QUERY ANSWERING WITH DESCRIPTION LOGIC ONTOLOGIES

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

NAVIGATIONAL QUERIES

Some very natural queries are not expressible as CQs:

- find dishes that contain something spicy
- is a a relative of b?
- is there a bus connection from x to y?

LIMITATIONS OF (U)CQS

Some very natural queries are not expressible as CQs:

- find dishes that contain something spicy
- is a a relative of b?
- is there a bus connection from x to y?

We need navigational queries that can flexibly explore our data (node- & edge-) labeled graphs

=

ABoxes

=

relational databases with unary and binary predicates only (node- & edge-) labeled graphs

=

ABoxes

=

relational databases with unary and binary predicates only

We are dealing with graph-structured data

- important in the database community
- can capture highly connected data with no fixed schema
- social, biological, chemical networks, pointer structures …

Regular Path Queries (RPQs): find pairs of objects that are connected by a chain of roles that comply with a given regular language

$$(\mathsf{hasCourse} \cup \mathsf{courseOf}^{-}) \cdot (\mathsf{hasIngred} \cup \mathsf{ingredOf}^{-})^{*} \cdot \mathsf{Spicy?}(x, y)$$

Regular Path Queries (RPQs): find pairs of objects that are connected by a chain of roles that comply with a given regular language

$$(\mathsf{hasCourse} \cup \mathsf{courseOf}^-) \cdot (\mathsf{hasIngred} \cup \mathsf{ingredOf}^-)^* \cdot \mathsf{Spicy?}(x, y)$$

Conjunctive RPQs: allow to join RPQs conjunctively

- similar to CQs, but each atom is an RPQ
- extend CQs with the navigational power of RPQs

$$\begin{aligned} q(x, x') = \exists y, z. \quad & \mathsf{serves} \cdot \mathsf{Menu?} \cdot (\mathsf{hasMain} \cup \mathsf{hasStarter})(x, y) \land \\ & \mathsf{serves} \cdot \mathsf{Menu?} \cdot (\mathsf{hasCourse} \cup \mathsf{courseOf}^-)(x', y) \land \\ & (\mathsf{hasIngred} \cup \mathsf{ingredOf}^-)^* \cdot \mathsf{Spicy?}(y, z) \end{aligned}$$

Both languages have 1-way and 2-way variants

OUR MOST EXPRESSIVE NAVIGATIONAL QUERIES: C2RPQS

Recall: $\mathsf{N}_\mathsf{R}^\pm$ contains all role names and their inverses.

A **conjunctive two-way regular path query (C2RPQ)** has the form

$$q(\vec{x}) = \exists \vec{y}. \bigwedge L(t, t') \wedge \bigwedge A(t)$$

where A is a concept name

t, t′ are variables or individuals (in $\mathsf{N}_\mathsf{I} \cup \vec{x} \cup \vec{y}$)

L is **regular language** over $\mathsf{N}_\mathsf{R}^\pm \cup \{A? \mid A \in \mathsf{N}_\mathsf{C}\}$

Recall: $\mathsf{N}_\mathsf{R}^{\pm}$ contains all role names and their inverses.

A **conjunctive two-way regular path query (C2RPQ)** has the form

$$q(\vec{x}) = \exists \vec{y}. \bigwedge L(t, t') \wedge \bigwedge A(t)$$

where A is a concept name
t, t′ are variables or individuals (in $\mathsf{N}_\mathsf{I} \cup \vec{x} \cup \vec{y}$)
L is **regular language** over $\mathsf{N}_\mathsf{R}^{\pm} \cup \{A? \mid A \in \mathsf{N}_\mathsf{C}\}$

Regular languages can be given as:

- regular expressions $\mathcal{E} \to r \in \mathsf{N}_\mathsf{R}^{\pm} \mid A? \mid \mathcal{E} \cdot \mathcal{E} \mid \mathcal{E} \cup \mathcal{E} \mid \mathcal{E}^*$
- non-deterministic finite automata NFA

Note: RegExps and NFAs are equivalent, but NFAs are more succinct

Conjunctive (one-way) regular path queries (CRPQs) disallow inverses
⇝ regular expressions use only (direct) role names

$$
\begin{aligned}
q(x, x') = \quad & \exists y, z.\mathsf{serves} \cdot \mathsf{Menu?hasCourse}(x, y) \wedge \\
& \mathsf{serves} \cdot \mathsf{Menu?} \cdot \mathsf{hasCourse}(x', y) \wedge \mathsf{hasIngred}^* \cdot \mathsf{Spicy?}(y, z) \\
q(x) = \quad & \exists y.\mathsf{hasIngred}^* \cdot \mathsf{Spicy?}(x, y)
\end{aligned}
$$

OTHER NAVIGATIONAL QUERY LANGUAGES

Conjunctive (one-way) regular path queries (CRPQs) disallow inverses
$\rightsquigarrow$ regular expressions use only (direct) role names

$$\begin{aligned} q(x, x') = \quad & \exists y, z.\mathsf{serves} \cdot \mathsf{Menu?hasCourse}(x, y) \land \\ & \mathsf{serves} \cdot \mathsf{Menu?} \cdot \mathsf{hasCourse}(x', y) \land \mathsf{hasIngred}^* \cdot \mathsf{Spicy?}(y, z) \\ q(x) = \quad & \exists y.\mathsf{hasIngred}^* \cdot \mathsf{Spicy?}(x, y) \end{aligned}$$

Two-way regular path queries (2RPQs) have only one atom and no existential variables $\rightsquigarrow$ both variables are answer variables

$$\begin{aligned} q(x, y) \quad & = (\mathsf{hasIngred} \cup \mathsf{ingredOf}^-)^* \cdot \mathsf{Spicy?}(x, y) \\ q(x, y) \quad & = (\mathsf{hasIngred} \cup \mathsf{ingredOf}^-)^* \cdot \mathsf{Spicy?} \cdot \Sigma^*(x, y) \end{aligned}$$

Conjunctive (one-way) regular path queries (CRPQs) disallow inverses
$\rightsquigarrow$ regular expressions use only (direct) role names

$$\begin{aligned} q(x, x') = \quad & \exists y, z.\text{serves} \cdot \text{Menu?hasCourse}(x, y) \land \\ & \text{serves} \cdot \text{Menu?} \cdot \text{hasCourse}(x', y) \land \text{hasIngred}^* \cdot \text{Spicy?}(y, z) \\ q(x) = \quad & \exists y.\text{hasIngred}^* \cdot \text{Spicy?}(x, y) \end{aligned}$$

Two-way regular path queries (2RPQs) have only one atom and no existential variables $\rightsquigarrow$ both variables are answer variables

$$\begin{aligned} q(x, y) &= (\text{hasIngred} \cup \text{ingredOf}^-)^* \cdot \text{Spicy?}(x, y) \\ q(x, y) &= (\text{hasIngred} \cup \text{ingredOf}^-)^* \cdot \text{Spicy?} \cdot \Sigma^*(x, y) \end{aligned}$$

(One-way) Regular path queries (RPQs) are 2RPQs with no inverses
$\rightsquigarrow$ all of the restrictions above

$$\begin{aligned} q(x, y) &= \text{hasIngred}^* \cdot \text{Spicy?}(x, y) \\ q(x, y) &= \text{hasCourse} \cdot \text{hasIngred}^* \cdot \text{Spicy?}(x, y) \end{aligned}$$

SEMANTICS OF C2RPQS

Satisfaction of atoms $L(t, t')$:

$(d, d') \in L^{\mathcal{I}}$ if there is an L-**path from** d **to** d', i.e.,

- a **sequence** $e_0 e_1 \ldots e_n$ objects from $\Delta^{\mathcal{I}}$ with $e_0 = d$ and $e_n = d'$
- a **word** $u_1 u_2 \ldots u_n \in L$ over $N_R^{\pm} \cup \{A? \mid A \in N_C\}$

such that, for every $1 \leq i \leq n$:

- if $u_i = A?$, then $e_{i-1} = e_i \in A^{\mathcal{I}}$
- if $u_i = R \in N_R^{\pm}$, then $(e_{i-1}, e_i) \in R^{\mathcal{I}}$

Satisfaction of atoms $L(t, t')$:

$(d, d') \in L^{\mathcal{I}}$ if there is an L-**path from** d **to** d', i.e.,

- a **sequence** $e_0 e_1 \dots e_n$ objects from $\Delta^{\mathcal{I}}$ with $e_0 = d$ and $e_n = d'$
- a **word** $u_1 u_2 \dots u_n \in L$ over $N_R^{\pm} \cup \{A? \mid A \in N_C\}$

such that, for every $1 \leq i \leq n$:

- if $u_i = A?$, then $e_{i-1} = e_i \in A^{\mathcal{I}}$
- if $u_i = R \in N_R^{\pm}$, then $(e_{i-1}, e_i) \in R^{\mathcal{I}}$

Match: mapping π from terms to elements that satisfies all atoms

As before: $\mathcal{I} \models_{\pi} q(\vec{a})$ if match π maps answer variables to $\vec{a}$

Satisfaction of atoms $L(t, t')$:

$(d, d') \in L^{\mathcal{I}}$ if there is an L-**path from** d **to** d', i.e.,

- a **sequence** $e_0 e_1 \ldots e_n$ objects from $\Delta^{\mathcal{I}}$ with $e_0 = d$ and $e_n = d'$
- a **word** $u_1 u_2 \ldots u_n \in L$ over $N_R^{\pm} \cup \{A? \mid A \in N_C\}$

such that, for every $1 \leq i \leq n$:

- if $u_i = A?$, then $e_{i-1} = e_i \in A^{\mathcal{I}}$
- if $u_i = R \in N_R^{\pm}$, then $(e_{i-1}, e_i) \in R^{\mathcal{I}}$

Match: mapping π from terms to elements that satisfies all atoms

As before: $\mathcal{I} \models_{\pi} q(\vec{a})$ if match π maps answer variables to $\vec{a}$

Certain answers defined as for CQs

Again suffices to **find match in universal model**

We focus on answering 2RPQs: one atom, no existential variables

We focus on answering 2RPQs: one atom, no existential variables

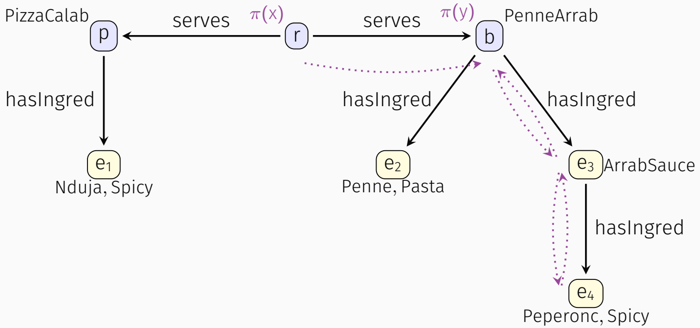
Bound on matches ranging over individuals only

We focus on answering 2RPQs: one atom, no existential variables

Bound on matches ranging over individuals only

Challenge: paths may need to go deep into the universal model

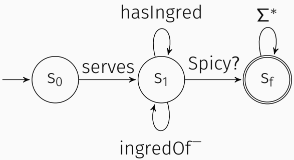
$$q(x,y) = \text{serves} \cdot (\text{hasIngred} \cup \text{ingredOf}^-)^* \cdot \text{Spicy?} \cdot \Sigma^*(x,y)$$



Goal: compact representation of all ways in which paths through the anonymous part can participate in matches

Goal: compact representation of **all ways** in which **paths through the anonymous part** can participate in **matches**

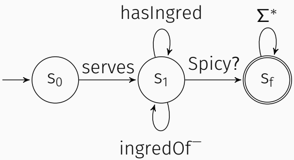
We use **NFA representation**



We write $M \in \text{Loop}_\alpha[s, s']$ iff $a \in M^{\mathcal{I}_\mathcal{K}}$ implies the existence of **a path** p **below** a that takes the **NFA** α **from** s **to** s', e.g.,

Goal: compact representation of **all ways** in which **paths through the anonymous part** can participate in **matches**

We use **NFA representation**



We write $M \in \mathsf{Loop}_\alpha[s, s']$ iff $a \in M^{\mathcal{I}_\mathcal{K}}$ implies the existence of **a path** p **below** a that takes the **NFA** α **from** s **to** s', e.g.,

$$\mathsf{PenneArrab} \in \mathsf{Loop}_\alpha[s_1, s_f]$$

because of

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

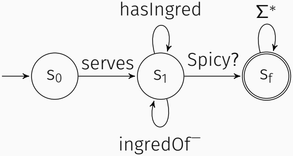
We can **explicitly compute** the **full** Loop_α **table** inductively:

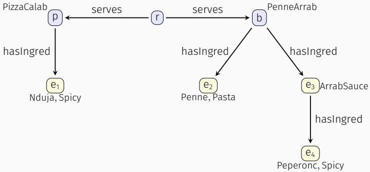
COMPUTING THE LOOP TABLE

We can **explicitly compute** the **full** Loop_α **table** inductively:

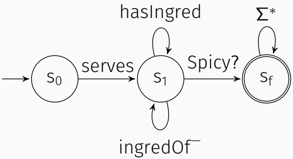
if s is a state, and $A \in N_C$	then $A \in \text{Loop}_\alpha[s, s]$
if $M_1 \in \text{Loop}_\alpha[s_1, s_2]$ and $M_2 \in \text{Loop}_\alpha[s_2, s_3]$	then $M_1 \sqcap M_2 \in \text{Loop}_\alpha[s_1, s_3]$
if $\mathcal{T} \models C_1 \sqcap \cdots \sqcap C_n \sqsubseteq A$ and $(s_1, A?, s_2) \in \delta$	then $C_1 \sqcap \cdots \sqcap C_n \in \text{Loop}_\alpha[s_1, s_2]$
if $\mathcal{T} \models C_1 \sqcap \cdots \sqcap C_n \sqsubseteq \exists R.D$, $\mathcal{T} \models R \sqsubseteq R'$, $\mathcal{T} \models R \sqsubseteq R''$, $(s_1, R', s_2) \in \delta$, $D \in \text{Loop}_\alpha[s_2, s_3]$, and $(s_3, R''^-, s_4) \in \delta$	then $C_1 \sqcap \cdots \sqcap C_n \in \text{Loop}_\alpha[s_1, s_4]$

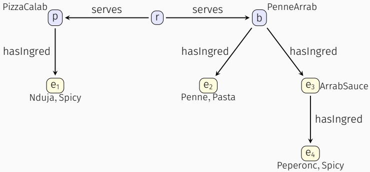
COMPUTING LOOPS: AN EXAMPLE





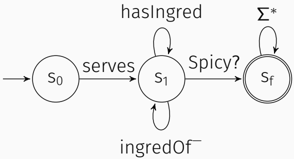
COMPUTING LOOPS: AN EXAMPLE

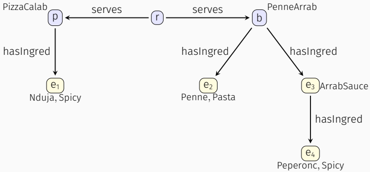




- $\text{Peperonc} \in \text{Loop}_\alpha[s_1, s_f]$ because $(s_1, \text{Spicy?}, s_f) \in \delta$ and

$$\text{Peperonc} \sqsubseteq \text{Spicy}$$





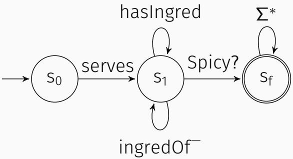
- $\mathsf{Peperonc} \in \mathsf{Loop}_\alpha[s_1, s_f]$ because $(s_1, \mathsf{Spicy?}, s_f) \in \delta$ and

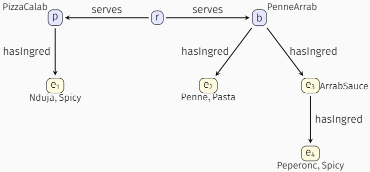
$$\mathsf{Peperonc} \sqsubseteq \mathsf{Spicy}$$

- $\mathsf{ArrabSauce} \in \mathsf{Loop}_\alpha[s_1, s_f]$ because $(s_1, \mathsf{hasIngred}, s_1), (s_f, \mathsf{hasIngred}^-, s_f) \in \delta$ and

$$\mathsf{ArrabSauce} \sqsubseteq \exists \mathsf{hasIngred}.\mathsf{Peperonc}$$
$$\mathsf{Peperonc} \in \mathsf{Loop}_\alpha[s_1, s_f]$$

COMPUTING LOOPS: AN EXAMPLE





- Peperonc $\in \text{Loop}_\alpha[s_1, s_f]$ because $(s_1, \text{Spicy?}, s_f) \in \delta$ and

  Peperonc $\sqsubseteq$ Spicy

- ArrabSauce $\in \text{Loop}_\alpha[s_1, s_f]$ because $(s_1, \text{hasIngred}, s_1), (s_f, \text{hasIngred}^-, s_f) \in \delta$ and

  ArrabSauce $\sqsubseteq \exists$hasIngred.Peperonc
  Peperonc $\in \text{Loop}_\alpha[s_1, s_f]$

- PenneArrab $\in \text{Loop}_\alpha[s_1, s_f]$ because $(s_1, \text{hasIngred}, s_1), (s_f, \text{hasIngred}^-, s_f) \in \delta$ and

  PenneArrab $\sqsubseteq \exists$hasIngred.ArrabSauce
  ArrabSauce $\in \text{Loop}_\alpha[s_1, s_f]$

Non-deterministic algorithm to decide $(a, b) \in cert(\alpha(x, y), \mathcal{K})$

Input: NFA $\alpha = (S, \Sigma, \delta, s_0, F)$, KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, (a, b) from $\mathcal{A}$

Non-deterministic algorithm to decide $(a, b) \in cert(\alpha(x, y), \mathcal{K})$

Input: NFA $\alpha = (S, \Sigma, \delta, s_0, F)$, KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, (a, b) from $\mathcal{A}$

- After checking consistency, we **start from** (a, s_0)
- At pair (c, s), **guess new pair** (d, s') together with one of:
  - **transition** (s, σ, s') — take a σ-step from c to d in ABox
    $\rightsquigarrow$ check if $(c, d) \in \sigma^{\mathcal{I}}$
  - **concepts** M **in** $Loop_\alpha[s, s']$ — stay at same individual, jump to s'
    $\rightsquigarrow$ check if $c = d \in M^{\mathcal{I}}$
- **Exit** when we get **pair** (b, s_f)
- **Use counter** to ensure termination (only need to consider each pair once)

Algorithm EvalAtom

INPUT: NFA $\alpha = (S, \Sigma, \delta, s_0, F)$ with $\Sigma \subseteq N_R^{\pm} \cup \{A? \mid A \in N_C\}$, $\mathcal{ELHI}_\bot$ KB $(\mathcal{T}, \mathcal{A})$, $(a, b) \in \mathsf{Ind}(\mathcal{A}) \times \mathsf{Ind}(\mathcal{A})$

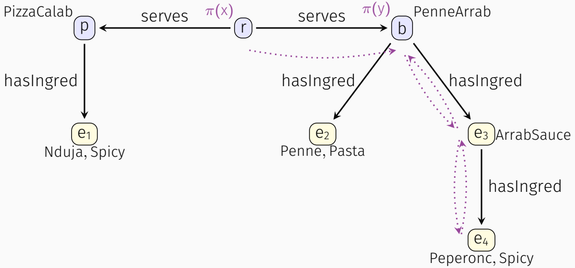
1. Test whether $(\mathcal{T}, \mathcal{A})$ is satisfiable, output **yes** if not.
2. Initialize current $= (a, s_0)$ and count $= 0$. Set max $= |\mathcal{A}| \cdot |S| + 1$.
3. While count $<$ max and current $\notin \{(b, s_f) \mid s_f \in F\}$
   - 3.1 Let current $= (c, s)$.
   - 3.2 Guess a pair $(d, s') \in \mathsf{Ind}(\mathcal{A}) \times S$ and either $(s, \sigma, s') \in \delta$ or $M \in \mathsf{Loop}_\alpha[s, s']$.
   - 3.3 If (s, σ, s') was guessed
     - If $\sigma \in N_R^{\pm}$, then verify that $\mathcal{T}, \mathcal{A} \models \sigma(c, d)$, and return **no** if not.
     - If $\sigma = A?$, then verify that $c = d$ and $\mathcal{T}, \mathcal{A} \models A(c)$, and return **no** if not.
   - 3.4 If M was guessed, then verify that $c = d$ and that $\mathcal{T}, \mathcal{A} \models B(c)$ for every concept name $B \in M$, and return **no** if not.
   - 3.5 Set current $= (d, s')$ and increment count.
4. If current $= (b, s_f)$ for some $s_f \in F$, return **yes**. Else return **no**.

EVALUATION ALGORITHM: EXAMPLE (1/2)

$$q(x, y) = \text{serves} \cdot (\text{hasIngred} \cup \text{ingredOf}^{-})^{*} \cdot \text{Spicy?} \cdot \Sigma^{*}(x, y)$$

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

$$\begin{aligned}
\text{PenneArrab} &\sqsubseteq \text{PastaDish} \sqcap \exists \text{hasIngred.ArrabSauce} \\
\text{PastaDish} &\sqsubseteq \text{Dish} \sqcap \exists \text{hasIngred.Pasta} \\
\text{ArrabSauce} &\sqsubseteq \exists \text{hasIngred.Peperonc} \\
\text{Peperonc} \sqcup \exists \text{hasIngred.Spicy} &\sqsubseteq \text{Spicy} \\
\text{Spicy} \sqcap \text{Dish} &\sqsubseteq \text{SpicyDish}
\end{aligned}$$

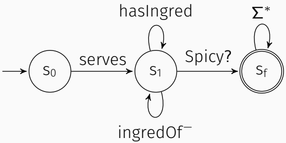


EVALUATION ALGORITHM: EXAMPLE (2/2)

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

$q(x, y) = \text{serves} \cdot (\text{hasIngred} \cup \text{ingredOf}^-)^* \cdot \text{Spicy?} \cdot \Sigma^*(x, y)$

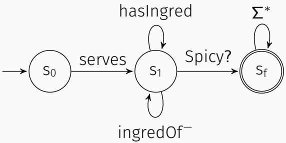
Peperonc $\in \text{Loop}_\alpha[s_1, s_f]$
ArrabSauce $\in \text{Loop}_\alpha[s_1, s_f]$
PenneArrab $\in \text{Loop}_\alpha[s_1, s_f]$



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

$q(x, y) = \text{serves} \cdot (\text{hasIngred} \cup \text{ingredOf}^-)^* \cdot \text{Spicy?} \cdot \Sigma^*(x, y)$

$\text{Peperonc} \in \text{Loop}_\alpha[s_1, s_f]$
$\text{ArrabSauce} \in \text{Loop}_\alpha[s_1, s_f]$
$\text{PenneArrab} \in \text{Loop}_\alpha[s_1, s_f]$



count:	0	1	2
Guess	(r, s_0)	(b, s_1)	(b, s_f)
		$(s_0, \text{serves}, s_1) \in \delta$	$\text{PenneArrab} \in \text{Loop}_\alpha[s_1, s_f]$
Test		$(r, b) \in \text{serves}^\mathcal{I}$	$b \in \text{PenneArrab}^\mathcal{I}$
			return **yes**

Theorem $(\mathsf{a},\mathsf{b}) \in \mathsf{cert}(\mathsf{q},\mathcal{K})$ iff there is **some execution of** $\mathsf{EvalAtom}(\alpha,\mathcal{K},(\mathsf{a},\mathsf{b}))$ that returns `yes`.

Theorem $(a, b) \in \mathrm{cert}(q, \mathcal{K})$ iff there is **some execution of** $\mathrm{EvalAtom}(\alpha, \mathcal{K}, (a, b))$ that returns `yes`.

- Iterations bounded by counter (poly. counter $\rightsquigarrow$ log space)
- We need calls to procedures for:
  satisfiability instance checking membership in Loop_α table
- These calls are in Exp for $\mathcal{ELHI}_\bot$
  Loop_α **computation:**
  exponentially many iterations (poly. in # states + # conjunctions)
  each one tests entailment

COMPLEXITY OF THE ALGORITHM

Theorem $(a,b) \in \mathrm{cert}(q,\mathcal{K})$ iff there is **some execution of** $\mathrm{EvalAtom}(\alpha,\mathcal{K},(a,b))$ that returns **yes**.

- Iterations bounded by counter (poly. counter $\rightsquigarrow$ log space)
- We need calls to procedures for:
  satisfiability   instance checking   membership in Loop_α table
- These calls are in Exp for $\mathcal{ELHI}_\bot$

  Loop_α **computation:**
  exponentially many iterations (poly. in # states + # conjunctions)
  each one tests entailment

Exp upper bound for $\mathcal{ELHI}_\bot$ (combined complexity)

COMPLEXITY OF THE ALGORITHM

Theorem $(a, b) \in \mathrm{cert}(q, \mathcal{K})$ iff there is **some execution of** $\mathrm{EvalAtom}(\alpha, \mathcal{K}, (a, b))$ that returns **yes**.

- Iterations bounded by counter (poly. counter $\rightsquigarrow$ log space)
- We need calls to procedures for:
  satisfiability instance checking membership in Loop_α table
- These calls are in EXP for $\mathcal{ELHI}_\bot$

  Loop_α **computation:**
  exponentially many iterations (poly. in # states + # conjunctions)
  each one tests entailment

EXP upper bound for $\mathcal{ELHI}_\bot$ (combined complexity)

For $\mathcal{ELH}$ and DL-Lite, we can obtain **P upper bound** (combined)

- modified Loop_α uses only **basic concepts** $A \in N_C$ **/** A, $\exists R$
- necessary tests (satisfiability, entailment, ...) **polynomial**

Data complexity:

- Loop_α **computation** in **constant time** (ABox independent)
- called procedures in **P for $\mathcal{ELH}$** and AC_0 **for DL-Lite$_\mathcal{R}$**
- EvalAtom needs **NLOGSPACE** (non-deterministic with poly counter)

Data complexity:

- Loop_α **computation** in **constant time** (ABox independent)
- called procedures in **P for $\mathcal{ELH}$** and AC_0 **for DL-Lite$_\mathcal{R}$**
- EvalAtom needs **NLOGSPACE** (non-deterministic with poly counter)

Theorem

- For $\mathcal{ELHI}_\bot$, 2RPQ answering is **EXP-complete** in **combined complexity** and **P-complete** in **data complexity**
- For **DL-Lite$_\mathcal{R}$ and $\mathcal{ELH}$**, the **combined complexity** drops to **P-complete**
- In **data complexity**, the problem is **NLOGSPACE-complete for DL-Lite$_\mathcal{R}$**, and **P-complete for $\mathcal{ELH}$**

Data complexity:

- Loop_α computation in constant time (ABox independent)
- called procedures in P for $\mathcal{ELH}$ and AC_0 for DL-Lite$_\mathcal{R}$
- EvalAtom needs NLogSpace (non-deterministic with poly counter)

Theorem

- For $\mathcal{ELHI}_\bot$, 2RPQ answering is Exp-complete in combined complexity and P-complete in data complexity
- For DL-Lite$_\mathcal{R}$ and $\mathcal{ELH}$, the combined complexity drops to P-complete
- In data complexity, the problem is NLogSpace-complete for DL-Lite$_\mathcal{R}$, and P-complete for $\mathcal{ELH}$

Most matching lower bounds from simpler problems:

Data complexity:

- Loop_α computation in constant time (ABox independent)
- called procedures in P for $\mathcal{ELH}$ and AC_0 for DL-Lite$_\mathcal{R}$
- EvalAtom needs NLOGSPACE (non-deterministic with poly counter)

Theorem

- For $\mathcal{ELHI}_\bot$, 2RPQ answering is EXP-complete in combined complexity and P-complete in data complexity
- For DL-Lite$_\mathcal{R}$ and $\mathcal{ELH}$, the combined complexity drops to P-complete
- In data complexity, the problem is NLOGSPACE-complete for DL-Lite$_\mathcal{R}$, and P-complete for $\mathcal{ELH}$

Most matching lower bounds from simpler problems:

instance checking graph reachability = RPQ over plain ABox

Data complexity:

- Loop_α computation in constant time (ABox independent)
- called procedures in P for $\mathcal{ELH}$ and AC_0 for DL-Lite$_\mathcal{R}$
- EvalAtom needs NLOGSPACE (non-deterministic with poly counter)

Theorem

- For $\mathcal{ELHI}_\bot$, 2RPQ answering is EXP-complete in combined complexity and P-complete in data complexity
- For DL-Lite$_\mathcal{R}$ and $\mathcal{ELH}$, the combined complexity drops to P-complete
- In data complexity, the problem is NLOGSPACE-complete for DL-Lite$_\mathcal{R}$, and P-complete for $\mathcal{ELH}$

Most matching lower bounds from simpler problems:

instance checking graph reachability = RPQ over plain ABox

P-hardness for DL-Lite$_\mathcal{R}$ non-trivial

For answering C2RPQs, we combine the earlier ideas:

- rewrite the query so that matches ranging over individuals suffice
- in each step, consider possibly deeper paths with Loop_α table

After rewriting, guess matches using individuals only and check them using EvalAtom on each atom

For answering C2RPQs, we combine the earlier ideas:

- rewrite the query so that matches ranging over individuals suffice
- in each step, consider possibly deeper paths with Loop_α table

After rewriting, guess matches using individuals only and check them using EvalAtom on each atom

Works for all DLs discussed and gives optimal complexity bounds

For answering C2RPQs, we combine the earlier ideas:

- rewrite the query so that matches ranging over individuals suffice
- in each step, consider possibly deeper paths with Loop_α table

After rewriting, guess matches using individuals only and check them using EvalAtom on each atom

Works for all DLs discussed and gives optimal complexity bounds

Answering C2RPQs is not much harder:

- combined complexity increases to PSPACE for DL-Lite$_{\mathcal{R}}$ and $\mathcal{ELH}$
- but most other bounds are the same as for RPQs and CQs
- even for very expressive DLs that are not Horn

Navigational queries provide more querying power at moderate computational cost

Good alternative to CQs, gaining increasing attention

Property paths in SPARQL

- included in the SPARQL 1.1 standard
- add regular paths as in C2RPQs

Ongoing quest for more flexible navigational languages

Expressible in Datalog, but computationally better behaved

COMPLEXITY OF ANSWERING (C)(2)RPQS

	2RPQs		C2RPQs	
	data complexity	combined complexity	data complexity	combined complexity
DL-Lite DL-Lite$_{\mathcal{R}}$	NLogSpace	P	NLogSpace	PSpace
$\mathcal{EL}, \mathcal{ELH}$	P	P	P	PSpace
$\mathcal{ELI}$, $\mathcal{ELHI}_{\bot}$, Horn-$\mathcal{SHOIQ}$	P	Exp	P	Exp
$\mathcal{ALC}$, $\mathcal{ALCHQ}$	coNP	Exp	coNP-hard	2Exp
$\mathcal{ALCI}$, $\mathcal{SH}$, $\mathcal{SHIQ}$	coNP	Exp	coNP-hard	2Exp
$\mathcal{SHOIQ}$	coNP	coNExp	coNP-hard[1]	coN2Exp-hard[1]

[1] decidability open

We can reduce emptiness of the intersection of regular languages to CRPQ answering in $\mathcal{EL}$ (or DL-Lite).

Given regular languages $L_1 \ldots L_n$ over alphabet Σ

We use a TBox to generate all words in Σ^*,

$$\mathcal{T} = \{\top \sqsubseteq \exists r_\sigma.\top \mid \sigma \in \Sigma\}$$

Then

$$\mathcal{L}(L_1) \cap \cdots \cap \mathcal{L}(L_n) \neq \emptyset \quad \text{iff} \quad \mathcal{T}, \{A(c)\} \models \exists x. L_1(c,x) \wedge \cdots \wedge L_n(c,x)$$

COMPARISON: COMPLEXITY OF ANSWERING (U)CQS

	IQs		CQs	
	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