Efficient reasoning in heterogeneous data integration systems

Maxime Buron

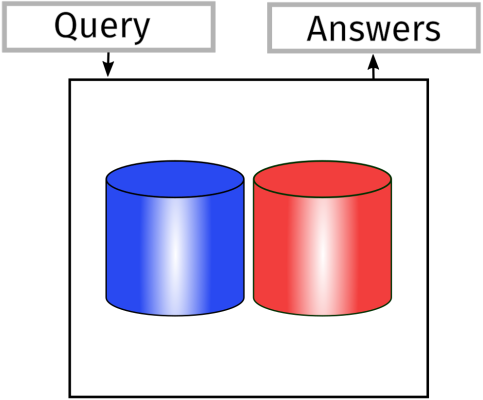
LIMOS

July 7, 2022

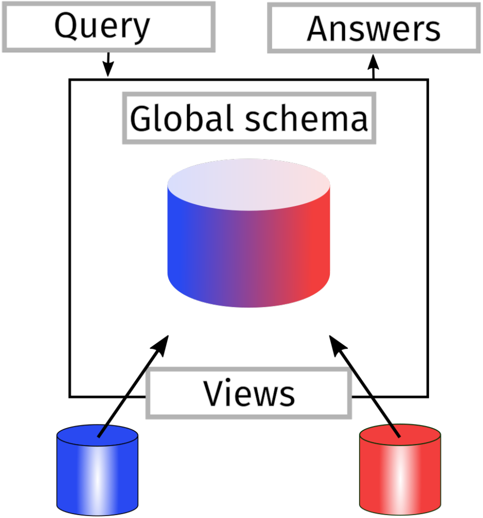
Part I

Introduction

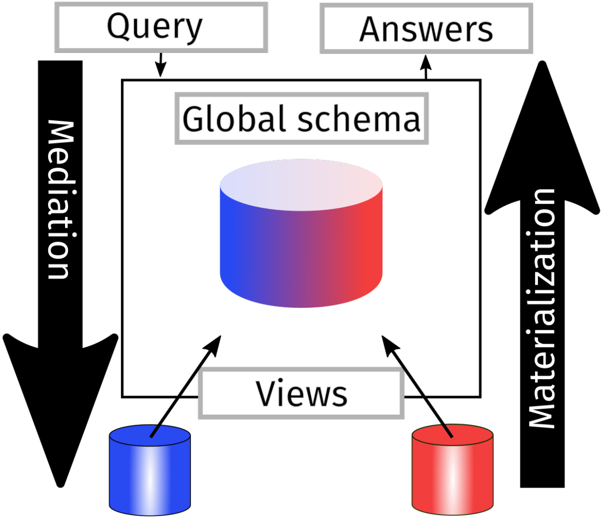
Heterogeneous data integration



Integration system



Materialization- and mediation-based integration



Mediation with semantics a.k.a. Ontology-Based Data Access

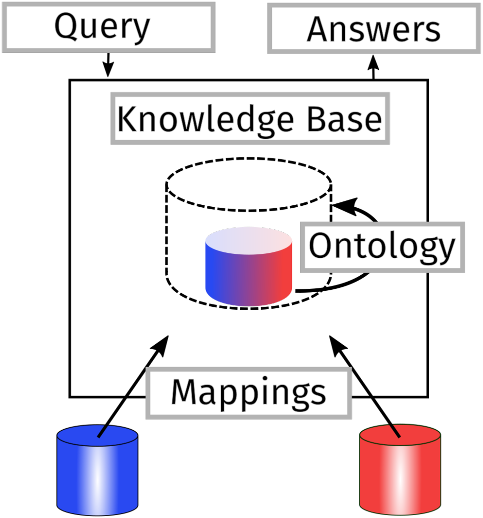


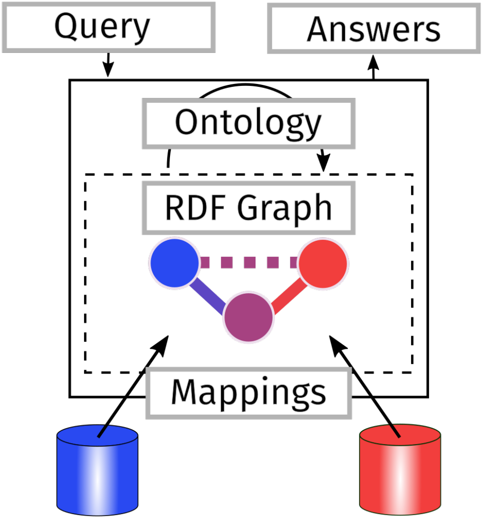
Table of Contents

Part II

RDF integration systems

thesis work supervised by François Goasdoué, Ioana Manolescu and Marie-Laure Mugnier

RDF integration systems



Preliminaries: querying in RDF graphs

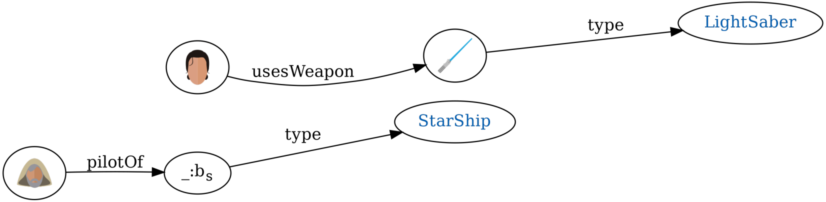
RDF triple

An RDF triple contains three values among:

- IRIs
- blank nodes
- literals

RDF graph: data and RDFS ontology

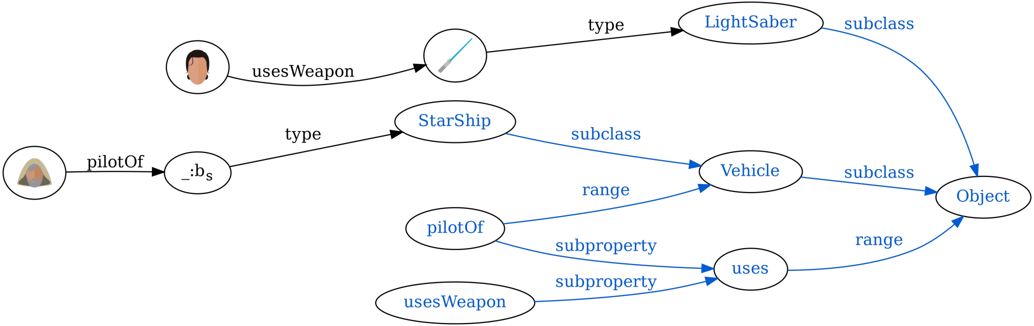
Data triples of an RDF graph G:



RDF graph: data and RDFS ontology

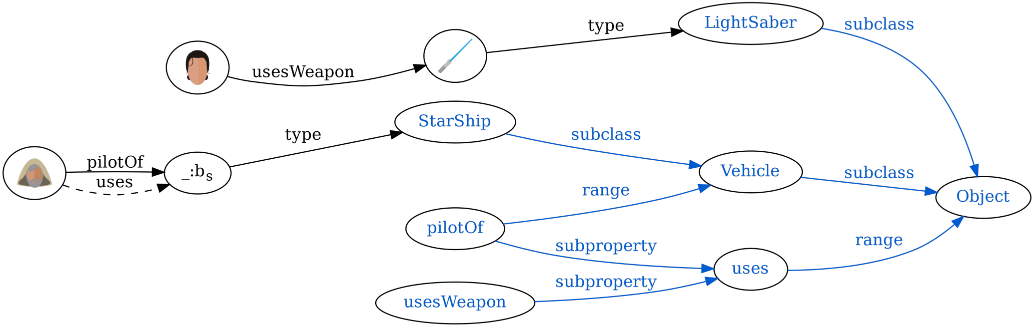
The RDFS triples use the built-in properties:

- :subclass
- :subproperty
- :domain
- :range



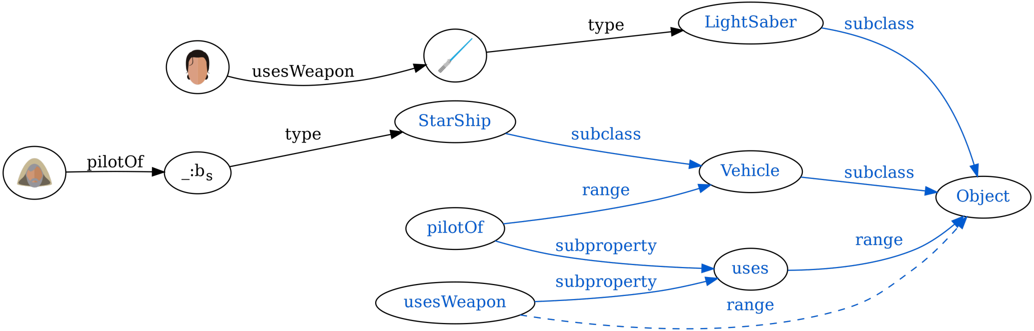
Data entailment using $\mathcal{R}_{\mathrm{data}}$

$$\mathcal{R}_{\mathrm{data}} = \{ \quad (\mathrm{p}_1, \mathrm{:subproperty}, \mathrm{p}_2), (\mathrm{s}, \mathrm{p}_1, \mathrm{o}) \rightarrow (\mathrm{s}, \mathrm{p}_2, \mathrm{o}) \quad \ldots \quad \}$$



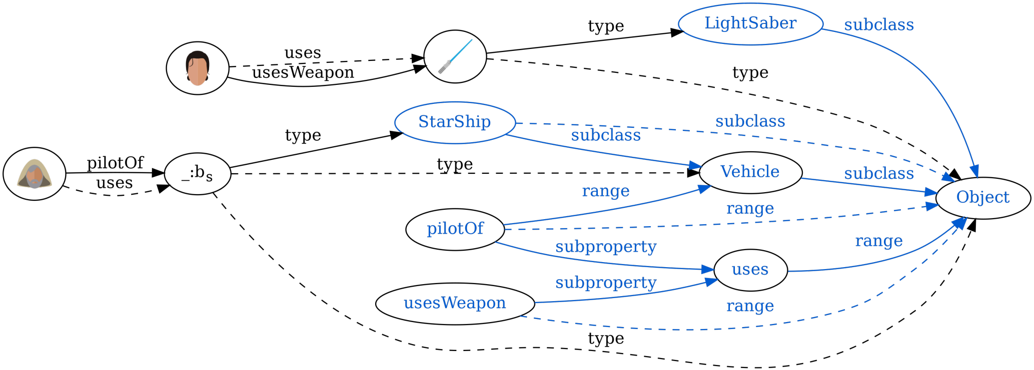
Ontological entailment using $\mathcal{R}_{\text{onto}}$

$$\mathcal{R}_{\text{onto}} = \{ \quad (\mathsf{p}, \text{:subproperty}, \mathsf{p}_1), (\mathsf{p}_1, \text{:range}, \mathsf{o}) \rightarrow (\mathsf{p}, \text{:range}, \mathsf{o}) \quad \ldots \quad \}$$



Full saturation of the graph w.r.t. $\mathcal{R}_{\mathrm{data}}$ and $\mathcal{R}_{\mathrm{onto}}$

The full saturation of G is $G^{\mathcal{R}_{\mathsf{onto}} \cup \mathcal{R}_{\mathsf{data}}}$:



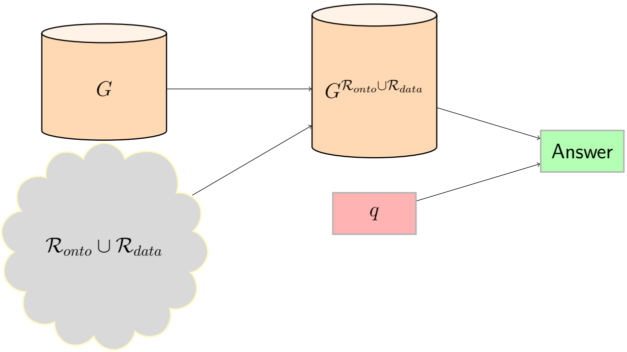
Basic Graph Pattern Queries

We consider conjunctive queries over the data and the ontology.

For instance: "Who is using what kind of object?"

$$q(x, y) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, y), (y, \text{:subclass}, \text{:Object})$$

The saturation-based query answering technique



Pros:

- Efficient: no reasoning at query time

Cons:

- The saturation requires time to be computed and extra-space to be materialized
- The saturation needs to be recomputed on updates $\rightarrow$ Saturation maintenance is needed

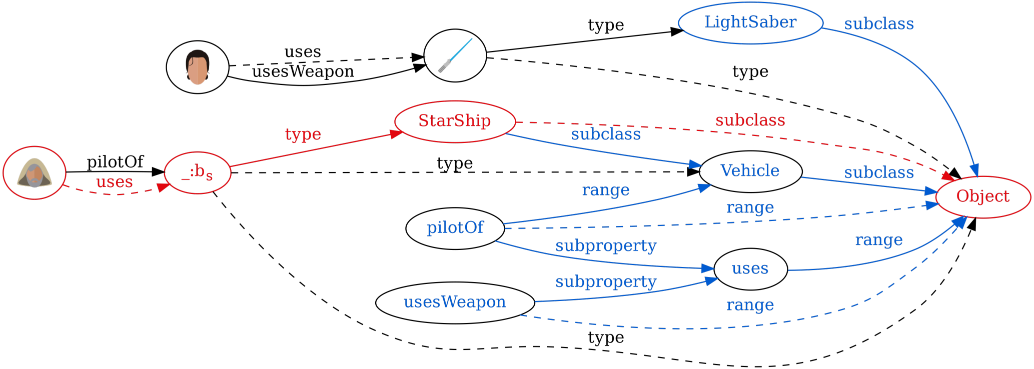
Saturation-Based Query Answering

$$q(x, y) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, y), \color{blue}{(y, \text{:subclass}, \text{:Object})}$$

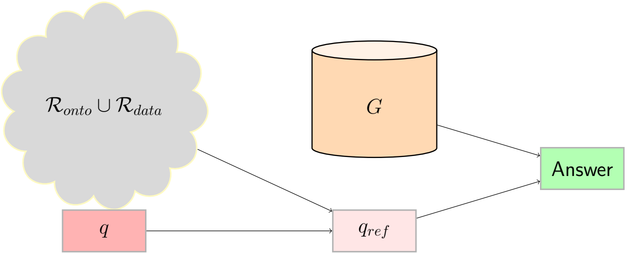
$q(G^{\mathcal{R}_{\text{data}} \cup \mathcal{R}_{\text{onto}}}) =$ $\{\langle$  $, \text{:LightSaber}\rangle$

$\langle$  $, \text{:Vehicle}\rangle$

$\color{red}{\langle}$  $\color{red}{, \text{:StarShip}\rangle}\}$



The reformulation-based query answering technique



Pros:

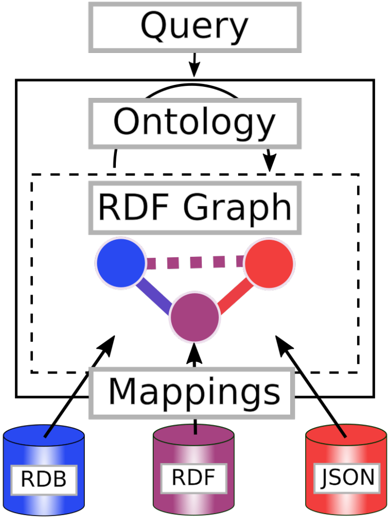
- data is always up-to-date (no need to compute and store the saturation)

Cons:

- Every incoming query needs to be reformulated (low overhead in practice)
- Reformulated queries may be complex, hence costly to evaluate, even by modern, highly optimized query engines $\rightarrow$ Query optimization is needed

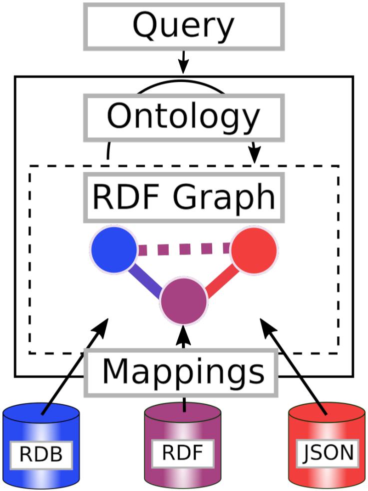
Query answering for RDF Integration Systems

Ontology-Based Data Access

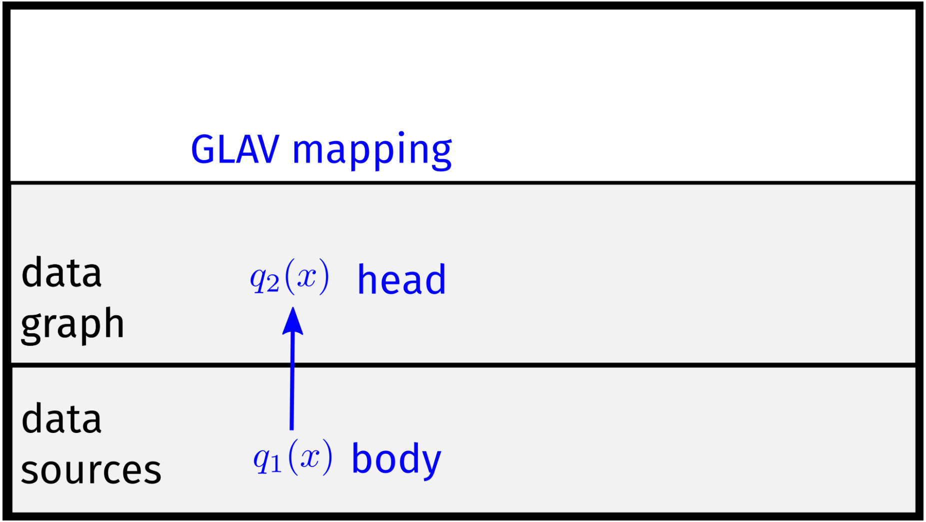


Contributions

1. More powerful integration setting:
   - Global-Local-As-View mappings in an OBDA context
   - Queries on the data and the ontology
2. A novel query answering strategy: shifting a part of the reasoning from query time to offline
3. Obi-Wan, a system implementing several query answering strategies



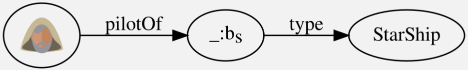
Global-Local-As-View mapping



Global-Local-As-View mapping example

data graph

$(\text{name}, \text{:pilotOf}, z),$
$(z, \text{:type}, \text{:StarShip})$

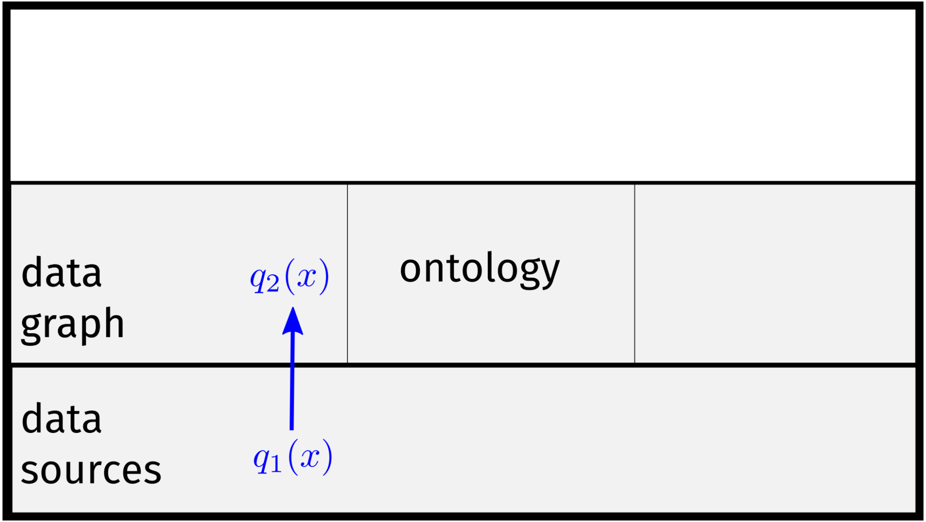


data sources

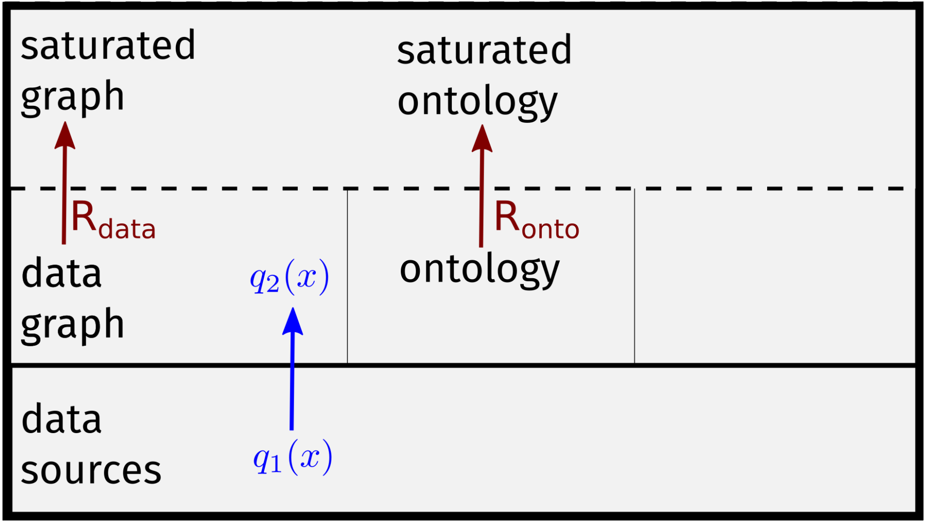
$q_{\text{SQL}}(\text{name})$

name	vehicle type
Luke	starship

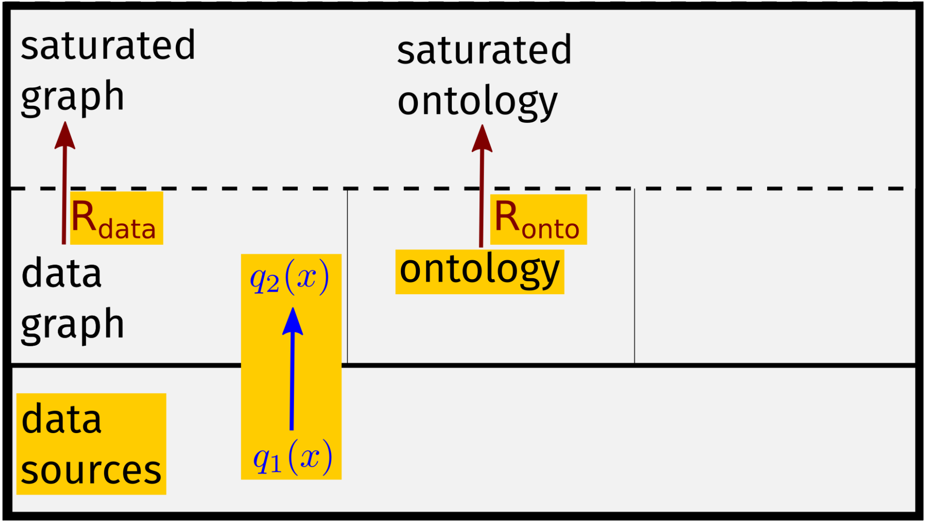
RDFS ontology



RDFS reasoning in the integrated graph



RDF Integration System



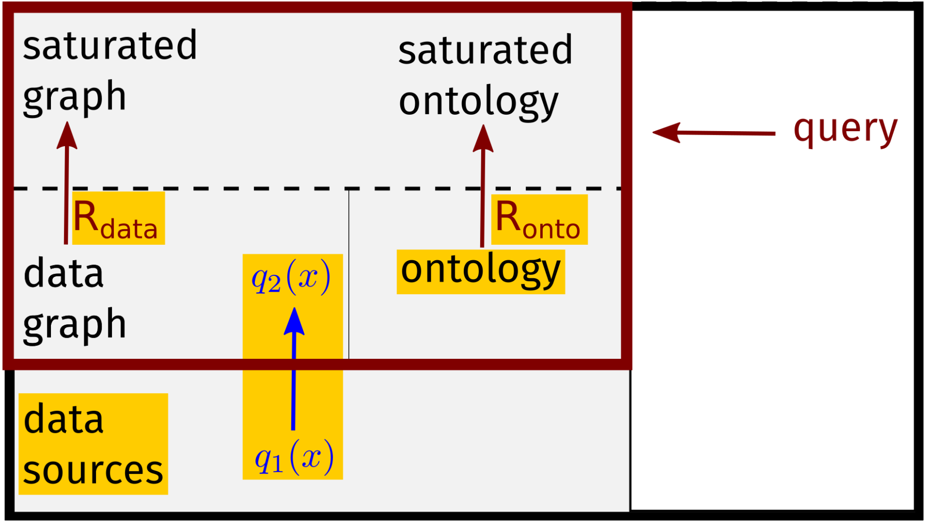
Obi-Wan: a RDF Integration System implementation

Features

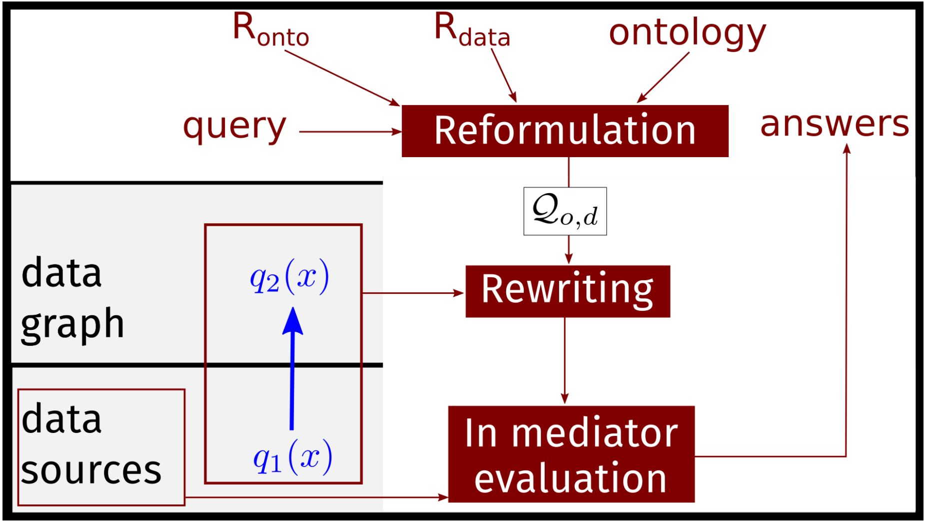
- supports GLAV mappings
- supports heterogeneous data sources: PostgreSQL, MongoDB, Jena TDB
- provides a RIS visualization

Demonstration

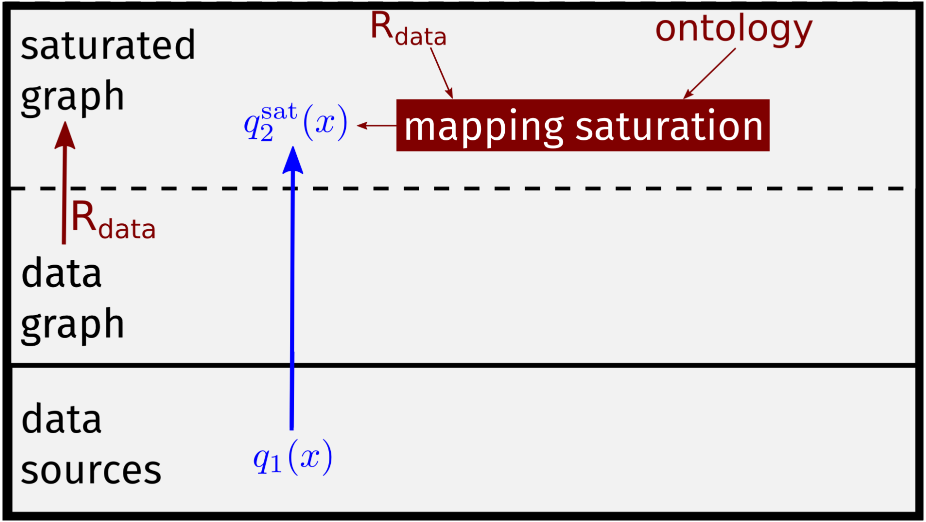
Query answering problem



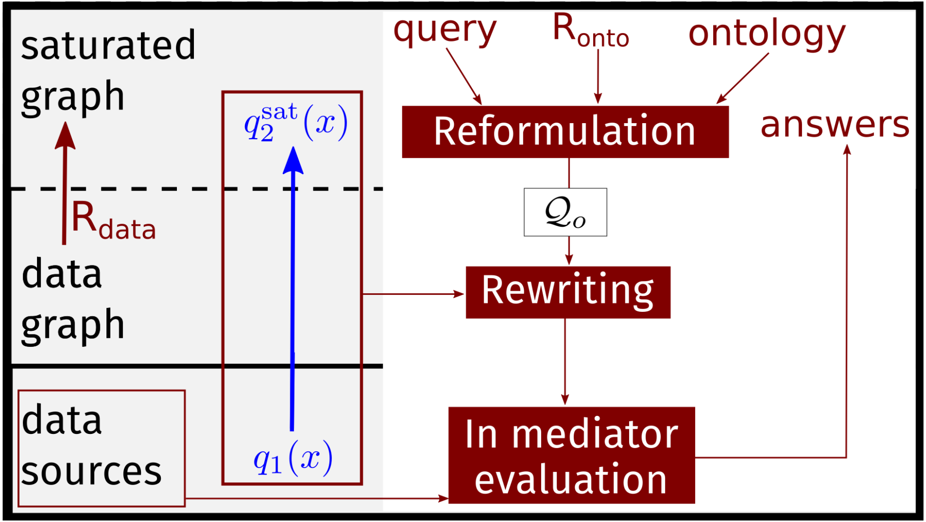
All reasoning at query time (REW-CA)



Some reasoning at query time (REW-C): preprocessing



Some reasoning at query time (REW-C): query time

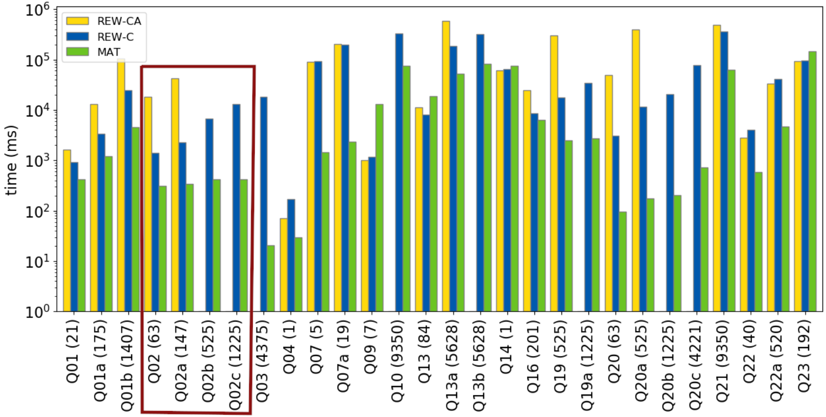


Experiment settings

- Obi-Wan dependencies:
  - OntoSQL (reformulation and materialization)
  - Graal (rewriting using mappings)
  - Tatooine (mediated query evaluation)
- RDF Integration System:
  - Extension of Berlin SPARQL BenMark
  - 3863 GLAV mappings
  - RDFS ontology of 2011 triples
  - Induced graph with 108M triples (185M triples when saturated)
  - Two data sources: One relational and one JSON

Sample comparison on an extension of BSBM

- Materialization (MAT) - kind of reference time
- Full reformulation + rewriting (REW-CA)
- Mapping saturation + partial reformulation + rewriting (REW-C)



Conclusion

1. Global-Local-As-View mappings in OBDA context
2. Queries on data and ontology
3. A new scalable query answering strategy using partial reformulation and saturated mappings
4. Obi-Wan: a query answering system supporting RDFS reasoning

Work with François Goasdoué, Ioana Manolescu, Marie-Laure Mugnier:

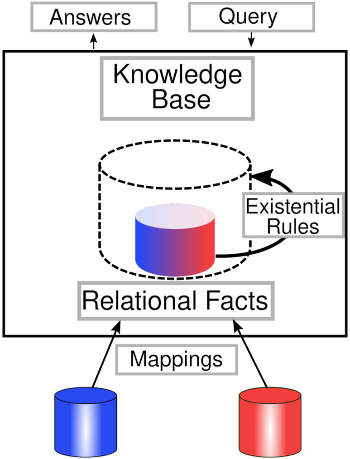
- Ontology-Based RDF Integration of Heterogeneous Data at EDBT 2020
- Obi-Wan demonstration at VLDB 2020: `https://gitlab.inria.fr/cedar/obi-wan`
- Tutorial at the summer school MDD 2022

Part III

Parallelisable existential rules: a story of pieces

joint work with Marie-Laure Mugnier and Michaël Thomazo

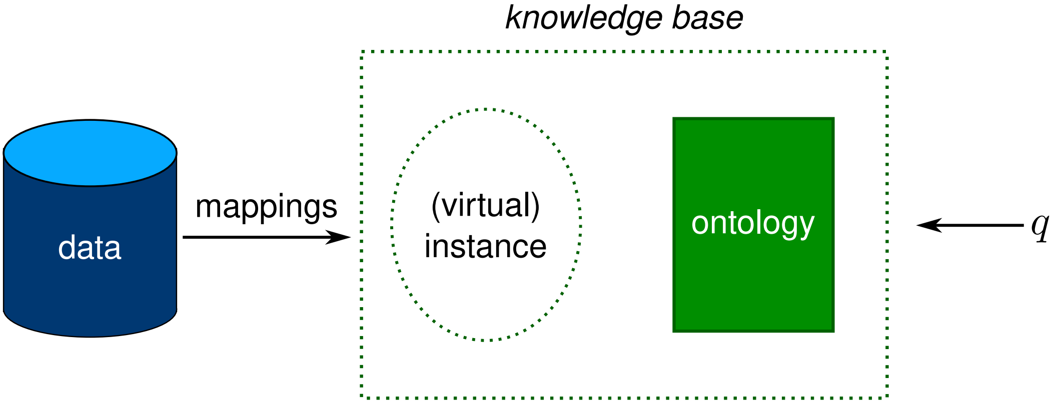
OBDA with existential rules



Motivation: how to answer a query in OBDA using only mappings ?

Context

Ontology-Based Data Access



Mappings as existential rules

Existential rules

$$\forall \vec{x}\ \forall \vec{y}\ (\ \text{Body}[\vec{x}, \vec{y}] \rightarrow \exists \vec{z}\ \text{Head}[\vec{x}, \vec{z}]\ )$$

GLAV mappings (aka source-to-target Tuple Generating Dependencies)

$$\forall \vec{x}\ (\ \exists \vec{y}\ \text{Body}[\vec{x}, \vec{y}] \rightarrow \exists \vec{z}\ \text{Head}[\vec{x}, \vec{z}]\ )$$

- Body is a conjunctive query on the data with answer variables $\vec{x}$
- Head is a conjunctive query on the vocabulary of the ontology with answer variables $\vec{x}$

In the following:

- Rules and mappings have no constants

Chasing with existential rules

Example

$\mathcal{M}$: $M_1 = s_1(x, y) \rightarrow t_1(x, y)$
$M_2 = s_2(x, y) \rightarrow t_2(x)$

$\mathcal{R}$: $R_1 = t_2(x) \rightarrow \exists z\ t_3(x, z)$
$R_2 = t_1(x, y) \wedge t_3(x, z) \rightarrow t_4(y)$

Chasing steps

- $\mathrm{chase}_0(D, \mathcal{M} \cup \mathcal{R}) = D = \{s_1(a, b), s_2(a, c)\}$

Chasing with existential rules

Example

$$\mathcal{M}: \quad M_1 = s_1(x,y) \rightarrow t_1(x,y) \qquad \mathcal{R}: \quad R_1 = t_2(x) \rightarrow \exists z\ t_3(x,z)$$
$$M_2 = s_2(x,y) \rightarrow t_2(x) \qquad R_2 = t_1(x,y) \wedge t_3(x,z) \rightarrow t_4(y)$$

Chasing steps

- $\text{chase}_0(D, \mathcal{M} \cup \mathcal{R}) = D = \{s_1(a,b), s_2(a,c)\}$
- $\text{chase}_1(D, \mathcal{M} \cup \mathcal{R}) = \text{chase}_0(D, \mathcal{M} \cup \mathcal{R}) \cup \{t_1(a,b), t_2(a)\}$

Chasing with existential rules

Example

$$\mathcal{M}: \quad \begin{array}{l} M_1 = s_1(x,y) \rightarrow t_1(x,y) \\ M_2 = s_2(x,y) \rightarrow t_2(x) \end{array} \qquad \mathcal{R}: \quad \begin{array}{l} R_1 = t_2(x) \rightarrow \exists z\ t_3(x,z) \\ R_2 = t_1(x,y) \wedge t_3(x,z) \rightarrow t_4(y) \end{array}$$

Chasing steps

- $\text{chase}_0(D, \mathcal{M} \cup \mathcal{R}) = D = \{s_1(a,b), s_2(a,c)\}$
- $\text{chase}_1(D, \mathcal{M} \cup \mathcal{R}) = \text{chase}_0(D, \mathcal{M} \cup \mathcal{R}) \cup \{t_1(a,b), t_2(a)\}$
- $\text{chase}_2(D, \mathcal{M} \cup \mathcal{R}) = \text{chase}_1(D, \mathcal{M} \cup \mathcal{R}) \cup \{t_3(a,z_0)\}$

Chasing with existential rules

Example

$\mathcal{M}$: $M_1 = s_1(x, y) \rightarrow t_1(x, y)$
$M_2 = s_2(x, y) \rightarrow t_2(x)$

$\mathcal{R}$: $R_1 = t_2(x) \rightarrow \exists z\ t_3(x, z)$
$R_2 = t_1(x, y) \wedge t_3(x, z) \rightarrow t_4(y)$

Chasing steps

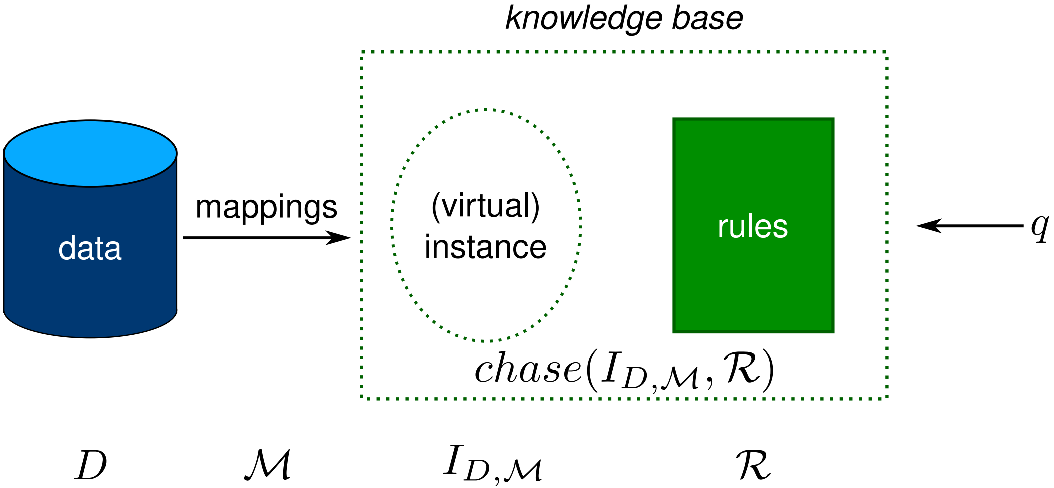
- $\text{chase}_0(D, \mathcal{M} \cup \mathcal{R}) = D = \{s_1(a, b), s_2(a, c)\}$
- $\text{chase}_1(D, \mathcal{M} \cup \mathcal{R}) = \text{chase}_0(D, \mathcal{M} \cup \mathcal{R}) \cup \{t_1(a, b), t_2(a)\}$
- $\text{chase}_2(D, \mathcal{M} \cup \mathcal{R}) = \text{chase}_1(D, \mathcal{M} \cup \mathcal{R}) \cup \{t_3(a, z_0)\}$
- $\text{chase}_3(D, \mathcal{M} \cup \mathcal{R}) = \text{chase}_2(D, \mathcal{M} \cup \mathcal{R}) \cup \{t_4(b)\}$

Virtual instance

$I_{D,\mathcal{M}} = \text{chase}_1(D, \mathcal{M})$

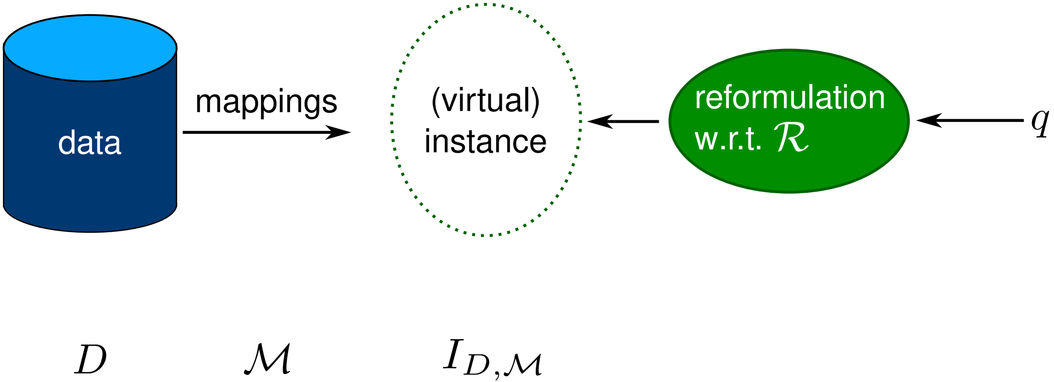
Context

Ontology-Based Data Access with existential rules



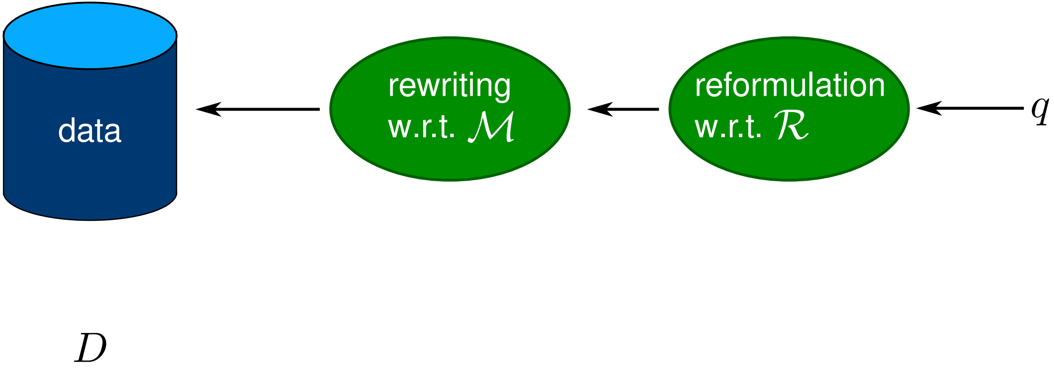
Context

OBDA classical mediation-based query answering method



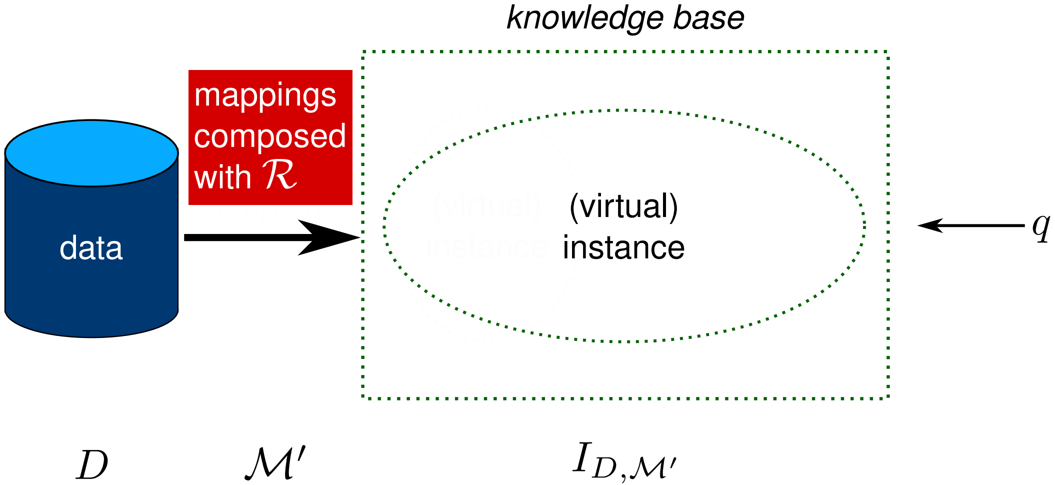
Context

OBDA classical mediation-based query answering method



Context

OBDA query answering by compiling the rules into the mappings



Example

Composing $\mathcal{M}$ with $\mathcal{R}$

$$\mathcal{M}\colon \quad \begin{aligned} M_1 &= s_1(x,y) \to t_1(x,y) \\ M_2 &= s_2(x,y) \to t_2(x) \end{aligned} \qquad \Bigg| \qquad \mathcal{R}\colon \quad \begin{aligned} R_1 &= t_2(x) \to \exists z\ t_3(x,z) \\ R_2 &= t_1(x,y) \wedge t_3(x,z) \to t_4(y) \end{aligned}$$

$$\mathcal{M}'\colon \begin{aligned} M_1 &= s_1(x,y) \to t_1(x,y) \\ M_2 &= s_2(x,y) \to t_2(x) \end{aligned}$$

Example

Composing $\mathcal{M}$ with $\mathcal{R}$

$$\mathcal{M}: \quad \begin{array}{l} M_1 = s_1(x, y) \rightarrow t_1(x, y) \\ M_2 = s_2(x, y) \rightarrow t_2(x) \end{array} \quad \Bigg| \quad \mathcal{R}: \quad \begin{array}{l} R_1 = t_2(x) \rightarrow \exists z \; t_3(x, z) \\ R_2 = t_1(x, y) \wedge t_3(x, z) \rightarrow t_4(y) \end{array}$$

$$\mathcal{M}': \begin{array}{l} M_1 = s_1(x, y) \rightarrow t_1(x, y) \\ M_2 = s_2(x, y) \rightarrow t_2(x) \\ M_3 = R_1 \circ M_2 = s_2(x, y) \rightarrow \exists z \; t_3(x, z) \end{array}$$

Example

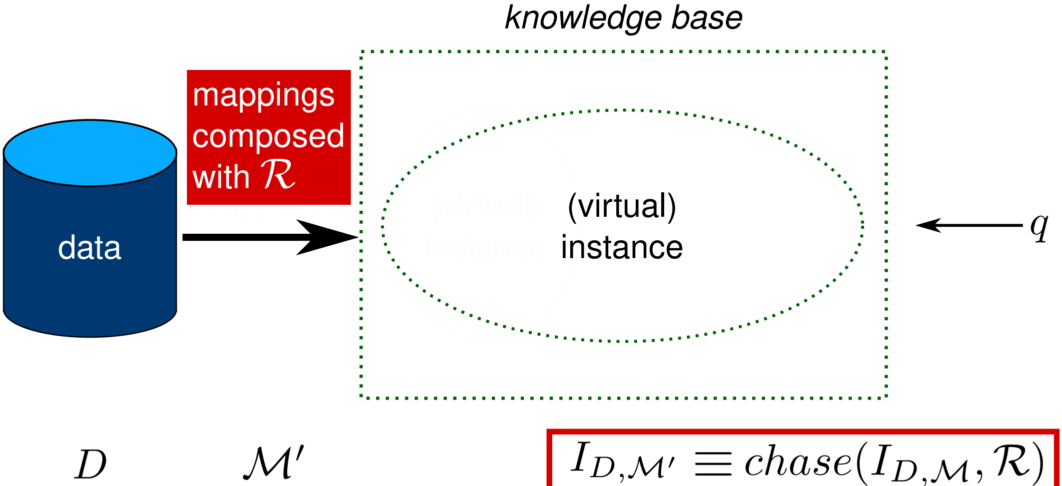
Composing $\mathcal{M}$ with $\mathcal{R}$

$$\mathcal{M}: \quad \begin{array}{l} M_1 = s_1(x, y) \rightarrow t_1(x, y) \\ M_2 = s_2(x, y) \rightarrow t_2(x) \end{array} \quad \Bigg| \quad \mathcal{R}: \quad \begin{array}{l} R_1 = t_2(x) \rightarrow \exists z \; t_3(x, z) \\ R_2 = t_1(x, y) \wedge t_3(x, z) \rightarrow t_4(y) \end{array}$$

$$\mathcal{M}': \begin{array}{l} M_1 = s_1(x, y) \rightarrow t_1(x, y) \\ M_2 = s_2(x, y) \rightarrow t_2(x) \\ M_3 = R_1 \circ M_2 = s_2(x, y) \rightarrow \exists z \; t_3(x, z) \\ M_4 = (R_2 \circ M_1) \circ M_3 = s_1(x, y) \wedge s_2(x, z) \rightarrow t_4(y) \end{array}$$

Context

OBDA query answering by compiling the rules into the mappings



$$\boxed{I_{D,\mathcal{M}'} \equiv chase(I_{D,\mathcal{M}}, \mathcal{R})}$$

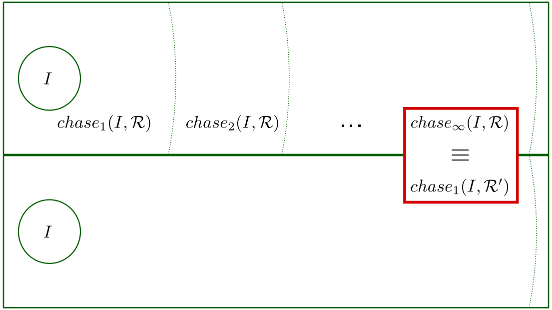
Characterization of the parallelisable rule sets

Research question and contributions

Research question: When can the chase be simulated in a single breadth-first step?

$\mathcal{R}$ is parallelisable if there exists a finite rule set independent from any instance able to produce an equivalent chase of $\mathcal{R}$ in a single step.

$\Rightarrow$ How to characterize parallelisable sets of rules?



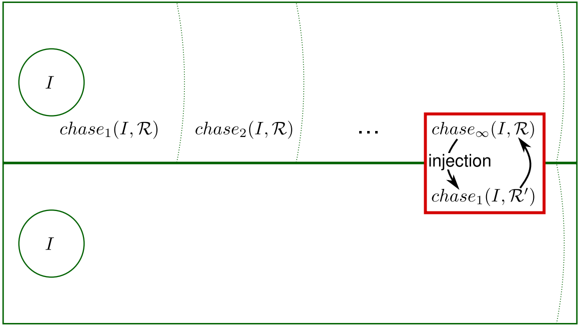
Contributions

- Parallelisable = Bounded + Pieceful
- Links between parallelisability and rule composition

Parallelisability

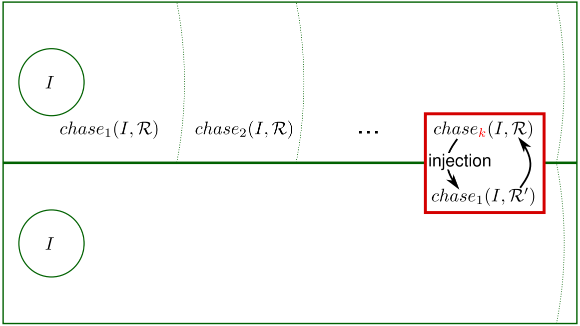
$\mathcal{R}$ is parallelisable if there exists a **finite** rule set $\mathcal{R}'$ such that for any instance I:

1. there is an **injective homomorphism** from $chase_\infty(I, \mathcal{R})$ to $chase_1(I, \mathcal{R}')$
2. there is a homomorphism from $chase_1(I, \mathcal{R}')$ to $chase_\infty(I, \mathcal{R})$



Parallelisability ensures boundedness

$\mathcal{R}$ is bounded if there is k s.t. for any instance I, $chase_k(I, \mathcal{R}) = chase_\infty(I, \mathcal{R})$



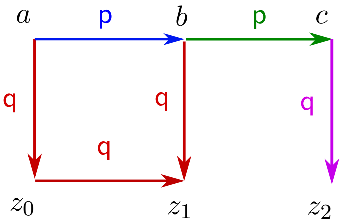
If $\mathcal{R}$ is parallelisable then it is bounded, but the converse does not hold

Key notion: Piece

Piece

Minimal set of atoms 'glued' by nulls in the chase or by existential variables in rule heads.

$$
\begin{aligned}
&p(a, b),\\
&p(b, c),\\
&q(a, z_0), q(z_0, z_1), q(b, z_1),\\
&q(c, z_2)
\end{aligned}
$$

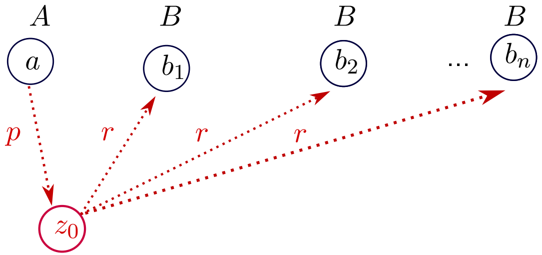


In the following:

We consider that the rules are decomposed in rules having a single-piece head.

Boundedness does not ensure parallelisability

$chase_\infty(I_n, \mathcal{R}) =$



Prime example (bounded)

$R_1 : A(x) \rightarrow \exists z\ p(x, z)$
$R_2 : p(x, z) \wedge B(y) \rightarrow r(z, y)$

$I_n = \{A(a), B(b_1), \ldots, B(b_n)\}$

For any n, $chase_\infty(I_n, \mathcal{R})$ contains a piece of $n + 1$ atoms, hence this rule set is not parallelisable.

A new class: Pieceful

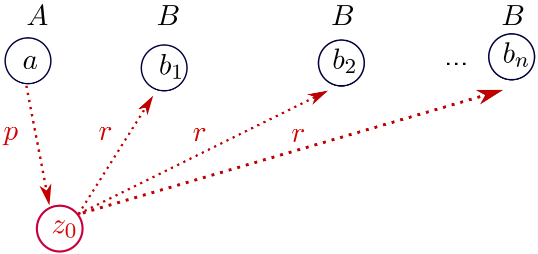
The frontier variables of a rule are the shared variables between its body and head.

$\mathcal{R}$ is pieceful if for any trigger (R, π) in any derivation with $\mathcal{R}$,

- either $\pi(frontier(R))$ belongs to the terms of the initial instance
- or $\pi(frontier(R))$ belongs to the terms of atoms brought by a single previous rule application.

Prime example is not pieceful

$chase_\infty(I_n, \mathcal{R}) =$



Prime example (bounded)

$R_1 : A(x) \rightarrow \exists z\ p(x, z)$
$R_2 : p(x, z) \wedge B(y) \rightarrow r(z, y)$

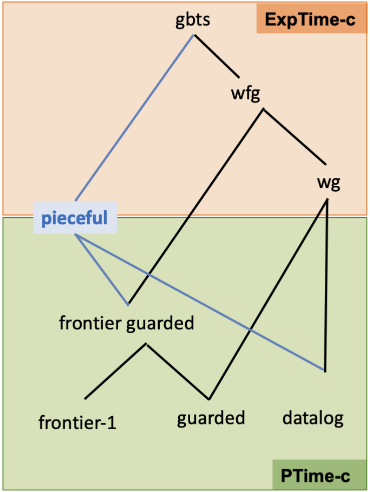
$I_n = \{A(a), B(b_1), \ldots, B(b_n)\}$

First trigger: $(R_1, \{x \mapsto a\}$; creates $p(a, z_0)$
Then: $(R_2, \{x \mapsto a, \mathbf{z \mapsto z_0, y \mapsto b_1}\})$

Parallelisability $\Rightarrow$ Piecefulness

Why? If a rule set $\mathcal{R}$ is not pieceful, one can create an instance I_n s.t. $chase(I_n, \mathcal{R})$ has a null that occurs in at least n atoms.

New landscape



(with data complexity of conjunctive query entailment)

Parallelisability = Boundedness + Piecefulness

What we have so far:

- Parallelisability $\Rightarrow$ Boundedness (but the converse is false: see prime example)
- Parallelisability $\Rightarrow$ Piecefulness (but the converse is false: see transitivity)

Parallelisability = Boundedness + Piecefulness

What we have so far:

- Parallelisability $\Rightarrow$ Boundedness (but the converse is false: see prime example)
- Parallelisability $\Rightarrow$ Piecefulness (but the converse is false: see transitivity)

Boundedness + Piecefulness $\Rightarrow$ Parallelisability

Parallelisability = Boundedness + Piecefulness

What we have so far:

- Parallelisability $\Rightarrow$ Boundedness (but the converse is false: see prime example)
- Parallelisability $\Rightarrow$ Piecefulness (but the converse is false: see transitivity)

Boundedness + Piecefulness $\Rightarrow$ Parallelisability

Parallelisabillity is undecidable

Since the piecefull includes Datalog and the boundedness in Datalog is undecidable.

Conclusion and perspectives

To conclude

- Parallelisable = Bounded + Pieceful
- Links between parallelisability and rule composition

Open issues

- Better understand rule composition to compute parallelisation in practice
- Better understand the properties of the pieceful class
- More succint rule composition based on rule skolemization?
  It would lead beyond (skolemized) existential rules when rules are not pieceful

RDF Entailment rules for data management

Rule name	Entailment rule
rdfs5	$(\mathtt{p}_1, \text{:subproperty}, \mathtt{p}_2), (\mathtt{p}_2, \text{:subproperty}, \mathtt{p}_3) \rightarrow (\mathtt{p}_1, \text{:subproperty}, \mathtt{p}_3)$
rdfs11	$(\mathtt{s}, \text{:subclass}, \mathtt{o}), (\mathtt{o}, \text{:subclass}, \mathtt{o}_1) \rightarrow (\mathtt{s}, \text{:subclass}, \mathtt{o}_1)$
ext1	$(\mathtt{p}, \text{:domain}, \mathtt{o}), (\mathtt{o}, \text{:subclass}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \text{:domain}, \mathtt{o}_1)$
ext2	$(\mathtt{p}, \text{:range}, \mathtt{o}), (\mathtt{o}, \text{:subclass}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \text{:range}, \mathtt{o}_1)$
ext3	$(\mathtt{p}, \text{:subproperty}, \mathtt{p}_1), (\mathtt{p}_1, \text{:domain}, \mathtt{o}) \rightarrow (\mathtt{p}, \text{:domain}, \mathtt{o})$
ext4	$(\mathtt{p}, \text{:subproperty}, \mathtt{p}_1), (\mathtt{p}_1, \text{:range}, \mathtt{o}) \rightarrow (\mathtt{p}, \text{:range}, \mathtt{o})$
rdfs2	$(\mathtt{p}, \text{:domain}, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{s}_1, \text{:type}, \mathtt{o})$
rdfs3	$(\mathtt{p}, \text{:range}, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{o}_1, \text{:type}, \mathtt{o})$
rdfs7	$(\mathtt{p}_1, \text{:subproperty}, \mathtt{p}_2), (\mathtt{s}, \mathtt{p}_1, \mathtt{o}) \rightarrow (\mathtt{s}, \mathtt{p}_2, \mathtt{o})$
rdfs9	$(\mathtt{s}, \text{:subclass}, \mathtt{o}), (\mathtt{s}_1, \text{:type}, \mathtt{s}) \rightarrow (\mathtt{s}_1, \text{:type}, \mathtt{o})$

$\mathcal{R}$

$\mathcal{R}_{\text{onto}}$

$\mathcal{R}_{\text{data}}$

View-based rewriting details

Global-Local-As-View Mapping Example (2)

data graph

$(\text{name}, :\text{usesWeapon}, \text{saber}),$
$(\text{saber}, :\text{type}, :\text{LightSaber})$

data sources

$q_{\text{MONGO}}(\text{name}, \text{saber})$

```
{
  "name": "Rey",
  "saber": "luke-saber"
}
```

Breaking Global-Local-As-View Mappings into Views

We decompose the GLAV mappings into GAV and LAV views.

1. $m_{\text{pilot}} = q_{SQL}(\text{name}) \rightarrow (\text{name}, \text{:pilotOf}, y), (y, \text{:type}, \text{:Starship})$

GAV $V_{\text{pilot}}(\text{name}) \leftarrow q_{SQL}(\text{name})$
LAV $V_{\text{pilot}}(\text{name}) \leftarrow (\text{name}, \text{:pilotOf}, y), (y, \text{:type}, \text{:Starship})$

Breaking Global-Local-As-View Mappings into Views

We decompose the GLAV mappings into GAV and LAV views.

1. $m_{\text{pilot}} = q_{SQL}(\text{name}) \rightarrow (\text{name}, \text{:pilotOf}, y), (y, \text{:type}, \text{:Starship})$

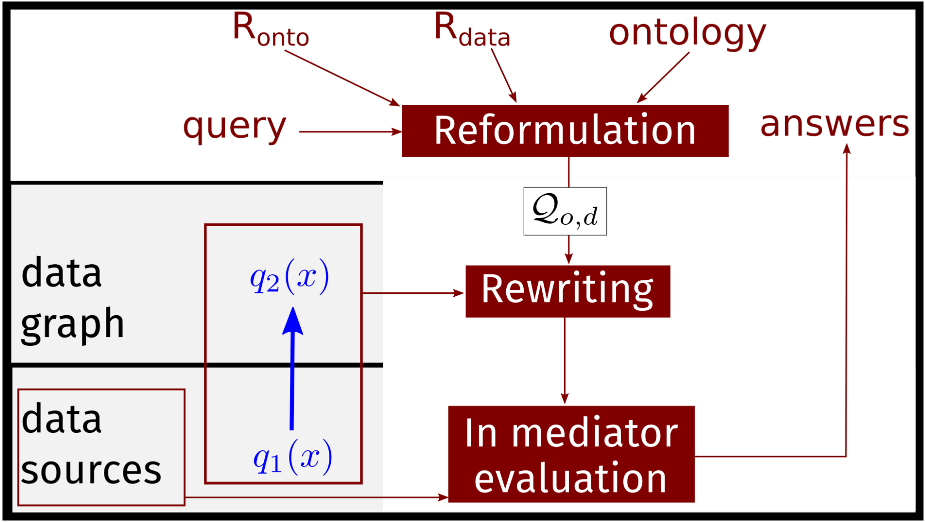
   GAV $V_{\text{pilot}}(\text{name}) \leftarrow q_{SQL}(\text{name})$
   LAV $V_{\text{pilot}}(\text{name}) \leftarrow (\text{name}, \text{:pilotOf}, y), (y, \text{:type}, \text{:Starship})$

2. $m_{\text{jedi}} = q_{MONGO}(\text{name}, \text{saber}) \rightarrow$
   $(\text{name}, \text{:usesWeapon}, \text{saber}), (\text{saber}, \text{:type}, \text{:LightSaber})$

   GAV $V_{\text{jedi}}(\text{name}, \text{saber}) \leftarrow q_{MONGO}(\text{name}, \text{saber})$
   LAV $V_{\text{jedi}}(\text{name}, \text{saber}) \leftarrow (\text{name}, \text{:usesWeapon}, \text{saber}), (\text{saber}, \text{:type}, \text{:LightSaber})$

LAV views will be used to perform a query rewriting.

All Reasoning at Query Time (REW-CA)

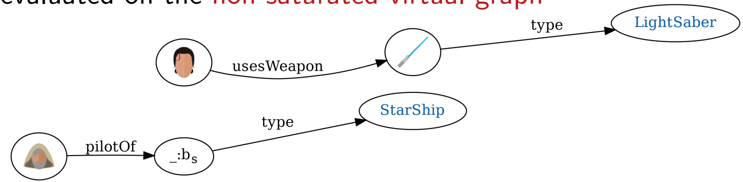


All Reasoning at Query Time (REW-CA): Example

The query reformulation of $q(x, y) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, y), (y, \text{:subclass}, \text{:Object})$

$$
\begin{aligned}
\mathcal{Q}_{o,d} \quad = \quad & q_{o,1}(x, \text{:LightSaber}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:LightSaber}) \\
\cup \quad & q_{d,1}(x, \text{:LightSaber}) \leftarrow (x, \text{:usesWeapon}, z), (z, \text{:type}, \text{:LightSaber}) \\
\cup \quad & q_{o,2}(x, \text{:Vehicle}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:Vehicle}) \\
\cup \quad & q_{d,6}(x, \text{:Vehicle}) \leftarrow (x, \text{:pilotOf}, z), (z, \text{:type}, \text{:StarShip}) \\
\cup \quad & q_{o,3}(x, \text{:StarShip}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:StarShip}) \\
\cup \quad & q_{d,8}(x, \text{:StarShip}) \leftarrow (x, \text{:pilotOf}, z), (z, \text{:type}, \text{:StarShip}) \\
\cup \quad & \ldots
\end{aligned}
$$

must be evaluated on the non-saturated virtual graph



using view-based query rewriting

All Reasoning at Query Time (REW-CA): Example

The query reformulation of $q(x, y) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, y), (y, \text{:subclass}, \text{:Object})$

$$
\begin{aligned}
\mathcal{Q}_{o,d} \quad = \quad & q_{o,1}(x, \text{:LightSaber}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:LightSaber}) \\
\cup \quad & q_{d,1}(x, \text{:LightSaber}) \leftarrow (x, \text{:usesWeapon}, z), (z, \text{:type}, \text{:LightSaber}) \\
\cup \quad & q_{o,2}(x, \text{:Vehicle}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:Vehicle}) \\
\cup \quad & q_{d,6}(x, \text{:Vehicle}) \leftarrow (x, \text{:pilotOf}, z), (z, \text{:type}, \text{:StarShip}) \\
\cup \quad & q_{o,3}(x, \text{:StarShip}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:StarShip}) \\
\cup \quad & q_{d,8}(x, \text{:StarShip}) \leftarrow (x, \text{:pilotOf}, z), (z, \text{:type}, \text{:StarShip}) \\
\cup \quad & \ldots
\end{aligned}
$$

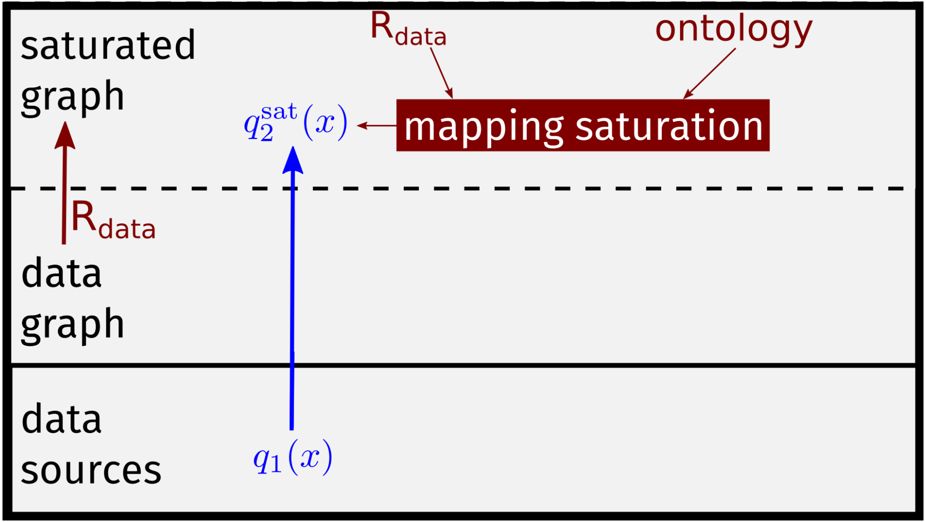
We use LAV views

$$
\begin{aligned}
V_{\text{pilot}}(\text{name}) \quad & \leftarrow \quad (\text{name}, \text{:pilotOf}, y), (y, \text{:type}, \text{:Starship}) \\
V_{\text{jedi}}(\text{name}, \text{saber}) \quad & \leftarrow \quad (\text{name}, \text{:usesWeapon}, \text{saber}), (\text{saber}, \text{:type}, \text{:LightSaber})
\end{aligned}
$$

to rewrite the reformulations into

$$
\begin{aligned}
\text{rew} \quad = \quad & \text{rew1}(x, \text{:LightSaber}) \leftarrow V_{\text{jedi}}(x, \text{saber}) \\
\cup \quad & \text{rew2}(x, \text{:Vehicle}) \leftarrow V_{\text{pilot}}(x) \\
\cup \quad & \text{rew3}(x, \text{:StarShip}) \leftarrow V_{\text{pilot}}(x)
\end{aligned}
$$

Some Reasoning at Query Time (REW-C): Preprocessing



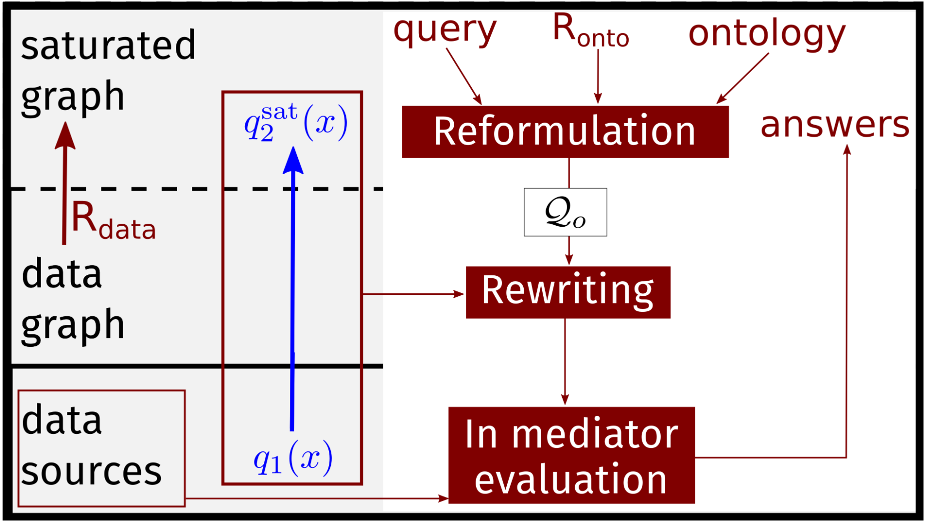
Some Reasoning at Query Time (REW-C): Mapping Saturation Example

We saturate the LAV view definitions using $\mathcal{R}_{\mathrm{data}}$ and the ontology.

$$V_{\mathrm{pilot}}(\mathrm{name}) \leftarrow (\mathrm{name}, :\mathrm{pilotOf}, y), (y, :\mathrm{type}, :\mathrm{Starship}), (\mathrm{name}, :\mathrm{uses}, y), (y, :\mathrm{type}, :\mathrm{Vehicle}), (y, :\mathrm{type}, :\mathrm{Object})$$

$$V_{\mathrm{jedi}}(\mathrm{name}, \mathrm{saber}) \leftarrow (\mathrm{name}, :\mathrm{usesWeapon}, \mathrm{saber}), (\mathrm{saber}, :\mathrm{type}, :\mathrm{LightSaber}) (\mathrm{name}, :\mathrm{uses}, \mathrm{saber}), (\mathrm{saber}, :\mathrm{type}, :\mathrm{Object})$$

Some Reasoning at Query Time (REW-C): Query Time

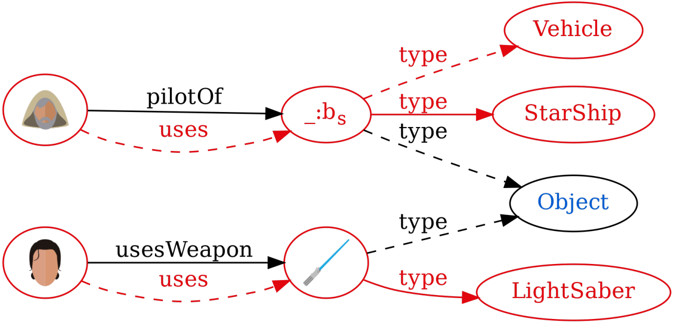


Some Reasoning at Query Time (REW-C): Example

The query reformulation of $q(x, y) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, y), (y, \text{:subclass}, \text{:Object})$

$$\begin{aligned} \mathcal{Q}_o &= q_{o,1}(x, \text{:LightSaber}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:LightSaber}) \\ &\cup \; q_{o,2}(x, \text{:Vehicle}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:Vehicle}) \\ &\cup \; q_{o,3}(x, \text{:StarShip}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:StarShip}) \end{aligned}$$

must be evaluated on the saturated virtual graph



using view-based query rewriting

Some Reasoning at Query Time (REW-C): Example

The query reformulation of $q(x, y) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, y), (y, \text{:subclass}, \text{:Object})$

$$\begin{aligned} \mathcal{Q}_o &= q_{o,1}(x, \text{:LightSaber}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:LightSaber}) \\ &\cup \quad q_{o,2}(x, \text{:Vehicle}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:Vehicle}) \\ &\cup \quad q_{o,3}(x, \text{:StarShip}) \leftarrow (x, \text{:uses}, z), (z, \text{:type}, \text{:StarShip}) \end{aligned}$$

We use the saturated LAV views

$$\begin{aligned} V_{\text{pilot}}(\text{name}) \quad &\leftarrow \quad (\text{name}, \text{:pilotOf}, y), (y, \text{:type}, \text{:Starship}), \\ & \qquad (\text{name}, \text{:uses}, y), (y, \text{:type}, \text{:Vehicle}), (y, \text{:type}, \text{:Object}) \\ V_{\text{jedi}}(\text{name}, \text{saber}) \quad &\leftarrow \quad (\text{name}, \text{:usesWeapon}, \text{saber}), (\text{saber}, \text{:type}, \text{:LightSaber}) \\ & \qquad (\text{name}, \text{:uses}, \text{saber}), (\text{saber}, \text{:type}, \text{:Object}) \end{aligned}$$

to rewrite the reformulation into

$$\begin{aligned} \text{rew} &= \text{rew1}(x, \text{:LightSaber}) \leftarrow V_{\text{jedi}}(x, \text{saber}) \\ &\cup \quad \text{rew2}(x, \text{:Vehicle}) \leftarrow V_{\text{pilot}}(x) \\ &\cup \quad \text{rew3}(x, \text{:StarShip}) \leftarrow V_{\text{pilot}}(x) \end{aligned}$$

Rule composition

Rule composition

Datalog unfolding

For datalog rules: parallelisability = boundedness

A parallelisation of $\mathcal{R}$ can be computed by 'unfolding' the rules from $\mathcal{R}$.
$\mathcal{R}^\star$: starting from $\mathcal{R}$, we repeatedly unfold a rule from $\mathcal{R}^\star$ with a rule from $\mathcal{R}$.

$\mathcal{R} = \{R_1, R_2, R_3\}$ $R_1 : A(x) \rightarrow B(x)$
$R_2 : C(x) \rightarrow D(x)$
$R_3 : B(x) \wedge D(x) \rightarrow G(x)$

Denoting $R_i \circ R_j$ the unfolding of R_i by R_j, we obtain:
$R_3 \circ R_1 : A(x) \wedge D(x) \rightarrow G(x)$

Rule composition

Datalog unfolding

For datalog rules: parallelisability = boundedness

A parallelisation of $\mathcal{R}$ can be computed by 'unfolding' the rules from $\mathcal{R}$.
$\mathcal{R}^\star$: starting from $\mathcal{R}$, we repeatedly unfold a rule from $\mathcal{R}^\star$ with a rule from $\mathcal{R}$.

$\mathcal{R} = \{R_1, R_2, R_3\}$ $R_1 : A(x) \rightarrow B(x)$
$R_2 : C(x) \rightarrow D(x)$
$R_3 : B(x) \wedge D(x) \rightarrow G(x)$

Denoting $R_i \circ R_j$ the unfolding of R_i by R_j, we obtain:
$R_3 \circ R_1 : A(x) \wedge D(x) \rightarrow G(x)$
$R_3 \circ R_2 : B(x) \wedge C(x) \rightarrow G(x)$

Rule composition

Datalog unfolding

For datalog rules: parallelisability = boundedness

A parallelisation of $\mathcal{R}$ can be computed by 'unfolding' the rules from $\mathcal{R}$.
$\mathcal{R}^\star$: starting from $\mathcal{R}$, we repeatedly unfold a rule from $\mathcal{R}^\star$ with a rule from $\mathcal{R}$.

$\mathcal{R} = \{R_1, R_2, R_3\}$ $R_1 : A(x) \to B(x)$
$R_2 : C(x) \to D(x)$
$R_3 : B(x) \wedge D(x) \to G(x)$

Denoting $R_i \circ R_j$ the unfolding of R_i by R_j, we obtain:
$R_3 \circ R_1 : A(x) \wedge D(x) \to G(x)$
$R_3 \circ R_2 : B(x) \wedge C(x) \to G(x)$
$(R_3 \circ R_1) \circ R_2 : A(x) \wedge C(x) \to G(x)$
$(R_3 \circ R_2) \circ R_1 = (R_3 \circ R_1) \circ R_2$.
$\mathcal{R}^\star = \mathcal{R} \cup \{R_3 \circ R_1, R_3 \circ R_2, (R_3 \circ R_1) \circ R_2\}$

Rule composition

Datalog unfolding

For datalog rules: parallelisability = boundedness

A parallelisation of $\mathcal{R}$ can be computed by 'unfolding' the rules from $\mathcal{R}$.
$\mathcal{R}^\star$: starting from $\mathcal{R}$, we repeatedly unfold a rule from $\mathcal{R}^\star$ with a rule from $\mathcal{R}$.

$\mathcal{R} = \{R_1, R_2, R_3\}$ $R_1 : A(x) \to B(x)$
$R_2 : C(x) \to D(x)$
$R_3 : B(x) \wedge D(x) \to G(x)$

Denoting $R_i \circ R_j$ the unfolding of R_i by R_j, we obtain:
$R_3 \circ R_1 : A(x) \wedge D(x) \to G(x)$
$R_3 \circ R_2 : B(x) \wedge C(x) \to G(x)$
$(R_3 \circ R_1) \circ R_2 : A(x) \wedge C(x) \to G(x)$
$(R_3 \circ R_2) \circ R_1 = (R_3 \circ R_1) \circ R_2$.
$\mathcal{R}^\star = \mathcal{R} \cup \{R_3 \circ R_1, R_3 \circ R_2, (R_3 \circ R_1) \circ R_2\}$

Remark: $\mathcal{R}^\star = \{\text{rewriting}(\text{body}(R)) \to \text{head}(R) \mid R \in \mathcal{R}\}$

Rule composition

Datalog unfolding

For datalog rules: parallelisability = boundedness

A parallelisation of $\mathcal{R}$ can be computed by 'unfolding' the rules from $\mathcal{R}$.
$\mathcal{R}^\star$: starting from $\mathcal{R}$, we repeatedly unfold a rule from $\mathcal{R}^\star$ with a rule from $\mathcal{R}$.

$\mathcal{R} = \{R_1, R_2, R_3\}$ $R_1 : A(x) \to B(x)$
$R_2 : C(x) \to D(x)$
$R_3 : B(x) \wedge D(x) \to G(x)$

Denoting $R_i \circ R_j$ the unfolding of R_i by R_j, we obtain:
$R_3 \circ R_1 : A(x) \wedge D(x) \to G(x)$
$R_3 \circ R_2 : B(x) \wedge C(x) \to G(x)$
$(R_3 \circ R_1) \circ R_2 : A(x) \wedge C(x) \to G(x)$
$(R_3 \circ R_2) \circ R_1 = (R_3 \circ R_1) \circ R_2$.
$\mathcal{R}^\star = \mathcal{R} \cup \{R_3 \circ R_1, R_3 \circ R_2, (R_3 \circ R_1) \circ R_2\}$

Remark: $\mathcal{R}^\star = \{\mathrm{rewriting}(\mathrm{body}(R)) \to \mathrm{head}(R) \mid R \in \mathcal{R}\}$

Rule composition

Existential rules

Unfolding extended to (single-piece) existential rules

- Based on piece-unifiers instead of classical unifiers
- Generates rules inducing every pieces of the chase (growing heads)
- Keeps single-piece rules

Rule composition

Existential rules

Unfolding extended to (single-piece) existential rules

- Based on piece-unifiers instead of classical unifiers
- Generates rules inducing every pieces of the chase (growing heads)
- Keeps single-piece rules

$\cancel{R_3 \circ R_1}$ impossible because of the piece-unifier

$\mathcal{R} = \{R_1, R_2, R_3\}$
$R_1 : A(x) \rightarrow \exists y\ p(x, y)$
$R_2 : p(x, y) \rightarrow \exists z\ s(y, z)$
$R_3 : p(x, y) \wedge s(y, z) \rightarrow B(z)$

Rule composition

Existential rules

Unfolding extended to (single-piece) existential rules

- Based on piece-unifiers instead of classical unifiers
- Generates rules inducing every pieces of the chase (growing heads)
- Keeps single-piece rules

$\mathcal{R} = \{R_1, R_2, R_3\}$
$R_1 : A(x) \rightarrow \exists y\ p(x, y)$
$R_2 : p(x, y) \rightarrow \exists z\ s(y, z)$
$R_3 : p(x, y) \wedge s(y, z) \rightarrow B(z)$

$\cancel{R_3 \circ R_1}$ impossible because of the piece-unifier
$R_2 \circ R_1 : A(x) \rightarrow \exists y, z\ p(x, y) \wedge s(y, z)$

Rule composition

Existential rules

Unfolding extended to (single-piece) existential rules

- Based on piece-unifiers instead of classical unifiers
- Generates rules inducing every pieces of the chase (growing heads)
- Keeps single-piece rules

$\mathcal{R} = \{R_1, R_2, R_3\}$
$R_1 : A(x) \rightarrow \exists y\ p(x, y)$
$R_2 : p(x, y) \rightarrow \exists z\ s(y, z)$
$R_3 : p(x, y) \wedge s(y, z) \rightarrow B(z)$

$\cancel{R_3 \circ R_1}$ impossible because of the piece-unifier
$R_2 \circ R_1 : A(x) \rightarrow \exists y, z\ p(x, y) \wedge s(y, z)$
$R_3 \circ (R_2 \circ R_1) : A(x) \rightarrow \exists y, z\ p(x, y) \wedge s(y, z), B(z)$

Rule composition

Existential rules

Unfolding extended to (single-piece) existential rules

- Based on piece-unifiers instead of classical unifiers
- Generates rules inducing every pieces of the chase (growing heads)
- Keeps single-piece rules

$\mathcal{R} = \{R_1, R_2, R_3\}$
$R_1 : A(x) \to \exists y\ p(x, y)$
$R_2 : p(x, y) \to \exists z\ s(y, z)$
$R_3 : p(x, y) \wedge s(y, z) \to B(z)$

$\cancel{R_3 \circ R_1}$ impossible because of the piece-unifier
$R_2 \circ R_1 : A(x) \to \exists y, z\ p(x, y) \wedge s(y, z)$
$R_3 \circ (R_2 \circ R_1) : A(x) \to \exists y, z\ p(x, y) \wedge s(y, z), B(z)$
$R_3 \circ R_2 : p(x, y) \to \exists z\ s(y, z), B(z)$
$(R_3 \circ R_2) \circ R_1 = R_3 \circ (R_2 \circ R_1)$
$\mathcal{R}^\star = \mathcal{R} \cup \{R_2 \circ R_1, R_3 \circ R_2, (R_3 \circ R_2) \circ R_1\}$

Rule composition

Existential rules

Unfolding extended to (single-piece) existential rules

- Based on piece-unifiers instead of classical unifiers
- Generates rules inducing every pieces of the chase (growing heads)
- Keeps single-piece rules

$\mathcal{R} = \{R_1, R_2, R_3\}$
$R_1 : A(x) \rightarrow \exists y\ p(x, y)$
$R_2 : p(x, y) \rightarrow \exists z\ s(y, z)$
$R_3 : p(x, y) \wedge s(y, z) \rightarrow B(z)$

$\cancel{R_3 \circ R_1}$ impossible because of the piece-unifier
$R_2 \circ R_1 : A(x) \rightarrow \exists y, z\ p(x, y) \wedge s(y, z)$
$R_3 \circ (R_2 \circ R_1) : A(x) \rightarrow \exists y, z\ p(x, y) \wedge s(y, z), B(z)$
$R_3 \circ R_2 : p(x, y) \rightarrow \exists z\ s(y, z), B(z)$
$(R_3 \circ R_2) \circ R_1 = R_3 \circ (R_2 \circ R_1)$
$\mathcal{R}^\star = \mathcal{R} \cup \{R_2 \circ R_1, R_3 \circ R_2, (R_3 \circ R_2) \circ R_1\}$

Soundness and completeness of $\mathcal{R}^\star$: $I, \mathcal{R} \models q$ iff $chase_1(I, \mathcal{R}^\star) \models q$

Details on existential rule composition $R_2 \circ R_1$

Given $R_1 : B_1 \rightarrow H_1$ and $R_2 : B_2 \rightarrow H_2$
and $\mu = (B'_2, H'_1, u)$ a piece-unifier of B_2 with R_1:

1. If $u(frontier(R_2)) \cap exist(R_1) = \emptyset$:

$$R_2 \circ_\mu R_1 = u(B_1) \cup u(B_2 \setminus B'_2) \rightarrow u(H_2)$$

2. Otherwise:

$$R_2 \circ_\mu R_1 = u(B_1) \cup u(B_2 \setminus B'_2) \rightarrow u(H_1) \cup u(H_2)$$

In short: if no frontier variable of R_2 is unified with an existential variable of R_1, the head of R_1 can be safely ignored, which allows to keep single-piece rules

Details on existential rule composition $R_2 \circ R_1$

Given $R_1 : B_1 \to H_1$ and $R_2 : B_2 \to H_2$
and $\mu = (B_2', H_1', u)$ a piece-unifier of B_2 with R_1:

1. If $u(frontier(R_2)) \cap exist(R_1) = \emptyset$:

$$R_2 \circ_\mu R_1 = u(B_1) \cup u(B_2 \setminus B_2') \to u(H_2)$$

2. Otherwise:

$$R_2 \circ_\mu R_1 = u(B_1) \cup u(B_2 \setminus B_2') \to u(H_1) \cup u(H_2)$$

In short: if no frontier variable of R_2 is unified with an existential variable of R_1, the head of R_1 can be safely ignored, which allows to keep single-piece rules

Definition of $\mathcal{R}^\star$ the composed rules from $\mathcal{R}$:

starting from $\mathcal{R}$, we repeatedly compose the rules in $\mathcal{R}^\star$ pairwise.

Rule composition on the prime example

$R_1 : A(x) \rightarrow \exists z\ p(x, z)$
$R_2 : p(x, z) \wedge B(y) \rightarrow r(z, y)$

Let us build $\mathcal{R}^\star$:
$R_2 \circ R_1 : A(x) \wedge B(y) \rightarrow \exists z\ p(x, z) \wedge r(z, y)$

Rule composition on the prime example

$R_1 : A(x) \to \exists z\ p(x,z)$
$R_2 : p(x,z) \wedge B(y) \to r(z,y)$

Let us build $\mathcal{R}^\star$:
$R_2 \circ R_1 : A(x) \wedge B(y) \to \exists z\ p(x,z) \wedge r(z,y)$
$R_2 \circ (R_2 \circ R_1) : A(x) \wedge B(y) \wedge B(y_1) \to \exists z\ p(x,z) \wedge r(z,y) \wedge r(z,y_1)$

Rule composition on the prime example

$R_1 : A(x) \rightarrow \exists z\ p(x, z)$
$R_2 : p(x, z) \wedge B(y) \rightarrow r(z, y)$

Let us build $\mathcal{R}^\star$:
$R_2 \circ R_1 : A(x) \wedge B(y) \rightarrow \exists z\ p(x, z) \wedge r(z, y)$
$R_2 \circ (R_2 \circ R_1) : A(x) \wedge B(y) \wedge B(y_1) \rightarrow \exists z\ p(x, z) \wedge r(z, y) \wedge r(z, y_1)$
etc.
At each step, a new rule $R_2 \circ R^*$, where R^* is the rule created at the preceding step:
$A(x) \wedge B(y) \wedge B(y_1) \dots B(y_i) \rightarrow \exists z\ p(x, z) \wedge r(z, y) \wedge r(z, y_1) \dots \wedge r(z, y_i)$

What this example shows:

- Completeness requires composition of the form $R \circ R^*$ (and not only $R^* \circ R$ as in datalog)
- $\mathcal{R}^\star$ may be infinite even if $\mathcal{R}$ is bounded, with no finite subset of $\mathcal{R}^\star$ being complete.

Parallelisation by rule composition

Completeness of $\mathcal{R}^\star$

If $\mathcal{R}$ is pieceful, then for any instance I, each piece of $chase_\infty(I, \mathcal{R})$ can be obtained by applying a rule from $\mathcal{R}^\star$ to I.

Conjecture: this is true even if $\mathcal{R}$ is not pieceful

Corollary

If $\mathcal{R}$ is parallelisable (ie pieceful and bounded)
then it is parallelisable by a (finite) subset of $\mathcal{R}^\star$

Another characterization of piecefulness

(Existential) stability

- For a piece-unifier of $body(R_2)$ with R_1: if a frontier variable of R_2 is unified with an existential variable of R_1, then the whole frontier of R_2 is unified
- For $\mathcal{R}$: all piece-unifiers with rules of $\mathcal{R}$ have the stability property

Existential stability may be lost when a composed rule is added

We say that $\mathcal{R}$ has the existential stability 'at the infinite' if $\mathcal{R}^\star$ has the existential stability

Piecefulness = Stability at the infinite

- If $\mathcal{R}$ is pieceful then it has the existential stability
- If $\mathcal{R}$ is pieceful then $\mathcal{R}^\star$ is pieceful (hence, $\mathcal{R}^\star$ has the existential stability)
- If $\mathcal{R}$ is stable at the infinite then it is pieceful