Path Query Patterns

CIP2017-02-06

a.k.a. (Conjunctive) Regular Path Queries

“Regular expressions over Graphs”

Tobias Lindaaker

tobias@neotechnology.com

Find complex connections

```
MATCH (a)-[~cousin]-(b)

MATCH (me)-[:ALLY | [:ENEMY :ENEMY]]-(friend)

MATCH (x)-[[:A :B]+ :C+ [:B :A]*]->(y)

MATCH p = (a)-[[:KNOWS | :LOVES]+]->()
```

Syntax design philosophy

- Reuse and extend existing syntax
- Direct notation for common simple patterns
- Allow for complex patterns
- Avoid repetition and ceremony

Improved since last time

- Shorthand syntax for node labels!
- Shorthand syntax for matching any edge!
- Shorthand syntax for property predicates!

- Declared Path Patterns are scoped per query
  And declared at the head of the query

- Non-repetitive syntax for Declared Path Patterns

Composition of Patterns

Regular Expressions over Paths

- Sequence / Concatenation:
  `()-[α β]-()`
- Alternative / Disjunction:
  `()-[α | β]-()`
- Transitive closure:
  ```
  ()-[α*]-()      // zero or more times
  ()-[α+]-()      // one  or more times
  ()-[α*n..]-()   // n    or more times
  ()-[α*n..m]-() // at least n, at most m
  ```
- Overriding direction for sub-pattern:
  `()-[<α]-()  ()-[α>]-()  ()-[<α>]-()`

Predicates for Path Patterns

- Edge Label
  `()-[:KNOWS]-()`
- Node Label
  `()-[(:Person)]-()`
- Edge Properties
  `()-[:ROAD{length:12}]-()`
- Node Properties
  `()-[({year:2017})]-()`
- Edges with any label, and matching empty path
  `()-[-]-()` `()-[()]-()`
- Grouping of expressions
  `()-[[:KNOWS :LOVES]+]-()`

Declared Path Patterns

- Referenced using ~

```
MATCH (a)-[~my_pattern]-(b)
```

- Declaration:

```
PATH PATTERN my_pattern =
     ()-[:FOO :BAR* :BAZ]-()
```

- Makes the pattern language Context Free

Advanced property tests

- ```
  PATH PATTERN same_color_nodes = (a)-[-]-(b)
       WHERE a.color = b.color

  MATCH (x)-[~same_color_nodes+]-(y)
  ```

- ```
  PATH PATTERN older_friends = (a)-[:KNOWS]-(b)
       WHERE b.birthday < a.birthday
  ```

Conjunctive Path Queries

- ```
  PATH PATTERN friends_in_same_city = (a)-[:KNOWS]-(b)
       WHERE EXISTS {
             (a)-[:LIVES_IN]->()<-[:LIVES_IN]-(b)
       }
  ```
- One pattern must be designated the *main* pattern
- Other patterns are put in `WHERE`
- The main pattern is presented when binding a matched path:
  ```
  MATCH friendships = (a)-[~friends_in_same_city+]-()
  ```
  ```

Balanced Path Matching

Classic Context Free use case

```
// treats siblings as “zeroth cousin”
PATH PATTERN cousin =
     ()-[:PARENT> [() | ~cousin] <:PARENT]-()

MATCH rel=(me)-[~cousin]-(cus)
WHERE length(rel) > 2 // filter away siblings
```

Expressive power compared to academic languages

- Covers all of GXPath - except negation
- Weaker scoping reach than Regular Expressions with Memory
  - Variables only reach within single Declared Path Pattern
- Defines a Context Free Language over paths rather than just a Regular Language
- More sophisticated data tests than GXPath

Sophisticated data test

```
PATH PATTERN same_pet_name = (a)-[:KNOWS]-(b)
     WHERE EXISTS {
           MATCH (a)-[:OWNS]-(petA:Animal),
                 (b)-[:OWNS]-(petB:Animal)
           WHERE petA.name = petB.name
     }
MATCH peculiar_friends = (x)-[~same_pet_name*]-()
```

Alternative syntax ideas

- Differentiate between single edge and path

```
MATCH (a)-[single:EDGE]-(and_a)-/~path*/-()
```

- Single edge matching allows variable binding
- Binding a path results in a whole path value bound

Next steps

“Battle testing”

- Prototype implementation
- Try to express interesting queries
  - Needs to find / define interesting queries
- Measure up against *“yardstick queries”*
- Academic evaluation of
  - expressive power
  - computational complexity

Yardstick queries

that I have not yet been able to express

- Paths where the property value differs in all nodes
  - *quite possibly not tractable*
- Paths where the property value in all subsequent nodes differs from the first node