# 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

Internship
Faculty of Mathe
Eindhoven Univ

Micheal Sch
George Fletch

# A Coq mechanised formal semantics for realistic SQL queries * Formally reconciling SQL and bag relational algebra

Véronique Benzaken, Évelyne Contejean

l formal semantics for realistic SQL queries
8. hal-01830255v2

55
330255v2
018

e ouverte pluridisciplinaire **HAL**, est
dépôt et à la diffusion de documents
de niveau recherche, publiés ou non,

# Toward a Verified Relational Database Management System *

Gregory Malecha Greg Morrisett Avraham Shinnar Ryan Wisnesky
Harvard University, Cambridge, MA, USA
{gmalecha, 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 realization of that model (B+ trees) implemented with the Ynot 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

ager would be proven correct with respect to this specification to ensure that a bug cannot lead to accidental corruption or disclosure. It is for these reasons that we see *verified* RDBMSs as a compelling challenge to the programming languages and software verification communities that moves beyond the now successful domains of verified compilers and theorem provers.

As a step towards this goal, we have constructed a verified, lightweight, in-memory RDBMS using the Coq proof assistant [2]. Currently, our RDBMS supports queries, written in a stylized subset of SQL, over an in-memory relational store that can be [de]serialized to disk. As such, it provides much of the function-

# 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

path pattern

match-clause
```
MATCH (u)-[e]->(v)
```
return-clause
```
RETURN *
```

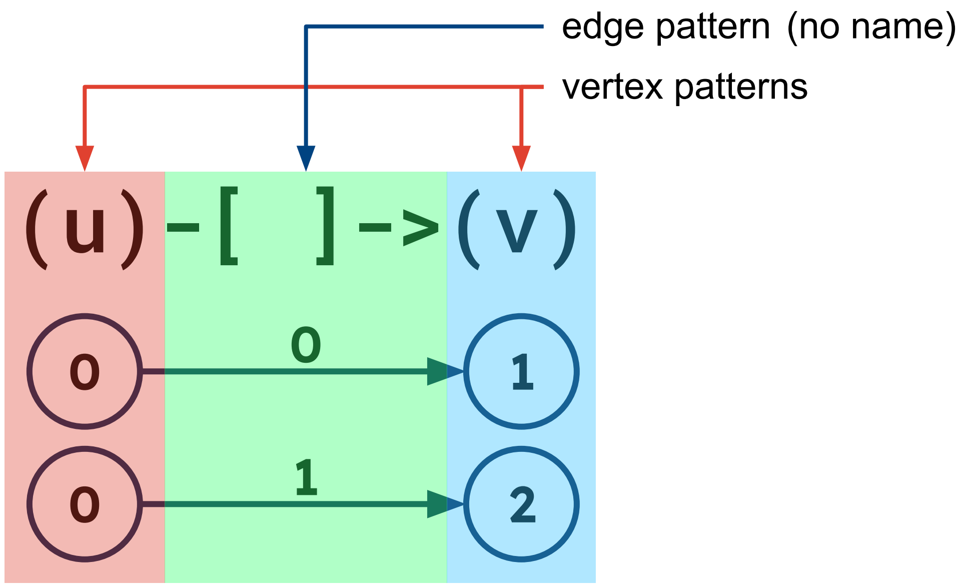
names

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

labels

properties

# GQL path patterns and resulting tables

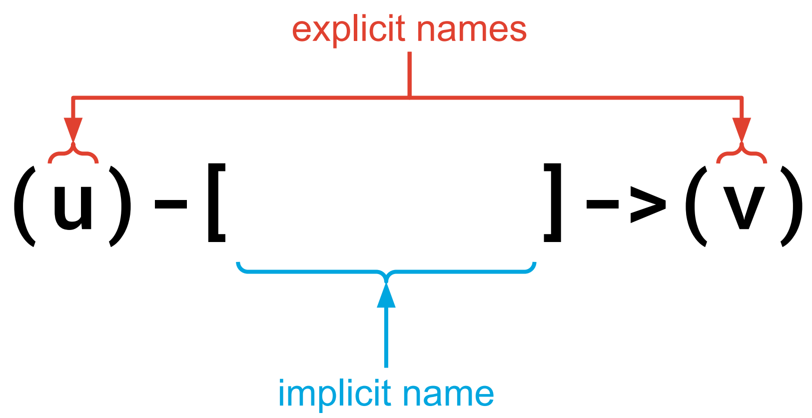



**Path pattern and matching paths**

| u:vertex | v:vertex |
|---|---|
| 0 | 1 |
| 0 | 2 |

**Resulting table**

# GQL pattern normalization

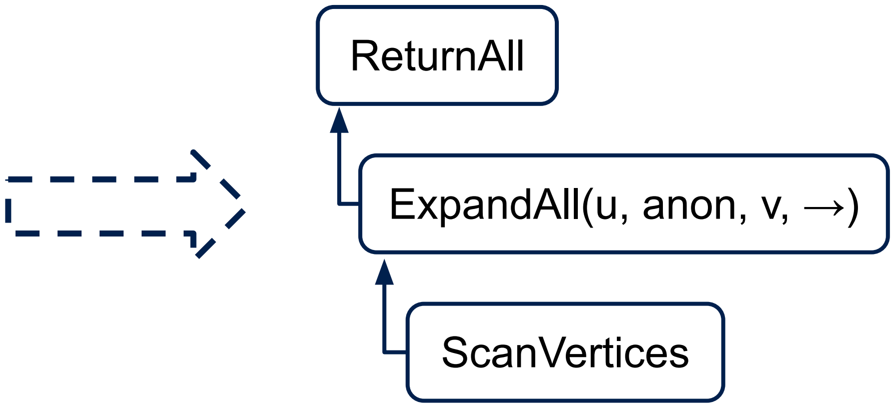



! In the specification all vertex and edge patterns have names but some of them may be marked as implicit

# Execution plans

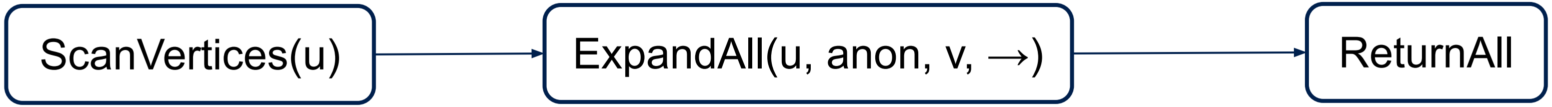
Databases translate queries into execution plans:

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



# Execution plans evaluation

Operations transform an intermediate table to produce the result:

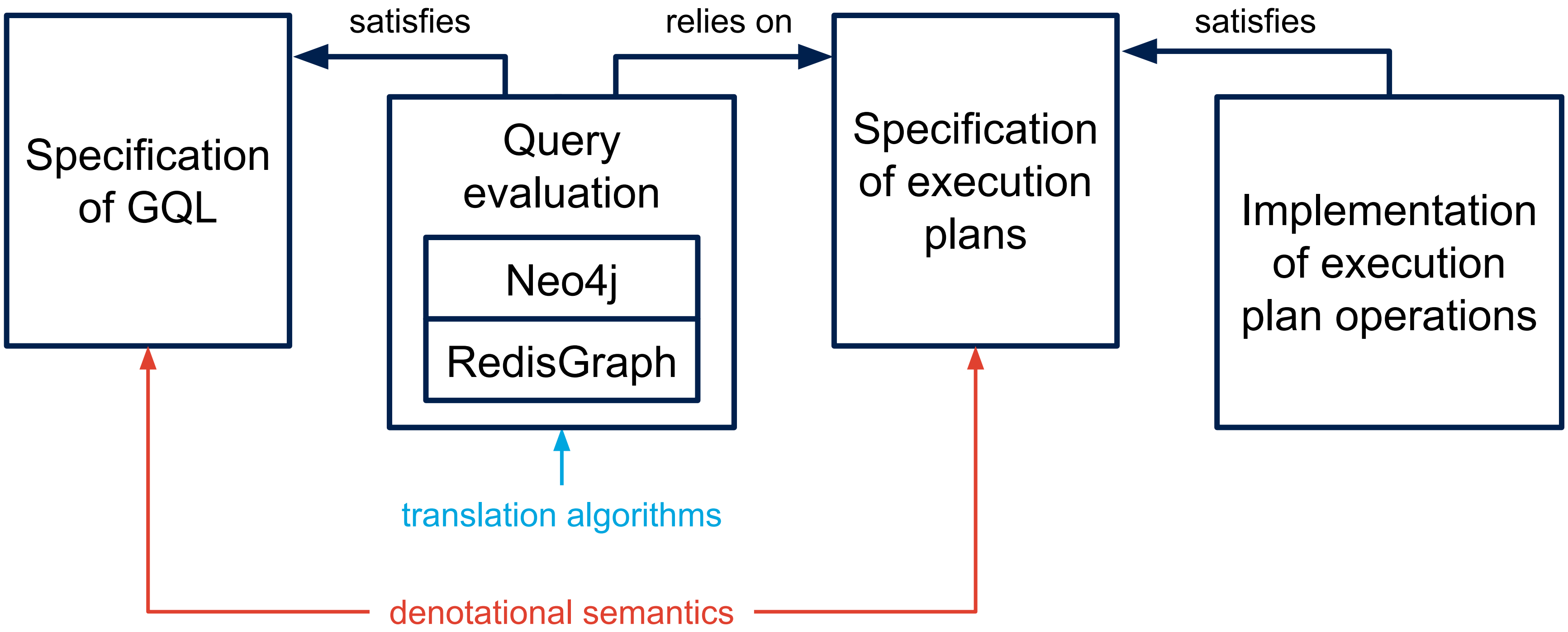



| u |
|---|
| 0 |
| 1 |
| 2 |

| u | anon | v |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 2 |

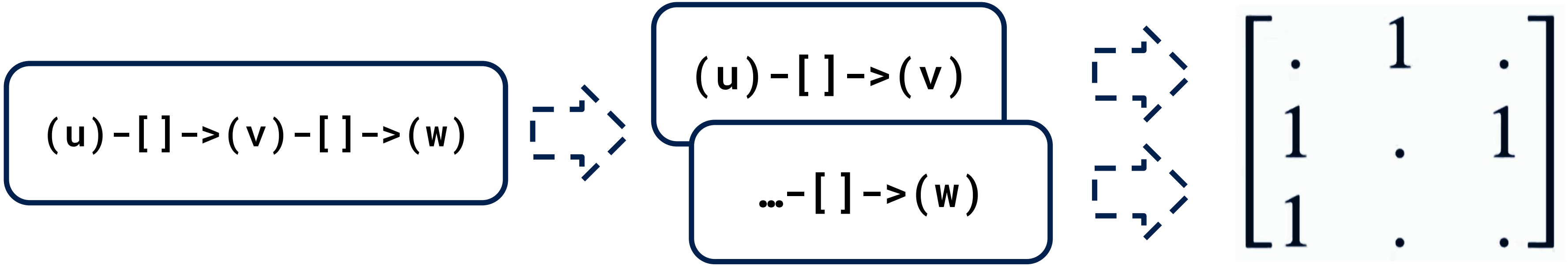
| u | v |
|---|---|
| 0 | 1 |
| 0 | 2 |

# The big picture



# Optimizations

RedisGraph optimizes the query evaluation using linear algebra:




Path pattern

Pattern slices

Matrix expressions

# 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

| u | v |
|---|---|
| 0 | 1 |
| 0 | 2 |

| u | v |
|---|---|
| 0 | 2 |
| 0 | 1 |
| 0 | 2 |

considered to be the same

SO…

```
ORDERED BY,
DISTINCT, count()
```

cannot be formalized

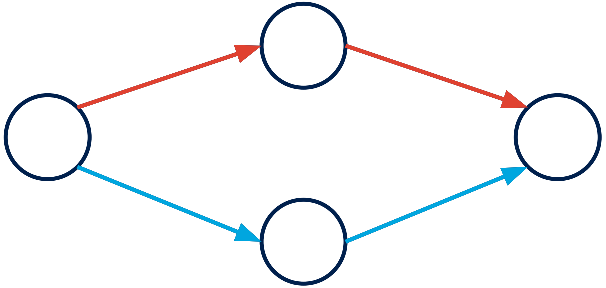
# A former “bug” in RedisGraph

```
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