cup \cdots \cup \mathsf{VertSet}(w_k)$$
$$E = \mathsf{EdgeSet}(w_1) \cup \cdots \cup \mathsf{EdgeSet}(w_k)$$

Note that $V \subseteq V_{\mathcal{D}}$ *and* $E \subseteq E_{\mathcal{D}}$. *The* characteristic database *of* $\{w_1, \dots, w_k\}$ *is the database*

$$\mathcal{C}_{w_1,\dots,w_k} = (L, V, E, \mathsf{Src}, \mathsf{Tgt}, \mathsf{Lbl})$$

where $L = E$, Lbl *is the identity function, and* Src *(resp.* Tgt*) is the restriction of* $\mathsf{Src}_{\mathcal{D}}$ *(resp.* $\mathsf{Tgt}_{\mathcal{D}}$*) to* E.
(One may verify that $\mathcal{C}_{w_1,\dots,w_k}$ *is independent of the database* $\mathcal{D}$ *chosen.)*

2. *Let* w *be a walk of positive length* k. *Then we denote by* P_w *the* characteristic regular expression *of* w, *defined as* $P_w = w[1] \cdots w[k]$.

Intuitively, the characteristic database of a finite set of compatible walks is just the smallest database containing them all, i.e. $\{w_1, \dots, w_k\} \subseteq \mathsf{Walks}(\mathcal{C}_{w_1,\dots,w_k})$, and in which edges are simply labeled with their own identifier. The characteristic expression of a walk is just the sequence of its edge identifiers.

These definitions are chosen in such a way that the characteristic expression of each w_i of positive length is matched exactly once ($\mathsf{Matches}(\mathcal{C}_{w_1,\dots,w_k}, P_{w_i}) = \{w_i\}$) and that the characteristic expression of any other walk of positive length is matched either once or not at all: $\mathsf{Matches}(\mathcal{C}_{w_1,\dots,w_k}, P_w$ is either empty or equal to $\{w\})$.

B Index & Acronyms