Comparing Path Pattern Syntaxes

A timeline overview of the evolution of the path pattern syntax of Cypher

Tobias Lindaaker
tobias@neo4j.com
The basic patterns of Cypher

MATCH ()--()  
MATCH ()-[r]--() with binding  
MATCH ()-[r:FOO]--() with specific edge  
MATCH ()-[:FOO]--() specific edge, no binding  
MATCH ()-[:FOO*]--() repeated  
MATCH ()-[r:FOO*]--() repeated with binding  
  binds r to a list of edges  
MATCH ()-[*]--() repeated any edge  
MATCH ()-[]--() single edge
The basic patterns of Cypher

MATCH ()-[[:A|B]]->() union of edge labels
MATCH ()-[[:A|B*]]->() repetition of union

• Cannot express repeated sequences
  or alternatives between sequences

• Binding repeated edges yields a different variable type
The first steps / The origins

Juxtaposed patterns (Cypher internal, Sept 2015)

MATCH (a)[
    (()-[:x]->()-[:y]->())* (()-[:z]->())]*(b)

• + Builds on previous pattern syntax
• − Large patterns require long strings
• − Nested patterns hard to understand
• + Pattern in one single place
• − No reuse of common pattern fragments

PGQL with declared patterns (first shared May 2016)

PATH X_Y := ()-[:x]->()-[:y]->()
PATH X_Y_Z := ()-/X_Y*/()-[:z]->()

MATCH (a)-/X_Y*/-(b)

• − New pattern grammar
• − Large patterns require lots of declarations
• + Nested patterns neatly decomposed
• − Pattern split over declaration and use
• + Easy reuse of declared patterns
Combining the two (February 2017)

- A pattern language over declared path predicates and shorthand syntax for simple edge labels

```
PATH PATTERN X_Y := ()-[[:x]]->()-[[:y]]->()
MATCH (a)-[/~X_Y* :Z]*/*/->(b)
```

- Puts focus on the pattern itself, avoids boilerplate, while adopting the benefits from PGQL
Studying examples - what syntax is needed?

- Most patterns focus on the types of relationships by which nodes are connected
  - Short syntax for grouping
- In some cases the pattern of connected nodes is more important
  - Short syntax for *Any Edge*
  - Short syntax for *Node predicates*
- Other times a combination of both
- Sometimes properties are important
  - Short syntax for property constant comparison
- Specifying direction (and bi-directional patterns) matters
The choice of delimiters

\((\cdot )-\lbrack\ldots\rbrack-(\cdot )\) \hspace{2cm} \((\cdot )-/\ldots/- (\cdot )\)

- + reuses/extends existing syntax
- - conflicts with existing syntax
- - complex rules for when variable can be bound
- - hard to declare empty pattern:
  - \((\cdot )-\lbrack\cdot \rbrack-(\cdot )\) already means single edge
- - unnatural operator precedence between disjunction and star

- - new pattern syntax
- + allows building good semantics

If we still use \([\cdot ]\) for grouping within a \((\cdot )-/\ldots/- (\cdot )\) pattern we free up the use of \((\cdot )\) for expressing node patterns. *This is a slight deviation from most other regular expression languages.*
Comparing to GXPath (February 2017)

**Cypher**

- Uses ascii-art (made for users)
- Only supports paths where a single continuous chain of edges can be bound as evidence.
- Supports all (other) capabilities of GXPath

**GXPath**

*L. Libkin, D. Vrgoč et al*

- Uses symbols and Greek letters (made for theoreticians)
- Supports arbitrary pairs
  - In particular pairs where a given pattern does not match.
  - Although omitting this makes GXPath tractable.
Comparing to RegExp on text (July 2017)

**Cypher**

- Atoms: Edge/Node labels
- Wildcard: – (any edge)
- Grouping using `[ ]`
- No short-form for disjunction of atoms - no need!

Otherwise the same constructs!

**RegExps over Text**

- Atoms: text literals
- Wildcard: .
- Grouping using `( )`
- Disjunction of atoms: `[ . . . ]`
- Negation of disjunction: `[ ^ . . . ]`
- Greedy, Lazy, Possessive...
Influencing GCore (August 2017)

- GCore pattern language defined mostly by: Tobias Lindaaker, Oskar van Rest, Peter Boncz, Pablo Barcelo
- Heavily influenced by Cypher - and the work done for Cypher (which of course means it is influenced by PGQL since Cypher is)

- GCore has the need to specify labels (and properties) for the path itself, thus encapsulates the actual pattern in further angle brackets:
  
  $$(\text{-}/p1<[^{\sim X \_ Y*}:Z]*>:\text{MyPath}\{\text{pathProp:'foo'}\}/-)$$

  If such details are not needed, they can of course be omitted:

  $$(\text{-}/<[^{\sim X \_ Y*}:Z]*>/-)$$
Complex cost functions

The influence back from GCore

• GCore deals with *Cheapest Paths*, thus needs to bind the cost
• There is also a need to specify the cost based on *branches* in the pattern:

```cypher
PATH PATTERN similar_friend = (a)-[:KNOWS]-(b),
    (a)-[al:LIKES]->(thing)<-[bl:LIKES]-(b)
// score is 1-5, we need '1-...' since we do cheapest
COST (1-al.score/5)*(1-bl.score/5)
MATCH ()-*/CHEAPEST similar_friend*/-()
```

• These are useful features that we bring back to Cypher
Other interesting comparisons to make

- Regular Expressions with Memory
- **Conjunctive Context Free Path Queries**
- Warded Datalog
  - Arenas et al., 2014
  - Gottlob and Pieris, 2015
  - *Used in VADALOG* [Bellomarini, Luigi, et al., 2017]
This is all documented in CIP 2017-02-06

(rendered document)