Formal semantics of Cypher:
towards a standard language for querying property graphs

N. Francis
V. Marsault
A. Green
T. Lindaaker
S. Plantikow
M. Rydberg
P. Selmer
A. Taylor
P. Guagliardo
L. Libkin
M. Schuster

1. Univ. Paris-Est Marne-la-Vallée, ENPC, ESIEE, CNRS
2. Neo Technology
3. University of Edinburgh

GT Alga
Inria Nord Europe, Lille
2018-10-15
1. Introduction

2. Property graphs

3. Regular Path Queries

4. Cypher by example

5. Principles of the semantics

6. Towards a standard language for querying property graphs
Most databases use the relational model
- Relational algebra in theory
- The language SQL in practice
Most databases use the relational model
- Relational algebra in theory
- The language SQL in practice

Some data have intrinsically the structure of graphs:
- Semantic web
- Social Networks
- Bioinformatic networks

Native representation of data as graphs allows:
- Efficient algorithms on graphs
- Pattern matching
- Optimisations
More and more ways to manipulate data as graphs

<table>
<thead>
<tr>
<th>Data model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property graphs, RDF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cypher, Gremlin, PGQL, SPARQL, G-Core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engines</th>
</tr>
</thead>
<tbody>
<tr>
<td>JanusGraph, Jena, Neo4j, Virtuoso</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraud detection, Investigative journalism</td>
</tr>
</tbody>
</table>
More and more ways to manipulate data as graphs

<table>
<thead>
<tr>
<th>Data model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property graphs, RDF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cypher, Gremlin, PGQL, SPARQL, G-Core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Engines</th>
</tr>
</thead>
<tbody>
<tr>
<td>JanusGraph, Jena, Neo4j, Virtuoso</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraud detection, Investigative journalism</td>
</tr>
</tbody>
</table>
Cypher

- Language for querying and updating *property graphs*

- Invented by Neo Technology
- Originally, part of engine Neo4j → commercial success
- Now, in multiple datagraph engines (e.g., SAP HANA Graph, Redis Graph, Agens Graph)
Cypher

- Language for querying and updating *property graphs*

- Invented by Neo Technology
- Originally, part of engine Neo4j → commercial success
- Now, in multiple datagraph engines (e.g., SAP HANA Graph, Redis Graph, Agens Graph)

The openCypher project

- Since 2015
- Seeks to standardise Cypher (SQL for property graphs?)
  - Community-led evolution
  - Complete specification
**Short term goal**

Full denotational semantics for the language Cypher.

- Industrial partnership Neo4j/University of Edinburgh
- Reverse engineering and formalisation from Neo4j
- **Done** semantics of the “core fragment” [Francis et al’18]
- **Soon**: semantics of the “update clauses”
Short term goal

Full denotational semantics for the language Cypher.

- Industrial partnership Neo4j/University of Edinburgh
- Reverse engineering and formalisation from Neo4j
- Done semantics of the “core fragment” [Francis et al’18]
- Soon: semantics of the “update clauses”

Long term goal

Design a standard language for querying property graphs: GQL.

- Merging Cypher, PGQL, G-Core.
- Involvement of Neo Technology, Oracle and LDBC.
1. Introduction

2. Property graphs

3. Regular Path Queries

4. Cypher by example

5. Principles of the semantics

6. Towards a standard language for querying property graphs
Property graph

- Nodes: $N_1, N_2, \ldots, N_5$

$N_1$ follows $N_2$

$N_3$ posted on: "05-14"

$N_4$ follows $N_5$

$N_5$ posted on: "05-15"

Nodes:

- $N_1$
- $N_2$
- $N_3$
- $N_4$
- $N_5$

Relationships:

- follows
- posted
- answers

Labels (de nœuds):

- User
- Message

Types (of relationships):

- follows
- posted
- answers

Properties (i.e. Key/Value pairs):

- name: "Alice"
- id: 22
- text: "Hello"
- id: 25
- text: "World"
Nodes: \( N_1, N_2, \ldots, N_5 \)

Relationships: \( r_1, r_2, \ldots, r_7 \)
Nodes: \(N_1, N_2, \cdots, N_5\)

Relationships: \(r_1, r_2, \cdots, r_7\)

Labels (de nœuds):
- User
- Message
Nodes: $N_1, N_2, \cdots, N_5$
Relationships: $r_1, r_2, \cdots, r_7$

- Labels (de nœuds):
  - User
  - Message
- Types (of relationships):
  - Follows
  - Posted
  - Answers
- **Nodes**: $N_1, N_2, \cdots, N_5$
- **Relationships**: $r_1, r_2, \cdots, r_7$

- **Labels (de nœuds)**:
  - *User*
  - *Message*

- **Types (of relationships)**:
  - *FOLLOWS*
  - *POSTED*
  - *ANSWERS*

- **Properties** *(i.e. Key/Value pairs)*:
  - **name**: "Alice"
  - **id**: 22
  - **text**: "Hello"
  - **id**: 25
  - **text**: "World"
Outline

1 Introduction

2 Property graphs

3 Regular Path Queries

4 Cypher by example

5 Principles of the semantics

6 Towards a standard language for querying property graphs
Data model for theoretical query languages

Graph database
- Relation bear types
- Node do not bear types (could be simulated)
- Neither bears property
Data model for theoretical query languages

Graph database

- Relation bear types
- Node do not bear types
  (could be simulated)
- Neither bears property
Data model for theoretical query languages

Graph database

- Relation bear types
- Node do not bear types (could be simulated)
- **Neither bears property**

Finite number of relation types (and node types)

→ “Dataless graph”
Regular Path Query (RPQ) – Definition

\[ A: \text{finite alphabet (of relation types)} \]
\[ N: \text{nodes in the graph} \]

Definition (RPQ \( R \)):
\[ R = (x, E, y) \]
\[
\begin{align*}
E & : \text{regular expr. over } A \\
x, y & : \text{two variables}
\end{align*}
\]

Example:
\[ \xrightarrow{(a + b)^*} \]

Definition (Answer to \( R \)):
Set of the functions
\[ F : \{x, y\} \rightarrow N, \ \exists u \in E, \ F(x) \xrightarrow{u} F(y) \]
Regular Path Query (RPQ) – Example 1

Query:

\[ x \xrightarrow{\text{POSTED}} (\text{ANSWERS})^* \xrightarrow{y} \]

Answers:

\[ F_1 : \ x \mapsto N_1 \quad y \mapsto N_4 \]
\[ F_2 : \ x \mapsto N_2 \quad y \mapsto N_5 \]
\[ F_3 : \ x \mapsto N_2 \quad y \mapsto N_4 \]
Regular Path Query (RPQ) – Example 1

Query:

\[ \text{ posted.}(\text{ answers})^* \]

Answers:

\[
\begin{align*}
F_1 : & \quad x \mapsto N_1 \quad y \mapsto N_4 \\
F_2 : & \quad x \mapsto N_2 \quad y \mapsto N_5 \\
F_3 : & \quad x \mapsto N_2 \quad y \mapsto N_4
\end{align*}
\]
Regular Path Query (RPQ) – Example 1

Query:

POSTED.(ANSWERS)*

Answers:

\[ F_1 : \ x \mapsto N_1 \quad y \mapsto N_4 \]

\[ F_2 : \ x \mapsto N_2 \quad y \mapsto N_5 \]

\[ F_3 : \ x \mapsto N_2 \quad y \mapsto N_4 \]
Regular Path Query (RPQ) – Example 1

Query:

\[ x \xrightarrow{\text{POSTED}.(\text{ANSWERS})^*} y \]

Answers:

\[ F_1 : \ x \mapsto N_1 \quad y \mapsto N_4 \]
\[ F_2 : \ x \mapsto N_2 \quad y \mapsto N_5 \]
\[ F_3 : \ x \mapsto N_2 \quad y \mapsto N_4 \]
Regular Path Query (RPQ) – Example 1

Query:

\[ x \xrightarrow{\text{POSTED} \cdot (\text{ANSWERS}^*)} y \]

Answers:

\[
\begin{align*}
F_1 : & \ x \mapsto N_1 \quad y \mapsto N_4 \\
F_2 : & \ x \mapsto N_2 \quad y \mapsto N_5 \\
F_3 : & \ x \mapsto N_2 \quad y \mapsto N_4
\end{align*}
\]
Regular Path Query (RPQ) – Example 2

Query:

\[(\text{FOLLOWS})^*\]

\[x \rightarrow \vdash \text{RPQ follows set-semantics}\]
Regular Path Query (RPQ) – Example 2

Query:

\[ (\text{FOLLOW})^* \]

\[
\begin{align*}
\text{Query:} & \\
\text{x} & \xrightarrow{(\text{FOLLOW})^*} \text{y} \\
\end{align*}
\]
Regular Path Query (RPQ) – Example 2

Query:
\[(\text{FOLLOWS}^*)\]

Answers:
- \(F_1 : x \mapsto N_1, y \mapsto N_1\)
- \(F_2 : x \mapsto N_1, y \mapsto N_2\)
- \(F_3 : x \mapsto N_1, y \mapsto N_3\)
- \(\vdots \vdots \vdots \vdots \)
- \(F_9 : x \mapsto N_3, y \mapsto N_3\)
**Regular Path Query (RPQ) – Example 2**

**Query:**

\[
\xrightarrow{(\text{FOLLOWS})^*}
\]

**Answers:**

- \( F_1 : x \mapsto N_1 \quad y \mapsto N_1 \)
- \( F_2 : x \mapsto N_1 \quad y \mapsto N_2 \)
- \( F_3 : x \mapsto N_1 \quad y \mapsto N_3 \)
- \( \vdots \quad \vdots \quad \vdots \)
- \( F_9 : x \mapsto N_3 \quad y \mapsto N_3 \)
Regular Path Query (RPQ) – Example 2

Query:
\[ x \xrightarrow{(\text{FOLLOWS})^*} y \]

Answers:
\[
F_1 : \ x \mapsto N_1 \quad y \mapsto N_1 \\
F_2 : \ x \mapsto N_1 \quad y \mapsto N_2 \\
F_3 : \ x \mapsto N_1 \quad y \mapsto N_3 \\
F_9 : \ x \mapsto N_3 \quad y \mapsto N_3
\]
Query:

\[ x \xrightarrow{(\textsc{follows})^*} y \]

Answers:

\[
\begin{align*}
F_1 & : \quad x \mapsto N_1 \quad y \mapsto N_1 \\
F_2 & : \quad x \mapsto N_1 \quad y \mapsto N_2 \\
F_3 & : \quad x \mapsto N_1 \quad y \mapsto N_3 \\
\vdots & \quad \vdots \quad \vdots \\
F_9 & : \quad x \mapsto N_3 \quad y \mapsto N_3
\end{align*}
\]
Regular Path Query (RPQ) – Example 2

Query:

(FOLLOW)∗

Answers:

\[
\begin{align*}
F_1 & : \ x \mapsto N_1 \quad y \mapsto N_1 \\
F_2 & : \ x \mapsto N_1 \quad y \mapsto N_2 \\
F_3 & : \ x \mapsto N_1 \quad y \mapsto N_3 \\
: & \quad : \quad : \\
F_9 & : \ x \mapsto N_3 \quad y \mapsto N_3
\end{align*}
\]

→ ∞-many path realise $F_1$
→ RPQ follows set-semantics
Conjunction of RPQs (CRPQ) – Definition

\( A \): alphabet (of relation types)
\( N \): nodes in the graph

Definition (a CRPQ \( C \)):
\[
C = (R_1 \land R_2 \land \cdots \land R_n)
\]
where \( R_1, \ldots, R_n \) are RPQs

Example:

\[ (a + b)^* \quad (ab)^* \]

Answer to \( C \):
Set of the functions \( F : \text{var}(C) \rightarrow N \)

such that \( \forall i, \ F|_{\text{var}(R_i)} \) is an answer to \( R_i \)
Conjunction of RPQs (CRPQ) – Example

Query:

\[
\begin{array}{c}
\text{POSTED} \\
\overset{r_4}{\longrightarrow} \\
\overset{r_3}{\longrightarrow} \\
\overset{r_2}{\longrightarrow} \\
\overset{r_1}{\longrightarrow} \\
\overset{r_5}{\longrightarrow} \\
\overset{r_6}{\longrightarrow} \\
\overset{r_7}{\longrightarrow} \\
\end{array}
\]

\(x\) \quad \(y\) \quad \(z\) 

\((\text{ANSWERS})^*\)

Answers:

\[
\begin{array}{c}
F_1 : \ N_1 \quad N_4 \quad N_4 \\
F_2 : \ N_2 \quad N_5 \quad N_4 \\
F_3 : \ N_2 \quad N_5 \quad N_5 \\
\end{array}
\]

User name: “Alice”

User name: “Bob”

User name, Admin name: “Charlie”

Message id: 22
Message text: “Hello”

Message id: 25
Message text: “World”

User 1 follows User 2
User 3 follows User 4
User 1 posted on: “05-14”
User 2 posted on: “05-15”

x \quad y \quad z
Union of CRPQs (UCRPQ) – Definition

\[ A: \text{alphabet (of relation types)} \]
\[ N: \text{nodes in the graph} \]

**Definition (UCRPQ Q)**

\[ Q = (C_1 \cup C_2 \cup \cdots \cup C_n) \]

where \( C_1, C_2, \ldots, C_n \) are CRPQs

**Answer to Q:**

Set of partial functions: \((\text{var}(C_1) \cup \cdots \cup \text{var}(C_n)) \rightarrow N\)

\[ \bigcup_{i=1}^{n} F_i , \]

where, \( \forall i, F_i \) is the answer to \( C_i \).
Outline

1 Introduction

2 Property graphs

3 Regular Path Queries

4 Cypher by example

5 Principles of the semantics

6 Towards a standard language for querying property graphs
Evaluating a Cypher query

**Example of Cypher query:**

MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
WHERE t1 = "Hello"

A Cypher statement
- is a sequence of *clauses*
Evaluating a Cypher query

Example of Cypher query:

MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
WHERE t1 = "Hello"

A Cypher statement
- is a sequence of clauses
- queries a graph
- returns a table
Query:
MATCH (u1:User)
Query:
MATCH (u1:User)

Result:

u1
N1
N2
N3
Query:
MATCH (u1:User:Admin)

Result:
__
<table>
<thead>
<tr>
<th>u1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
</tr>
</tbody>
</table>

MATCHING NODES (2)

- **User, Admin**
  - Name: "Charlie"

- **User**
  - Name: "Alice"

- **User**
  - Name: "Bob"

- **Message**
  - ID: 22
  - Text: "Hello"

- **Message**
  - ID: 25
  - Text: "World"

- Follows
  - User 1 (Alice) follows User 2 (Bob)
  - User 3 (Charlie) follows User 2 (Bob)

- POSTED
  - On: "05-14"
  - On: "05-15"
Matching nodes (3)

Query:
MATCH (u1{id:22})

Result:
\[ \text{u1} \]

\[ N_4 \]
Matching relations (1)

Query:
MATCH ()-[p1]->()

Result:

```
p1
  r1
  r2
  r3
  r4
  r5
  r6
  r7
```
Matching relation (2)

Query:
MATCH (u1)-[p1:POSTED]->(m1)

Result:

<table>
<thead>
<tr>
<th>u1</th>
<th>p1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>r5</td>
<td>N4</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>N5</td>
</tr>
</tbody>
</table>

Message
- id: 22
- text: "Hello"

Message
- id: 25
- text: "World"
Query:
MATCH (u1)-[:FOLLOWS]->()

Result:

Cypher has bag semantics:
$N_2$ has two outgoing $FOLLOWS$ relations $\Rightarrow$ two lines $N_2$
Query:
MATCH (u1:Admin)
-[:FOLLOWS*]->(m1)

Result:
<table>
<thead>
<tr>
<th>u1</th>
<th>l1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>[r4]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1]</td>
<td>N2</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r2]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r3]</td>
<td>N3</td>
</tr>
</tbody>
</table>

Cypher-Morphism
Each rel. matched \( \leq 1 \) time \( \Rightarrow \) Finitely many results
Matching paths (1)

**Query:**

MATCH (u1:Admin)
-[:FOLLOWS*]-(m1)

**Result:**

<table>
<thead>
<tr>
<th>u1</th>
<th>l1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>[r4]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1]</td>
<td>N2</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r2]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r3]</td>
<td>N3</td>
</tr>
</tbody>
</table>

- User name: "Charlie"
- User name: "Alice"
- User name: "Bob"
- Message id: 22, text: "Hello"
- Message id: 25, text: "World"
Query:
MATCH (u1:Admin)
  -[l1:FOLLOWS*]->(m1)

Result:
<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>l1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>N3</td>
</tr>
<tr>
<td>N3</td>
<td>[r4]</td>
<td>N1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[r4, r1]</td>
<td>N2</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r2]</td>
<td>N1</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r3]</td>
<td>N3</td>
<td></td>
</tr>
</tbody>
</table>

**PyTorch:**
```python
MATCH (u1:Admin)
  -[l1:FOLLOWS*]->(m1)
RETURN u1, l1, m1
```

**Cypher-Morphism:**

Each rel. matched ≤ 1 time ⇒ Finitely many results
Matching paths (1)

Query:
MATCH (u1:Admin)
    -[l1:FOLLOWS*]-(m1)

Result:
<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>l1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>[r4]</td>
<td>N1</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1]</td>
<td>N2</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r2]</td>
<td>N1</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r3]</td>
<td>N3</td>
<td></td>
</tr>
</tbody>
</table>
Query:
MATCH (u1:Admin) -[r1:FOLLOWS*]- (m1)

Result:

<table>
<thead>
<tr>
<th>u1</th>
<th>r1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>[r4]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1]</td>
<td>N2</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r2]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r3]</td>
<td>N3</td>
</tr>
</tbody>
</table>

MATCHING PATHS

**User**, Admin
name: "Charlie"

**User**
name: "Alice"

**User**
name: "Bob"

**Message**
id: 22
text: "Hello"

**Message**
id: 25
text: "World"
Matching paths (1)

Query:
MATCH (u1:Admin) -[l1:FOLLOWS*]-(m1)

Result:
<table>
<thead>
<tr>
<th>u1</th>
<th>l1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>[r4]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1]</td>
<td>N2</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r2]</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>[r4, r1, r3]</td>
<td>N3</td>
</tr>
</tbody>
</table>

Cypher-Morphism
- Each rel. matched \( \leq 1 \) time
⇒ Finitely many results
Matching paths (2)

Query:
MATCH (u1)-[:FOLLOWS]->() -[:POSTED]->(m1)

Result:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>m1</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>N5</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N4</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>N5</td>
<td></td>
</tr>
</tbody>
</table>
Query:
```
MATCH (u1)-[:FOLLOWS]->() -[:POSTED]->(m1)
```

Result:
```
<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N5</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N4</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>N5</td>
<td></td>
</tr>
</tbody>
</table>
```
Chaining clauses

Query:
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[[:FOLLOWS]-](u1)
-[:FOLLOWS]->(u3)
Chaining clauses

Query:
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
-[:FOLLOWS]->(u3)

Table after first MATCH:

<table>
<thead>
<tr>
<th>u1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N4</td>
</tr>
<tr>
<td>N2</td>
<td>N5</td>
</tr>
</tbody>
</table>

Table after second MATCH:
Chaining clauses

Query:
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
-[:FOLLOWS]->(u3)

Table after first MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N1</td>
<td>N4</td>
</tr>
<tr>
<td>N2</td>
<td>N2</td>
<td>N5</td>
</tr>
</tbody>
</table>

Table after second MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
<th>u2</th>
<th>u3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chaining clauses

Query:
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
-[:FOLLOWS]->(u3)

Table after first MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N4</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N5</td>
<td></td>
</tr>
</tbody>
</table>

Table after second MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
<th>u2</th>
<th>u3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chaining clauses

Query:
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
-[:FOLLOWS]->(u3)

Table after first MATCH:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>m1</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>N4</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table after second MATCH:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>m1</td>
<td>u2</td>
<td>u3</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>N4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- User(name: "Alice")
- User(name: "Bob")
- User(name: "Charlie")
- POSTED(on: "05-14")
- POSTED(on: "05-15")
- Answers
- Message(id: 22, text: "Hello")
- Message(id: 25, text: "World")
- Follows
- Follows
- Follows
- Follows
Chaining clauses

Query:
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
-[:FOLLOWS]->(u3)

Table after first MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N4</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N5</td>
<td></td>
</tr>
</tbody>
</table>

Table after second MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
<th>u2</th>
<th>u3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>N5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

User, Admin
name: "Charlie"

User
name: "Alice"

User
name: "Bob"

POSTED
on: "05-14"

POSTED
on: "05-15"

Message
id: 22
text: "Hello"

Message
id: 25
text: "World"
Chaining clauses

Query:
MATCH (u1)-[r1]:POSTED]->(m1)
MATCH (u2)<-[r2]:FOLLOWS]--(u1)
  -[r3]:FOLLOWS]->(u3)

Table after first MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N4</td>
<td>N5</td>
</tr>
</tbody>
</table>

Table after second MATCH:

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>m1</th>
<th>u2</th>
<th>u3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>N5</td>
<td>N1</td>
<td>N3</td>
<td>N1</td>
</tr>
</tbody>
</table>

User
name:"Alice"

User
name:"Bob"

User , Admin
name:"Charlie"

POSTED
on:"05-14"

POSTED
on:"05-15"

Message
id:22
text:"Hello"

Message
id:25
text:"World"

Message
id:22
text:"Hello"

Message
id:25
text:"World"
Column manipulation (**WITH clause**)

Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

After the MATCH clause

```
<table>
<thead>
<tr>
<th>u1</th>
<th>p1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>r5</td>
<td>N4</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>N5</td>
</tr>
</tbody>
</table>
```

User name: "Charlie"

User name: "Alice"

User name: "Bob"

Message id: 22
- text: "Hello"

Message id: 25
- text: "World"
Column manipulation (WITH clause)

Query:
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1

After the MATCH clause
```
<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>p1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N1</td>
<td>r5</td>
<td>N4</td>
</tr>
<tr>
<td>2</td>
<td>N2</td>
<td>r6</td>
<td>N5</td>
</tr>
</tbody>
</table>
```

Execution of the WITH clause
```
<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>p1</th>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N1</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>N2</td>
<td>r6</td>
<td></td>
</tr>
</tbody>
</table>
```

- User, Admin
  - name: "Charlie"
- User
  - name: "Alice"
- User
  - name: "Bob"
- POSTED
  - on: "05-14"
- POSTED
  - on: "05-15"
- Message
  - id: 22
  - text: "Hello"
- Message
  - id: 25
  - text: "World"
Column manipulation (WITH clause)

Query:

MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1

After the MATCH clause

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>p1</td>
<td>m1</td>
</tr>
<tr>
<td>N1</td>
<td>r5</td>
<td>N4</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>N5</td>
</tr>
</tbody>
</table>

Execution of the WITH clause

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>p1</td>
<td>t1</td>
</tr>
<tr>
<td>N1</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td></td>
</tr>
</tbody>
</table>
Query:
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1

After the MATCH clause

Execution of the WITH clause

Column manipulation (WITH clause)
Column manipulation (WITH clause)

Query:
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1

After the MATCH clause
---------
<table>
<thead>
<tr>
<th>u1</th>
<th>p1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>r5</td>
<td>N4</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>N5</td>
</tr>
</tbody>
</table>

Execution of the WITH clause
---------
<table>
<thead>
<tr>
<th>u1</th>
<th>p1</th>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>r5</td>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>&quot;World&quot;</td>
</tr>
</tbody>
</table>
Column manipulation (WITH clause)

Query:
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1

After the MATCH clause

<table>
<thead>
<tr>
<th>u1</th>
<th>p1</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>r5</td>
<td>N4</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>N5</td>
</tr>
</tbody>
</table>

Final result

<table>
<thead>
<tr>
<th>u1</th>
<th>p1</th>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>r5</td>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>&quot;World&quot;</td>
</tr>
</tbody>
</table>
Line filtering (WHERE clause)

Query:
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
WHERE t1 = "Hello"

After the WITH clause

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>p1</td>
<td>t1</td>
</tr>
</tbody>
</table>
| N1    | r5    | "Hello"
| N2    | r6    | "World"

User, Admin
name: "Charlie"

User
name: "Alice"

User
name: "Bob"

POSTED
on: "05-14"

POSTED
on: "05-15"

Message
id: 22
text: "Hello"

Message
id: 25
text: "World"
Line filtering (**WHERE** clause)

**Query:**

MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
WHERE t1 = "Hello"

After the **WITH** clause

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>p1</th>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td></td>
<td></td>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>N2</td>
<td></td>
<td></td>
<td>&quot;World&quot;</td>
</tr>
</tbody>
</table>

**Execution of the **WHERE** clause**

<table>
<thead>
<tr>
<th></th>
<th>u1</th>
<th>p1</th>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td></td>
<td></td>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>N2</td>
<td></td>
<td></td>
<td>&quot;World&quot;</td>
</tr>
</tbody>
</table>

**Diagram:**

- **User** node
  - Name: "Alice"
- **User** node
  - Name: "Bob"
- **User** node
  - Name: "Charlie"
- **POSTED** node
  - On: "05-14"
- **POSTED** node
  - On: "05-15"
- **FOLLOW**s relationship
  - u1 to p1
  - p1 to m1
  - m1 to t1

**Nodes:**

- N1
- N2
- N3
- N4
- N5
Line filtering (WHERE clause)

Query:
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
WHERE t1 = "Hello"

After the WITH clause

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>p1</td>
<td>t1</td>
</tr>
<tr>
<td>N1</td>
<td>r5</td>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>N2</td>
<td>r6</td>
<td>&quot;World&quot;</td>
</tr>
</tbody>
</table>

Final result

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>p1</td>
<td>t1</td>
</tr>
<tr>
<td>N1</td>
<td>r5</td>
<td>&quot;Hello&quot;</td>
</tr>
</tbody>
</table>
Query:
MATCH (a{name: "Bob"})
-[*]->(b{id: 22})
<-*-(a)

Question
What does this computes?
A last read-only example

Query:
MATCH (a{name:"Bob"})-[*]->(b{id:22})<-[*]-(a)

Question
What does this computes ?

Answer
The # of pairs of disjoint paths from $N_2$ to $N_4$. 
Query:

MATCH (a {name: "Bob"})
-[*]-(b {id: 22})
<-[*]-(a)

Question:
What does this computes?

Answer:
The # of pairs of disjoint paths from $N_2$ to $N_4$.

$\Rightarrow$ Evaluation of one constant
MATCH is NP-HARD
**Node and relation creation (CREATE)**

**Query:**
```
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]-> (b:User)
```

**Table after MATCH clause:**
```
+--+-
| a | N3 |
+--+-
```

**Table after CREATE clause:**
```
+------+-+----------+
| a     | b        |
|-------+-+----------|
| N1    | N2       |
+-------+-+----------|
| N3    | N3       |
|       | N3       |
|       | N3       |
+-------+-+----------+

**Nodes and Relations:**
- **User 1:** name: "Alice"
- **User 2:** name: "Bob"
- **User 3:** name: "Charlie"
- **Message 1:** id: 22, text: "Hello"
- **Message 2:** id: 25, text: "World"
- **Relation 1:** r1
- **Relation 2:** r2
- **Relation 3:** r3
- **Relation 4:** r4
- **Relation 5:** r5
- **Relation 6:** r6
- **Relation 7:** r7
- **Follows:**
  - N1 -> N2
  - N2 -> N3
  - N3 -> N4
  - N3 -> N5
- **Posted:**
  - N1 -> N4 on: "05-14"
  - N2 -> N5 on: "05-15"
Node and relation creation (CREATE)

Query:
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]-> (b:User)

Table after MATCH clause:

| a
| N3 |

Table after CREATE clause:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>N3</td>
</tr>
</tbody>
</table>
Node and relation creation (CREATE)

Query:
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]-> (b:User)

Table after MATCH clause:
___
a
___
N3

Table after CREATE clause:
___
a  b
___
N3  N6
The example graph stored as CREATE clauses

Query:

CREATE

(n1:User{name:"Alice"}),
(n2:User{name:"Bob"}),
(n3:User:Admin

{name:"Charlie"}),
(n4:Message {id:22,

    text:"Hello"}),
(n5:Message {id:25,

    text:"World"})

CREATE

(n1)-[:FOLLOWS]->(n2),
(n1)-[:POSTED

    {on:"05-04"}]->(n4),
(n2)-[:FOLLOWS]->(n1),
(n2)-[:FOLLOWS]->(n3),
(n2)-[:POSTED

    {on:"05-04"}]->(n5),
(n3)-[:FOLLOWS]->(n1),
(n5)-[:ANSWERS]->(n4),
Query:
MATCH (a{name: "Charlie"})
CREATE (a)-[:FOLLOWS]-> (b:User)
SET b:Admin, b.name="Eve"

Table after CREATE clause:
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>N6</td>
<td></td>
</tr>
</tbody>
</table>

Node/Relation modification (SET clause)
Query:
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]-> (b:User)
SET b:Admin, b.name="Eve"

Table after CREATE clause:
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>N6</td>
<td></td>
</tr>
</tbody>
</table>
The MERGE clause: MATCH else CREATE

**Input table:**

- **a**
- **n**

<table>
<thead>
<tr>
<th>N3</th>
<th>&quot;Alice&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
</tr>
</tbody>
</table>

**Query:**

```
MERGE (a)-[:FOLLOWS]-> (b:User {name:n})
```

**Output table:**

<table>
<thead>
<tr>
<th>a</th>
<th>n</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Alice&quot;</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
<td></td>
</tr>
</tbody>
</table>

**User, Admin**
- **name:** "Charlie"

**User**
- **name:** "Alice"
- **FOLLOWS**
- **on:** "05-14"

**User**
- **name:** "Bob"
- **FOLLOWS**
- **on:** "05-15"

**Message**
- **id:** 22
- **text:** "Hello"

**Message**
- **id:** 25
- **text:** "World"
The MERGE clause: MATCH else CREATE

Input table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>n</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Alice&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Query:

MERGE (a)-[:FOLLOWS]-> (b:User {name:n})

Output table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>n</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Alice&quot;</td>
<td>N1</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The MERGE clause: MATCH else CREATE

Input table:

<table>
<thead>
<tr>
<th>N3</th>
<th>&quot;Alice&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
</tr>
</tbody>
</table>

Query:

MERGE (a)-[:FOLLOWS]-> (b:User {name:n})

Output table:

<table>
<thead>
<tr>
<th>a</th>
<th>n</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Alice&quot;</td>
<td>N1</td>
</tr>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
<td>N6</td>
</tr>
</tbody>
</table>

Message

id: 22
text: "Hello"

Message

id: 25
text: "World"
The MERGE clause: MATCH else CREATE

Input table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>n</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Alice&quot;</td>
<td>N1</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
<td>N6</td>
<td></td>
</tr>
</tbody>
</table>

Query:

```
MERGE (a)-[:FOLLOWS]->
(b:User {name:n})
```

Output table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>n</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>&quot;Alice&quot;</td>
<td>N1</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>&quot;Eve&quot;</td>
<td>N6</td>
<td></td>
</tr>
</tbody>
</table>
**Other constructs**

- **DELETE** deletes node and relations.
  
  Ex: `MATCH (a{name:"Eve"}) DELETE a`

- **REMOVE** removes labels or properties.
  
  Ex: `MATCH (a{name:"Charlie"}) REMOVE a:Admin,a.name`

- **WITH** allows to perform aggregations.
  
  Ex: `MATCH (a)-[[:FOLLOWS]->(b) WITH a, count(b) as c`

- **ORDER BY** limits size of table.
  
  Ex: `MATCH (a:User) ORDER BY a.name LIMIT 1`
Outline

1. Introduction

2. Property graphs

3. Regular Path Queries

4. Cypher by example

5. Principles of the semantics

6. Towards a standard language for querying property graphs
Record (table row)

A *record* is a partial function from variables to values.

Example: \((x \mapsto \text{"Bob"} ; y \mapsto 1)\)
Record (table row)

A *record* is a partial function from variables to values.

Example: \((x \mapsto \text{"Bob"}; y \mapsto 1)\)

Table

A *table* is a multi-set (or bag) of records with the same domain.

Example:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>(y)</td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;Alice&quot;</td>
<td>999</td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>
Modelling Tables

Record (table row)

A *record* is a partial function from variables to values.

Example: \( (x \mapsto "Bob" ; y \mapsto 1) \)

Table

A *table* is a multi-set (or bag) of records with the same domain.

Example:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>(y)</td>
<td></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>&quot;Alice&quot;</td>
<td>999</td>
<td></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

= 

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(y)</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>&quot;Alice&quot;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
<td></td>
</tr>
</tbody>
</table>
### Semantics brackets – Read-Only

<table>
<thead>
<tr>
<th>G: a graph</th>
</tr>
</thead>
</table>

#### Semantics of expressions

\[
\boxed{\cdot}_{u,G} : \text{expression} \mapsto \text{value} \quad (\text{where } u \text{ is a record})
\]

#### Semantics of clauses

\[
\boxed{\cdot}_{G} : \text{clause} \mapsto (\text{function: Tables} \rightarrow \text{Tables})
\]

#### Semantics of queries

\[
\boxed{\cdot}_{G} : \text{query} \mapsto (\text{function: Tables} \rightarrow \text{Tables})
\]

output : (Graphs \times \text{Queries}) \mapsto \text{Tables}
Principle of Read-Only Semantics

\[ G: \text{a graph} \]
\[ Q: \text{a query} \]

To compute the output of \( Q \)

- \( Q \) is a sequence of clauses \( Q = C_1 C_2 \cdots C_n \)
**Principle of Read-Only Semantics**

\( G \): a graph  
\( Q \): a query

To compute the output of \( Q \)

- \( Q \) is a sequence of clauses \( Q = C_1 \ C_2 \ldots \ C_n \)
- Compute \( \begin{bmatrix} C_1 \end{bmatrix}_G \), \( \begin{bmatrix} C_2 \end{bmatrix}_G \), \ldots , \( \begin{bmatrix} C_n \end{bmatrix}_G \)
Principle of Read-Only Semantics

- $G$: a graph
- $Q$: a query

To compute the output of $Q$

- $Q$ is a sequence of clauses $Q = C_1 C_2 \cdots C_n$
- Compute $\left[ C_1 \right]_G$, $\left[ C_2 \right]_G$, $\cdots$, $\left[ C_n \right]_G$
- Let $\left[ Q \right]_G = \left[ C_n \right]_G \circ \cdots \circ \left[ C_2 \right]_G \circ \left[ C_1 \right]_G$
$G$: a graph
$Q$: a query

To compute the output of $Q$

- $Q$ is a sequence of clauses $Q = C_1 C_2 \cdots C_n$
- Compute $\begin{bmatrix} C_1 \end{bmatrix}_G, \begin{bmatrix} C_2 \end{bmatrix}_G, \cdots, \begin{bmatrix} C_n \end{bmatrix}_G$
- Let $\begin{bmatrix} Q \end{bmatrix}_G = \begin{bmatrix} C_n \end{bmatrix}_G \circ \cdots \circ \begin{bmatrix} C_2 \end{bmatrix}_G \circ \begin{bmatrix} C_1 \end{bmatrix}_G$
- $\text{output}(G, Q) = \begin{bmatrix} Q \end{bmatrix}_G \left( T_{\text{unit}} \right)$

where $T_{\text{unit}}$ is the 1-line 0-column table.
Excerpt of Read-Only semantics (2)

- \[\left[ \text{WHERE } e \right]_G(T) = \left\{ u \in T \mid \left[ e \right]_{G,u} = \text{true} \right\}\]

- \[\left[ \text{MATCH } \vec{\pi} \right]_G(T) = \bigcup_{u \in T} \left\{ u \cdot u' \mid u' \in \text{match}(\vec{\pi}, G, u) \right\}\]

- \[\left[ \text{MATCH } \vec{\pi} \text{ WHERE } e \right]_G(T) = \left[ \text{WHERE } e \right]_G(\left[ \text{MATCH } \vec{\pi} \right]_G(T))\]

- \[\left[ \text{OPTIONAL MATCH } \vec{\pi} \text{ WHERE } e \right]_G(T) = \bigcup_{u \in T} \left\{ \left[ \text{MATCH } \vec{\pi} \text{ WHERE } e \right]_G(\{u\}) \mid \left[ \text{MATCH } \vec{\pi} \text{ WHERE } e \right]_G(\{u\}) \neq \emptyset \right\}\]

- \[\left[ \text{OPTIONAL MATCH } \vec{\pi} \right]_G(T) = \left[ \text{OPTIONAL MATCH } \vec{\pi} \text{ WHERE true} \right]_G(T)\]
Excerpt of Read-Only semantics (2)

- \[ \left[ \text{WITH } \ast \right]_G (T) = T \text{ if } T \text{ has at least one column} \]

- \[ \left[ \text{WITH } \ast, e_1 \text{ [AS } a_1\text{],} \ldots, e_m \text{ [AS } a_m\text{]} \right]_G (T) = \left[ \text{WITH } b_1 \text{ AS } b_1, \ldots, b_q \text{ AS } b_q, e_1 \text{ [AS } a_1\text{],} \ldots, e_m \text{ [AS } a_m\text{]} \right]_G (T) \]

- \[ \left[ \text{WITH } e_1 \text{ [AS } a_1\text{],} \ldots, e_m \text{ [AS } a_m\text{]} \right]_G (T) = \bigcup_{u \in T} \left\{ (a'_1 : \left[ e_1 \right]_{G,u}, \ldots, a'_m : \left[ e_m \right]_{G,u}) \right\} \]

- \[ \left[ \text{UNWIND } e \text{ AS } a \right]_G (T) = \bigcup_{u \in T} \bigcup_{v \in E_u} \{ (u, a : v) \}, \]

with \( E_u = \begin{cases} \bigcup_{0 \leq i < m} \{ v_i \} & \text{if } \left[ e \right]_{G,u} = \text{list}(v_0, \ldots, v_{m-1}) \\ \{ \} & \text{if } \left[ e \right]_{G,u} = \text{list()} \\ \left[ e \right]_{G,u} \right) \end{cases} \) otherwise
Semantics brackets – Read-Write

Semantics of expressions: (Nothing changes)
\[
\boxed{\cdot}_{u,G} : \text{expression} \mapsto \text{value} \quad (\text{where } u \text{ is a record})
\]

Semantics of clauses:
\[
\boxed{\cdot} : \text{clause} \mapsto (\text{function}: (\text{Graphs} \times \text{Tables}) \rightarrow (\text{Graphs} \times \text{Tables}))
\]

Semantics of queries:
\[
\boxed{\cdot} : \text{query} \mapsto (\text{function}: (\text{Graphs} \times \text{Tables}) \rightarrow (\text{Graphs} \times \text{Tables}))
\]
output : (Graphs \times Queries) \mapsto (Graphs \times \text{Tables})

(Computed just like RO \rightarrow \text{composition of clause semantics})
### Atomicity:
Each clause is executed as a single unit

### Consistency:
Each clause should on valid Graph/Table pair

### General scheme of the semantics if a clause is:
1. Ensure that the input Graph/Table is valid w.r.t. clause
2. Compute output Table and all changes to graph
3. Apply all changes to Graph
4. Ensure validity of output Graph/Table

If any fails, semantics is undefined.
Desirable properties of the semantics at Clause-level

Atomicity:
Each clause is executed as a single unit

Consistency:
Each clause should on valid Graph/Table pair

General scheme of the semantics if a clause is:
1. Ensure that the input Graph/Table is valid w.r.t. clause
2. Compute output Table and all changes to graph
3. Apply all changes to Graph
4. Ensure validity of output Graph/Table
If any fails, semantics is undefined.

In Neo4j, Atomicity and Consistency are verified at query level only.
In Neo4j, SET violate Atomicity

Query:
MATCH (m1)-[:ANSWERS]->(m2)
SET m1.id = m2.id,
m2.id = m1.id

Table before SET clause:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>m2</td>
</tr>
<tr>
<td>N5</td>
<td>N4</td>
</tr>
</tbody>
</table>

Neo4j sets both id to 25.
Semantics exchange id values
In Neo4j, `SET` violate Atomicity

Query:
```
MATCH (m1)-[:ANSWERS]->(m2)
SET m1.id = m2.id,
    m2.id = m1.id
```

Table before `SET` clause:

```
<table>
<thead>
<tr>
<th>m1</th>
<th>m2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N5</td>
<td>N4</td>
</tr>
</tbody>
</table>
```

Neo4j sets both `id` to 25.

Semantics exchange `id` values
In Neo4j, **MERGE** is highly non-deterministic.

**Input Table:**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N₁</td>
<td>N₂</td>
<td>N₃</td>
</tr>
<tr>
<td>2</td>
<td>N₂</td>
<td>N₃</td>
<td>N₁</td>
</tr>
<tr>
<td>3</td>
<td>N₃</td>
<td>N₁</td>
<td>N₂</td>
</tr>
</tbody>
</table>

**Query:**

```sql
MERGE (a)-[:FOLLOWING]->(b)
-[:FOLLOWING]->(c)
```

**Neo4j creates 4 edges, Semantics propose different semantics creating either 3 or 6.**
In Neo4j, **MERGE** is highly non-deterministic.

**Input Table:**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N₁</td>
<td>N₂</td>
<td>N₃</td>
</tr>
<tr>
<td>2</td>
<td>N₂</td>
<td>N₃</td>
<td>N₁</td>
</tr>
<tr>
<td>3</td>
<td>N₃</td>
<td>N₁</td>
<td>N₂</td>
</tr>
</tbody>
</table>

**Query:**

```cypher
MERGE (a)-[FOLLOWS]->(b)
-FOLLOWS-(c)
```

**Neo4j creates 4 edges**
In Neo4j, **MERGE** is highly non-deterministic.

Input Table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>N2</td>
<td>N3</td>
<td>N1</td>
</tr>
<tr>
<td>N2</td>
<td>N3</td>
<td>N1</td>
<td>N2</td>
</tr>
</tbody>
</table>

Query:

```
MERGE (a)-[:FOLLOWS]->(b)
-[:FOLLOWS]->(c)
```

Neo4j creates 4 edges.

Semantics propose different semantics creating either 3 or 6.
In Neo4j, **DELETE** violates clause-level consistency.

Query:
```
MATCH (a {name: "Alice"})
MATCH (a)-[r]-()
DELETE a

.DELETE r  // Arbitrary clauses
RETURN a.name AS n
```

- Execution of arbitrary code on invalid graph
- Access to deleted-entity
In Neo4j, DELETE violates clause-level consistency

Query:

MATCH (a {name: "Alice"})
MATCH (a)-[r]-()
DELETE a

[..] // Arbitrary clauses
DELETE r
RETURN a.name AS n

- Execution of arbitrary code on invalid graph
- Access to deleted-entity

Semantics

- is undefined if it should produce invalid graphs
- replaces deleted entities by null in the table
1. Introduction
2. Property graphs
3. Regular Path Queries
4. Cypher by example
5. Principles of the semantics
6. Towards a standard language for querying property graphs
## Cypher vs RPQ

### Cypher queries vs UCRPQs

- Cypher has bag + cypher-morphism semantics
  - (Set+ standard morphism semantics may be emulated...)
- Cypher has relationship/path variables
- (Data model is different)

### RPQs not expressible in Cypher

- \((ab)^*\): no concatenation under star in Cypher
- \((a^*)^*\): no nested stars in Cypher
- \((a + b^{-1})^*\): some unions are not allowed under star in Cypher

\[
(ab + cd)^3: \text{ nested alternations of concatenations and unions require multi-exponential blow-up of the query}
\]
\[
\rightarrow (ab)^3 + (ab)^2 cd + ab(cd)^2 + (cd)^3
\]
 PGQL (Property Graph Query Language)

- Designed by Oracle Inc.
- Support full UCRPQ
- ASCII-art representation of patterns (similar to Cypher)
- Syntax close to SQL

Example: (from http://pgql-lang.org/)

```sql
SELECT p1.name
FROM facebook_graph /* In the Facebook graph,.. */
MATCH (p1:Person) /* ..find persons such that.. */
WHERE NOT EXISTS ( /* ..there does not exist.. */
    SELECT p2
    FROM twitter_graph /* ..in the Twitter graph.. */
    MATCH (p2:Person) /* ..a person. */
    WHERE p1.name = p2.name /* ..with the same name. */
)
```
- Designed by LDBC
- Experimental
- Support full UCRPQ
- ASCII-art representation of patterns (similar to Cypher)
- Support Multiple graphs, graph views, and paths cost
- Multiple semantics

Example:

CONSTRUCT  (n)-[:To{distance: c}]->(m)
MATCH  (n) -/SHORTEST  p<:KNOWS*>  COST  c/->(m)
  ON  facebook_graph
MATCH  (n),(m)  ON  upem_graph
cf. GQL manifesto: http://gql.today
cf. GQL manifesto: http://gql.today

To be continued...