The case for Regular Path Queries

Tobias Lindaaker @ neo4j

CIR-2017-179

Why should we add Path Queries to Cypher?

- Find complex connections
  - Repetitions of patterns and not just single edge label
  - Alternatives between patterns and not just single edge label

- Express patterns as patterns instead of combinations of queries
  (Using `UNION` to express disjunctions hides what is actually going on)

Path Pattern Queries
RPQs in Cypher

Tobias Lindaaker @ neo4j

CIP 2017-02-06

Path Patterns in Cypher

- Predicates on Edge Label: `()-/:FOO/-()`
- Predicates on Nodes: `()-/(:Alpha{beta:'gamma'})/-()`
- Alternatives: `()-/:FOO | :BAR | :BAZ/-()`
- Sequence: `()-/:FOO :BAR :BAZ/-()`
- Grouping: `()-/:FOO | [:BAR :BAZ]/-()`
- Direction: `()-/<:FOO :BAR <:BAZ>/->()`
- Any Edge: `()-/-/-()`
- Repetition: `()-/:FOO? :BAR+ :BAZ* :FOO*3.. :BAR*1..5/-()`
- Predicates on Edge Properties: `()-/ [- {some:'value'}] /-()`
  (applies to ***all*** edges matched by the group)
  (for more complex predicates use *Named Path Patterns*)
- Negation: `()-/[^:FOO :BAR]/-()`
  (negation groups only allows edge labels - no other combinators or properties)
- Named Path Patterns: `()-/~foobar/-()`
  after declaring: `PATH PATTERN foobar = ()-/:FOO :BAR/-()`

Example

```
()-/[^:FOO :BAR]/-()

()-/[^:FOO|:BAR]/-()

()-/~not_foo_or_bar/-()

PATH PATTERN not_foo_or_bar = ()-[x]-()
     WHERE label(x) <> 'FOO' AND label(x) <> 'BAR'
```

Atomic Patterns vs Path Patterns

Edge Patterns are delimited by square brackets `[ ... ]` and denotes a single edge
Edge Patterns can be used both for matching an edge and for creating an edge
Edge Patterns can bind a matched edge to a variable

Path Patterns are delimited by slashes `/ ... /` and denotes a path of zero or more edges
Path Patterns can only be used for matching paths, and cannot be used for creating entities in a graph
Path Patterns can not bind individual edges (or nodes) to variables

Compared to Regular Expressions (over text)

Regular expression over text

- Atoms: text literals
- Disjunction: `|`
- Concatenation: *juxtaposition*
- Grouping: `( ... )`
- Quantification:
  - `?` - zero or one
  - `*` - zero or more
  - `+` - one or more
  - `{n,m}` - at least `n`, at most `m`
- Wildcard: `.` (any character)
- String start/end
- Disjunction of atoms short form: `[ ... ]`
- Negation of disjunction of atoms: `[^...]`
- Greedy, Lazy, Possessive

Cypher Path Patterns

- Atoms: Edge / Node (label) predicates
- Disjunction: `|`
- Concatenation: *juxtaposition*
- Grouping: `[ ... ]`
- Quantification:
  - `?` - zero or one
  - `*` - zero or more
  - `+` - one or more
  - `*n..m` - at least `n`, at most `m`
- Wildcard: `–` (any edge)
- *not applicable*
- *not available*
- *not supported*
- *not explored*

Compared to GXPath *(path expressions)*

- $\llbracket \varepsilon \rrbracket^G$ – $\{(v,v) \mid v \in V\}$ ***- yes:*** `(v)-/()/->(v)`
- $\llbracket \_ \rrbracket^G$ – $\{(v,w) \mid (v,a,w) \in E$ for some a$\}$
  - yes: `(v)-/-/->(w)`
- $\llbracket a \rrbracket^G$ – $\{(v,w) \mid (v,a,w) \in E\}$ ***- yes:*** `(v)-/:a/->(w)`
- $\llbracket a^- \rrbracket^G = \{(v,w) \mid (w,a,v) \in E\}$ ***- yes:*** `(v)-/<:a/->(w)`
- $\llbracket \alpha^* \rrbracket^G =$ reflexive transitive closure of α
  - yes: `()-[α*]->()`
- $\llbracket \alpha \cdot \beta \rrbracket^G = \llbracket \alpha \rrbracket^G \circ \llbracket \beta \rrbracket^G$ ***- yes:*** `()-[α β]->()`
- $\llbracket \alpha \cup \beta \rrbracket^G = \llbracket \alpha \rrbracket^G \cup \llbracket \beta \rrbracket^G$ ***- yes:*** `()-[α|β]->()`
- $\llbracket \neg\alpha \rrbracket^G = V \times V - \llbracket \alpha \rrbracket^G$ ***- NO! the*** `PATH PATTERN` ***syntax is limited to connected patterns***
- $\llbracket [\varphi] \rrbracket^G = \{(v,v) \mid v \in \llbracket \varphi \rrbracket^G\}$
  - yes: `PATH PATTERN phi = (v) WHERE φ(v)`
- $\llbracket \alpha^{n,m} \rrbracket^G = \bigcup_{k=n}^{m} (\llbracket \alpha \rrbracket^G)^k$ ***- yes:*** `()-[α*n..m]->()`
- $\llbracket \alpha_= \rrbracket^G = \{(v,w) \in \llbracket \alpha_= \rrbracket^G \mid \rho(v)=\rho(w)\}$
  - yes: `PATH PATTERN alpha_eq = (v)-[α]->(w) WHERE v.ρ = w.ρ`
- $\llbracket \alpha_\neq \rrbracket^G = \{(v,w) \in \llbracket \alpha_= \rrbracket^G \mid \rho(v)\neq\rho(w)\}$
  - yes: `PATH PATTERN alpha_not_eq = (v)-[α]->(w) WHERE v.ρ <> w.ρ`
- **Conjunctions *(for CRPQs)*:** $\llbracket \alpha \cap \beta \rrbracket^G = \llbracket \alpha \rrbracket^G \cup \llbracket \beta \rrbracket^G$
  - yes:
  ```
  PATH PATTERN alpha_and_beta = (v)-[α]->(w)
                    WHERE EXISTS { (v)-[β]->(w) }
  ```

Cartesian product with negation

```
MATCH (a), (b)
WHERE NOT EXISTS {
    (a)-/.../-(b)
}
```

Compared to GXPath *(node tests)*

- $[[\alpha]]^G = \{v \mid \exists w\ (v,w) \in [[\alpha]]^G\}$
  - ***yes:*** `PATH PATTERN has_alpha = (v) WHERE EXIST { (v)-[ α]->() }`
- $[[\neg\varphi]]^G = V - [[\varphi]]^G$
  - ***yes:*** `PATH PATTERN not_phi = (v) WHERE NOT φ(v)`
- $[[\varphi \wedge \psi]]^G = [[\varphi]]^G \cap [[\psi]]^G$
  - ***yes:*** `PATH PATTERN phi_and_psi = (v) WHERE φ(v) AND ψ(v)`
- $[[\varphi \vee \psi]]^G = [[\varphi]]^G \cup [[\psi]]^G$
  - ***yes:*** `PATH PATTERN phi_or_psi = (v) WHERE φ(v) OR ψ(v)`
- $[[c^=]]^G = \{v \in V \mid \rho(v) = c\}$
  - ***yes:*** `PATH PATTERN rho_is_c = (v) WHERE v.ρ = c`
- $[[c^\neq]]^G = \{v \in V \mid \rho(v) \neq c\}$
  - ***yes:*** `PATH PATTERN rho_is_not_c = (v) WHERE v.ρ <> c`
- $[[\alpha = \beta]]^G = \{v \in V \mid \exists w,y\ (v,w)\in[[\alpha]]^G,\ (v,y)\in[[\beta]]^G,\ \rho(w)=\rho(y)\}$
  - ***yes:*** `PATH PATTERN alpha_eq_beta = (v) WHERE EXIST { MATCH (v)-[α]-(w), (v)-[β]-(y) WHERE w.ρ = y.ρ }`
- $[[\alpha \neq \beta]]^G = \{v \in V \mid \exists w,y\ (v,w)\in[[\alpha]]^G,\ (v,y)\in[[\beta]]^G,\ \rho(w)\neq\rho(y)\}$
  - ***yes:*** `PATH PATTERN alpha_not_eq_beta = (v) WHERE EXIST { MATCH (v)-[α]-(w), (v)-[β]-(y) WHERE w.ρ <> y.ρ }`

Paths where property value differs from first

```
PATH PATTERN not_first_foo = (first)-/-*/-()
     WHERE all( n IN nodes(not_first_foo)
                WHERE n = first OR n.foo <> first.foo )


MATCH x = (a)-/~not_first_foo/-(b)
RETURN a, b, length(x)
```

Remaining work

- Incorporate negation into CIP
- Comparison with Conjunctive Context-Free Path Queries
- Are there things we can express in Named Path Patterns (WHERE) that cannot be implemented using REMs?
- Make sure that the referenced papers are linked from the CIP
- Idea: `:-` instead of `-` for (edge label) wildcard
- Binding groups (paths), a better syntax than:

  ```
  PATH PATTERN foo = ()-[x]-(b), p=(b)-/-*/-()
  ```

  (at least a discussion section on it - either including or explaining why not)
  - Can we use bare words for that?
- Quite possibly juxtaposition will be used elsewhere for conjunction
  so perhaps we should allow that here as well, and use a separate operator for sequences

Further reading

- The Cypher Improvement Proposal: CIP 2017-02-06 (rendered text)
- Presentation from oCIM 1
- Short presentation on Path Patterns
- Presentation from oCIM 2