QUERY ANSWERING WITH DESCRIPTION LOGIC ONTOLOGIES

Meghyn Bienvenu (*CNRS & Université de Montpellier*)
Magdalena Ortiz (*Vienna University of Technology*)

CONJUNCTIVE QUERIES

(UNIONS OF) CONJUNCTIVE QUERIES

IQs quite restricted: No selections and joins as in DB queries

(UNIONS OF) CONJUNCTIVE QUERIES

IQs quite restricted: No selections and joins as in DB queries

Most work on OMQA adopts (unions of) conjunctive queries (CQs)

IQs quite restricted: No selections and joins as in DB queries

Most work on OMQA adopts (unions of) conjunctive queries (CQs)

A conjunctive query (CQ) is a first-order query $q(\vec{x})$ of the form

$$\exists \vec{y}. P_1(\vec{t_1}) \wedge \cdots \wedge P_n(\vec{t_n})$$

where every variable in some $\vec{t_i}$ appears in either $\vec{x}$ or $\vec{y}$ and every P_i is either a concept or role name

(UNIONS OF) CONJUNCTIVE QUERIES

IQs quite restricted: No selections and joins as in DB queries

Most work on OMQA adopts (unions of) conjunctive queries (CQs)

A conjunctive query (CQ) is a first-order query $q(\vec{x})$ of the form

$$\exists \vec{y}. P_1(\vec{t_1}) \wedge \cdots \wedge P_n(\vec{t_n})$$

where every variable in some $\vec{t_i}$ appears in either $\vec{x}$ or $\vec{y}$ and every P_i is either a concept or role name

A union of CQs (UCQ) is a first-order query $q(\vec{x})$ of the form

$$q_1(\vec{x}) \vee \cdots \vee q_n(\vec{x})$$

where the $q_i(\vec{x})$ are CQs with same tuple $\vec{x}$ of free vars

WHAT CAN WE EXPRESS AS CQS?

Find pairs of restaurants and dishes they serve which contain an spicy ingredient:

$$q_1(x, y) = \quad \exists z.\mathsf{serves}(x, y) \wedge \mathsf{Dish}(y) \wedge \mathsf{hasIngred}(y, z) \wedge \mathsf{Spicy}(z)$$

Find restaurants that serve a vegetarian menu and a menu with a spicy main dish, and that both have the same cake as dessert:

$$\begin{aligned} q_2(x) = \quad & \exists y_1, y_2, z_1, z_2.\mathsf{serves}(x, y_1) \wedge \mathsf{vegMenu}(y_1) \wedge \\ & \quad \mathsf{hasDessert}(y_1, z_1) \wedge \mathsf{Cake}(z_1) \wedge \\ & \quad \mathit{serves}(x, y_2) \wedge \mathsf{Menu}(y_2) \wedge \mathsf{hasMain}(y_2, z_2) \wedge \\ & \quad \mathsf{Spicy}(z_2) \wedge \mathsf{hasDessert}(y_2, z_1) \end{aligned}$$

WHAT CAN WE EXPRESS AS CQS?

Find pairs of restaurants and dishes they serve which contain an spicy ingredient:

$$q_1(x,y) = \quad \exists z.\mathsf{serves}(x,y) \wedge \mathsf{Dish}(y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{Spicy}(z)$$

Find restaurants that serve a vegetarian menu and a menu with a spicy main dish, and that both have the same cake as dessert:

$$\begin{aligned} q_2(x) = \quad & \exists y_1, y_2, z_1, z_2.\mathsf{serves}(x,y_1) \wedge \mathsf{vegMenu}(y_1) \wedge \\ & \quad \mathsf{hasDessert}(y_1,z_1) \wedge \mathsf{Cake}(z_1) \wedge \\ & \quad serves(x,y_2) \wedge \mathsf{Menu}(y_2) \wedge \mathsf{hasMain}(y_2,z_2) \wedge \\ & \quad \mathsf{Spicy}(z_2) \wedge \mathsf{hasDessert}(y_2,z_1) \end{aligned}$$

In general, not expressible as instance queries!

Find restaurants that serve a dish that contains an spicy ingredient, or that contains an ingredient that contains an spicy ingredient:

$$
\begin{aligned}
q_1(x) = \quad & (\exists y, z.\mathsf{serves}(x, y) \wedge \mathsf{Dish}(y) \wedge \mathsf{hasIngred}(y, z) \wedge \mathsf{Spicy}(z)) \\
& \vee \\
& (\exists y_1, y_2, z.\mathsf{serves}(x, y_1) \wedge \mathsf{Dish}(y_1) \wedge \\
& \qquad \mathsf{hasIngred}(y_1, y_2) \wedge \mathsf{hasIngred}(y_2, z) \wedge \mathsf{Spicy}(z))
\end{aligned}
$$

CQs correspond to:

- select-project-join queries of relational algebra / SQL
- basic graph patterns of SPARQL

Alternatively, CQs and UCQs can be seen as Datalog rules

CQs:

$$q(\vec{x}) = \exists \vec{y}. P_1(\vec{t_1}) \wedge \cdots \wedge P_n(\vec{t_n}) \quad \rightsquigarrow \quad q(\vec{x}) \leftarrow P_1(\vec{t_1}), \ldots, P_n(\vec{t_n})$$

CQs:

$$q(\vec{x}) = \exists \vec{y}. P_1(\vec{t_1}) \wedge \cdots \wedge P_n(\vec{t_n}) \quad \rightsquigarrow \quad q(\vec{x}) \leftarrow P_1(\vec{t_1}), \ldots, P_n(\vec{t_n})$$

UCQs:

$$\begin{array}{rlcl} q(\vec{x}) = & \exists \vec{y_1}. P^1_1(\vec{t^1_1}) \wedge \cdots \wedge P^1_{n_1}(\vec{t^1_{n_1}}) & & q(\vec{x}) \leftarrow P^1_1(\vec{t^1_1}), \ldots, P^1_{n_1}(\vec{t^1_{n_1}}) \\ & \vee\ \exists \vec{y_2}. P^2_1(\vec{t^2_1}) \wedge \cdots \wedge P^2_{n_2}(\vec{t^2_{n_2}}) & & q(\vec{x}) \leftarrow P^2_1(\vec{t^2_1}), \ldots, P^2_n(\vec{t^2_n}) \\ & \vdots & \rightsquigarrow & \vdots \\ & \vee\ \exists \vec{y_\ell}. P^\ell_1(\vec{t^\ell_1}) \wedge \cdots \wedge P^\ell_{n_\ell}(\vec{t^\ell_{n_\ell}}) & & q(\vec{x}) \leftarrow P^\ell_1(\vec{t^\ell_1}), \ldots, P^\ell_{n_\ell}(\vec{t^\ell_{n_\ell}}) \end{array}$$

Recall that $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\vec{a} \in \mathsf{ans}(q, \mathcal{I})$ for every model $\mathcal{I}$ of $\mathcal{K}$

- A CQ $q(\vec{x})$ is an **FO formula**, its **satisfaction in an interpretation** is clear

$$\vec{a} \in \mathsf{ans}(q, \mathcal{I}) \text{ iff } \mathcal{I} \models q(\vec{x} \mapsto \vec{a})$$

Recall that $\vec{a} \in \mathrm{cert}(q, \mathcal{K})$ iff $\vec{a} \in \mathrm{ans}(q, \mathcal{I})$ for every model $\mathcal{I}$ of $\mathcal{K}$

- A CQ $q(\vec{x})$ is an **FO formula**, its **satisfaction in an interpretation** is clear

$$\vec{a} \in \mathrm{ans}(q, \mathcal{I}) \text{ iff } \mathcal{I} \models q(\vec{x} \mapsto \vec{a})$$

- We can also use the notion of a match

Recall that $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\vec{a} \in \mathsf{ans}(q, \mathcal{I})$ for every model $\mathcal{I}$ of $\mathcal{K}$

- A CQ $q(\vec{x})$ is an **FO formula**, its **satisfaction in an interpretation** is clear

$$\vec{a} \in \mathsf{ans}(q, \mathcal{I}) \text{ iff } \mathcal{I} \models q(\vec{x} \mapsto \vec{a})$$

- We can also use the notion of a **match**

  A **match** for $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$ in an **interpretation** $\mathcal{I}$ is a **mapping** π from the **variables** in $\vec{x} \cup \vec{y}$ **to objects** in $\Delta^{\mathcal{I}}$ such that:

  - $\pi(t) \in A^{\mathcal{I}}$ for every atom $A(t) \in q$
  - $\pi(t, t') \in r^{\mathcal{I}}$ for every atom $r(t, t') \in q$

Recall that $\vec{a} \in \text{cert}(q, \mathcal{K})$ iff $\vec{a} \in \text{ans}(q, \mathcal{I})$ for every model $\mathcal{I}$ of $\mathcal{K}$

- A CQ $q(\vec{x})$ is an **FO formula**, its **satisfaction in an interpretation** is clear

$$\vec{a} \in \text{ans}(q, \mathcal{I}) \text{ iff } \mathcal{I} \models q(\vec{x} \mapsto \vec{a})$$

- We can also use the notion of a **match**

  A **match** for $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$ in an **interpretation** $\mathcal{I}$ is a **mapping** π from the **variables** in $\vec{x} \cup \vec{y}$ **to objects** in $\Delta^{\mathcal{I}}$ such that:
  - $\pi(t) \in A^{\mathcal{I}}$ for every atom $A(t) \in q$
  - $\pi(t, t') \in r^{\mathcal{I}}$ for every atom $r(t, t') \in q$

We write $\mathcal{I} \models_{\pi} q(\vec{a})$ if π **is a match for** $q(\vec{x})$ **in** $\mathcal{I}$ and $\pi(\vec{x}) = \vec{a}$

$\vec{a} \in \mathrm{cert}(q, \mathcal{K})$

iff

for every model $\mathcal{I}$ of $\mathcal{K}$ we have $\vec{a} \in \mathrm{ans}(q, \mathcal{I})$

iff

for every model $\mathcal{I}$ of $\mathcal{K}$ there exists a **match** π such that $\mathcal{I} \models_\pi q(\vec{a})$

$\vec{a} \in \mathsf{cert}(q, \mathcal{K})$

iff

for every model $\mathcal{I}$ of $\mathcal{K}$ we have $\vec{a} \in \mathsf{ans}(q, \mathcal{I})$

iff

for every model $\mathcal{I}$ of $\mathcal{K}$ there exists a match π such that $\mathcal{I} \models_\pi q(\vec{a})$

Answering CQs = deciding if there is a match in every model

$$\vec{a} \in \mathsf{cert}(q, \mathcal{K})$$

iff

for every model $\mathcal{I}$ of $\mathcal{K}$ we have $\vec{a} \in \mathsf{ans}(q, \mathcal{I})$

iff

for every model $\mathcal{I}$ of $\mathcal{K}$ there exists a **match** π such that $\mathcal{I} \models_\pi q(\vec{a})$

Answering CQs = deciding if there is a **match in every model**

Challenge: how do we check that?

infinitely many models   models can be infinite

For Horn DLs, each satisfiable $\mathcal{K}$ has a universal model $\mathcal{I}_\mathcal{K}$

$\mathcal{I}_\mathcal{K}$ is **'contained'** in every model $\mathcal{I}$ of $\mathcal{K}$

$\rightsquigarrow$ formally, there is a homomorphism from $\mathcal{I}_\mathcal{K}$ to $\mathcal{I}$

For **Horn** DLs, each satisfiable $\mathcal{K}$ has a universal model $\mathcal{I}_{\mathcal{K}}$

$\mathcal{I}_{\mathcal{K}}$ is **'contained'** in every model $\mathcal{I}$ of $\mathcal{K}$

$\rightsquigarrow$ formally, there is a **homomorphism** from $\mathcal{I}_{\mathcal{K}}$ to $\mathcal{I}$

An **answer** to a (U)CQ q **in** $\mathcal{I}_{\mathcal{K}}$ is an **answer** to q in **every model of** $\mathcal{K}$

$\rightsquigarrow$ matches of (U)CQs are **preserved under homomorphisms**

$$\vec{a} \in \mathrm{cert}(q, \mathcal{K}) \text{ iff } \vec{a} \in \mathrm{ans}(q, \mathcal{I}_{\mathcal{K}})$$

So: $\mathcal{I}_{\mathcal{K}}$ **gives us the certain answers to** q **over** $\mathcal{K}$

For **Horn** DLs, each satisfiable $\mathcal{K}$ has a **universal model** $\mathcal{I}_\mathcal{K}$

$\mathcal{I}_\mathcal{K}$ is **'contained'** in every model $\mathcal{I}$ of $\mathcal{K}$

$\rightsquigarrow$ formally, there is a **homomorphism** from $\mathcal{I}_\mathcal{K}$ to $\mathcal{I}$

An **answer** to a (U)CQ q **in** $\mathcal{I}_\mathcal{K}$ is an **answer** to q in **every model of** $\mathcal{K}$

$\rightsquigarrow$ matches of (U)CQs are **preserved under homomorphisms**

$$\vec{a} \in \mathrm{cert}(q, \mathcal{K}) \text{ iff } \vec{a} \in \mathrm{ans}(q, \mathcal{I}_\mathcal{K})$$

So: $\mathcal{I}_\mathcal{K}$ **gives us the certain answers to** q **over** $\mathcal{K}$

Note: due to the **universal model property**, answering **UCQs** is **not harder** than answering **CQs** why?

Use the saturation of $(\mathcal{T}, \mathcal{A})$ for building a universal model $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

CONSTRUCTING A UNIVERSAL MODEL

Use the saturation of $(\mathcal{T}, \mathcal{A})$ for building a universal model $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

Intuition:
- $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ contains the saturated ABox $\mathcal{A}'$
- if an object satisfies M and $M \sqsubseteq \exists R.M' \in \mathrm{sat}(\mathcal{T})$, a fresh object witnessing this is created

Use only logically strongest inclusions in $\mathrm{sat}(\mathcal{T})$, denoted $\mathrm{sat}^{\mathrm{str}}(\mathcal{T})$

CONSTRUCTING A UNIVERSAL MODEL

Use the saturation of $(\mathcal{T}, \mathcal{A})$ for building a universal model $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

Intuition:
- $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ contains the saturated ABox $\mathcal{A}'$
- if an object satisfies M and $M \sqsubseteq \exists R.M' \in \mathsf{sat}(\mathcal{T})$, a fresh object witnessing this is created

Use only logically strongest inclusions in $\mathsf{sat}(\mathcal{T})$, denoted $\mathsf{sat}^{\mathsf{str}}(\mathcal{T})$

Formally, $\Delta^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ contains words

$$aR_1M_1 \dots R_nM_n$$

with $a \in \mathsf{Ind}(\mathcal{A})$ and:

- R_i are roles and M_i are conjunctions of concept names
- there exists $M \sqsubseteq \exists R_1.M_1 \in \mathsf{sat}^{\mathsf{str}}(\mathcal{T})$ such that $\mathcal{T}, \mathcal{A} \models M(a)$
- for every $1 \leq i < n$, exists $M_i' \sqsubseteq \exists R_{i+1}.M_{i+1} \in \mathsf{sat}^{\mathsf{str}}(\mathcal{T})$ with $M_i' \subseteq M_i$

Defining the interpretation function is straightforward:

- $a^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} = a$,
- $a \in A^{\mathcal{I}}$ iff $A(a) \in \mathsf{sat}(\mathcal{T}, \mathcal{A})$,
- $eRM \in A^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ iff $A \in M$,
- $(a, b) \in r^{\mathcal{I}}$ iff $r(a, b) \in \mathsf{sat}(\mathcal{T}, \mathcal{A})$,
- $(e, eRM) \in r^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ iff $R \sqsubseteq r \in \mathsf{sat}(\mathcal{T})$, and
- $(eRM, e) \in r^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ if $R \sqsubseteq r^{-} \in \mathsf{sat}(\mathcal{T})$

Remark: For readability, in the examples we use shorter names instead of the long words

EXAMPLE OF THE CANONICAL MODEL CONSTRUCTION (1/3)

TBox:

$$\begin{array}{rcl}
\mathsf{PenneArrab} & \sqsubseteq & \exists \mathsf{hasIngred}.\mathsf{Penne} \\
\mathsf{Penne} & \sqsubseteq & \mathsf{Pasta} \\
\mathsf{PenneArrab} & \sqsubseteq & \exists \mathsf{hasIngred}.\mathsf{ArrabSauce} \\
\mathsf{ArrabSauce} & \sqsubseteq & \exists \mathsf{hasIngred}.\mathsf{Peperonc} \\
\mathsf{Peperonc} & \sqsubseteq & \mathsf{Spicy} \\
\mathsf{PizzaCalab} & \sqsubseteq & \exists \mathsf{hasIngred}.\mathsf{Nduja} \\
\mathsf{Nduja} & \sqsubseteq & \mathsf{Spicy}
\end{array}$$

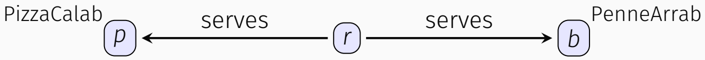
ABox: $\mathsf{serves}(r, b)$ $\mathsf{serves}(r, p)$ $\mathsf{PenneArrab}(b)$ $\mathsf{PizzaCalab}(p)$

The saturated TBox additionally contains:

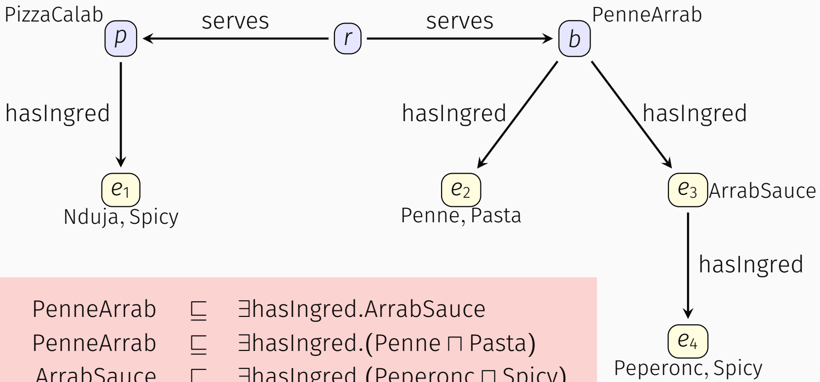
$$\begin{array}{rcl}
\mathsf{PenneArrab} & \sqsubseteq & \exists \mathsf{hasIngred}.(\mathsf{Penne} \sqcap \mathsf{Pasta}) \\
\mathsf{ArrabSauce} & \sqsubseteq & \exists \mathsf{hasIngred}.(\mathsf{Peperonc} \sqcap \mathsf{Spicy}) \\
\mathsf{PizzaCalab} & \sqsubseteq & \exists \mathsf{hasIngred}.(\mathsf{Nduja} \sqcap \mathsf{Spicy})
\end{array}$$

$\mathcal{I}_{\mathcal{T},\mathcal{A}}$ contains the ABox and is closed under inclusions

serves(r, b)	serves(r, p)	PenneArrab(b)	PizzaCalab(p)



The **anonymous objects** witnessing existential concepts form **trees**



$$
\begin{aligned}
\mathsf{PenneArrab} &\sqsubseteq \exists \mathsf{hasIngred}.\mathsf{ArrabSauce}\\
\mathsf{PenneArrab} &\sqsubseteq \exists \mathsf{hasIngred}.(\mathsf{Penne} \sqcap \mathsf{Pasta})\\
\mathsf{ArrabSauce} &\sqsubseteq \exists \mathsf{hasIngred}.(\mathsf{Peperonc} \sqcap \mathsf{Spicy})\\
\mathsf{PizzaCalab} &\sqsubseteq \exists \mathsf{hasIngred}.(\mathsf{Nduja} \sqcap \mathsf{Spicy})
\end{aligned}
$$

To answer CQ q, it suffices to test whether it has a match in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

But this is still challenging!
- $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ contains assertions and objects not present in $\mathcal{A}$
- we cannot build $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ explicitly: can be infinite!

To answer CQ q, it suffices to test whether it has a match in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

But this is still challenging!
- $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ contains assertions and objects not present in $\mathcal{A}$
- we cannot build $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ explicitly: can be infinite!

Our approach: use query rewriting!

To answer CQ q, it suffices to test whether it has a match in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$

But this is still challenging!
- $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ contains **assertions** and **objects** not present in $\mathcal{A}$
- we cannot build $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ explicitly: can be **infinite!**

Our approach: use **query rewriting**!

Formally: given a CQ q, we construct a UCQ $REW_{\mathcal{T}}(q)$ such that

$\vec{a} \in \mathrm{ans}(q, \mathcal{I}_{\mathcal{T},\mathcal{A}})$
iff
there is a **match** π for a disjunct q' of $\mathrm{rew}_{\mathcal{T}}(q)$ such that
$\mathcal{I}_{\mathcal{T},\mathcal{A}} \models_{\pi} q'(\vec{a})$ and π **sends all vars to individuals** from $\mathcal{A}$

Idea of the 1-step rewriting of q into q':

1. Choose **leaf variable** x so that no vars are mapped below it in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$
2. Find $M \sqsubseteq \exists R.N$ **in** $\mathsf{sat}(\mathcal{T})$ that ensures **all atoms with** x
3. **Drop** from q all atoms with x
4. **Merge all neighbours of** x and **add** M for the merged variable

REWRITING THE QUERY

Idea of the 1-step rewriting of q into q':

1. Choose **leaf variable** x so that no vars are mapped below it in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$
2. Find $M \sqsubseteq \exists R.N$ **in** $\mathsf{sat}(\mathcal{T})$ that ensures **all atoms with** x
3. **Drop** from q all atoms with x
4. **Merge all neighbours of** x and **add** M for the merged variable

Properties:

- Every **match** π' **for** q' can be easily modified into **match** π **for** q

REWRITING THE QUERY

Idea of the 1-step rewriting of q into q':

1. Choose **leaf variable** x so that no vars are mapped below it in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$
2. Find $M \sqsubseteq \exists R.N$ **in** sat($\mathcal{T}$) that ensures **all atoms with** x
3. **Drop** from q all atoms with x
4. **Merge all neighbours of** x and **add** M for the merged variable

Properties:

- Every **match** π' **for** q' can be easily modified into **match** π **for** q
- For each **match** π **for** q, there is some q' **produced by the rewriting step** and **a match** π' **for** q'

Idea of the 1-step rewriting of q into q':

1. Choose **leaf variable** x so that no vars are mapped below it in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$
2. Find $M \sqsubseteq \exists R.N$ **in** sat($\mathcal{T}$) that ensures **all atoms with** x
3. **Drop** from q all atoms with x
4. **Merge all neighbours of** x and **add** M for the merged variable

Properties:

- Every **match** π' **for** q' can be easily modified into **match** π **for** q
- For each **match** π **for** q, there is some q' **produced by the rewriting step** and **a match** π' **for** q'
- The matches π **and** π' **are essentially the same**, but π' matches x **closer to the ABox** than π

REWRITING THE QUERY

Idea of the 1-step rewriting of q into q':

1. Choose **leaf variable** x so that no vars are mapped below it in $\mathcal{I}_{\mathcal{T},\mathcal{A}}$
2. Find $M \sqsubseteq \exists R.N$ **in** sat($\mathcal{T}$) that ensures **all atoms with** x
3. **Drop** from q all atoms with x
4. **Merge all neighbours of** x and **add** M for the merged variable

Properties:

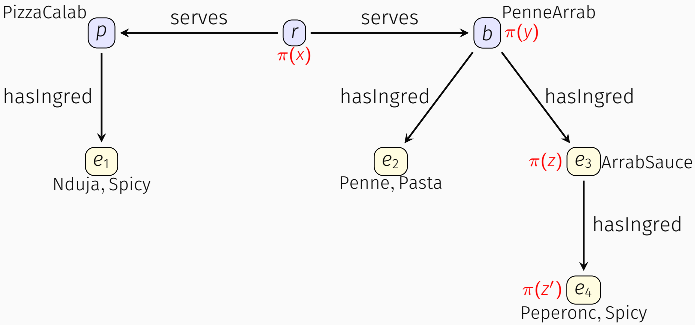
- Every **match** π' **for** q' can be easily modified into **match** π **for** q
- For each **match** π **for** q, there is some q' **produced by the rewriting step** and **a match** π' **for** q'
- The matches π **and** π' **are essentially the same**, but π' matches x **closer to the ABox** than π

We **repeatedly apply the rewriting step** to obtain a set of queries whose relevant matches **range over ABox individuals**

Example of a rewriting step (1/2)

$$q(y,x) = \exists z, z'.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$$

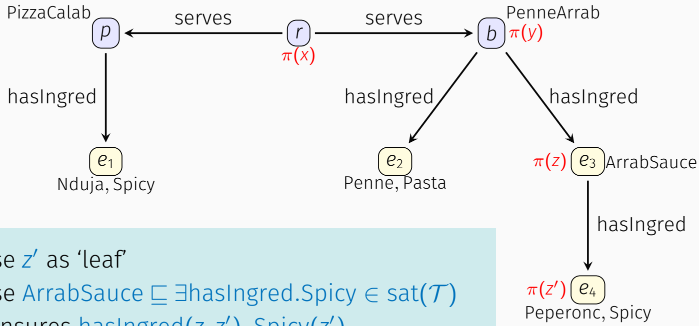
We have $\mathcal{I}_{\mathcal{K}} \models_{\pi} q(b,r)$ with $\pi(x) = r$, $\pi(y) = b$, $\pi(z) = e_3$, $\pi(z') = e_4$



EXAMPLE OF A REWRITING STEP (1/2)

$$q(y, x) = \exists z, z'.\mathsf{serves}(x, y) \wedge \mathsf{hasIngred}(y, z) \wedge \mathsf{hasIngred}(z, z') \wedge \mathsf{Spicy}(z')$$

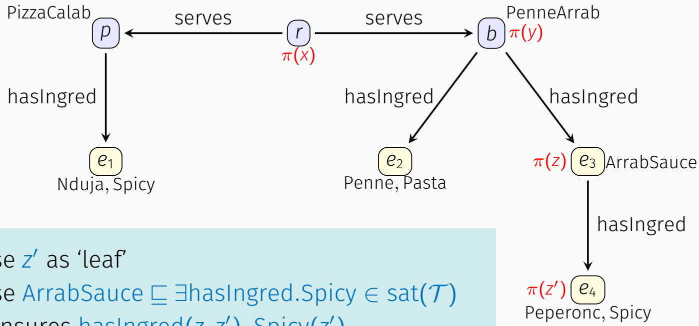
We have $\mathcal{I}_{\mathcal{K}} \models_{\pi} q(b, r)$ with $\pi(x) = r$, $\pi(y) = b$, $\pi(z) = e_3$, $\pi(z') = e_4$



- Choose z' as 'leaf'
- Choose $\mathsf{ArrabSauce} \sqsubseteq \exists \mathsf{hasIngred}.\mathsf{Spicy} \in \mathsf{sat}(\mathcal{T})$
- RHS ensures $\mathsf{hasIngred}(z, z')$, $\mathsf{Spicy}(z')$
- We replace these atoms by $\mathsf{ArrabSauce}(z)$

$$q(y,x) = \exists z, z'.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$$

We have $\mathcal{I}_{\mathcal{K}} \models_\pi q(b,r)$ with $\pi(x)=r$, $\pi(y)=b$, $\pi(z)=e_3$, $\pi(z')=e_4$



- Choose z' as 'leaf'
- Choose $\mathsf{ArrabSauce} \sqsubseteq \exists\mathsf{hasIngred}.\mathsf{Spicy} \in \mathsf{sat}(\mathcal{T})$
- RHS ensures $\mathsf{hasIngred}(z,z')$, $\mathsf{Spicy}(z')$
- We replace these atoms by $\mathsf{ArrabSauce}(z)$

$$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$$

EXAMPLE OF A REWRITING STEP (2/2)

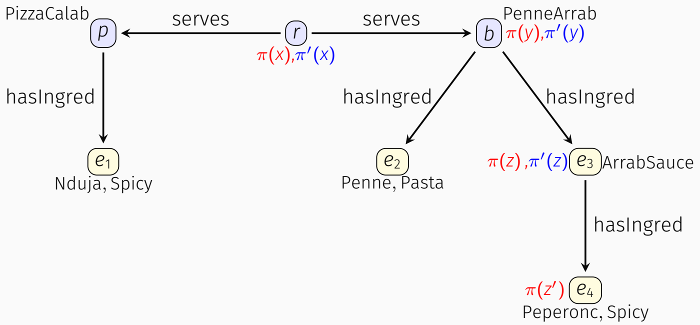
$q(y,x) = \exists z, z'.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$
$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$

EXAMPLE OF A REWRITING STEP (2/2)

$q(y,x) = \exists z,z'.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$
$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$

$\mathcal{I}_{\mathcal{K}} \models_{\pi} q(b,r)$

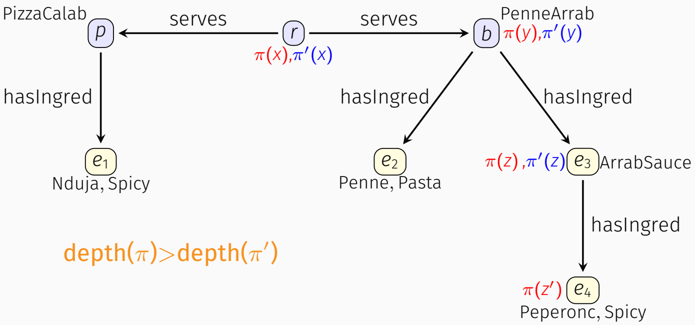
$\mathcal{I}_{\mathcal{K}} \models_{\pi'} q'(b,r)$



$q(y,x) = \exists z, z'.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$
$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$

$\mathcal{I}_{\mathcal{K}} \models_{\pi} q(b,r)$

$\mathcal{I}_{\mathcal{K}} \models_{\pi'} q'(b,r)$

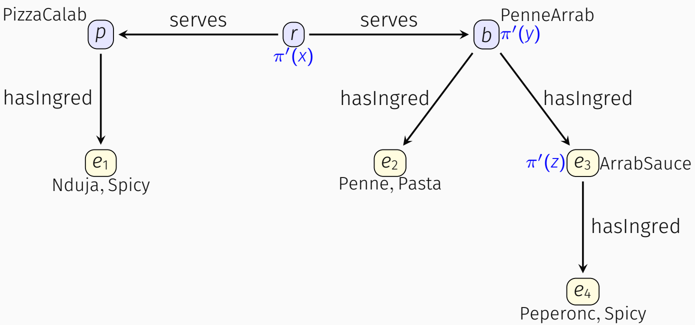


$\mathsf{depth}(\pi) > \mathsf{depth}(\pi')$

ANOTHER REWRITING STEP (1/2)

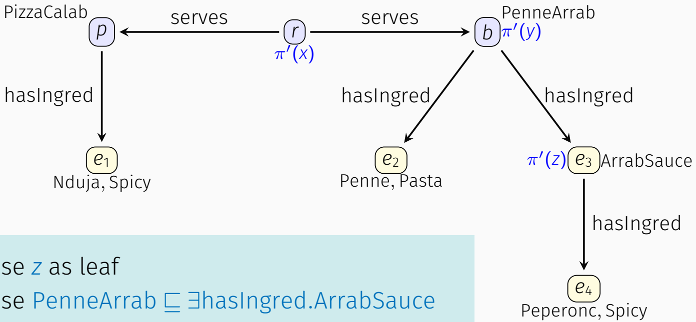
$q'(y, x) = \exists z.\mathsf{serves}(x, y) \land \mathsf{hasIngred}(y, z) \land \mathsf{ArrabSauce}(z)$

$\mathcal{I}_{\mathcal{K}} \models_{\pi'} q'(b, r)$



$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$

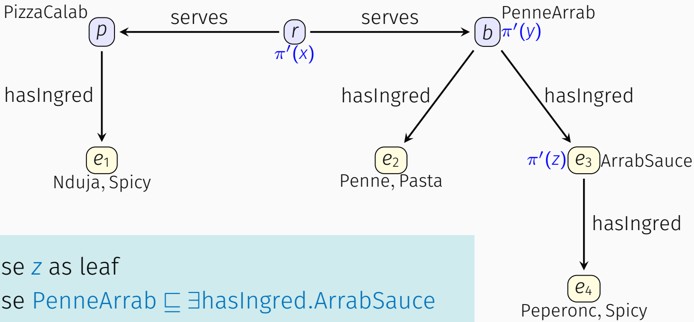
$\mathcal{I}_{\mathcal{K}} \models_{\pi'} q'(b,r)$



- Choose z as leaf
- Choose PenneArrab $\sqsubseteq \exists$hasIngred.ArrabSauce
- RHS yields hasIngred(y,z) and ArrabSauce(z)
- We replace these atoms by PenneArrab(y)

$$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$$

$$\mathcal{I}_{\mathcal{K}} \models_{\pi'} q'(b,r)$$



- Choose z as leaf
- Choose $\mathsf{PenneArrab} \sqsubseteq \exists\mathsf{hasIngred}.\mathsf{ArrabSauce}$
- RHS yields $\mathsf{hasIngred}(y,z)$ and $\mathsf{ArrabSauce}(z)$
- We replace these atoms by $\mathsf{PenneArrab}(y)$

$$q''(y,x) = \mathsf{serves}(x,y) \wedge \mathsf{PenneArrab}(y)$$

ANOTHER REWRITING STEP (2/2)

$q(y,x) = \exists z, z' \mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$
$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$
$q''(y,x) = \mathsf{serves}(x,y) \wedge \mathsf{PenneArrab}(y)$

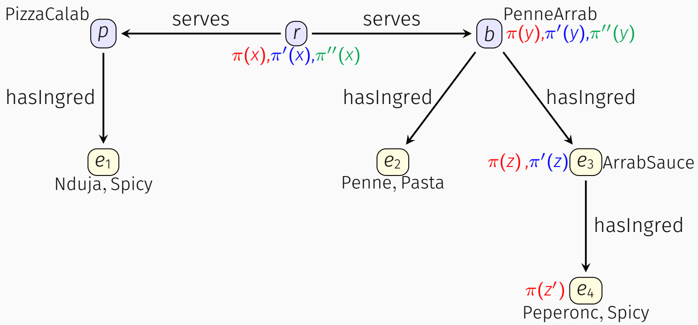
ANOTHER REWRITING STEP (2/2)

$q(y,x) = \exists z, z' \text{serves}(x,y) \land \text{hasIngred}(y,z) \land \text{hasIngred}(z,z') \land \text{Spicy}(z')$
$q'(y,x) = \exists z.\text{serves}(x,y) \land \text{hasIngred}(y,z) \land \text{ArrabSauce}(z)$
$q''(y,x) = \text{serves}(x,y) \land \text{PenneArrab}(y)$

$\mathcal{I}_\mathcal{K} \models_\pi q(b,r)$ $\mathcal{I}_\mathcal{K} \models_{\pi'} q'(b,r)$ $\mathcal{I}_\mathcal{K} \models_{\pi''} q''(b,r)$

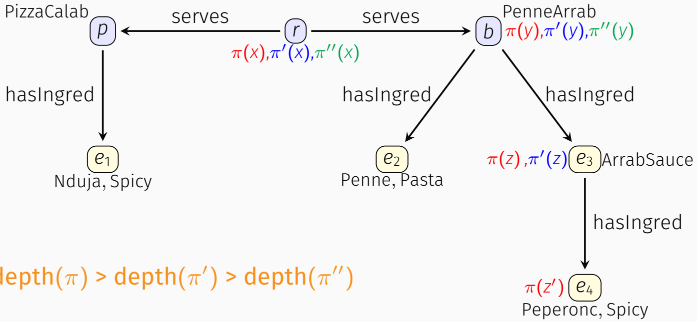
$q(y,x) = \exists z, z' \mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$
$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$
$q''(y,x) = \mathsf{serves}(x,y) \wedge \mathsf{PenneArrab}(y)$

$\mathcal{I}_\mathcal{K} \models_\pi q(b,r)$ $\mathcal{I}_\mathcal{K} \models_{\pi'} q'(b,r)$ $\mathcal{I}_\mathcal{K} \models_{\pi''} q''(b,r)$



$q(y,x) = \exists z, z' \mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{hasIngred}(z,z') \wedge \mathsf{Spicy}(z')$
$q'(y,x) = \exists z.\mathsf{serves}(x,y) \wedge \mathsf{hasIngred}(y,z) \wedge \mathsf{ArrabSauce}(z)$
$q''(y,x) = \mathsf{serves}(x,y) \wedge \mathsf{PenneArrab}(y)$

$\mathcal{I}_\mathcal{K} \models_\pi q(b,r)$ | $\mathcal{I}_\mathcal{K} \models_{\pi'} q'(b,r)$ | $\mathcal{I}_\mathcal{K} \models_{\pi''} q''(b,r)$

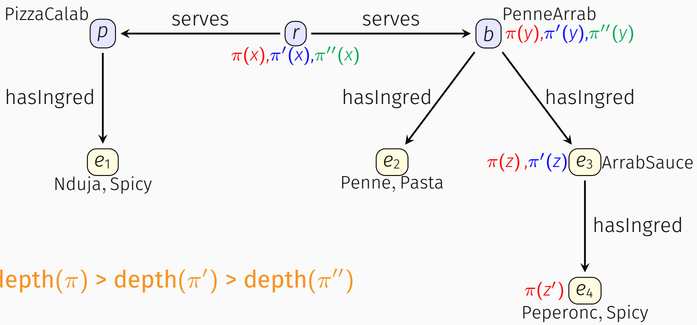


$\mathsf{depth}(\pi) > \mathsf{depth}(\pi') > \mathsf{depth}(\pi'')$

ANOTHER REWRITING STEP (2/2)

$q(y,x) = \exists z, z' \text{serves}(x,y) \wedge \text{hasIngred}(y,z) \wedge \text{hasIngred}(z,z') \wedge \text{Spicy}(z')$
$q'(y,x) = \exists z.\text{serves}(x,y) \wedge \text{hasIngred}(y,z) \wedge \text{ArrabSauce}(z)$
$q''(y,x) = \text{serves}(x,y) \wedge \text{PenneArrab}(y)$

$\mathcal{I}_{\mathcal{K}} \models_{\pi} q(b,r)$ $\mathcal{I}_{\mathcal{K}} \models_{\pi'} q'(b,r)$ $\mathcal{I}_{\mathcal{K}} \models_{\pi''} q''(b,r)$



$\text{depth}(\pi) > \text{depth}(\pi') > \text{depth}(\pi'')$

In π'' all variables are mapped to individuals

Theorem For every satisfiable $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and CQ $q(\vec{x})$: $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\mathcal{I}_\mathcal{K} \models_\pi q'(\vec{a})$ for some $q' \in \mathsf{rew}_\mathcal{T}(q)$ and some π that maps all variables to individuals in $\mathcal{A}$.

Theorem For every satisfiable $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and CQ $q(\vec{x})$: $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\mathcal{I}_\mathcal{K} \models_\pi q'(\vec{a})$ for some $q' \in \mathsf{rew}_\mathcal{T}(q)$ and some π that maps all variables to individuals in $\mathcal{A}$.

There is a bounded number of such restricted matches π

Checking if π is match reduces to linearly many instance checks

Theorem For every satisfiable $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and CQ $q(\vec{x})$: $\vec{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\mathcal{I}_\mathcal{K} \models_\pi q'(\vec{a})$ for some $q' \in \mathsf{rew}_\mathcal{T}(q)$ and some π that maps all variables to individuals in $\mathcal{A}$.

There is a bounded number of such restricted matches π

Checking if π is match reduces to linearly many instance checks

Yields terminating, sound, and complete CQ answering procedure

Combined complexity:

sat($\mathcal{T}$) and rew$_{\mathcal{T}}(q)$ can be constructed in single exponential time

single exponential bound on candidate matches π

$\rightsquigarrow$ instance checking in single exponential time

Combined complexity:

$\mathsf{sat}(\mathcal{T})$ and $\mathsf{rew}_{\mathcal{T}}(q)$ can be constructed in single exponential time

single exponential bound on candidate matches π

$\rightsquigarrow$ instance checking in single exponential time

Data complexity:

$\mathsf{sat}(\mathcal{T})$ and $\mathsf{rew}_{\mathcal{T}}(q)$ are ABox independent

polynomial bound on candidate matches π

$\rightsquigarrow$ instance checking in polynomial time

Combined complexity:

sat($\mathcal{T}$) and rew$_{\mathcal{T}}(q)$ can be constructed in single exponential time

single exponential bound on candidate matches π

$\rightsquigarrow$ instance checking in single exponential time

Data complexity:

sat($\mathcal{T}$) and rew$_{\mathcal{T}}(q)$ are ABox independent

polynomial bound on candidate matches π

$\rightsquigarrow$ instance checking in polynomial time

Theorem CQ answering in $\mathcal{ELHI}_{\bot}$ and Horn-$\mathcal{SHIQ}$ is Exp-complete in combined complexity and P-complete in data complexity.

OPTIMAL BOUNDS FOR LIGHTWEIGHT DLS

Adapting our technique gives optimal bounds for lightweight DLs:

For $\mathcal{ELH}$ and DL-Lite$_{\mathcal{R}}$ we get **NP** in **combined complexity**:

- compute sat($\mathcal{T}$) in polynomial time
- non-deterministically build the right $q' \in \text{rew}_{\mathcal{T}}(q)$
- guess a candidate π
- check if it is a match $\rightsquigarrow$ instance checking in polynomial time

CQ answering is **NP-hard** over ABox alone seen as DB (no TBox)

Adapting our technique gives optimal bounds for lightweight DLs:

For $\mathcal{ELH}$ and DL-Lite$_{\mathcal{R}}$ we get **NP** in **combined complexity**:

- compute sat($\mathcal{T}$) in polynomial time
- non-deterministically build the right $q' \in \mathrm{rew}_{\mathcal{T}}(q)$
- guess a candidate π
- check if it is a match $\rightsquigarrow$ instance checking in polynomial time

CQ answering is **NP-hard** over ABox alone seen as DB (no TBox)

For $\mathcal{EL}$ in **data complexity**, yields **P** membership
$\rightsquigarrow$ optimal since instance queries already **P-hard**

Adapting our technique gives optimal bounds for lightweight DLs:

For $\mathcal{ELH}$ and DL-Lite$_{\mathcal{R}}$ we get **NP** in **combined complexity**:

- compute sat($\mathcal{T}$) in polynomial time
- non-deterministically build the right $q' \in \text{rew}_{\mathcal{T}}(q)$
- guess a candidate π
- check if it is a match $\rightsquigarrow$ instance checking in polynomial time

CQ answering is **NP-hard** over ABox alone seen as DB (no TBox)

For $\mathcal{EL}$ in **data complexity**, yields P membership
$\rightsquigarrow$ optimal since instance queries already **P-hard**

In DL-Lite$_{\mathcal{R}}$, we get a FO-rewriting (later) $\rightsquigarrow$ **in** AC_0 for data compl.

OPTIMAL BOUNDS FOR LIGHTWEIGHT DLS

Adapting our technique gives optimal bounds for lightweight DLs:

For $\mathcal{ELH}$ and DL-Lite$_{\mathcal{R}}$ we get **NP** in **combined complexity**:

- compute sat($\mathcal{T}$) in polynomial time
- non-deterministically build the right $q' \in \text{rew}_{\mathcal{T}}(q)$
- guess a candidate π
- check if it is a match $\rightsquigarrow$ instance checking in polynomial time

CQ answering is **NP-hard** over ABox alone seen as DB (no TBox)

For $\mathcal{EL}$ in **data complexity**, yields **P** membership

$\rightsquigarrow$ optimal since instance queries already **P-hard**

In DL-Lite$_{\mathcal{R}}$, we get a FO-rewriting (later) $\rightsquigarrow$ **in** AC_0 for data compl.

Theorem CQ answering in $\mathcal{ELH}$ and DL-Lite$_{\mathcal{R}}$ is NP-complete in combined complexity. For $\mathcal{ELH}$ the data complexity is P-complete, and for **DL-Lite$_{\mathcal{R}}$** the data complexity is in AC_0.

Our procedure yields a Datalog rewriting:

- $\mathrm{rew}_{\mathcal{T}}(q)$ is a UCQ $\rightsquigarrow$ translate into set of **Datalog rules** $\Pi_{\mathrm{rew}(q)}$
  - use Q in head of rules
- the program $\Pi(\mathcal{T}, \Sigma)$ (from earlier) computes all **entailed ABox assertions**

Our procedure yields a Datalog rewriting:

- $\mathsf{rew}_{\mathcal{T}}(q)$ is a UCQ $\rightsquigarrow$ translate into set of **Datalog rules** $\Pi_{\mathsf{rew}(q)}$
  - use Q in head of rules
- the program $\Pi(\mathcal{T},\Sigma)$ (from earlier) computes all **entailed ABox assertions**

$$(\Pi_{\mathsf{rew}(q)} \cup \Pi(\mathcal{T},\Sigma), Q)$$

is a **Datalog rewriting** of q w.r.t. $\mathcal{T}$ relative to consistent Σ-ABoxes

Alternatively, view as a combined approach: saturation + rewriting

Alternatively, view as a **combined approach: saturation + rewriting**

Know that it suffices to **evaluate the UCQ** $\mathrm{rew}_{\mathcal{T}}(q)$ over the set of **ABox assertions entailed** from the KB $\mathcal{K}$

Alternatively, view as a **combined approach: saturation + rewriting**

Know that it suffices to **evaluate the UCQ** $\text{rew}_{\mathcal{T}}(q)$ over the set of **ABox assertions entailed** from the KB $\mathcal{K}$

Also know: assertions entailed from $\mathcal{K}$ = assertions in $\text{sat}(\mathcal{K})$

Alternatively, view as a **combined approach: saturation + rewriting**

Know that it suffices to **evaluate the UCQ** $\mathrm{rew}_{\mathcal{T}}(q)$ over the set of **ABox assertions entailed** from the KB $\mathcal{K}$

Also know: assertions entailed from $\mathcal{K}$ = assertions in $\mathrm{sat}(\mathcal{K})$

Materialize assertions in $\mathrm{sat}(\mathcal{K})$ and view result as **database**

+ only need to evaluate a UCQ
  can use standard relational database systems
- materializing not always convenient
  saturation needs to be updated if data changes

AN FO REWRITING APPROACH FOR CQS IN DL-LITE

For DL-Lite$_{\mathcal{R}}$ we can generate an FO-rewriting as follows.

Replace in all $q' \in \mathrm{rew}_{\mathcal{T}}(q)$ each atom by its FO-rewriting for instance checking:

For DL-Lite$_{\mathcal{R}}$ we can generate an FO-rewriting as follows.

Replace in all $q' \in \mathrm{rew}_{\mathcal{T}}(q)$ each atom by its FO-rewriting for instance checking:

- replace each $A(t)$ by $\mathrm{RewriteIQ}(A, \mathcal{T})$
- replace each $r(t, t')$ by $\mathrm{RewriteIQ}(r, \mathcal{T})$

For DL-Lite$_\mathcal{R}$ we can generate an FO-rewriting as follows.

Replace in all $q' \in \mathsf{rew}_\mathcal{T}(q)$ each atom by its FO-rewriting for instance checking:

- replace each $A(t)$ by $\mathsf{RewriteIQ}(A, \mathcal{T})$
- replace each $r(t, t')$ by $\mathsf{RewriteIQ}(r, \mathcal{T})$

Resulting FO formula:

- positive, can be transformed into a UCQ
- it is a rewriting of q and $\mathcal{T}$ (relative to consistent ABoxes)
- yields AC_0 upper bound in data complexity

- Similar results hold for other dialects of DL-Lite and $\mathcal{EL}$
- Also for more expressive Horn DLs, like Horn-$\mathcal{SHOIQ}$
- For answering CQs and UCQs in Horn DLs, usually we have:
  - Data complexity is in P
  - The combined complexity is either:
    - NP-complete for tractable DLs
    - the same as for instance queries in richer Horn DLs
- With complex role inclusions the complexity increases
  - CQs undecidable for $\mathcal{EL}++$
  - If suitably restricted, PSPACE-complete

UNDECIDABILITY OF CQ ANSWERING WITH COMPLEX ROLE INCLUSIONS

We can reduce emptiness of the intersection of two CF languages to CQ answering in $\mathcal{EL}$ (or DL-Lite) with complex role inclusions

$$r_1 \circ \cdots \circ r_n \sqsubseteq s$$

Given two CFGs (the non-terminals N_1 and N_2 are disjoint)

$$\mathsf{G_i} = (\mathsf{N_i}, \mathsf{T}, \mathsf{P_i}, \mathsf{S_i}) \qquad i \in \{1, 2\}$$

We define a TBox

$$\mathcal{T} = \{\top \sqsubseteq \exists r_t.\top \mid t \in T\} \cup \{r_{A_1} \circ \cdots \circ r_{A_n} \sqsubseteq r_A \mid A \to A_1 \cdots A_n \in P_1 \cup P_2\}$$

Then

$$\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset \quad \text{iff} \quad \mathcal{T}, \{A(c)\} \models \exists x.S_1(c, x) \wedge S_2(c, x)$$

- No universal model property
- CQ answering usually exponentially harder than instance queries
  - exponential blow-up in the size of the query
- Different techniques:
  - Automata on infinite trees
  - Reductions to satisfiability using treefications and rolling-up
  - Resolution, decompositions, typ/knot elimination, etc.
- Often best-case exponential, implementations still not in sight
- Usually bounds for UCQs and CQs coincide, and even for positive existential queries
- For the well-known $\mathcal{SHOIQ}$ decidability and complexity elusive

	IQs		CQs, UCQs	
	data complexity	combined complexity	data complexity	combined complexity
DL-Lite DL-Lite$_{\mathcal{R}}$	in AC_0	NLOGSPACE	in AC_0	NP
$\mathcal{EL}, \mathcal{ELH}$	P	P	P	NP
$\mathcal{ELI}, \mathcal{ELHI}_{\bot}$, Horn-$\mathcal{SHOIQ}$	P	EXP	P	EXP
$\mathcal{ALC}$, $\mathcal{ALCHQ}$	coNP	EXP	coNP	EXP
$\mathcal{ALCI}, \mathcal{SH}$, $\mathcal{SHIQ}$	coNP	EXP	coNP	2EXP
$\mathcal{SHOIQ}$	coNP	coNEXP	coNP-hard[1]	coN2EXP-hard[1]

[1] decidability open