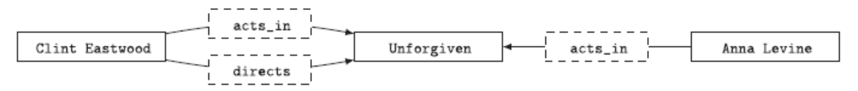
1.6. Graph Databases[8]

- A graph database is a database that, in contrast to the tables of a relational database, uses graph structures for storing and querying data.
- As a consequence, graph databases support convenient and fast retrieval of complex structures that may be cumbersome to model in relational systems.
- The recently growing popularity of graph databases stems from the realisation that there are a variety of domains for which graph databases offer a more intuitive conceptualization than relational databases. For example, one can view a social network as a graph of people who know each other, or view transport networks, biological pathways, citation networks, and so on, as a graph.

[8] Renzo Angles, Marcelo Arenas, Pablo Barcelo, Aidan Hogan, Juan Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50, 5, Article 68 (September 2017), 40 pages. https://doi.org/10.1145/3104031

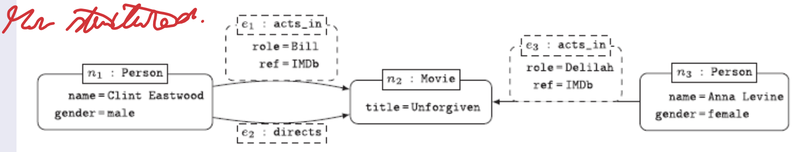
Edge-labelled graphs vs. property graphs

- A simple edge-labelled graph:



- A simple property graph:

Mehr strukturiert.



Definitions

- An edge-labelled graph G is a pair (V, E), where
  (1) V is a finite set of *vertices*,
  (2) E is a finite set of *edges*, $E \subseteq V \times Lab \times V$, where Lab is a set of *labels*

- A property graph G is a tuple $(V, E, \rho, \lambda, \sigma)$, where
  (1) V is a finite set of *vertices*,
  (2) E is a finite set of *edges* such that $V \cap E = \emptyset$,
  (3) $\rho : E \to (V \times V)$ is a total function. $\rho(e) = (v_1, v_2)$ indicates a directed edge from v_1 to v_2.
  (4) $\lambda : (V \cup E) \to Lab$ is a total function assigning labels $l \in Lab$ to vertices and edges.
  (5) $\sigma : (V \cup E) \times Prop \to Val$ is a partial function with $Prop$ a finite set of *properties* p and Val a set of *values* s.
  Remark: We say also σ assignes a set of *key-value*-pairs (p, s) to the respective vertice v or edge e.

Querying: pattern matching vs. navigation

- Graph pattern matching: A query is stated as a graph with variables; *matches* of variables determine the answers to the query.
    - ELG: variables may appear as nodes and edge labels;
    - PG: variables may appear in place of any constant.

  ?

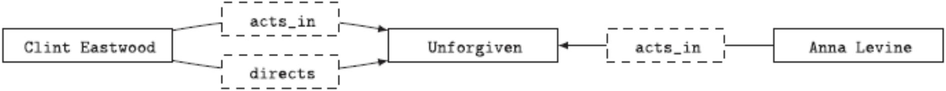
  A *match* is a mapping from variables to constants such that when the mapping is applied the result is "*contained*" in the original graph database.
- Graph navigation: A query is stated as a regular expression describing a graph topology; answers return either existance of the described structure in the original graph database, or the corresponding subgraph according to "*certain*" criteria.

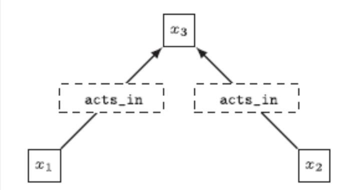
Let Q be a query and G a graph.

- *Set semantics*: $Q(G)$ is defined as a set of matches; in other words, the result of evaluating Q over G cannot contain duplicate matches.
- *Bag semantics*: $Q(G)$ is defined as a bag of matches; more specifically, the number of times a match appears in the result corresponds with the number of unique mappings that witness the match.

Graph pattern matching

Edge-labelled graph and pattern matching query





Graph-Homomorphism

Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be graphs.

A function $f : V(G) \to V(H)$ is called *graph-homomorphism*, iff

$$(u, v) \in E(G) \Longrightarrow (f(u), f(v)) \in E(H).$$

Graph-Isomorphism

Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be graphs.

A bijective function $f : V(G) \to V(H)$ is called *graph-isomorphism*, iff

$$(u, v) \in E(G) \Longleftrightarrow (f(u), f(v)) \in E(H).$$

Subgraph-Isomorphism

Let $G = (V(G), E(G))$ and $H = (V(H), E(H))$ be graphs.

A injective function $f : V(G) \to V(H)$ is called *subgraph-isomorphism*, if

$$(u, v) \in E(G) \Longrightarrow (f(u), f(v)) \in E(H).$$

G is isomorphic to a subgraph of H.

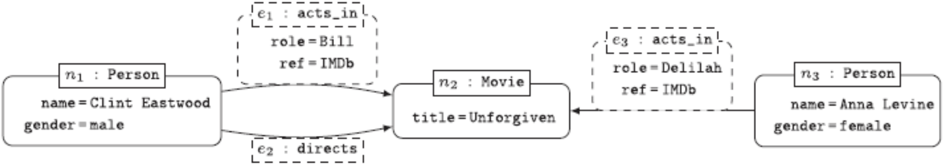
Match

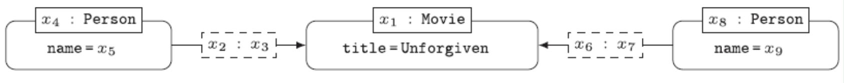
Given an edge-labelled graph $G = (V, E)$ and a *graph-pattern* $Q = (V', E')$, a *match* h of Q in $G = (V, E)$ is a mapping $h : Const \cup Var \rightarrow Const$ such that:

- for each constant $a \in Const : h(a) = a$.
- for each edge $(b, l, c) \in E' : (h(b, h(l), h(c)) \in E$.

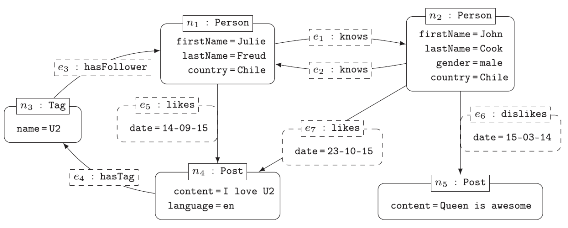
Match is defined analogously for property graphs: mapping h maps constants to themselves and variables to constants; if the image of some Q under h is contained within graph G, then h is a match.

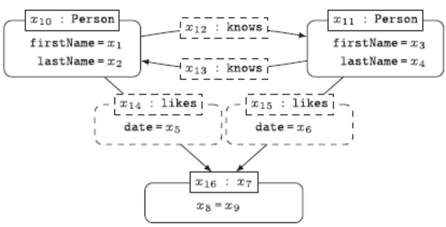
Property graph and pattern matching query



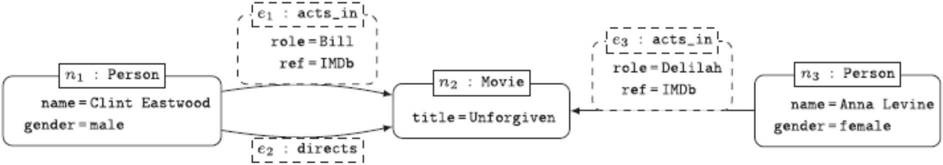


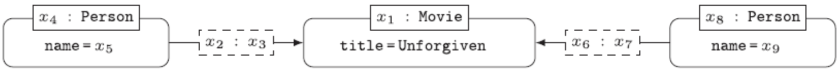
Property graph and pattern matching query





Homomorphism- vs. isomorphism-based semantics

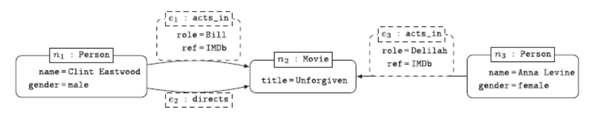


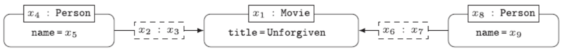


Results based on all homomorphisms from query Q to PG G.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_2	*directs*	n_1	*ClintEastwood*
n_2	e_1	*acts_in*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
				⋮				

Homomorphism- vs. isomorphism-based semantics

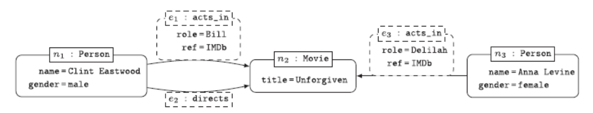


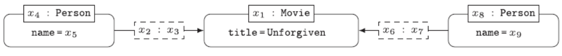


Results based on all homomorphisms from query Q to PG G.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_2	*directs*	n_1	*ClintEastwood*
n_2	e_1	*acts_in*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_1	*acts_in*	n_1	*ClintEastwood*
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_1	*acts_in*	n_1	*ClintEastwood*
n_2	e_1	*acts_in*	n_1	*ClintEastwood*	e_2	*directs*	n_1	*ClintEastwood*
n_2	e_1	*acts_in*	n_1	*ClintEastwood*	e_1	*acts_in*	n_1	*ClintEastwood*
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_2	*directs*	n_1	*ClintEastwood*
n_2	e_3	*acts_in*	n_1	*AnnaLevine*	e_3	*acts_in*	n_1	*AnnaLevine*

Homomorphism- vs. isomorphism-based semantics





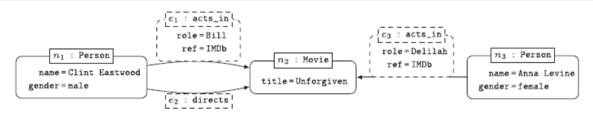
(Subgraph-) Isomorphism-based semantics

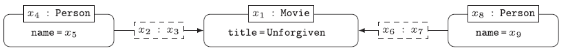
(I1) No-repeated-anything semantics: mappings must be injective,

no-repeated-anything:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_2	*directs*	n_1	*ClintEastwood*

Homomorphism- vs. isomorphism-based semantics





(Subgraph-) Isomorphism-based semantics

(I2) No-repeated-node-semantics: injective w.r.t. to variables that map to nodes only,

no-repeated-node:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_2	*directs*	n_1	*ClintEastwood*
n_2	e_1	*acts_in*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_1	*acts_in*	n_1	*ClintEastwood*

(Subgraph-) Isomorphism-based semantics

(I3) No-repeated-edge semantics: injective w.r.t. to variables that map to edges only.

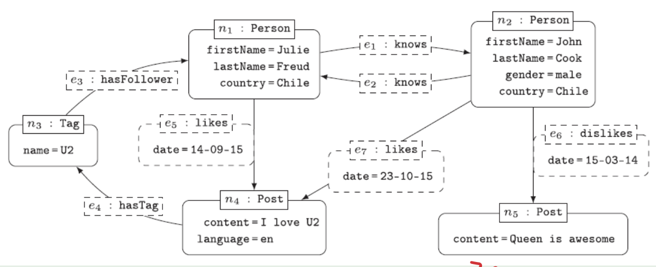
no-repeated-edge:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_2	*directs*	n_1	*ClintEastwood*
n_2	e_1	*acts_in*	n_1	*ClintEastwood*	e_3	*acts_in*	n_3	*AnnaLevine*
n_2	e_3	*acts_in*	n_3	*AnnaLevine*	e_1	*acts_in*	n_1	*ClintEastwood*
n_2	e_2	*directs*	n_1	*ClintEastwood*	e_1	*acts_in*	n_1	*ClintEastwood*
n_2	e_1	*acts_in*	n_1	*ClintEastwood*	e_2	*directs*	n_1	*ClintEastwood*

Graph navigation

A query is stated as a regular expression describing a graph topology.

Examples

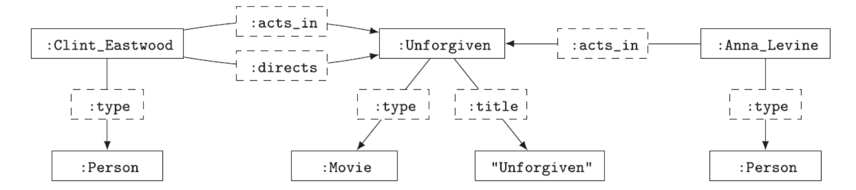


≥ 1

$$P_1 := x \xrightarrow{knows^*} y; \quad P_2 := x \xrightarrow{knows^+ \cdot likes} y$$

$$P_3 := x \xrightarrow{knows^+(likes|dislikes)} y; \quad P_4 := x \xrightarrow{*} y$$

Examples



:Clint_Eastwood

:acts_in

:directs

:Unforgiven

:acts_in

:Anna_Levine

:type

:type

:title

:type

:Person

:Movie

"Unforgiven"

:Person

$$P_5 := x \xrightarrow{acts\_in \cdot acts\_in^-} y$$

Reverse.

$$P_6 := x \xrightarrow{(acts\_in \cdot acts\_in^-)^+} y$$

Evaluation of a regular path query (RPG)

A path π in a edge-labelled graph G is a sequence:

$$n_1 a_1 n_2 a_2 n_3 \dots n_{k-1} a_{k-1} n_k$$

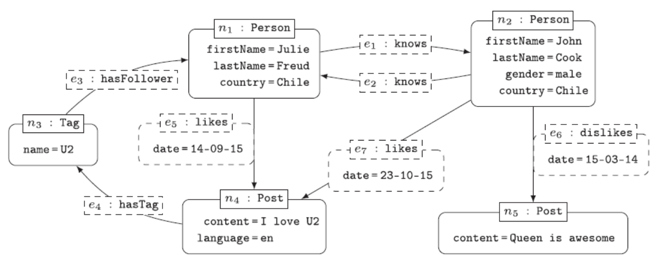
where (n_i, a_i, n_{i+1}) is an edge in G, $i < k$.

The label of π is given as $Lab(\pi) = a_1 a_2 \dots a_{k-1}$.

Let a path query $P = x \xrightarrow{\alpha} y$, where α is a regular expression over the set of labels, and a labelled graph G be given.

The *evaluation* of P over G consists of all paths π in G whose label $Lab(\pi)$ satisfies α.

Navigational queries semantics



$$P := x \xrightarrow{knows^{+}\cdot likes\cdot hasTag} y$$

Infinite result:

$n_1 \quad e_1 \quad n_2 \quad e_7 \quad n_4 \quad e_4 \quad n_3$

$n_1 \quad e_1 \quad n_2 \quad e_2 \quad n_1 \quad e_5 \quad n_4 \quad e_4 \quad n_3$

$n_1 \quad e_1 \quad n_2 \quad e_2 \quad n_1 \quad e_1 \quad n_2 \quad e_7 \quad n_4 \quad e_4 \quad n_3$

$n_1 \quad e_1 \quad n_2 \quad e_2 \quad n_1 \quad e_1 \quad n_2 \quad e_2 \quad n_1 \quad e_5 \quad n_4 \quad e_4 \quad n_3$

...

Navigational queries semantics

- Arbitrary path semantics: All paths considered.
  Result may be infinite - however of practical interest for testing existence of a path, or returning pairs of nodes being connected.
- Shortest path semantics: Paths of minimal length are returned.
  Results are witnesses for connectedness.
- No-repeated-node semantics: Each nodes appears at most once in the result.
  Simple paths.
- No-repeated-edge semantics: Each edge appears at most once in the result.

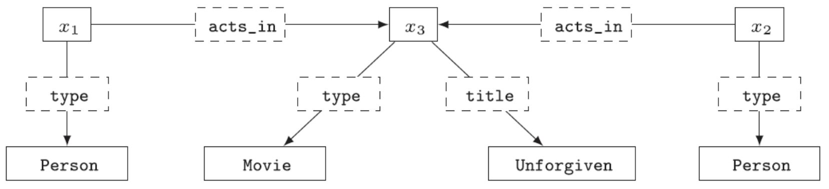
Graph Database Query Languages

SPARQL is a declarative language recommended by W3C for querying RDF graphs, a variant of edge-labelled graphs.

Find all co-stars in movie "Unforgiven".

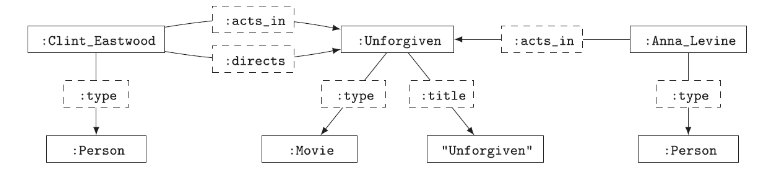
```
SELECT ?x1 ?x2
WHERE {
?x1 :acts_in ?x3 . ?x1 :type :Person .
?x2 :acts_in ?x3 . ?x2 :type :Person .
?x3 :title "Unforgiven" . ?x3 :type :Movie . FILTER(?x1 != ?x2)}
```

Query as a graph:



Find people with a finite Bacon-Number.

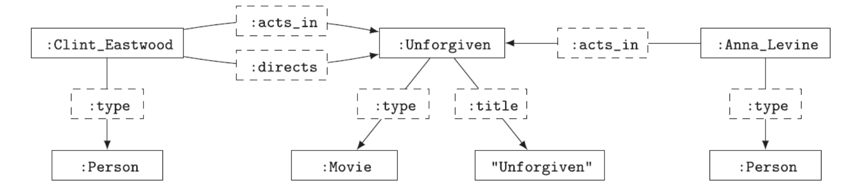
SELECT ?x
WHERE { ?x (:acts_in/^:acts_in) * :Kevin_Bacon . }



Cycles, indefinite paths?

Find movies that Clint Eastwood has acted or directed in.

SELECT ?x
WHERE {{ :Clint_Eastwood :acts_in ?x . }
UNION { :Clint_Eastwood :directs ?x . }}

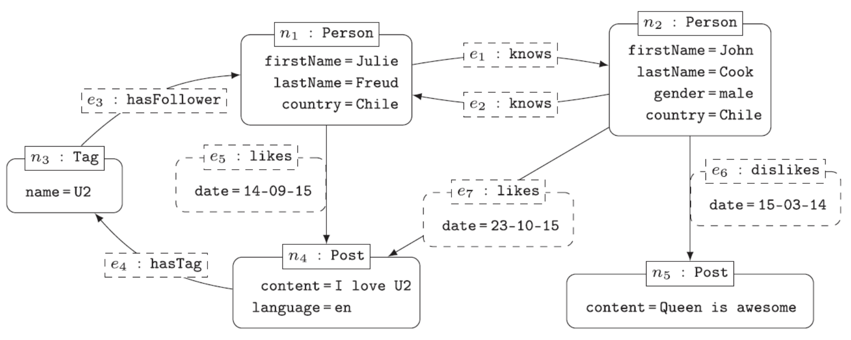


Both patterns to the left and right of the UNION will be evaluated independently and their results unioned. The result :Unforgiven will be returned twice, since SPARQL, by default, adopts a bag semantics.

The openCypher project[9] aims to deliver a full and open specification of the industry's most widely adopted graph database query language: Cypher.

Find all friends-of-friends of Julie that liked a post with a tag Julie follows.

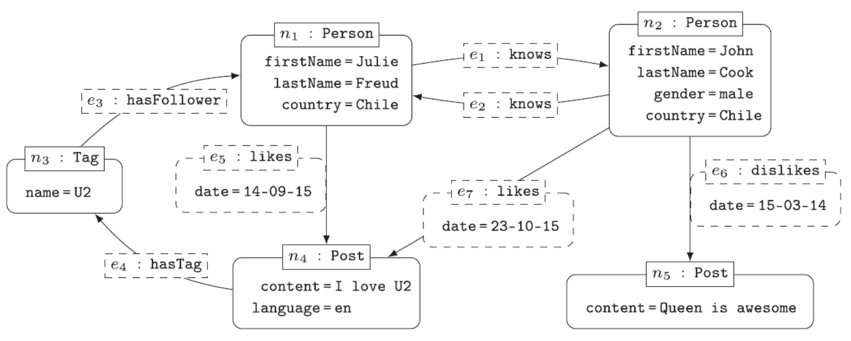
MATCH (x1:Person firstName:"Julie") -[:knows*]-> (x2:Person)
MATCH (x2) -[:likes]-> () -> [:hasTag] -> (x3)
MATCH (x3) -[:hasFollower]-> (x1) RETURN x2



[9] https://www.opencypher.org/

Find all friends-of-friends of Julie together with a path witnessing the friendship.

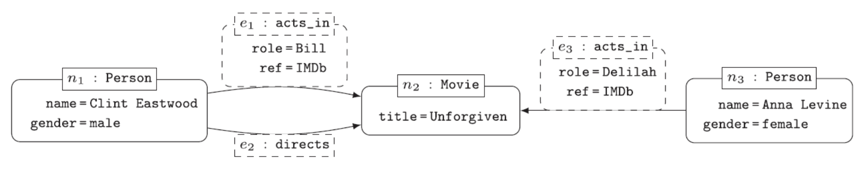
MATCH p = (:Person firstName:"Julie") -[:knows*]-> (x:Person)
RETURN x, p



Gremlin is the query language of the Apache Tinker- Pop3 graph Framework. Although Gremlin is also specified with the property graph model in mind, it has a more "functional" feel: while SPARQL and Cypher have obvious influences from SQL, Gremlin feels more like a programming language interface. Likewise, its focus is on navigational queries rather than matching patterns; however, amongst the "graph traversal" operations that it defines, we can find familiar graph pattern matching features.

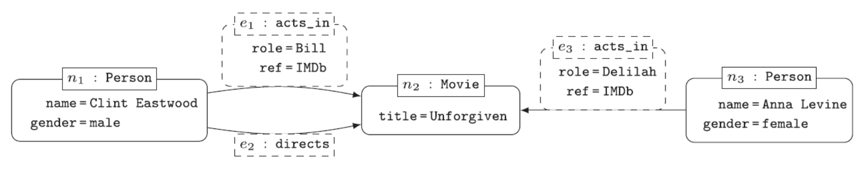
Find all co-actors of Clint Eastwood.

```
G.V().hasLabel('Person').has('name','Clint Eastwood')
.out('acts_in').hasLabel('Movies')
.in('acts_in').hasLabel('Person')
```



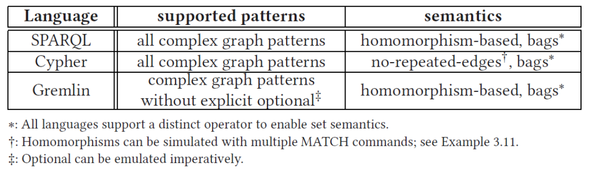
Find actors that are linked to Clint Eastwood by a path of length 2.

```
G.V().hasLabel('Person').has('name','Clint Eastwood').repeat(
out('acts_in').hasLabel('Movies')
.in('acts_in').hasLabel('Person')
).times(2)
```



GraphDB Query Language Semantics[10]

Pattern matching queries semantics.



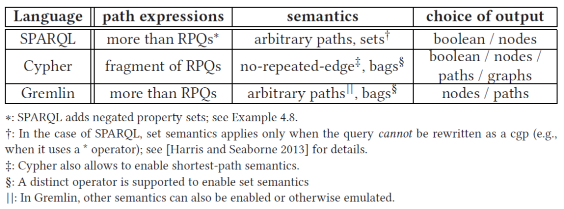
Language	supported patterns	semantics
SPARQL	all complex graph patterns	homomorphism-based, bags[*]
Cypher	all complex graph patterns	no-repeated-edges[†], bags[*]
Gremlin	complex graph patterns without explicit optional[‡]	homomorphism-based, bags[*]

∗: All languages support a distinct operator to enable set semantics.

†: Homomorphisms can be simulated with multiple MATCH commands; see Example 3.11.

‡: Optional can be emulated imperatively.

Navigational queries semantics.



Language	path expressions	semantics	choice of output
SPARQL	more than RPQs[*]	arbitrary paths, sets[†]	boolean / nodes
Cypher	fragment of RPQs	no-repeated-edge[‡], bags[§]	boolean / nodes / paths / graphs
Gremlin	more than RPQs	arbitrary paths[||], bags[§]	nodes / paths

∗: SPARQL adds negated property sets; see Example 4.8.

†: In the case of SPARQL, set semantics applies only when the query *cannot* be rewritten as a cgp (e.g., when it uses a * operator); see [Harris and Seaborne 2013] for details.

‡: Cypher also allows to enable shortest-path semantics.

§: A distinct operator is supported to enable set semantics

||: In Gremlin, other semantics can also be enabled or otherwise emulated.

[10] Renzo Angles et al. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50, 5.