Schema Mappings for Data Graphs

Nadime Francis[1] Leonid Libkin[2]

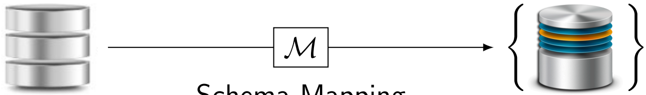
[1]Université Paris-Est Marne-la-Vallée
[2]University of Edinburgh

BDA 2017
Nancy, November, 16th

Results originally published in PODS 2017

Introduction

Data Exchange

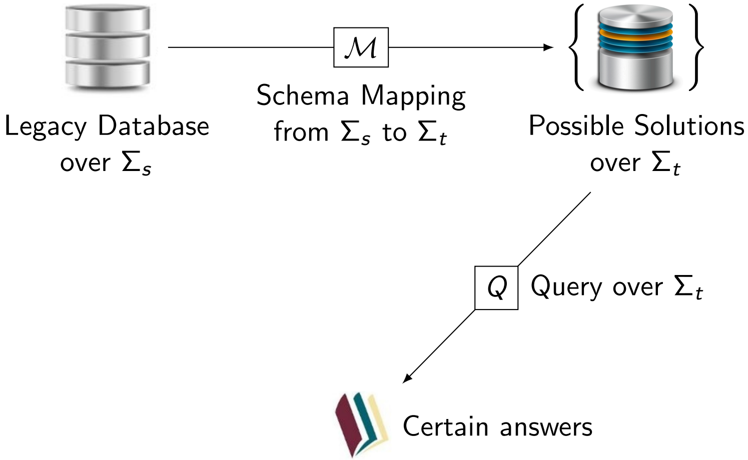


Legacy Database over Σ_s

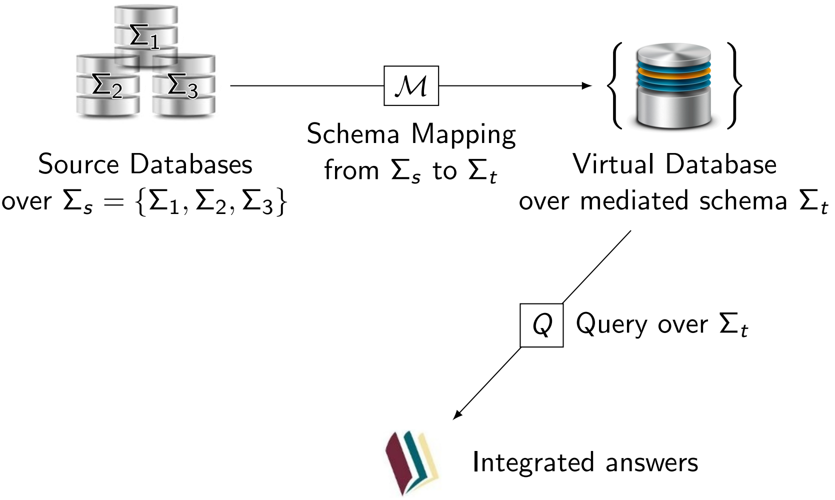
Schema Mapping from Σ_s to Σ_t

Possible Solutions over Σ_t

Data Exchange



Data Exchange as Virtual Data Integration



Key questions

Solution Existence

- Input: A database S and a mapping $\mathcal{M}$
- Question: Does there exist a solution T to S under $\mathcal{M}$?

Key questions

Solution Existence

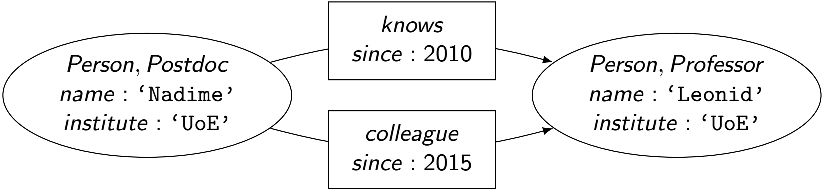
- Input: A database S and a mapping $\mathcal{M}$
- Question: Does there exist a solution T to S under $\mathcal{M}$?

Query Answering

- Input: A database S, a tuple $\bar{v} \in S$, a mapping $\mathcal{M}$, a query Q
- Question: Is $\bar{v}$ a "certain answer" to Q over the solution to S under $\mathcal{M}$?

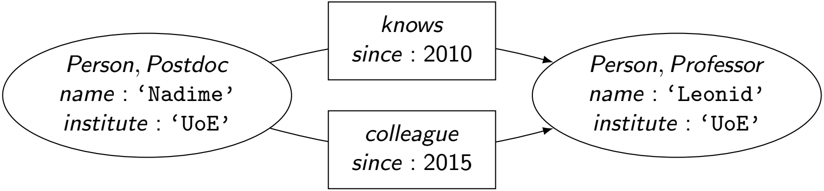
Why look at graph data?

- Data naturally presented as graphs:
  $\rightarrow$ Social networks, Internet, biological data, Semantic Web...
- Queries care about the topology of the graph:
  $\rightarrow$ Paths, cycles, connected components, graph patterns...
- Known techniques for relational databases do not apply



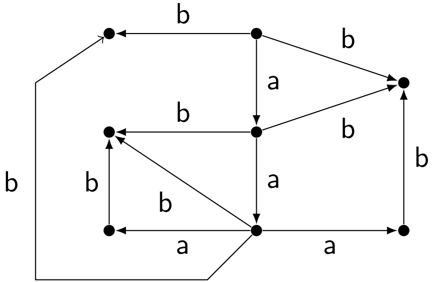
Property Graphs, Graph Databases and Data Graphs

- **Property graphs**: real life model, used in Neo4j (Cypher), advocated by LDBC.
- Graph databases: theoretical modelization as edge-labelled graph. Topology **only**. Does not capture real life scenarios.
- Data graphs: edge-labelled graphs with nodes carrying values from an infinite domain. Better abstraction of real life cases. Topology **and** data.



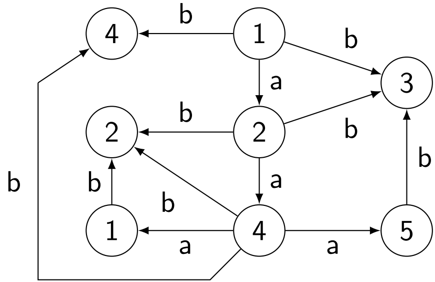
Property Graphs, Graph Databases and Data Graphs

- Property graphs: real life model, used in Neo4j (Cypher), advocated by LDBC.
- **Graph databases**: theoretical modelization as edge-labelled graph. Topology **only**. Does not capture real life scenarios.
- Data graphs: edge-labelled graphs with nodes carrying values from an infinite domain. Better abstraction of real life cases. Topology **and** data.



Property Graphs, Graph Databases and Data Graphs

- Property graphs: real life model, used in Neo4j (Cypher), advocated by LDBC.
- Graph databases: theoretical modelization as edge-labelled graph. Topology **only**. Does not capture real life scenarios.
- **Data graphs**: edge-labelled graphs with nodes carrying values from an infinite domain. Better abstraction of real life cases. Topology **and** data.



Data Exchange and Query Answering

Known Results

Data Exchange for Relational Databases

- Extensively investigated over the last 20 years
- “Book material”:
  - Foundations of Data Exchange [Arenas, Barceló, Libkin, Murlak, 2014]
  - Principles of Data Integration [Doan, Halevy, Ives, 2012]
- When and how solutions can be efficiently built and queried

Known Results

Data Exchange for Relational Databases

- Extensively investigated over the last 20 years
- "Book material":
  - Foundations of Data Exchange [Arenas, Barceló, Libkin, Murlak, 2014]
  - Principles of Data Integration [Doan, Halevy, Ives, 2012]
- When and how solutions can be efficiently built and queried

Data Exchange for Graph Data

- Schema Mappings and Data Exchange for Graph Databases [Barceló, Pérez, Reutter, 2013]
- Answering queries is typically coNP, tractability requires RPQs in mappings to be "rigid" (no $*$, no $\vee$)

Known Results

Data Exchange for Relational Databases

- Extensively investigated over the last 20 years
- "Book material":
  - Foundations of Data Exchange [Arenas, Barceló, Libkin, Murlak, 2014]
  - Principles of Data Integration [Doan, Halevy, Ives, 2012]
- When and how solutions can be efficiently built and queried

Data Exchange for Graph Data

- Schema Mappings and Data Exchange for Graph Databases [Barceló, Pérez, Reutter, 2013]
- Answering queries is typically coNP, tractability requires RPQs in mappings to be "rigid" (no $*$, no $\vee$)

$\rightarrow$ No literature for data graphs!

Chosen Setting

Key parameters

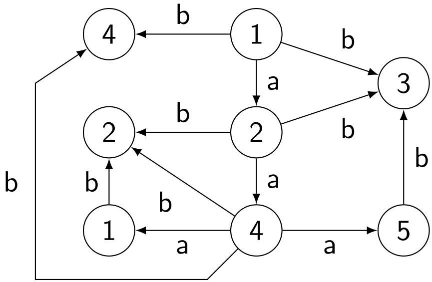
- Source (G_s) and target (G_t) databases: Data graphs
- Schema mapping ($\mathcal{M}$): Pairs of RPQs
- Query (Q): Data RPQs

Chosen Setting

Key parameters

- Source (G_s) and target (G_t) databases: **Data graphs**
- Schema mapping ($\mathcal{M}$): Pairs of RPQs
- Query (Q): Data RPQs

- Source: data graph over Σ_s and $\mathcal{D}$
- Target: data graph over Σ_t and $\mathcal{D}$



Chosen Setting

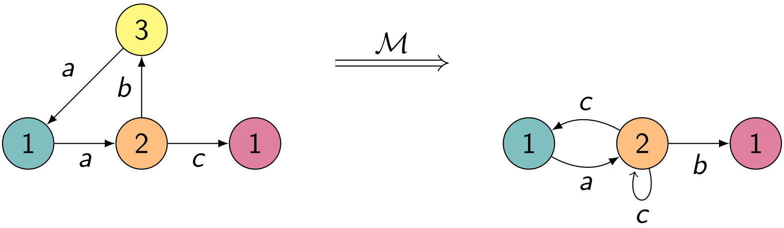
Key parameters

- Source (G_s) and target (G_t) databases: Data graphs
- Schema mapping ($\mathcal{M}$): **Pairs of RPQs**
- Query (Q): Data RPQs

$\mathcal{M} = \{(q_i, q_i')\}$ $\quad (G_s, G_t) \models \mathcal{M}$ iff $q_i(G_s) \subseteq q_i'(G_t)$

Note: nodes of G_s are exported together with their data values

Ex: $\mathcal{M} = \{(ba^+, c); (ac, ab^*)\}$



Chosen Setting

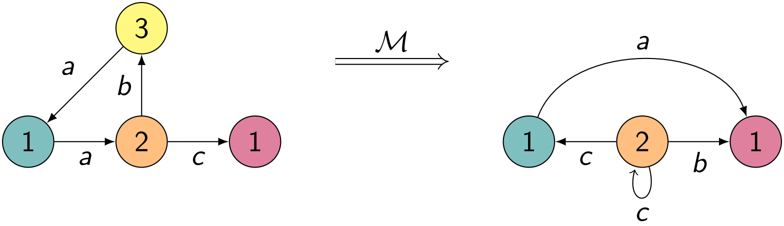
Key parameters

- Source (G_s) and target (G_t) databases: Data graphs
- Schema mapping ($\mathcal{M}$): **Pairs of RPQs**
- Query (Q): Data RPQs

$\mathcal{M} = \{(q_i, q_i')\}$ $\quad (G_s, G_t) \models \mathcal{M}$ iff $q_i(G_s) \subseteq q_i'(G_t)$

Note: nodes of G_s are exported together with their data values

Ex: $\mathcal{M} = \{(ba^+, c); (ac, ab^*)\}$



Chosen Setting

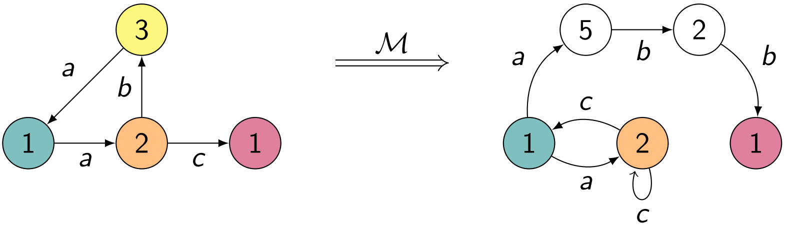
Key parameters

- Source (G_s) and target (G_t) databases: Data graphs
- Schema mapping ($\mathcal{M}$): **Pairs of RPQs**
- Query (Q): Data RPQs

$\mathcal{M} = \{(q_i, q_i')\}$ $\quad (G_s, G_t) \models \mathcal{M}$ iff $q_i(G_s) \subseteq q_i'(G_t)$

Note: nodes of G_s are exported together with their data values

Ex: $\mathcal{M} = \{(ba^+, c); (ac, ab^*)\}$



Chosen Setting

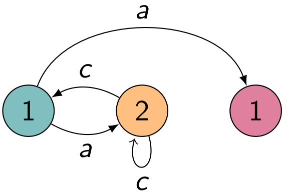
Key parameters

- Source (G_s) and target (G_t) databases: Data graphs
- Schema mapping ($\mathcal{M}$): Pairs of RPQs
- Query (Q): **Data RPQs**

Most general: regular expressions with memory

Ex: $Q = \Sigma_t^* \cdot \downarrow x.(\Sigma_t^+[x^=]) \cdot (\Sigma_t^+[x^=]) \cdot \Sigma_t^* \cdot (a \mid b)$
A data value repeats three times along a path finishing with a or b.

$Q(G_t) = \{(\bullet, \bullet); (\bullet, \bullet); (\bullet, \bullet); (\bullet, \bullet)\}$



Query Answering: Certain Answers Semantics

Certain Answers

Let G_s be a source database, $\mathcal{M}$ be a schema mapping from Σ_s to Σ_t, and Q be a query over Σ_t. Then:

$$\Box_{\mathcal{M}}(Q, G_s) = \bigcap_{G_t \mid (G_s, G_t) \models \mathcal{M}} Q(G_t)$$

Query Answering: Certain Answers Semantics

Certain Answers

Let G_s be a source database, $\mathcal{M}$ be a schema mapping from Σ_s to Σ_t, and Q be a query over Σ_t. Then:

$$\Box_{\mathcal{M}}(Q, G_s) = \bigcap_{G_t \ \mid \ (G_s, G_t) \models \mathcal{M}} Q(G_t)$$

PROBLEM	:	QUERY ANSWERING
INPUT	:	A data graph G_s, a tuple $\bar{v}$ of nodes of G_s a schema mapping $\mathcal{M}$ and a query Q
QUESTION	:	Is $\bar{v}$ in $\Box_{\mathcal{M}}(Q, G_s)$?

Query Answering: Certain Answers Semantics

Certain Answers

Let G_s be a source database, $\mathcal{M}$ be a schema mapping from Σ_s to Σ_t, and Q be a query over Σ_t. Then:

$$\Box_{\mathcal{M}}(Q, G_s) = \bigcap_{G_t \mid (G_s, G_t) \models \mathcal{M}} Q(G_t)$$

PROBLEM	:	QUERY ANSWERING$(\mathcal{M}, Q)$ Data complexity
INPUT	:	A data graph G_s, a tuple $\bar{v}$ of nodes of G_s
		~~a schema mapping $\mathcal{M}$ and a query Q~~
QUESTION	:	Is $\bar{v}$ in $\Box_{\mathcal{M}}(Q, G_s)$?

Contributions

Undecidability and Intractability

- Query answering is undecidable for RPQ mappings and data RPQ queries. (Already true for very simple mappings)
- Query answering is coNP-complete for word mappings and data RPQ queries. (Already true for paths with tests)

Recovering Tractability

- Query answering is in NLogSpace for word mappings and data RPQ queries under the presence of SQL nulls.
- Query answering is in NLogSpace for word mappings and data RPQ queries with equality only.

UNDECIDABILITY AND INTRACTABILITY

Undecidability

Theorem

There exist a very simple mapping $\mathcal{M}$ and an RPQ with equality Q such that $\textsc{Query Answering}(\mathcal{M}, Q)$ is undecidable.

Undecidability

Theorem

There exist a very simple mapping $\mathcal{M}$ and an RPQ with equality Q such that $\textsc{Query Answering}(\mathcal{M}, Q)$ is undecidable.

Very simple:
Rules of $\mathcal{M}$ are of two possible forms:

- Either (a, b), with $a \in \Sigma_s$ and $b \in \Sigma_t$;
- Or (a, Σ_t^*), with $a \in \Sigma_s$.

Undecidability

Theorem

There exist a very simple mapping $\mathcal{M}$ and an RPQ with equality Q such that $\textsc{Query Answering}(\mathcal{M}, Q)$ is undecidable.

RPQ with equality:

- Query defined by an expression with $e_=$ and $e_{\neq}$ subexpressions
  Ex: $\Sigma^* \cdot (a^+)_= \cdot \Sigma^*$
  A data value occurs twice with only a's in between.
- Strictly weaker than data RPQs.

Undecidability

Theorem

There exist a very simple mapping $\mathcal{M}$ and an RPQ with equality Q such that $\textsc{Query Answering}(\mathcal{M}, Q)$ is undecidable.

For these fixed (data complexity) $\mathcal{M}$ and Q, given:

- a database G_s over Σ_s,
- a pair of nodes (x, y) of G_s,

it is undecidable whether $(x, y) \in \square_{\mathcal{M}}(Q, G_s)$.

Intractability

Theorem

Let $\mathcal{M}$ be a relational mapping and a Q a data RPQ. Then $\textsc{Query Answering}(\mathcal{M}, Q)$ is coNP-complete.

Intractability

Theorem

Let $\mathcal{M}$ be a relational mapping and a Q a data RPQ. Then $\text{Query Answering}(\mathcal{M}, Q)$ is coNP-complete.

Relational mapping:
Rules of $\mathcal{M}$ are of the form (q, w), where:

- q is an arbitrary RPQ over Σ_s;
- w is a single word over Σ_t.

It is already coNP-hard when q is a single symbol $a \in \Sigma_s$.

Intractability

Theorem

Let $\mathcal{M}$ be a relational mapping and a Q a data RPQ. Then $\textsc{Query Answering}(\mathcal{M}, Q)$ is coNP-complete.

Data RPQ:

- Any query defined by a register automaton or a regular expression with memory.
- It is already hard if Q is a path with tests.
  Ex: $((ab)_{=}c)_{\neq}$. No disjunction, no transitive closure.

Intractability

Theorem

Let $\mathcal{M}$ be a relational mapping and a Q a data RPQ. Then $\textsc{Query Answering}(\mathcal{M}, Q)$ is coNP-complete.

For these fixed (data complexity) $\mathcal{M}$ and Q, given:

- a database G_s over Σ_s,
- a pair of nodes (x, y) of G_s,

it is coNP-complete to decide whether $(x, y) \in \Box_{\mathcal{M}}(Q, G_s)$.

Tractable Fragments

Introducing SQL Nulls

SQL Behavior

- SQL: Missing values modelled with a single null value $\mathbf{N}$.
- Three valued logic:
  - $x = y$ returns *unknown* if either $x = \mathbf{N}$ or $y = \mathbf{N}$;
  - Otherwise, $x = y$ evaluates as usual.
- At query answering time, *unknown* collapses to *false*.

Introducing SQL Nulls

SQL Behavior

- SQL: Missing values modelled with a single null value $\mathbf{N}$.
- Three valued logic:
  - $x = y$ returns *unknown* if either $x = \mathbf{N}$ or $y = \mathbf{N}$;
  - Otherwise, $x = y$ evaluates as usual.
- At query answering time, *unknown* collapses to *false*.

Translating to data RPQs

- Replace $\mathcal{D}$ with $\mathcal{D}_{\mathbf{N}} = \mathcal{D} \cup \{\mathbf{N}\}$.
- Comparisons: as in SQL.

Query Answering Approximation via SQL Nulls

Theorem

Let $\mathcal{M}$ be a relational mapping and Q be a data RPQ. Then, given a source database G_s and (x, y) in G_s, it can be decided in NLogSpace whether $(x, y) \in \square^{\mathbf{N}}_{\mathcal{M}}(Q, G_s)$.

Query Answering Approximation via SQL Nulls

Theorem

Let $\mathcal{M}$ be a relational mapping and Q be a data RPQ. Then, given a source database G_s and (x, y) in G_s, it can be decided in NLogSpace whether $(x, y) \in \square^{\mathbf{N}}_{\mathcal{M}}(Q, G_s)$.

$$\square^{\mathbf{N}}_{\mathcal{M}}(Q, G_s) = \bigcap_{\substack{G_t \text{ over } \mathcal{D}_{\mathbf{N}} \\ (G_s, G_t) \models \mathcal{M}}} Q(G_t)$$

Query Answering Approximation via SQL Nulls

Theorem

Let $\mathcal{M}$ be a relational mapping and Q be a data RPQ. Then, given a source database G_s and (x, y) in G_s, it can be decided in NLogSpace whether $(x, y) \in \Box^{\mathbf{N}}_{\mathcal{M}}(Q, G_s)$.

$$\Box^{\mathbf{N}}_{\mathcal{M}}(Q, G_s) = \bigcap_{\substack{G_t \text{ over } \mathcal{D}_{\mathbf{N}} \\ (G_s, G_t) \models \mathcal{M}}} Q(G_t)$$

Motivations

- $\Box^{\mathbf{N}}_{\mathcal{M}}(Q, G_s)$ is an underapproximation of $\Box_{\mathcal{M}}(Q, G_s)$.
- It matches the implementation of traditional DBMSs, and thus the expected behavior if G_t is materialized as such.

Queries without Inequalities

Data RPQs without Inequalities

- Expressions with memory: only $[x^{=}]$.
- Expressions with equality: only $e_{=}$.

Queries without Inequalities

Data RPQs without Inequalities

- Expressions with memory: only $[x^=]$.
- Expressions with equality: only $e_=$.

Results

- $\textsc{Query Answering}(\mathcal{M}, Q)$ is in NLogSpace for relational mappings and data RPQs without inequalities.
- $\textsc{Query Answering}(\mathcal{M}, Q)$ is in coNP for RPQ mappings and RPQ with equalities and no inequalities.
- The question remains open when Q is an RPQ with memory and no inequalities.

PERSPECTIVES

- Very first steps in Data Exchange for Data Graphs
  - Early picture of decidability / tractability
  - Other tasks (metadata management)
- Application to real life graph data
  - Property graphs and real-life query languages
  - Neo Technology: Neo4j and Cypher
- Back to relational data exchange
  - Use of SQL nulls instead of marked nulls
  - Get efficient approximations

Thank you!