Schema validation and evolution for PGs

Eugenia Oshurko (ENS Lyon)
7 March 2019
Main ideas

- Schema as a PG
- Schema validation via graph homomorphisms
  - Schema nodes define node types
  - Schema relations define relations allowed between types
  - Properties on schema elements define sets of allowed properties
- Schema/instance homomorphism allows to use existing tools from the graph transformation community
Update semantics

- Sesqui-pushout (SqPO) rewriting approach
- Rewriting given a rule and a matching of its left-hand side
- Two phases
  - **restrictive** update: delete node/edge/property + clone node
  - **expansive** update: add node/edge/property + merge nodes

\[
\begin{align*}
L & \leftarrow P \rightarrow R \\
G & \leftarrow G^- \rightarrow G^+
\end{align*}
\]
Schema/instance coupled updates

Prescriptive updates

- Deletion of schema elements can be propagated to data
- We can clone schema nodes (split concepts) \( \rightsquigarrow \) clone of data nodes

Descriptive updates

- Creation of data elements can be propagated to schema
- We can merge nodes of different types \( \rightsquigarrow \) merge of schema concepts

Result: instance respects schema at all times
Prescriptive updates: example
Prescriptive updates: example

- **Message**
  - `imageFile`: STRING
  - `type`: {post, comment}

- **Post**
  - `imageFile`: STRING
  - `type`: post

- **Comment**
  - `type`: comment

- **Person**
  - `firstName`: STRING
  - `lastName`: STRING
  - `gender`: STRING
  - `birthday`: DATE

- **Post**
  - `imageFile`: STRING?
  - `creationDate`: STRING
  - `browserUsed`: STRING

- **Comment**
  - `creationDate`: STRING
  - `browserUsed`: STRING

- **LIKES**
- **KNOWS**
- **HAS CREATOR**
- **REPLY**
- **OF**
Prescriptive updates: example

\[
L \xrightarrow{m} P \xrightarrow{h} G
\]

\[
\text{imageFile: STRING} \quad \text{type: \{post, comment\}}
\]

\[
\text{creationDate: 2010-10-16} \quad \text{browserUsed: Firefox}
\]

\[
\text{firstName: Bryn} \quad \text{lastName: Davies}
\]

\[
\text{imageFile: photo33711.jpg}
\]

\[
\text{firstName: Jose} \quad \text{lastName: Alonso}
\]

\[
\text{creationDate: 2010-10-30} \quad \text{browserUsed: Safari}
\]

\[
\text{firstName: Jane} \quad \text{lastName: Murray}
\]

\[
\text{creationDate: 2010-10-30} \quad \text{browserUsed: Safari}
\]

\[
\text{firstName: Jose} \quad \text{lastName: Alonso}
\]

\[
\text{creationDate: 2010-10-16} \quad \text{browserUsed: Firefox}
\]

\[
\text{firstName: Jane} \quad \text{lastName: Murray}
\]

\[
\text{creationDate: 2010-10-30} \quad \text{browserUsed: Safari}
\]

\[
\text{firstName: Bryn} \quad \text{lastName: Davies}
\]

\[
\text{creationDate: 2010-10-30} \quad \text{browserUsed: \{Firefox, Safari\}}
\]

\[
\text{LIKES} \quad \text{KNOWS} \quad \text{HAS CREATOR} \quad \text{REPLY OF}
\]

\[
\text{LIKES} \quad \text{LIKES} \quad \text{HAS CREATOR} \quad \text{REPLY OF}
\]
Descriptive updates: example
Descriptive updates: example
Prototype system implemented in the regraph Python library (based on Neo4j PGs, see on GitHub https://github.com/Kappa-Dev/ReGraph)

- We convert rules + matchings to openCypher queries
- regraph generalizes schema/data relation to hierarchies of graphs.
Remarks

Couple of technical issues:

- no native support for node cloning and node merge (we wrote huge unreadable queries);
- no way to set properties by dynamic keys (we use APOC).

Pre-print available:
Angela Bonifati (UCBL1), Peter Furniss (Neo4j), Alastair Green (Neo4j), Russ Harmer (CNRS), Eugenia Oshurko (ENSL), Hannes Voigt (Neo4j), *Schema Validation and Evolution for Graph Databases*, https://arxiv.org/abs/1902.06427