Mechanizing the GQL semantics in Coq

Semyon Panenkov – JetBrains Research
Anton Podkopaev – JetBrains Research, Constructor University Bremen
Semyon Grigorev
Correctness problems during standardization

A new language **GQL** is currently being standardized.

However, the standard may have something:
- specified incorrectly
- underspecified

which may make implementing the standard hard or nearly impossible
Formalization and mechanization

One of the solutions is to mathematically **formalize** the standard. However, doing this is still prone-to-errors and notorious.

The next step is to **mechanize** the formalization in a proof-assistant. We then have machine-checked proofs which we can automate to some extent.
A Semantics of GQL
A New Query Language for Property Graphs
Formalized

O. H. Morra
o.h.morra@student.tue.nl

A Coq mechanised formal semantics for realistic SQL queries * Formally reconciling SQL and bag relational algebra
Véronique Benzaken, Évelyne Contejean

Toward a Verified Relational Database Management System *
Gregory Malecha  Greg Morrisett  Avraham Shinnar  Ryan Wisnesky
Harvard University, Cambridge, MA, USA
{malecha, greg, shinnar, ryan}@cs.harvard.edu

Abstract
We report on our experience implementing a lightweight, fully verified relational database management system (RDBMS). The functional specification of RDBMS behavior, RDBMS implementation, and proof that the implementation meets the specification are all written and verified in Coq. Our contributions include: (1) a complete specification of the relational algebra in Coq; (2) an efficient implementation of that model (8+ trees) implemented with the Yocto extension to Coq; and (3) a set of simple query optimizations proven to respect both semantics and run-time cost. In addition to describing the design and implementation of these artifacts, we highlight
The goal

We want to mechanize the core subset of GQL and capture the key implementation details of Neo4j and RedisGraph.
The subset of GQL we formalize

MATCH (u)-[e]->(v)
RETURN *

path pattern

match-clause
MATCH (u)-[e]->(v)

return-clause
RETURN *

(p:Person)-[e:WORKS_FOR]->(v:Company {name: "JB"})

names

labels

properties
GQL path patterns and resulting tables

Path pattern and matching paths

Resulting table

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
GQL pattern normalization

In the specification all vertex and edge patterns have names but some of them may be marked as implicit.
Execution plans

Databases translate queries into execution plans:

```
MATCH (u)-[anon]->(v)
RETURN *
```

```
ScanVertices
```

```
ExpandAll(u, anon, v, →)
```

```
ReturnAll
```
Execution plans evaluation

Operations transform an intermediate table to produce the result:

ScanVertices(u) → ExpandAll(u, anon, v, →) → ReturnAll

<table>
<thead>
<tr>
<th>u</th>
<th>anon</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
The big picture

Specification of GQL

Query evaluation
- Neo4j
- RedisGraph

Translation algorithms

Specification of execution plans

Implementation of execution plan operations

Satisfies

Relies on

Denotational semantics
RedisGraph optimizes the query evaluation using linear algebra:

Path pattern: $\langle u \rangle - [ \ ] - \langle v \rangle - [ \ ] - \langle w \rangle$

Pattern slices:
- $\langle u \rangle - [\ ] - \langle v \rangle$
- $\ldots - [\ ] - \langle w \rangle$

Matrix expressions:

\[
\begin{bmatrix}
\ldots & 1 & \\
1 & 1 & \\
1 & 1 & \\
\end{bmatrix}
\]
The results

1. Mechanized the specification of the core subset of the GQL standard
2. Mechanized the specification of the execution plan
3. Implemented and proved the correctness of the translation of the queries
4. Provided an example implementation of the execution plan evaluation

Correctness means that, according to the specification of the execution plan, the evaluation of translated queries satisfies the specification of GQL.
Limitations

<table>
<thead>
<tr>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

ORDERED BY, DISTINCT, count()

considered to be the same

SO...

cannot be formalized
A former “bug” in RedisGraph

MATCH p=(u)-[ ]->()-[ ]->(v)
RETURN count(p)

... returns 1

MATCH p=(u)-[ ]->(x)-[ ]->(v)
RETURN count(p)

... returns 2
The results

1. Mechanized the specification of the core subset of the GQL standard ✔
2. Mechanized the specification of the execution plan ✔
3. Implemented and proved the correctness of the translation of the queries ✔
4. Provided an example implementation of the execution plan evaluation ✔

Future plans

- Constrain the order and the number of repetitions
- Expand the covered subset of GQL
- Make the framework more extensible