Formal Semantics for Cypher Queries and Updates

Martin Schuster

University of Edinburgh

oCIM 4
Overview

Background:
- Ongoing project on formal definition of Cypher
- Started January 2017
- Edinburgh database group and Neo4j Cypher group

Status:
- Core is done (to be published in SIGMOD)
- Next stage: Looking at extensions
- This talk: Issues discovered at this stage
What we do:

- Mathematically model behaviour of Cypher
- Establish formal semantics

Why we do it:

- Provable insights into properties of Cypher
- Formal semantics is unambiguous
- Yields exact specification for implementers
Our goals

Our semantics aims to be:

- **Denotational**
  - Describes *what* Cypher statements do, not *how*
  - “how” left up to implementation

- **Deterministic**
  - Same Cypher statement on same graph/table should always yield same result
  - Not as trivial to achieve as it sounds (examples later)

- **Consistent**
  - Similar Cypher statements should be formalised similarly
  - Example: `MATCH/CREATE/MERGE`
Overview

1. Motivation

2. Our work so far
   - Query semantics
   - Reference implementation

3. Issues and open problems
   - Minor issues
   - Atomicity and determinism issues
Overview

1 Motivation

2 Our work so far
   - Query semantics
   - Reference implementation

3 Issues and open problems
   - Minor issues
   - Atomicity and determinism issues
Basics of query semantics (1)

Basic objects:
- Property graphs
- Tables (i.e. bags of records)

Query parts:
- Query: (Union of) clause sequence(s) terminated with RETURN clause
- Clause: “Atomic” statement, e.g.
  \[\text{[OPTIONAL]} \ \text{MATCH} \ \text{pattern\_tuple} \ [\text{WHERE} \ \text{expr} ]\]
- Subclause: Dependent part of clause, e.g. \[\text{[WHERE} \ \text{expr} \ ]\]
Basic properties:

- **Tables are unordered**

\[\{(a : 1, b : 3), (a : "Martin", b : "UoE"), (a : 1, b : 3)\} = \{(a : 1, b : 3), (a : 1, b : 3), (a : "Martin", b : "UoE")\}\]

- **Semantics of query \(Q\) given graph \(G\) maps tables to tables:**

\[\llbracket Q \rrbracket_G : \text{Tables} \rightarrow \text{Tables}\]

- **Output of \(Q\) on a \(G\): Result of evaluating \(Q\) on empty table given \(G\) (i.e. \(\llbracket Q \rrbracket_G(\{\})\)).**

Basics of update semantics

For updates: Slight change to query semantics

- Cypher statements (consisting of query and update clauses) map graphs and tables to graphs and tables
  \[ [S] : (\text{Graphs} \times \text{Tables}) \rightarrow (\text{Graphs} \times \text{Tables}) \]

- Queries are statements that do not change the graph
  \[ [Q](G, T) = (G, [Q]_G(T)) \]
Core fragment of Cypher queries already formalised:
UNION+[OPTIONAL ]MATCH+WHERE+UNWIND+WITH
(without aggregation)

- Accepted for publication in SIGMOD 2018

Current work (so far unpublished): Adding order, aggregation, updates.

⇝ arXiv version will be updated as more features get added
Reference implementation

- Parallel to work on semantics
- Direct (non-optimised) implementation of semantics
- Intention: Allow implementers to directly compare their implementation with formal semantics
- Current status: Core query fragment and order implemented; aggregation is WIP
Overview

1 Motivation

2 Our work so far
   - Query semantics
   - Reference implementation

3 Issues and open problems
   - Minor issues
   - Atomicity and determinism issues
Unification of CREATE and MERGE syntax

**CREATE** allows for…
- Path pattern tuples
- Directed relationships
- Unit-length relationships

**MERGE** allows for…
- Single path patterns
- Undirected relationships
- Unit-length relationships

Proposal: Unify syntax
- Path pattern tuples
- Directed relationships (avoid nondeterminism)
- Possible extension: Fixed-length relationships
Cypher 9 reference:
“WHERE adds constraints to the patterns in a MATCH or OPTIONAL MATCH clause or filters the results of a WITH clause.”

Different roles of WHERE as dependent subclause vs “stand-alone” clause

Current state: Filter anywhere but with OPTIONAL MATCH
(extreme example: WHERE false).

Proposal:
- WHERE only as subclause of OPTIONAL MATCH
- Introduce FILTER clause that can be used anywhere (not just after MATCH/WITH)
- For backward compatibility: Alias “stand-alone” WHERE to FILTER.
Consider `MATCH (a),(b) SET a.prop = b.prop`
What happens if `a` and `b` match multiple nodes?

$\rightsquigarrow$ Nondeterminism

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>$n_1$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>$n_2$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$n_1$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$n_2$</td>
</tr>
</tbody>
</table>

vs.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_2$</td>
<td>$n_1$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$n_2$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>$n_2$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>$n_1$</td>
</tr>
</tbody>
</table>

Current semantics approach:

- Leave ambiguous cases undefined
- Behaviour may depend on implementation
Atomicity of SET

Consider \texttt{SET a.prop = b.prop, b.prop = a.prop};
assume \texttt{a}’s and \texttt{b}’s are uniquely matched (i.e. no nondeterminism)

What should this do?

- Set \texttt{a.prop} to \texttt{b.prop}, then \texttt{b.prop} to \texttt{a.prop}? (i.e. like \texttt{SET a.prop = b.prop SET b.prop = a.prop})
- Or switch the values of \texttt{a.prop} and \texttt{b.prop}?

Current Neo4j implementation does the former, we propose the latter
Atomicity/semantics of DELETE

Consider $\text{MATCH (a)-[r]->(b) DELETE a,b ... DELETE r}$
Should this be allowed?

What does $\text{CREATE (n :label {num:1}) DELETE n RETURN n}$ return?

Proposal:

- **DELETE** is atomic, but order-independent inside clauses
  - $\text{MATCH (a)-[r]->(b) DELETE a,b,r}$ allowed
  - $\text{MATCH (a)-[r]->(b) DELETE a,b DELETE r}$ forbidden

- Strict semantics (as in CIR-2017-263): Deleted entities inaccessible immediately after deletion, retrieval attempts yield null or fail
Consider \texttt{CREATE (a \{num:1\})-[:r]->(b \{num:a.num\})}
Should this be allowed?

What about \texttt{CREATE (a \{num:1\}), (a)-[:r]->(b \{num:a.num\})}?  
What about \texttt{CREATE (a \{num:1\}), (a)-[:r]->(a :label)}?  

Is \texttt{CREATE \pi_1, \pi_2} the same as \texttt{CREATE \pi_1 CREATE \pi_2}?  

Proposal:
- \texttt{CREATE} may only access the graph prior to its execution  
- Labels/properties of newly created nodes must be stated at “first occurrence” of creation pattern
Atomicity/determinism of MERGE (1)

Given graph created by `CREATE (:a {num:1})-[r]->(:a {num:2})`

What should `MATCH (n:a) MERGE (n)-[r]->()-[r]->()` do?

What about `MERGE (n)-[r]-(m)` with example table (and no existing relationships)?

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>n₁</td>
<td>n₁</td>
</tr>
<tr>
<td>n₁</td>
<td>n₂</td>
</tr>
<tr>
<td>n₂</td>
<td>n₁</td>
</tr>
</tbody>
</table>

Should `MERGE` see...

- All of its own creations?
- Or only some?
- Or none at all?

This is still very much open (CIR-2018-296).
Atomicity/determinism of MERGE (2)

Proposal:

- **MERGE** can see prior graph $G$ and a change graph $G_c$ (initially empty).
- **MERGE** $\pi$ (with pattern $\pi$).
  - ...tries to match $\pi$ in $G$,
  - ...otherwise, tries to match $\pi$ in $G_c$,
  - ...otherwise, creates $\pi$ in $G_c$.
- Finally, **MERGE** inserts $G_c$ into $G$, attached at distinguished nodes.

Fixes nondeterminism for path pattern tuples with directed fixed-length relationships.
Summary

Query semantics: Mostly formalised, minor issues

Update semantics:
- Crucial issues: atomicity and determinism of statements
- Main factor in this: Driving tables are unordered

Any feedback is appreciated!