CIP2017-01-18

Configurable Pattern Matching Semantics

Stefan Plantikow, Mats Rydberg, Petra Selmer

Outline

Current Semantics

Paths, Morphisms, and Walks

Proposed Semantics

Extensions

Summary

Current Semantics

Simple patterns

```
MATCH <patterns>

MATCH ()                // node pattern

MATCH ()-[]->()         // relationship pattern
MATCH ()-[]-()          // (undirected version)

MATCH p=...             // path binding
```

What happens if we name patterns?

```
MATCH (a)-[r]->(b)
```

====> All matches spread across three fields: a, r, b

What happens if we combine patterns?

```
MATCH (a), (b)
```

====> Cross product over: a, b

What happens if we connect them?

```
MATCH (a)-[r1]->(b)<-[r2]-(c)
```

<===> This is the same as

```
MATCH (a)-[r1]->(b)
MATCH (b2)<-[r2]-(c)
WITH a, r1, b, r2, c
```

====> WHERE b = b2: Implicit join on b

====> AND r1 <> r2: Uniqueness

Graph Matching Morphisms

Homomorphism	Repeated nodes, Repeated relationships
Cyphermorphism (Relationship-Isomorphism)	Repeated nodes, No repeated relationships
Node-Isomorphism	No repeated nodes, No repeated relationships

Cyphermorphism in Cypher

Coined by Oskar van Rest from Oracle at oCIM 1:
"Cyphermorphism is really good"

All relationships matched by the same clause must be different

```
MATCH ()-[rel]->()-[rel_list*]->()<-[]<-[*]-()
```

- Doesn't matter if bound to a variable for a single relationship
- Doesn't matter if bound to a variable for a relationship list
- Doesn't matter if not bound to a variable

Benefits of Cyphermorphism

Coined by Oskar van Rest from Oracle at oCIM 1:
"Cyphermorphism is really good"

- **GOOD**: Fewer results by default
- **GOOD**: Never returns infinite results (never "stuck in a loop")
- **GOOD**: Proven in practice

Issues with Fixed Cyphermorphism

- Not always the right choice:
  Sometimes all matches are requested by the user
  - Opting out for simple patterns is cumbersome (split **MATCH** clause)
    `MATCH (a)-[r1]->(b) MATCH (b)-[r2]->(c), ...`
  - Can't opt out for unbounded variable length or shortest path patterns
- Occasionally confusing for new users; why do these patterns interact?
  `MATCH p1=(a)-[*]->(b), p2=(b)-[*]->(c)`

What is the next step?

- Should we have picked homomorphism as default back then?
  - Homomorphism more efficient for some path patterns (RPQs)
  - On the other hand: May lead to infinite results when enumerating all paths!

- In any case, let's enable users to switch semantics easily!

CIR-2017-174 Isomorphic pattern matching and configurable uniqueness

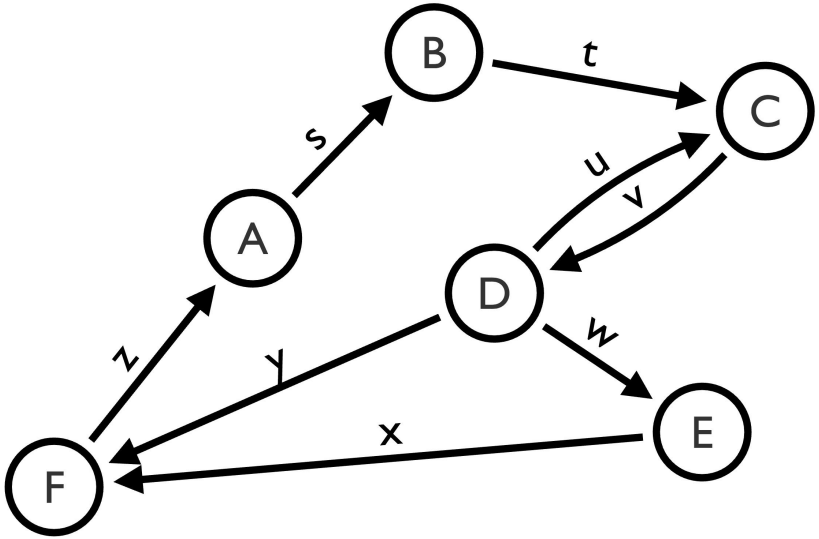
CIP-2017-01-18 **Configurable Pattern Matching Semantics**

Paths, Morphisms, and Walks

What's a path?

- Sequence of alternating nodes and relationships
- Starts with a node
- Ends with a node

...and that's where consensus stops :)



We mostly use definitions from

D. Jungnickel. *Graphs, Networks and Algorithms*. Springer Publishing Company, 2010

(Rosen seems to be less prevalent; we borrow "tidy path" though)

What's a walk?

Walk	Repeated nodes, Repeated relationships
Trail	Repeated nodes, No repeated relationships
(Tidy) Path	No repeated nodes, No repeated relationships
Open|Closed	Are start node and end node allowed to be the same node

Every tidy path is a trail

Every trail is a walk

Graph-Matching Morphisms vs Kinds of Walks

Homomorphism **<===> Walk**

Cyphermorphism **<===> Trail**

(Node-)Isomorphism **<===> Path**

Let's leverage this symmetry!

Proposed Semantics

Approach

Configurable semantics per walk

Default semantics that minimize breaking existing queries

STEP 1

Change to Pattern Variable Uniqueness

Pattern Variables

```
MATCH  p=...
       ^^^
```

Let's call this a **pattern variable** henceforth

Note: We're going to use `++` for path concatenation in the slides only

(This could go into the future CIP2017-05-18 Plus Operator Reform)

Today: Clause Uniqueness

```
MATCH p1=()-[r1]->(), p2=()-[r2]->()-[r3]->()
RETURN p1, p2
```

<===>

```
MATCH p1=()-[r1]->()
MATCH pa=()-[r2]->(x)
MATCH pb=(x)-[r3]->()
WITH * WHERE r1 <> r2 AND r2 <> r3 AND r1 <> r3
RETURN p1, pa++pb AS p2
```

Proposal: Pattern Variable Uniqueness

```
MATCH p1=()-[r1]->(), p2=()-[r2]->()-[r3]->()
RETURN p1, p2
```

<===>

```
MATCH p1=()-[r1]->()
MATCH pa=()-[r2]->(x)
MATCH pb=(x)-[r3]->()
WITH * WHERE r2 <> r3
RETURN p1, pa++pb AS p2
```

STEP 2

Introduce
Pattern Variable Class

Pattern Variable Classes

Key Idea:
If *morphisms correspond to different kinds of walks,
then configurable kinds of walks provide **configurable morphisms.**

`MATCH WALK`	Walk	Homomorphism
`MATCH TRAIL`	Trail	(Relationship-)Isomorphism
`MATCH PATH`	Path	(Node-)Isomorphism

Default Pattern Variable Class

- **`MATCH TRAILS`** aka Cyphermorphism remains the proven default
- Implementations are free to provide options for changing this
- Proposal suggests using **`MATCH WALKS`** for path patterns only

STEP 3

Introduce
Pattern Match Mode

Advanced Patterns

```
// variable length patterns
MATCH ()-[*]->()                // unbounded
MATCH ()-[*..2]->()             // bounded

// shortest path patterns
MATCH shortestPath(...)         // single (any)
MATCH allShortestPaths(...)     // all
```

Pattern Match Modes

Change which subset of all walks, trails, paths is to be matched

`MATCH ALL ...`	Every ...
`MATCH ALL SHORTEST ...`	Every shortest ...
`MATCH SHORTEST ...`	Single (any) shortest ...

Default Pattern Match Mode

Path patterns will often be used with shortest path but we don't want to switch to shortest path only, therefore we *default per sub-pattern*:

```
MATCH ()-[]->()  <==> MATCH ALL TRAILS ()-[]->()

MATCH ()-[*]->() <==> MATCH ALL TRAILS ()-[*]->()

MATCH ()-//->()  <==> MATCH ALL SHORTEST WALKS ()-//->()
```

Nice, concise syntax for shortest path by default!

Efficient path patterns by default!

Pattern
Variable Class
+ Match Mode

Configurable
Match Semantics

Infinite Results

```
MATCH WALKS ()-[*]->() // Error!
```

Some patterns produce infinite number of walks for cyclic graphs. To avoid:

(1) Must be requested explicitly by specifying the **ALL** match mode
(2) Implementations expected to generate warning

```
MATCH ALL WALKS ()-[*]->() // Ok, but dangerous
```

Extensions

Utility Functions

`isOpen(p)`	check if the source and target nodes of `p` are distinct
`isClosed(p)`	check if the source and target nodes of `p` are equal
`toTrail(p)`	`p` if `p` contains no duplicate relationships, `null` otherwise
`toPath(p)`	`toTrail(p)` if `p` contains no duplicate nodes at all besides the source and target nodes of `p`, `null` otherwise

Pre-Parser Option

What if existing applications need a different default?
Per-Parser Option to the rescue!

```
CYPHER match=all-trails MATCH ...
```

Change

default pattern variable class,
default pattern match mode,
or both!

More Match Modes upcoming

```
MATCH CHEAPEST BY ...
MATCH ALL CHEAPEST BY ...
```

More Pattern Variable Class Modifiers

```
// retains clause uniqueness
MATCH UNIQUE NODES ...
MATCH UNIQUE RELS ...
```

```
// reachability semantics if not bound
MATCH DISTINCT (a)-[*]->(b)
```

Summary

- Process Status
  - CIP drafted
  - Companion CIP for **MATCH CHEAPEST** upcoming
  - Next CIP (Multiple Graphs Syntax):
    Aim to finish 1 week before oCIG call for review

- Is this the right approach?
- Is this the right syntax? Is it too graph theory oriented?
    - **CON** Pattern variable uniqueness will break some queries
    - **PRO** Enables efficient RPQs / path patterns
    - **PRO** Grounded in graph theory
    - **PRO** Gives more control to users
    - **PRO** More intuitive uniqueness scope
    - **PRO** Extensible
    - ...

Thank you