Formal Specification of Cypher

Nadime Francis

University of Edinburgh

Wednesday, May, 10th
A property graph is a tuple $G = (N, R, s, t, \iota, \lambda, \tau)$, where:

- $N \subseteq \mathcal{N}$: finite set of nodes
- $R \subseteq \mathcal{R}$: finite set of relationships
- $s : R \rightarrow N$: maps each relationship to its source
- $t : R \rightarrow N$: maps each relationship to its target
- $\iota : (N \cup R) \times \mathcal{K} \rightarrow \mathcal{V}$: maps each $x$ and $k$ to $x.k$.
- $\lambda : N \rightarrow 2^\mathcal{L}$: associates a set of label to each node
- $\tau : R \rightarrow \mathcal{T}$: associates a type to each relationship
Records and Tables

A record is a tuple with named fields: \((a_1 : v_1, \ldots, a_n : v_n)\).

A table is a bag of uniform records.

Example: \(\{ (a : 1, b : 3), (a : 'oCIM 2', b : 'London'), (a : 'oCIM', b : 'Walldorf'), (a : 1, b : 3) \} \)

\[
\begin{array}{|c|c|}
\hline
a & b \\
\hline
1 & 3 \\
'oCIM 2' & 'London' \\
oCIM & 'Walldorf' \\
1 & 3 \\
\hline
\end{array} = \begin{array}{|c|c|}
\hline
a & b \\
\hline
'oCIM' & 'Walldorf' \\
1 & 3 \\
oCIM 2' & 'London' \\
1 & 3 \\
\hline
\end{array}
\]
Operations and Expressions
An Example

MATCH (n : Person) − [ : knows ]−> (m : Person)
WHERE n.institute = m.institute
RETURN n.name, m.name, n.institute AS institute
Operations and Expressions

An Example

MATCH (n : Person) − [: knows]−> (m : Person)
WHERE n.institute = m.institute
RETURN n.name, m.name, n.institute AS institute
Operations and Expressions
An Example

MATCH (n : Person) − [: knows]−> (m : Person)
WHERE n.institute = m.institute
RETURN n.name, m.name, n.institute AS institute

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
</tr>
</thead>
</table>
Operations and Expressions
An Example

MATCH (n : Person) − [: knows]−→ (m : Person)
WHERE n.institute = m.institute
RETURN n.name, m.name, n.institute AS institute

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{name : 'Nadime', institute : 'UoE'}</td>
<td>{name : 'Leonid', institute : 'UoE'}</td>
</tr>
<tr>
<td></td>
<td>{name : 'Paolo', institute : 'UoE'}</td>
<td>{name : 'Nadime', institute : 'UoE'}</td>
</tr>
<tr>
<td></td>
<td>{name : 'Alastair', institute : 'Neo'}</td>
<td>{name : 'Stefan', institute : 'Neo'}</td>
</tr>
</tbody>
</table>
Operations and Expressions
An Example

MATCH (n : Person) - [: knows] - (m : Person)
WHERE n.institute = m.institute
RETURN n.name, m.name, n.institute AS institute

<table>
<thead>
<tr>
<th>n.name</th>
<th>m.name</th>
<th>institute</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Nadime’</td>
<td>‘Leonid’</td>
<td>UoE</td>
</tr>
<tr>
<td>‘Paolo’</td>
<td>‘Nadime’</td>
<td>UoE</td>
</tr>
<tr>
<td>‘Alastair’</td>
<td>‘Stefan’</td>
<td>Neo</td>
</tr>
</tbody>
</table>
Operations and Expressions

\[
Q = \begin{cases}
\alpha: \text{MATCH} \ (n : \text{Person}) - [: \text{knows}] \rightarrow (m : \text{Person}) \\
\beta: \text{WHERE} \ n.\text{institute} = m.\text{institute} \\
\gamma: \text{RETURN} \ n.\text{name}, m.\text{name}, n.\text{institute} \ \text{AS} \ \text{institute}
\end{cases}
\]
Operations and Expressions

\[
Q = \begin{cases} 
(\alpha) \text{ MATCH } (n : Person) - [[: \text{knows}]] - \text{\rightarrow} (m : Person) \\
(\beta) \text{ WHERE } n.\text{institute} = m.\text{institute} \\
(\gamma) \text{ RETURN } n.\text{name}, m.\text{name}, n.\text{institute} \text{ AS institute}
\end{cases}
\]

Operations

- \([op]_G : Tables \rightarrow Tables\)
- Semantics of a query by composition

\textbf{Ex: } \([Q]_G = [\alpha]_G \circ [\beta]_G \circ [\gamma]_G\)
- Answers to \(Q\) on \(G\): \([Q]_G(\{\})\)
Operations and Expressions

$$Q = \begin{cases} 
(\alpha) & \text{MATCH } (n : \text{Person}) - [: \text{knows}] \to (m : \text{Person}) \\
(\beta) & \text{WHERE } n.\text{institute} = m.\text{institute} \\
(\gamma) & \text{RETURN } n.\text{name}, m.\text{name}, n.\text{institute} \text{ AS institute} 
\end{cases}$$

Operations

- \([\text{op}]_G : \text{Tables} \to \text{Tables}\)
- Semantics of a query by composition

Ex: \([Q]_G = (\alpha)_G \circ (\beta)_G \circ (\gamma)_G\)
- Answers to \(Q\) on \(G\): \([Q]_G(\{\})\)

Expressions

- \([\text{exp}]_{G, u} \in \mathcal{V}\) where \(u\) is a record, giving binding to variables

Ex: \([\beta]_G(T) = \{ u \in T \mid [n.\text{institute} = m.\text{institute}]_{G, u} = \text{true} \} \)
Pattern Matching

Rigid pattern satisfaction

- **Rigid** path pattern: no variable length edge patterns.

  **Ex:** 
  $\langle n : Person \rangle \rightarrow [\text{knows} * 2] \rightarrow () \rightarrow [\text{likes}] \rightarrow \langle m : Movie \rangle$

- Unique way for a path $p$ to satisfy a rigid pattern $\pi$ wrt $G, u$. Notation: $(p, G, u) \models \pi$
Pattern Matching

### Rigid pattern satisfaction

- **Rigid** path pattern: no variable length edge patterns.

**Ex:** \((n : Person) \rightarrow [: knows \ast 2] \rightarrow () \rightarrow [: likes] \rightarrow (m : Movie)\)

- Unique way for a path \(p\) to satisfy a \textbf{rigid} pattern \(\pi\) wrt \(G, u\).
  
  Notation: \((p, G, u) \models \pi\)

### Variable-length paths and free variables

- \(\text{rigid}(\pi) = \{\pi' | \pi' \text{ is rigid and } \pi \sqsubseteq \pi'\}\)

**Ex:** \((()) \rightarrow ([*2] \rightarrow ()) \sqsubseteq (()) \rightarrow ([*4] \rightarrow ()) \rightarrow ()\)

- \(\text{free}(\pi, u): \) all names that occur in \(\pi\) and not in \(u\)
Pattern Matching

Rigid pattern satisfaction

- **Rigid** path pattern: no variable length edge patterns.

  **Ex:** \((n : Person) \rightarrow [: knows * 2] \rightarrow () \rightarrow [: likes] \rightarrow (m : Movie)\)

- Unique way for a path \(p\) to satisfy a **rigid** pattern \(\pi\) wrt \(G, u\). Notation: \((p, G, u) \models \pi\)

Variable-length paths and free variables

- \(\text{rigid}(\pi) = \{ \pi' \mid \pi' \text{ is rigid and } \pi \sqsubseteq \pi' \}\)

  **Ex:** \((() \rightarrow [*2] \rightarrow () \rightarrow [*4] \rightarrow () \sqsubset () \rightarrow [*1..3] \rightarrow () \rightarrow [*] \rightarrow ()\)

- \(\text{free}(\pi, u)\): all names that occur in \(\pi\) and not in \(u\)

\[
\boxed{\text{MATCH } \pi}_G(T) = \bigcup_{\pi' \in \text{rigid}(\pi)} \left\{ (u, u') \mid u' \text{ is uniform with } \text{free}(\pi', u) \text{ and } (p, G, (u, u')) \models \pi' \right\}
\]
Ambiguous and Edge Cases
Nulls in Patterns

MATCH (n: Person {name: null})
RETURN (n)
Nulls in Patterns

MATCH (n : Person {name : null})
RETURN (n)

1. Every node \( n \) with a name property?
2. Every node \( n \) such that \( n.name \) IS NULL = true?
3. Nothing?
Nulls in Patterns

MATCH (n : Person {name : null})
RETURN (n)

1. Every node $n$ with a name property?

2. Every node $n$ such that $n$.name IS NULL = true?

3. Nothing!

Because $Q$ is actually equivalent to:

MATCH (n : Person)
WHERE n.name = null
RETURN (n)
Map Comparisons

When does \( \{ k_1 : v_1, \ldots, k_n : v_n \} = \{ \ell_1 : w_1, \ldots, \ell_m : w_m \} \) return true, false or null?

\[
\{ \text{name : null} \} = \{ \}
\]

\[
\{ a : 1, b : 2 \} = \{ b : 2, a : 1 \}
\]

\[
\{ \text{name : null} \} = \{ \text{name : null} \}
\]

\[
\{ a : 1, a : 2 \} = \{ a : 2 \}
\]
Map Comparisons

When does \( \{ k_1 : v_1, \ldots, k_n : v_n \} = \{ \ell_1 : w_1, \ldots, \ell_m : w_m \} \) return true, false or null?

- **False**
  \( \{ \text{name} : \text{null} \} = \{ \} \)

- **True**
  \( \{a : 1, b : 2\} = \{b : 2, a : 1\} \)
  \( \{ \text{name} : \text{null} \} = \{ \text{name} : \text{null} \} \)
  \( \{a : 1, a : 2\} = \{a : 2\} \)

Neither purely syntactic, nor \( \forall k, m_1.k = m_2.k \).
Setting Properties using a Map

```
WITH {name : null} AS map
CREATE (n)
SET n = map
RETURN (n)
```

Returns `n` as `{}`.
The property map of `n` is not equal to the map it was set to.
In particular, `n.{}` = `map` returns `false`.

Setting Properties using a Map

WITH \{name : null\} AS map
CREATE (n)
SET n = map
RETURN (n)

Returns \( n \) as \{\}. The property map of \( n \) is not equal to the map it was set to. In particular, \( n \{\ast\} = map \) returns false.
MATCH with no Free Variables

MATCH ()
RETURN *

MATCH ()
RETURN 1

is not allowed with no variable in scope.

Returns as many copies of 1 as nodes in the database.

After MATCH(), the active table is a bag containing multiple copies of the empty record.
MATCH with no Free Variables

Fail
MATCH ()
RETURN *

RETURN * is not allowed with no variable in scope.

Pass
MATCH ()
RETURN 1

Returns as many copies of 1 as nodes in the database.

After MATCH (), the active table is a bag containing multiple copies of the empty record.
Incomplete and Inconsistent Cases
Repeating UNWINDs

```
UNWIND [1, 2, 3] AS r
UNWIND r AS s
RETURN s
```

```
UNWIND [[1, 2], 3] AS r
UNWIND r AS s
UNWIND s AS t
UNWIND t AS u
RETURN u
```

Type mismatch, expected List but was Integer.

Returns a column with 1, 2 and 3 as rows.
Repeating UNWINDs

<table>
<thead>
<tr>
<th>Fail</th>
<th>Pass</th>
</tr>
</thead>
</table>
| **UNWIND** [1, 2, 3] **AS** r  
**UNWIND** r **AS** s  
**RETURN** s |  
**UNWIND** [[1, 2], 3] **AS** r  
**UNWIND** r **AS** s  
**RETURN** s |

Type mismatch, expected List but was Integer.

Returns a column with 1, 2 and 3 as rows.
Repeating UNWINDs

<table>
<thead>
<tr>
<th>Fail</th>
<th>Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNWIND [1, 2, 3] AS r</strong></td>
<td><strong>UNWIND [[1, 2], 3] AS r</strong></td>
</tr>
<tr>
<td><strong>UNWIND r AS s</strong></td>
<td><strong>UNWIND r AS s</strong></td>
</tr>
<tr>
<td><strong>RETURN s</strong></td>
<td><strong>RETURN s</strong></td>
</tr>
</tbody>
</table>

Type mismatch, expected List but was Integer.

Returns a column with 1, 2 and 3 as rows.

**UNWIND [[1, 2], 3] AS r**
**UNWIND r AS s**
**UNWIND s AS t**
**UNWIND t AS u**
**RETURN u**

Actually works, and returns a column with 1, 2 and 3 as rows.
Violating Cyphermorphism

\[ Q_0 = \text{MATCH } (x) - [r] - (y) - [r] - (z) \]
\[ \text{RETURN } x, y, z \]
Violating Cyphermorphism

\[ Q_0 = \text{MATCH } (x) - [r] - (y) - [r] - (z) \]
\[ \text{RETURN } x, y, z \]

**Error:** cannot use the same relationship variable ‘r’ for multiple patterns.
Violating Cyphermorphism

\[ Q_0 = \text{MATCH } (x) - [r] - (y) - [r] - (z) \]
\[ \text{RETURN } x, y, z \]

**Error**: cannot use the same relationship variable ‘r’ for multiple patterns.

\[ Q_1 = \text{MATCH } (x) - [r^*] - (y) - [r^*] - (z) \]
\[ \text{RETURN } x, y, z \]
Violating Cyphermorphism

\[ Q_0 = \text{MATCH } (x) - [r] - (y) - [r] - (z) \]
\[ \text{RETURN } x, y, z \]

**Error:** cannot use the same relationship variable ‘r’ for multiple patterns.

\[ Q_1 = \text{MATCH } (x) - [r^*] - (y) - [r^*] - (z) \]
\[ \text{RETURN } x, y, z \]

Works and enforces the paths from \( x \) to \( y \) and from \( y \) to \( z \) to use the same sequence of relationships, violating **Cyphermorphism**.
Violating Cyphermorphism

\[ Q_0 = \text{MATCH} \ (x) - [r] - (y) - [r] - (z) \]
\[ \text{RETURN} \ x, y, z \]

Error: cannot use the same relationship variable ‘r’ for multiple patterns.

\[ Q_1 = \text{MATCH} \ (x) - [r*] - (y) - [r*] - (z) \]
\[ \text{RETURN} \ x, y, z \]

Works and enforces the paths from \( x \) to \( y \) and from \( y \) to \( z \) to use the same sequence of relationships, violating Cyphermorphism. It is not included in the query \( Q_2 \) below:

\[ Q_1 = \text{MATCH} \ (x) - [r*] - (y) - [s*] - (z) \]
\[ \text{RETURN} \ x, y, z \]
Semantics Overview

Ambiguous and Edge Cases

Incomplete and Inconsistent Cases

Two Small Issues: Nulls as Indices and Keys, and Naming

RETURN 1 AS '0', 0
Two Small Issues: Nulls as Indices and Keys, and Naming

Fail

```
RETURN 1 AS '0', 0
```

Multiple result columns with the same name are not supported. Need to specify how expression naming is handled.
Two Small Issues: Nulls as Indices and Keys, and Naming

Fail

```
RETURN 1 AS '0', 0
```

Multiple result columns with the same name are not supported. Need to specify how expression naming is handled.

```
[1, 2, 3][null]  [1, 2, 3][null..4]
{name : 'Nadime'}[null]
```
Two Small Issues: Nulls as Indices and Keys, and Naming

Fail

```
RETURN 1 AS '0', 0
```

Multiple result columns with the same name are not supported. Need to specify how expression naming is handled.

Fail

```
[1, 2, 3][null]  [1, 2, 3][null..4]
{ name : 'Nadime' }[null]
```

No error, never terminates.
Thank you!