Model-driven engineering of an openCypher engine: using graph queries to compile graph queries

József Marton, Gábor Szárnyas, Márton Búr

MTA-BME Lendület Research Group on Cyber-Physical Systems and Database Laboratory
Budapest University of Technology and Economics
McGill University

October 10, 2017

Motivation and overview

- Increasing adaptation of graph databases
- Property graph data model well adapted
- No standard language
- Cypher by Neo4j $\rightarrow$ openCypher, 1.5 years
- Open implementations are limited
- Formalisation in relational graph algebra
- Keep compiler simple, add transformations: MDE
- Goal: incremental graph query engine

Running example and the property graph data model

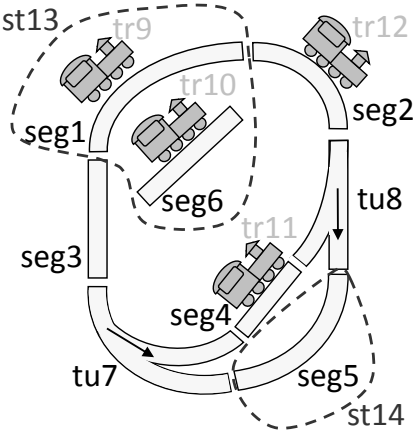


Figure: MoDeS3 example graphical syntax.

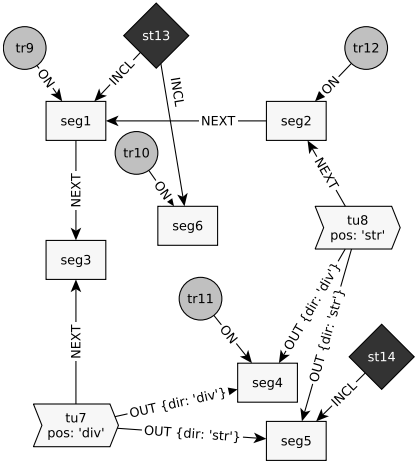
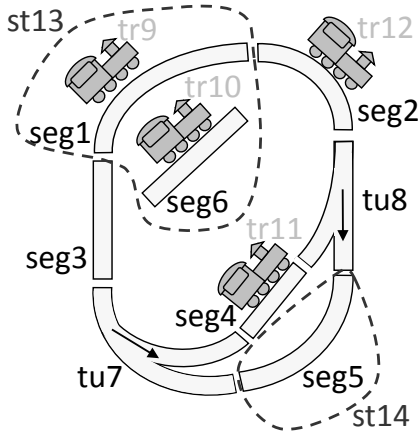


Figure: MoDeS3 example graph.

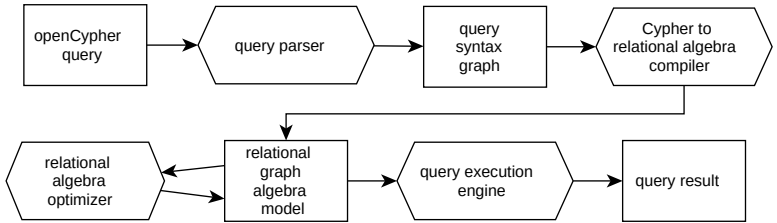
Monitoring objectives



- Close proximity
- Station w/ free track
- Busy station

Figure: MoDeS3 example graphical syntax.

Overview of our approach



Utilizing MDE techniques

- Xtext
- specific DSLs: Eclipse EMF, Ecore
- Model transformations: VIATRA

openCypher

- Cypher: declarative graph query language
- syntax resembles an actual graph → fast learning curve
- openCypher
  - open specification for cypher
  - long term goal: open standard
  - initiative by Neo4j

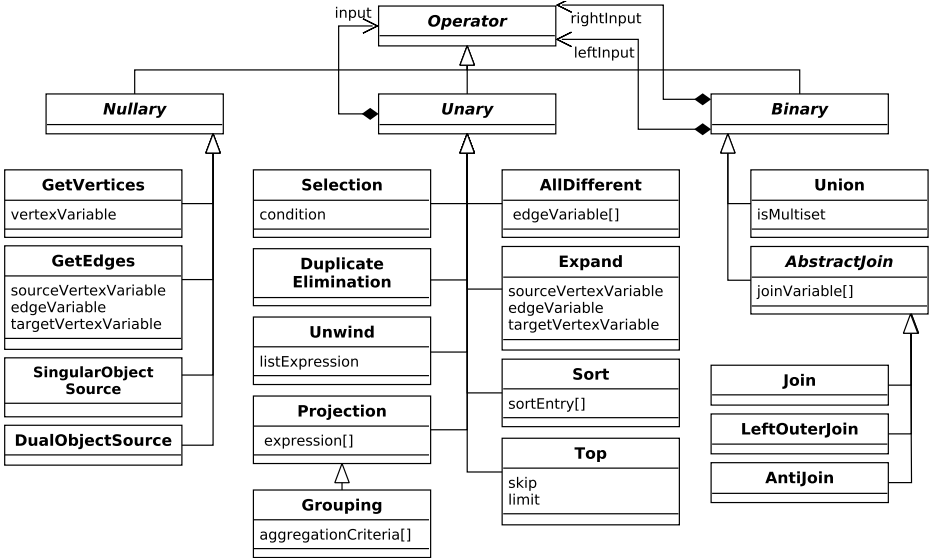
```
1 MATCH (tr:Train)-[:ON]->(seg:Segment)
2 RETURN tr, seg
```

- result: graph relation

Relational graph algebra

- Relational algebra
  - π, σ, $\bowtie$, $\rtimes$, $\triangleright$
  - γ, δ
  - λ, τ, ...
- Nullary operator instead of base relations
  - $\bigcirc_{(\texttt{tr}:\mathsf{Train})}$
- Extended w/ graph-specific operations
  - $\bigcirc_{(\texttt{tr}:\mathsf{Train})}$
  - $\uparrow^{(\texttt{seg}:\mathsf{Segment})}_{(\texttt{tr})}[\_\texttt{e1} : \mathsf{ON}]$
  - $\not\equiv_{\texttt{tr},\texttt{seg}}$
- Reference
  - Hölsch J., Grossniklaus M. (2016) An Algebra and Equivalences to Transform Graph Patterns in Neo4j. EDBT/ICDT 2016
  - Marton J., Szárnyas G., Varró D. (2017) Formalising openCypher Graph Queries in Relational Algebra. ADBIS 2017

Operator metamodel of relational graph algebra



Compilation of a Multipart Query

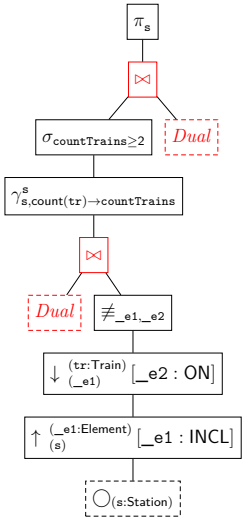
```
1 // identifies stations with at least two trains residing on its
2 // corresponding tracks
3 MATCH (s:Station)-[:INCL]->(:Element)<-[:ON]-(tr:Train)
4 WITH s, count(tr) AS countTrains
5 WHERE countTrains >= 2
6 RETURN s
```

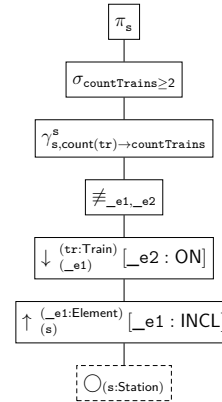
⟦r⟧ WITH «x1» WHERE «condition»

⟦s⟧ RETURN «x2»

$$\pi_{x2}\left(\left(\sigma_{\text{condition}}\pi_{x1}\left(r\right)\right) \bowtie s\right)$$

Query plans for Busy station





Transformation for removing unnecessary joins I.

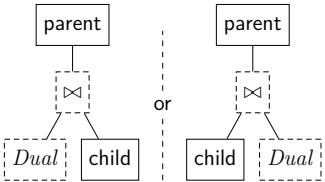


Figure: Left-hand side.

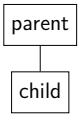


Figure: Right-hand side.

```
1  def changeChildOperator(Operator parentOp, Operator currentOp, Operator
       newOp) {
2    switch parentOp {
3      UnaryOperator:
4        parentOp.input = newOp
5      BinaryOperator: {
6        if (parentOp.getLeftInput.equals(currentOp))
7          parentOp.leftInput = newOp
8        if (parentOp.getRightInput.equals(currentOp))
9          parentOp.rightInput = newOp
10     }
11   }
12 }
```

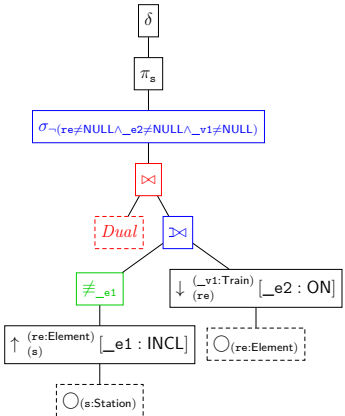
Transformation for removing unnecessary joins II.

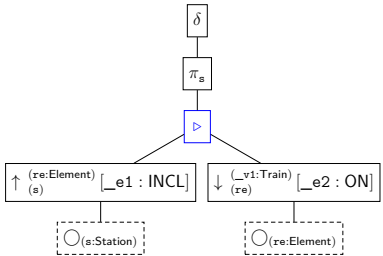
```
pattern unnecessaryJoin(childOp: Operator, joinOp: JoinOperator, parentOp:
    Operator) {
  find parentOperator(joinOp, parentOp);
  DualObjectSourceOperator(dualOp);
  JoinOperator.leftInput(joinOp, dualOp);
  JoinOperator.rightInput(joinOp, childOp);
} or {
  find parentOperator(joinOp, parentOp);
  DualObjectSourceOperator(dualOp);
  JoinOperator.leftInput(joinOp, childOp);
  JoinOperator.rightInput(joinOp, dualOp);
}
```

```
def removeUnnecessaryJoinOperator() {
  createRule()
    .precondition(UnnecessaryJoinMatcher.querySpecification)
    .action [
      changeChildOperator(parentOp, joinOp, otherInputOp)
    ].build
}
```

Identifying Antijoin Operators: Station with free track

```
1 // monitoring objective finds stations that have at least one free track
2 // available
3 MATCH (s:Station)-[:INCL]->(re:Element)
4 WHERE NOT (re)<-[:ON]-(:Train)
5 RETURN DISTINCT s
```





Summary

- defined an openCypher compilation workflow
- mapping of openCypher core by mapping to relational graph algebra
- enhanced compilation w/ transformation rules

Future work

- add transformation rules for expressions
- extend coverage for openCypher: CUD, list and map operations
- follow openCypher development
- formal specification of incremental operators
- create openCypher compatible distributed graph query engine