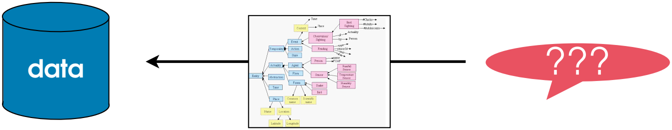
ONTOLOGY-MEDIATED QUERY ANSWERING WITH OWL 2 QL ONTOLOGIES

Succinctness and Complexity Landscapes

Meghyn Bienvenu (*LaBRI - CNRS & University of Bordeaux*)

Joint work with Stanislav Kikot, Roman Kontchakov, Vladimir Podolskii, and Michael Zakharyaschev

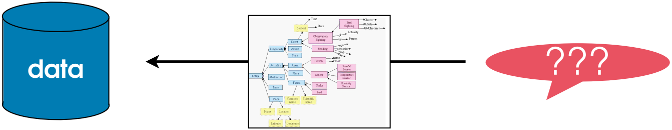


incomplete database

ontology

domain knowledge

user query



patient data

"Melanie has listeriosis"
"Paul has Lyme disease"

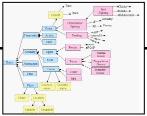
medical knowledge

"Listeriosis & Lyme disease are bacterial infections"

user query

"Find all patients with bacterial infections"

***expected answers*: Melanie, Paul**



patient data
"Melanie has listeriosis"
"Paul has Lyme disease"

medical knowledge
"Listeriosis & Lyme disease are bacterial infections"

user query
"Find all patients with bacterial infections"

***expected answers*: Melanie, Paul**

Why use an ontology?

- extend the vocabulary (making queries easier to formulate)
- provide a unified view of multiple data sources
- obtain more answers to queries (by exploiting domain knowledge)

Conjunctive queries (CQs) ∼ select-project-join queries in SQL

conjunctions of atoms, some variables can be existentially quantified

$$\exists y.\, \text{Faculty}(x) \wedge \text{Teaches}(x, y)$$

(find all faculty members that teach something)

Conjunctive queries (CQs) $\sim$ select-project-join queries in SQL

conjunctions of atoms, some variables can be existentially quantified

$$\exists y.\, \text{Faculty}(x) \wedge \text{Teaches}(x, y)$$

(find all faculty members that teach something)

OWL 2 QL ontologies

- W3C standardized ontology language
- based upon DL-Lite$_\mathcal{R}$ description logic
- designed for querying large datasets
- simple yet useful language

A (somewhat simplified) definition in FOL syntax

Ontology = finite set of FOL sentences (called **axioms**) of the forms:

$$
\begin{array}{ll}
\forall x\,\big(\tau(x) \to \tau'(x)\big) & \forall x\,\big(\tau(x) \land \tau'(x) \to \bot\big) \\
\forall x, y\,\big(\varrho(x,y) \to \varrho'(x,y)\big) & \forall x, y\,\big(\varrho(x,y) \land \varrho'(x,y) \to \bot\big) \\
\forall x\,\varrho(x,x) & \forall x\,\big(\varrho(x,x) \to \bot\big)
\end{array}
$$

where the formulas $\tau(x)$ and $\varrho(x,y)$ are defined by the grammars

$$
\begin{array}{lll}
\tau(x) & ::= \; A(x) \mid \exists y\,\varrho(x,y) & (A \text{ unary predicate}) \\
\varrho(x,y) & ::= \; P(x,y) \mid P(y,x) & (P \text{ binary predicate})
\end{array}
$$

For readability, we'll drop the universal quantifiers

A (somewhat simplified) definition in FOL syntax

Ontology = finite set of FOL sentences (called **axioms**) of the forms:

$$\begin{array}{ll} \forall x\,(\tau(x) \to \tau'(x)) & \forall x\,(\tau(x) \land \tau'(x) \to \bot) \\ \forall x, y\,(\varrho(x,y) \to \varrho'(x,y)) & \forall x, y\,(\varrho(x,y) \land \varrho'(x,y) \to \bot) \\ \forall x\,\varrho(x,x) & \forall x\,(\varrho(x,x) \to \bot) \end{array}$$

where the formulas $\tau(x)$ and $\varrho(x,y)$ are defined by the grammars

$$\begin{array}{lll} \tau(x) & ::= \; A(x) \mid \exists y\,\varrho(x,y) & (A \text{ unary predicate}) \\ \varrho(x,y) & ::= \; P(x,y) \mid P(y,x) & (P \text{ binary predicate}) \end{array}$$

For readability, we'll drop the universal quantifiers

Professors and **fellows** are **subclasses of faculty**

$$\text{Prof}(x) \rightarrow \text{Faculty}(x) \qquad \text{Fellow}(x) \rightarrow \text{Faculty}(x)$$

Professors and fellows are subclasses of faculty

$$\text{Prof}(x) \rightarrow \text{Faculty}(x) \qquad \text{Fellow}(x) \rightarrow \text{Faculty}(x)$$

Professors and fellows are disjoint classes

$$\text{Prof}(x) \land \text{Fellow}(x) \rightarrow \bot$$

Professors and fellows are subclasses of faculty

$$\text{Prof}(x) \rightarrow \text{Faculty}(x) \qquad \text{Fellow}(x) \rightarrow \text{Faculty}(x)$$

Professors and fellows are disjoint classes

$$\text{Prof}(x) \wedge \text{Fellow}(x) \rightarrow \bot$$

Professors must teach something

$$\text{Prof}(x) \rightarrow \exists y \text{Teaches}(x, y)$$

EXAMPLE AXIOMS

Professors and fellows are subclasses of faculty

$$\mathrm{Prof}(x) \rightarrow \mathrm{Faculty}(x) \qquad \mathrm{Fellow}(x) \rightarrow \mathrm{Faculty}(x)$$

Professors and fellows are disjoint classes

$$\mathrm{Prof}(x) \wedge \mathrm{Fellow}(x) \rightarrow \bot$$

Professors must teach something

$$\mathrm{Prof}(x) \rightarrow \exists y \mathrm{Teaches}(x, y)$$

Everything that is taught is a course

$$\exists x\, \mathrm{Teaches}(x, y) \rightarrow \mathrm{Course}(y)$$

EXAMPLE AXIOMS

Professors and fellows are subclasses of faculty

$$\text{Prof}(x) \rightarrow \text{Faculty}(x) \qquad \text{Fellow}(x) \rightarrow \text{Faculty}(x)$$

Professors and fellows are disjoint classes

$$\text{Prof}(x) \wedge \text{Fellow}(x) \rightarrow \bot$$

Professors must teach something

$$\text{Prof}(x) \rightarrow \exists y \text{Teaches}(x, y)$$

Everything that is taught is a course

$$\exists x\, \text{Teaches}(x, y) \rightarrow \text{Course}(y)$$

Being head of a team/lab/dept implies being a member

$$\text{HeadOf}(x, y) \rightarrow \text{MemberOf}(x, y)$$

OMQA SEMANTICS

Knowledge base (KB) = ontology $\mathcal{O}$ + dataset $\mathcal{D}$ (unary & binary facts)

Knowledge base (KB) = ontology $\mathcal{O}$ + dataset $\mathcal{D}$ (unary & binary facts)

Classical FOL semantics, based upon **interpretations** $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$

- function $\cdot^\mathcal{I}$ maps each unary predicate A to $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, each binary predicate R to $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and each constant a to $a^\mathcal{I} \in \Delta^\mathcal{I}$
- satisfaction of axioms, facts, or ground query in $\mathcal{I}$: as usual

Knowledge base (KB) = ontology $\mathcal{O}$ + dataset $\mathcal{D}$ (unary & binary facts)

Classical FOL semantics, based upon **interpretations** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

- function $\cdot^{\mathcal{I}}$ maps each unary predicate A to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each binary predicate R to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each constant a to $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- satisfaction of axioms, facts, or ground query in $\mathcal{I}$: as usual

Model = interpretation that **satisfies all axioms and facts in the KB**

- open-world assumption, facts not in the dataset can still be true

Knowledge base (KB) = ontology $\mathcal{O}$ + dataset $\mathcal{D}$ (unary & binary facts)

Classical FOL semantics, based upon **interpretations** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

- function $\cdot^{\mathcal{I}}$ maps each unary predicate A to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each binary predicate R to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each constant a to $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- satisfaction of axioms, facts, or ground query in $\mathcal{I}$: as usual

Model = interpretation that **satisfies all axioms and facts in the KB**

- open-world assumption, facts not in the dataset can still be true

Certain answers of query q w.r.t. KB $(\mathcal{O}, \mathcal{D})$:

- tuples of constants $\vec{a}$ (of same arity as q) such that $q(\vec{a})$ holds in **every model** of $(\mathcal{O}, \mathcal{D})$
- corresponds to a form of entailment, we'll write $\mathcal{O}, \mathcal{D} \models q(\vec{a})$

Knowledge base (KB) = ontology $\mathcal{O}$ + dataset $\mathcal{D}$ (unary & binary facts)

Classical FOL semantics, based upon **interpretations** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

- function $\cdot^{\mathcal{I}}$ maps each unary predicate A to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each binary predicate R to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each constant a to $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- satisfaction of axioms, facts, or ground query in $\mathcal{I}$: as usual

Model = interpretation that **satisfies all axioms and facts in the KB**

- open-world assumption, facts not in the dataset can still be true

Certain answers of query q w.r.t. KB $(\mathcal{O}, \mathcal{D})$:

- tuples of constants $\vec{a}$ (of same arity as q) such that $q(\vec{a})$ holds in **every model** of $(\mathcal{O}, \mathcal{D})$
- corresponds to a form of entailment, we'll write $\mathcal{O}, \mathcal{D} \models q(\vec{a})$

Ontology-mediated query answering: computing certain answers

OMQA EXAMPLE

Ontology:

$$\text{Prof}(x) \to \text{Faculty}(x) \qquad \text{Fellow}(x) \to \text{Faculty}(x)$$
$$\text{Prof}(x) \to \exists y \text{Teaches}(x, y) \qquad \exists x \text{Teaches}(x, y) \to \text{Course}(y)$$

Dataset:

$$\{\text{Prof}(\text{anna}), \text{Fellow}(\text{tom}), \text{Teaches}(\text{tom}, \text{cs101})\}$$

Query: $q(x) = \exists y.\text{Faculty}(x) \wedge \text{Teaches}(x, y)$

OMQA EXAMPLE

Ontology:

$$\mathsf{Prof}(x) \rightarrow \mathsf{Faculty}(x) \qquad \mathsf{Fellow}(x) \rightarrow \mathsf{Faculty}(x)$$
$$\mathsf{Prof}(x) \rightarrow \exists y \mathsf{Teaches}(x, y) \qquad \exists x \mathsf{Teaches}(x, y) \rightarrow \mathsf{Course}(y)$$

Dataset:

$$\{\mathsf{Prof}(\mathsf{anna}), \mathsf{Fellow}(\mathsf{tom}), \mathsf{Teaches}(\mathsf{tom}, \mathsf{cs101})\}$$

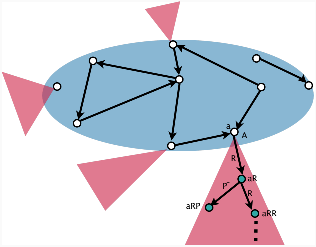
Query: $q(x) = \exists y.\mathsf{Faculty}(x) \wedge \mathsf{Teaches}(x, y)$

Get the following certain answers:

- anna $\qquad \mathsf{Prof}(\mathsf{anna}) + \mathsf{Prof}(x) \rightarrow \mathsf{Faculty}(x) + \mathsf{Prof}(x) \rightarrow \exists y \mathsf{Teaches}(x, y)$
- tom $\qquad \mathsf{Fellow}(\mathsf{tom}) + \mathsf{Fellow}(x) \rightarrow \mathsf{Faculty}(x) + \mathsf{Teaches}(\mathsf{tom}, \mathsf{cs101})$

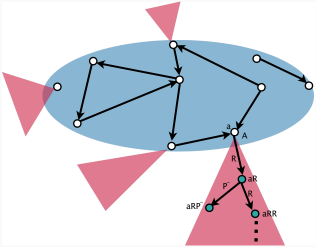
For Horn ontologies (no form of disjunction) like OWL 2 QL: enough to consider a single canonical model

- idea: exhaustively apply ontology axioms to dataset
- possibly infinite ($A(x) \rightarrow \exists y R(x,y)$, $R(x,y) \rightarrow A(y)$)
- forest-shaped (dataset + new tree structures for ∃-axioms)
- give correct answer to all CQs



For Horn ontologies (no form of disjunction) like OWL 2 QL: enough to consider a single canonical model

- idea: exhaustively apply ontology axioms to dataset
- possibly infinite ($A(x) \rightarrow \exists y R(x,y)$, $R(x,y) \rightarrow A(y)$)
- forest-shaped (dataset + new tree structures for ∃-axioms)
- give correct answer to all CQs



OMQA in OWL 2 QL =

finding ways to map the query into the canonical model

OMQA viewed as a decision problem (yes-or-no question):

INPUT: An n-ary query q, a dataset $\mathcal{D}$, a ontology $\mathcal{O}$, and a candidate answer tuple $\vec{a}$

QUESTION: Does $\mathcal{O}, \mathcal{D} \models q(\vec{a})$?

OMQA viewed as a decision problem (yes-or-no question):

INPUT: An n-ary query q, a dataset $\mathcal{D}$, a ontology $\mathcal{O}$, and a candidate answer tuple $\vec{a}$

QUESTION: Does $\mathcal{O}, \mathcal{D} \models q(\vec{a})$?

Combined complexity: in terms of size of whole input

Data complexity: in terms of size of $\mathcal{D}$ only

- view rest of input as fixed (of constant size)
- motivation: data typically much larger than rest of input

data complexity $\leq$ combined complexity

Idea: reduce OMQA to database query evaluation

- rewriting step: ontology $\mathcal{O}$ + query $q \rightsquigarrow$ first-order (SQL) query q'
- evaluation step: evaluate query q' using relational DB system

Advantage: harness efficiency of relational database systems

Idea: reduce OMQA to database query evaluation

- rewriting step: ontology $\mathcal{O}$ + query $q \rightsquigarrow$ first-order (SQL) query q'
- evaluation step: evaluate query q' using relational DB system

Advantage: harness efficiency of relational database systems

Key notion: first-order (FO) rewriting

- FO query q' is an FO-rewriting of OMQ $(\mathcal{O}, q)$ iff for every dataset $\mathcal{D}$:

$$\mathcal{O}, \mathcal{D} \models q(\vec{a}) \quad \Leftrightarrow \quad DB_{\mathcal{D}} \models q'(\vec{a})$$

Informally: evaluating q' over $\mathcal{D}$ (viewed as DB) gives correct result

QUERY REWRITING

Idea: reduce OMQA to database query evaluation

- rewriting step: ontology $\mathcal{O}$ + query $q \rightsquigarrow$ first-order (SQL) query q'
- evaluation step: evaluate query q' using relational DB system

Advantage: harness efficiency of relational database systems

Key notion: first-order (FO) rewriting

- FO query q' is an FO-rewriting of OMQ $(\mathcal{O}, q)$ iff for every dataset $\mathcal{D}$:

$$\mathcal{O}, \mathcal{D} \models q(\vec{a}) \quad \Leftrightarrow \quad DB_{\mathcal{D}} \models q'(\vec{a})$$

Informally: evaluating q' over $\mathcal{D}$ (viewed as DB) gives correct result

Good news: every CQ and OWL 2 QL ontology has an FO-rewriting

Reconsider the ontology $\mathcal{O}$:

$$\mathrm{Prof}(x) \rightarrow \mathrm{Faculty}(x) \qquad \mathrm{Fellow}(x) \rightarrow \mathrm{Faculty}(x)$$
$$\mathrm{Prof}(x) \rightarrow \exists y \mathrm{Teaches}(x, y) \qquad \mathrm{Teaches}(x, y) \rightarrow \mathrm{Course}(y)$$

and the query $q(x) = \exists y.\mathrm{Faculty}(x) \wedge \mathrm{Teaches}(x, y)$

Reconsider the ontology $\mathcal{O}$:

$$\begin{array}{ll} \text{Prof}(x) \rightarrow \text{Faculty}(x) & \text{Fellow}(x) \rightarrow \text{Faculty}(x) \\ \text{Prof}(x) \rightarrow \exists y \text{Teaches}(x,y) & \text{Teaches}(x,y) \rightarrow \text{Course}(y) \end{array}$$

and the query $q(x) = \exists y.\text{Faculty}(x) \wedge \text{Teaches}(x,y)$

The following query is a rewriting of $q(x)$ w.r.t. $\mathcal{O}$:

$$q(x) \quad \vee \quad \text{Prof}(x) \quad \vee \quad \exists y.\text{Fellow}(x) \wedge \text{Teaches}(x,y)$$

Reconsider the ontology $\mathcal{O}$:

$$\begin{array}{ll} \text{Prof}(x) \rightarrow \text{Faculty}(x) & \text{Fellow}(x) \rightarrow \text{Faculty}(x) \\ \text{Prof}(x) \rightarrow \exists y \text{Teaches}(x,y) & \text{Teaches}(x,y) \rightarrow \text{Course}(y) \end{array}$$

and the query $q(x) = \exists y.\text{Faculty}(x) \wedge \text{Teaches}(x,y)$

The following query is a rewriting of $q(x)$ w.r.t. $\mathcal{O}$:

$$q(x) \quad \vee \quad \text{Prof}(x) \quad \vee \quad \exists y.\text{Fellow}(x) \wedge \text{Teaches}(x,y)$$

Evaluating the rewritten query over the earlier dataset

$$\{\text{Prof}(\text{anna}), \text{Fellow}(\text{tom}), \text{Teaches}(\text{tom}, \text{cs101})\}$$

produces the two certain answers: anna and tom

Data-independent reduction of OMQA to DB query evaluation

- inherit low data complexity ($AC_0 \subsetneq$ PTIME) of FO query evaluation

Data-independent reduction of OMQA to DB query evaluation

- inherit low data complexity ($AC_0 \subsetneq$ PTIME) of FO query evaluation

However, experiments with several rewriting algorithms showed that the generated rewritings can be huge!

- can be difficult / impossible to generate and evaluate

Data-independent reduction of OMQA to DB query evaluation

- inherit low data complexity ($AC_0 \subsetneq$ PTIME) of FO query evaluation

However, experiments with several rewriting algorithms showed that the generated rewritings can be huge!

- can be difficult / impossible to generate and evaluate

To make the technique work in practice: want to generate reasonably small rewritings that are not too difficult to evaluate

Data-independent reduction of OMQA to DB query evaluation

- inherit low data complexity ($AC_0 \subsetneq$ PTIME) of FO query evaluation

However, experiments with several rewriting algorithms showed that the generated rewritings can be huge!

- can be difficult / impossible to generate and evaluate

To make the technique work in practice: want to generate reasonably small rewritings that are not too difficult to evaluate

This raises the following questions:

Succinctness When can we guarantee polynomial-size rewritings?

Complexity More generally, when is OMQA tractable?

Optimality Can query rewriting achieve optimal complexity?

SUCCINCTNESS OF REWRITINGS

Many of the proposed rewriting algorithms produce unions of conjunctive queries (UCQs = $\vee$ of CQs)

Many of the proposed rewriting algorithms produce unions of conjunctive queries (UCQs = $\vee$ of CQs)

Not hard to see smallest UCQ-rewriting may be exponentially large:

EASY NEGATIVE RESULT FOR UCQ-REWRITINGS

Many of the proposed rewriting algorithms produce unions of conjunctive queries (UCQs = $\vee$ of CQs)

Not hard to see smallest UCQ-rewriting may be exponentially large:

- Query: $A_1^0(x) \wedge \ldots \wedge A_n^0(x)$
- Ontology: $A_1^1(x) \rightarrow A_1^0(x) \quad A_2^1(x) \rightarrow A_2^0(x) \quad \ldots \quad A_n^1(x) \rightarrow A_n^0(x)$
- Rewriting: $\bigvee_{(i_1,\ldots,i_n)\in\{0,1\}} A_1^{i_1}(x) \wedge A_1^{i_1}(x) \wedge \ldots \wedge A_1^{i_1}(x)$

EASY NEGATIVE RESULT FOR UCQ-REWRITINGS

Many of the proposed rewriting algorithms produce unions of conjunctive queries (UCQs = $\vee$ of CQs)

Not hard to see smallest UCQ-rewriting may be exponentially large:

- Query: $A_1^0(x) \wedge \ldots \wedge A_n^0(x)$
- Ontology: $A_1^1(x) \rightarrow A_1^0(x) \quad A_2^1(x) \rightarrow A_2^0(x) \quad \ldots \quad A_n^1(x) \rightarrow A_n^0(x)$
- Rewriting: $\bigvee_{(i_1,\ldots,i_n)\in\{0,1\}} A_1^{i_1}(x) \wedge A_1^{i_1}(x) \wedge \ldots \wedge A_1^{i_1}(x)$

But: simple polysize FO-rewriting does exist! $\bigwedge_{i=1}^{n}(A_i^0(x) \vee A_i^1(x))$

Many of the proposed rewriting algorithms produce unions of conjunctive queries (UCQs = $\vee$ of CQs)

Not hard to see smallest UCQ-rewriting may be exponentially large:

- Query: $A_1^0(x) \wedge \ldots \wedge A_n^0(x)$
- Ontology: $A_1^1(x) \rightarrow A_1^0(x) \quad A_2^1(x) \rightarrow A_2^0(x) \quad \ldots \quad A_n^1(x) \rightarrow A_n^0(x)$
- Rewriting: $\bigvee_{(i_1,\ldots,i_n)\in\{0,1\}} A_1^{i_1}(x) \wedge A_1^{i_1}(x) \wedge \ldots \wedge A_1^{i_1}(x)$

But: simple polysize FO-rewriting does exist! $\bigwedge_{i=1}^{n}(A_i^0(x) \vee A_i^1(x))$

To get positive results, need to go beyond UCQs

PE-rewritings: positive existential queries (only $\exists$, $\wedge$, $\vee$)

$$(r(x,y) \vee s(y,x)) \wedge (A(x) \vee (B(x) \wedge \exists z\, p(x,z))) \wedge (A(y) \vee (B(y) \wedge \exists z\, p(y,z)))$$

DIFFERENT FORMS OF REWRITINGS

PE-rewritings: positive existential queries (only $\exists$, $\wedge$, $\vee$)

$$(r(x,y) \vee s(y,x)) \wedge (A(x) \vee (B(x) \wedge \exists z\, p(x,z))) \wedge (A(y) \vee (B(y) \wedge \exists z\, p(y,z)))$$

NDL-rewritings: non-recursive Datalog queries

$$q_1(x,y), q_2(x), q_2(y) \rightarrow \mathrm{goal}(x,y)$$

$$r(x,y) \rightarrow q_1(x,y) \qquad A(x) \rightarrow q_2(x)$$

$$s(y,x) \rightarrow q_1(x,y) \qquad B(x), p(x,z) \rightarrow q_2(x)$$

PE-rewritings: positive existential queries (only $\exists$, $\wedge$, $\vee$)

$$(r(x,y) \vee s(y,x)) \wedge (A(x) \vee (B(x) \wedge \exists z\, p(x,z))) \wedge (A(y) \vee (B(y) \wedge \exists z\, p(y,z)))$$

NDL-rewritings: non-recursive Datalog queries

$$q_1(x,y), q_2(x), q_2(y) \rightarrow \mathrm{goal}(x,y)$$

$$r(x,y) \rightarrow q_1(x,y) \qquad A(x) \rightarrow q_2(x)$$

$$s(y,x) \rightarrow q_1(x,y) \qquad B(x), p(x,z) \rightarrow q_2(x)$$

FO-rewritings: first-order queries (can also use $\forall$, $\neg$)

PE-rewritings: positive existential queries (only $\exists$, $\wedge$, $\vee$)

$$(r(x,y) \vee s(y,x)) \wedge (A(x) \vee (B(x) \wedge \exists z\, p(x,z))) \wedge (A(y) \vee (B(y) \wedge \exists z\, p(y,z)))$$

NDL-rewritings: non-recursive Datalog queries

$$q_1(x,y), q_2(x), q_2(y) \rightarrow \mathsf{goal}(x,y)$$

$$r(x,y) \rightarrow q_1(x,y) \qquad A(x) \rightarrow q_2(x)$$

$$s(y,x) \rightarrow q_1(x,y) \qquad B(x), p(x,z) \rightarrow q_2(x)$$

FO-rewritings: first-order queries (can also use $\forall$, $\neg$)

What if we replace UCQs by PE / NDL / FO?
Do we get polysize rewritings?

FIRST NEGATIVE RESULTS [KKPZ12]

Exponential blowup unavoidable for PE / NDL-rewritings

Formally: sequence of CQs q_n and OWL 2 QL ontologies $\mathcal{O}_n$ such that

- PE- and NDL-rewritings of $(\mathcal{O}_n, q_n)$ exponential in $|q_n| + |\mathcal{O}_n|$
- FO-rewritings of $(\mathcal{O}_n, q_n)$ superpolynomial unless $\mathsf{NP}/\mathrm{poly} \subseteq \mathsf{NC}^1$

Exponential blowup unavoidable for PE / NDL-rewritings

Formally: sequence of CQs q_n and OWL 2 QL ontologies $\mathcal{O}_n$ such that

- PE- and NDL-rewritings of $(\mathcal{O}_n, q_n)$ exponential in $|q_n| + |\mathcal{O}_n|$
- FO-rewritings of $(\mathcal{O}_n, q_n)$ superpolynomial unless $\mathsf{NP}/\mathrm{poly} \subseteq \mathsf{NC}^1$

Key proof step: reduce CNF satisfiability to OMQA

- ontology generates full binary tree, leaves represent valuations
  - depth of tree = number of variables
- tree-shaped query* selects valuation, checks clauses are satisfied
  - number of leaves / branches in query = number of clauses

* tree-shaped (acyclic) = undirected graph induced by query is a tree

Depth of ontology =
maximum depth of generated trees in canonical model

- $\mathcal{O}$ has finite depth $\leftrightarrow$ applying axioms in $\mathcal{O}$ always terminates

Depth of ontology =
maximum depth of generated trees in canonical model

- $\mathcal{O}$ has finite depth $\leftrightarrow$ applying axioms in $\mathcal{O}$ always terminates

Does restricting ontology depth suffice for polysize rewritings?

Depth of ontology =
maximum depth of generated trees in canonical model

- $\mathcal{O}$ has finite depth $\leftrightarrow$ applying axioms in $\mathcal{O}$ always terminates

Does restricting ontology depth suffice for polysize rewritings?

Unfortunately not...

Depth of ontology =
maximum depth of generated trees in canonical model

- $\mathcal{O}$ has finite depth $\leftrightarrow$ applying axioms in $\mathcal{O}$ always terminates

Does restricting ontology depth suffice for polysize rewritings?

Unfortunately not...

Depth 2 ontologies:

- no polysize PE- or NDL-rewritings
- no polysize FO-rewritings unless $\text{NP}/\mathrm{poly} \subseteq \text{NC}^1$

Depth 1 ontologies:

- no polysize PE- or NDL-rewritings
- no polysize FO-rewritings unless $\text{NL}/\mathrm{poly} \subseteq \text{NC}^1$

Depth of ontology =
maximum depth of generated trees in canonical model

- $\mathcal{O}$ has finite depth $\leftrightarrow$ applying axioms in $\mathcal{O}$ always terminates

Does restricting ontology depth suffice for polysize rewritings?

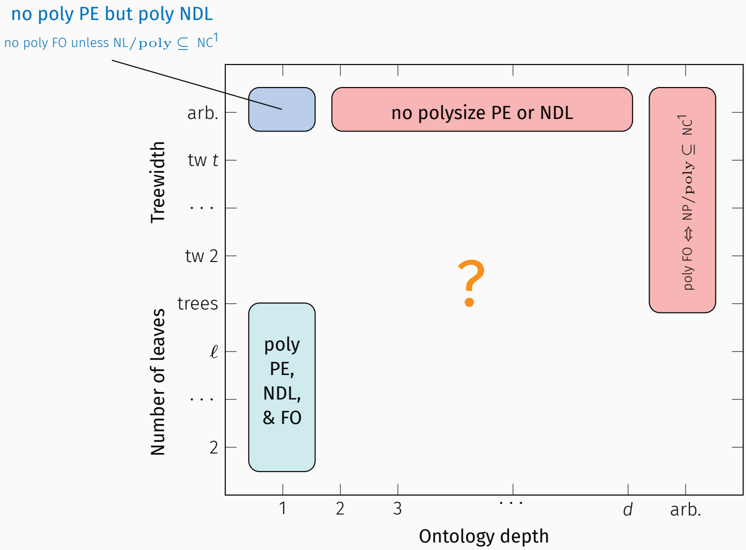
Unfortunately not...

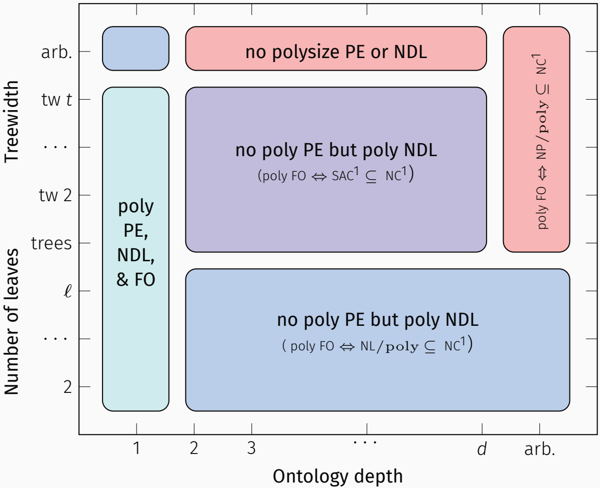
Depth 2 ontologies:

- no polysize PE- or NDL-rewritings
- no polysize FO-rewritings unless NP/$\mathrm{poly} \subseteq$ NC^1

Depth 1 ontologies:

- no polysize PE- or NDL-rewritings
- no polysize FO-rewritings unless NL/$\mathrm{poly} \subseteq$ NC^1
- but: polysize PE-rewritings for tree-shaped queries


no poly PE but poly NDL
no poly FO unless NL/poly ⊆ NC^1
arb.
tw t
...
tw 2
trees
ℓ
...
2
Treewidth
Number of leaves
no polysize PE or NDL
poly FO ⇔ NP/poly ⊆ NC^1
?
poly PE, NDL, & FO
1
2
3
...
d
arb.
Ontology depth


Treewidth
Number of leaves
arb.
tw t
...
tw 2
trees
ℓ
...
2
no polysize PE or NDL
poly PE, NDL, & FO
no poly PE but poly NDL
(poly FO ⇔ SAC¹ ⊆ NC¹)
poly FO ⇔ NP/poly ⊆ NC¹
no poly PE but poly NDL
(poly FO ⇔ NL/poly ⊆ NC¹)
1
2
3
...
d
arb.
Ontology depth

Strong negative result for PE-rewritings

- no polysize PE-rewritings for depth 2 ontologies + linear CQs

Conditional negative results for FO-rewritings

- polysize FO-rewritings exist iff
  - $\mathsf{SAC}^1 \subseteq \mathsf{NC}^1$ — bounded depth + bounded treewidth CQs
  - $\mathsf{NL}/\mathrm{poly} \subseteq \mathsf{NC}^1$ — bounded-leaf tree-shaped CQs

Strong negative result for PE-rewritings

- no polysize PE-rewritings for depth 2 ontologies + linear CQs

Conditional negative results for FO-rewritings

- polysize FO-rewritings exist iff
  - $\mathsf{SAC}^1 \subseteq \mathsf{NC}^1$ bounded depth + bounded treewidth CQs
  - $\mathsf{NL}/\mathrm{poly} \subseteq \mathsf{NC}^1$ bounded-leaf tree-shaped CQs

Positive results for NDL-rewritings

- bounded depth ontology + bounded treewidth CQs
- bounded-leaf tree-shaped CQs (+ arbitrary ontology)

Takeaway: NDL good target language for rewritings

BRIEF GLIMPSE AT PROOF TECHNIQUES (1)

Standard **computational complexity not the right tool**

- can be used to show no polytime-computable rewriting
- … but not that no polysize rewriting exists

BRIEF GLIMPSE AT PROOF TECHNIQUES (1)

Standard **computational complexity not the right tool**

- can be used to show no polytime-computable rewriting
- … but not that no polysize rewriting exists

Instead: establish **tight connections to circuit complexity**

- branch of complexity that classifies Boolean functions wrt. size / depth of Boolean circuits / formulas that compute them
- recall k-ary Boolean function maps tuples from $\{0,1\}^k$ to $\{0,1\}$

Standard **computational complexity not the right tool**

- can be used to show no polytime-computable rewriting
- … but not that no polysize rewriting exists

Instead: establish **tight connections to circuit complexity**

- branch of complexity that classifies Boolean functions wrt. size / depth of Boolean circuits / formulas that compute them
- recall k-ary Boolean function maps tuples from $\{0, 1\}^k$ to $\{0, 1\}$

Example: **function** $\textsc{Reach}_n$

- **input**: a Boolean vector representing the adjacency matrix of a directed graph G with n vertices including special vertices s and t
- **output**: 1 iff encoded graph G contains a directed path from s to t

BRIEF GLIMPSE AT PROOF TECHNIQUES (1)

Standard **computational complexity not the right tool**

- can be used to show no polytime-computable rewriting
- … but not that no polysize rewriting exists

Instead: establish **tight connections to circuit complexity**

- branch of complexity that classifies Boolean functions wrt. size / depth of Boolean circuits / formulas that compute them
- recall k-ary Boolean function maps tuples from $\{0,1\}^k$ to $\{0,1\}$

Example: **function** $\textsc{Reach}_n$

- **input**: a Boolean vector representing the adjacency matrix of a directed graph G with n vertices including special vertices s and t
- **output**: 1 iff encoded graph G contains a directed path from s to t

No family of polysize mon. Boolean formulas computing $\textsc{Reach}_n$

Types of rewritings $\rightsquigarrow$	ways of representing Boolean functions
PE-rewritings	monotone Boolean formulas ($\land, \lor$)
NDL-rewritings	monotone Boolean circuits ($\lor$- and $\land$-gates)
FO-rewritings	Boolean formulas ($\land, \lor, \neg$)

BRIEF GLIMPSE AT PROOF TECHNIQUES (2)

Types of rewritings $\rightsquigarrow$	ways of representing Boolean functions
PE-rewritings	monotone Boolean formulas $(\land, \lor)$
NDL-rewritings	monotone Boolean circuits ($\lor$- and $\land$-gates)
FO-rewritings	Boolean formulas $(\land, \lor, \neg)$

Associate Boolean functions with ontology-mediated query $(\mathcal{O}, q)$

Types of rewritings $\rightsquigarrow$ ways of representing Boolean functions

PE-rewritings	monotone Boolean formulas $(\wedge, \vee)$
NDL-rewritings	monotone Boolean circuits ($\vee$- and $\wedge$-gates)
FO-rewritings	Boolean formulas $(\wedge, \vee, \neg)$

Associate Boolean functions with ontology-mediated query $(\mathcal{O}, q)$

'Lower bound' function $f^{\mathsf{LB}}_{\mathcal{O},q}$ $\Rightarrow$ lower bounds on rewriting size

- transform rewriting of $(\mathcal{O}, q)$ into formula / circuit that computes $f^{\mathsf{LB}}_{\mathcal{O},q}$

'Upper bound' function $f^{\mathsf{UB}}_{\mathcal{O},q}$ $\Rightarrow$ upper bounds on rewriting size

- transform formula / circuit that computes $f^{\mathsf{UB}}_{\mathcal{O},q}$ into rewriting of $(\mathcal{O}, q)$

Types of rewritings $\rightsquigarrow$ ways of representing Boolean functions

PE-rewritings	monotone Boolean formulas $(\wedge, \vee)$
NDL-rewritings	monotone Boolean circuits ($\vee$- and $\wedge$-gates)
FO-rewritings	Boolean formulas $(\wedge, \vee, \neg)$

Associate Boolean functions with ontology-mediated query $(\mathcal{O}, q)$

'Lower bound' function $f^{\mathsf{LB}}_{\mathcal{O},q}$ $\Rightarrow$ lower bounds on rewriting size

- transform rewriting of $(\mathcal{O}, q)$ into formula / circuit that computes $f^{\mathsf{LB}}_{\mathcal{O},q}$

'Upper bound' function $f^{\mathsf{UB}}_{\mathcal{O},q}$ $\Rightarrow$ upper bounds on rewriting size

- transform formula / circuit that computes $f^{\mathsf{UB}}_{\mathcal{O},q}$ into rewriting of $(\mathcal{O}, q)$

Exploit circuit complexity results about (in)existence of small formulas / circuits computing different classes of Boolean functions

- which functions expressible as $f^{\mathsf{LB}}_{q,\mathcal{O}}$ / $f^{\mathsf{UB}}_{q,\mathcal{O}}$ for given OMQ class?
  - intermediate computational model: hypergraph programs

A **hypergraph program (HGP)** is a hypergraph $H = (V, E)$, where:

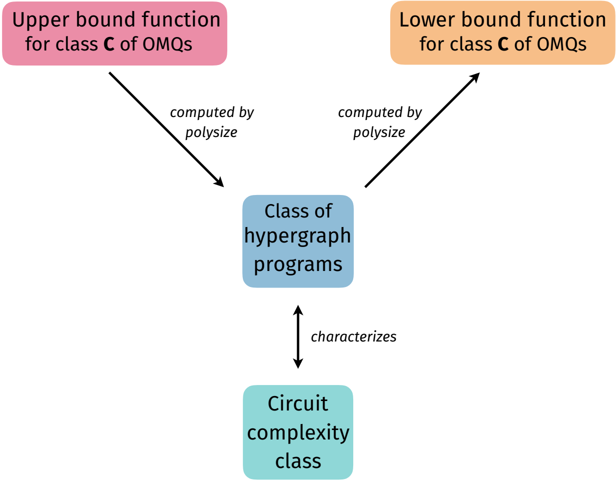
- vertices labelled by 0, 1, or literal $(\neg)p_i$
- input: valuation of $p_0, \ldots, p_n$
- **outputs 1** $\Leftrightarrow$ set of non-overlapping hyperedges that 'covers all zeros' (i.e. contains all vertices whose label evaluates to 0)

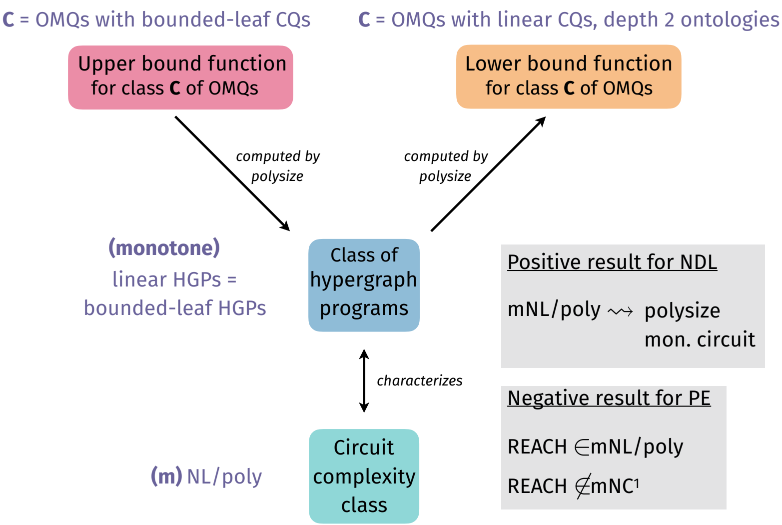
Restricted HGPs: monotone, bounded degree, tree / linear

Hypergraph associated with **ontology-mediated query** $(\mathcal{O}, q)$:

- **vertices = atoms in** q
- **hyperedges = subqueries of** q **'relevant' for** $\mathcal{O}$
  - roughly: can be satisfied by tree-shaped structure of canonical model

BACK TO GLIMPSE AT PROOF TECHNIQUES




C = OMQs with bounded-leaf CQs
C = OMQs with linear CQs, depth 2 ontologies
Upper bound function for class C of OMQs
Lower bound function for class C of OMQs
computed by polysize
computed by polysize
(monotone)
linear HGPs =
bounded-leaf HGPs
Class of hypergraph programs
characterizes
(m) NL/poly
Circuit complexity class
Positive result for NDL
mNL/poly ⇝ polysize mon. circuit
Negative result for PE
REACH ∈ mNL/poly
REACH ∉ mNC1

COMPLEXITY AND OPTIMALITY

Small rewritings do not guarantee low combined complexity

- need to consider cost of producing and evaluating the rewriting

Large rewritings do not guarantee high combined complexity

- maybe query rewriting is not the most efficient approach

WHAT DOES ALL THIS MEAN FOR THE COMPLEXITY OF OMQA?

Small rewritings do not guarantee low combined complexity

- need to consider cost of producing and evaluating the rewriting

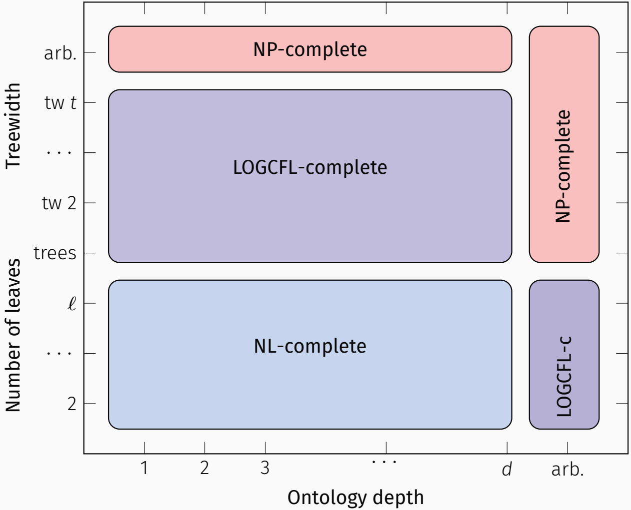
Large rewritings do not guarantee high combined complexity

- maybe query rewriting is not the most efficient approach

Motivated the study of the complexity landscape of query answering

Focus on combined complexity (data complexity same in all cases)

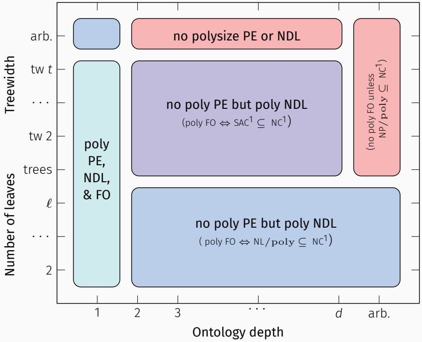
COMBINED COMPLEXITY LANDSCAPE FOR DL-LITE [BKP15], [BKKPZ18]



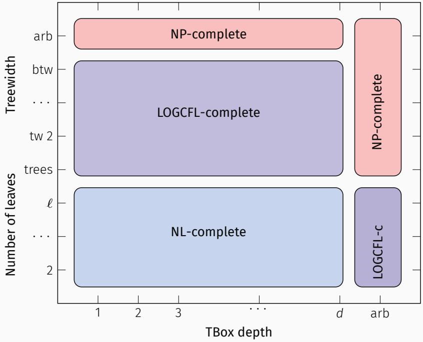
NL $\subseteq$ LOGCFL $\subseteq$ PTIME $\subseteq$ NP

COMPARING SUCCINCTNESS & COMPLEXITY LANDSCAPES

Size of rewritings



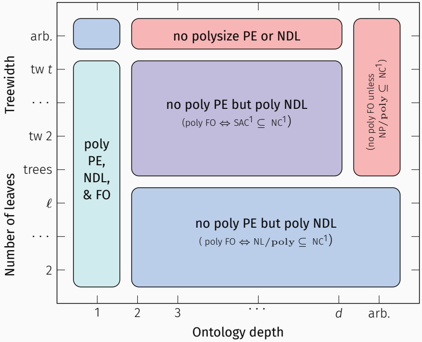
Combined complexity of OMQA



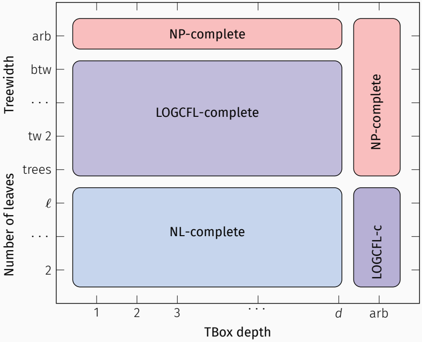
polysize NDL-rewritings ∼ polynomial (LOGCFL / NL) complexity

COMPARING SUCCINCTNESS & COMPLEXITY LANDSCAPES

Size of rewritings



Combined complexity of OMQA



polysize NDL-rewritings ∼ polynomial (LOGCFL / NL) complexity

Can we marry the positive succinctness & complexity results?

For the three well-behaved classes of OMQs, define **NDL-rewritings of optimal complexity**:

- rewriting can be constructed by L^C transducer
- evaluating the rewriting can be done in C

with $C \in \{\text{NL}, \text{LOGCFL}\}$ the complexity of the OMQ class

For the three well-behaved classes of OMQs, define
NDL-rewritings of optimal complexity:

- rewriting can be constructed by L^C transducer
- evaluating the rewriting can be done in C

with $C \in \{\mathrm{NL}, \mathrm{LOGCFL}\}$ the complexity of the OMQ class

Preliminary experiments with **simple OMQs** (**depth 1, linear CQs**):

- compared with other NDL-rewritings (Clipper, Rapid, Presto)
- **our rewritings grow linearly** with increasing query size
- **other systems** produce rewritings that **grow exponentially**

For the three well-behaved classes of OMQs, define
NDL-rewritings of optimal complexity:

- rewriting can be constructed by L^C transducer
- evaluating the rewriting can be done in C

with $C \in \{\mathrm{NL}, \mathrm{LOGCFL}\}$ the complexity of the OMQ class

Preliminary experiments with **simple OMQs** (**depth 1, linear CQs**):

- compared with other NDL-rewritings (Clipper, Rapid, Presto)
- **our rewritings grow linearly** with increasing query size
- **other systems** produce rewritings that **grow exponentially**

Take-away: optimal complexity achievable via query rewriting

CONCLUSION

Ontology-mediated query answering:

- new paradigm for intelligent information systems
- offers many advantages, but also computational challenges

Query rewriting promising algorithmic approach

Many interesting problems related to OMQA and query rewriting:

- succinctness of rewritings (Boolean functions, circuit complexity)
- existence of FO and Datalog rewritings (automata, CSP)
- other tools: parameterized complexity, word rewriting

Active area with lots left to explore!

QUESTIONS?

[KKPZ12] S. Kikot, R. Kontchakov, V. Podolskii, and M. Zakharyaschev: Exponential Lower Bounds and Separation for Query Rewriting. 39th International Colloquium on Automata, Languages, and Programming (ICALP'12), 2012.

[GS12] G. Gottlob and T. Schwentick: Rewriting Ontological Queries into Small Nonrecursive Datalog Programs. 13th International Conference on the Principles of Knowledge Representation and Reasoning (KR'12), 2012.

[GKKPSZ14] G. Gottlob, S. Kikot, R. Kontchakov, V. Podolskii, T. Schwentick, and M. Zakharyaschev: The Price of Query Rewriting in Ontology-based Data Access. Artificial Intelligence (AIJ), 2014.

[KKPZ14] S. Kikot, R. Kontchakov, V. Podolskii, and M. Zakharyaschev: On the Succinctness of Query Rewriting over Shallow Ontologies. 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'14), 2014.

[BKP15] M. Bienvenu, S. Kikot, V. Podolskii: Tree-like Queries in OWL 2 QL: Succinctness and Complexity Results. 30th Annual ACM/IEEE Symposium on Logic in Computer Science (**LICS'15**), 2015.

[BKKPRZ17] M. Bienvenu, S. Kikot, R. Kontchakov, V. Podolskii, V. Ryzhikov and M. Zakharyaschev: The Complexity of Ontology-Based Data Access with OWL 2 QL and Bounded Treewidth Queries. Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (**PODS'17**), 2017.

[BKKPZ18] M. Bienvenu, S. Kikot, R. Kontchakov, V. Podolskii, and M. Zakharyaschev: Ontology-Mediated Queries: Combined Complexity and Succinctness of Rewritings via Circuit Complexity. Journal of the ACM (**JACM**), 2018.

Original definition: class of decision problems logspace-reducible to the membership problem for context-free languages

Characterization in terms of circuits: solvable by uniform family of polysize, logarithmic-depth circuits, whose AND gates have fan-in 2

(called SAC^1 circuits)

Yet another definition: problems solvable by non-deterministic polytime logspace-bounded TM augmented with a stack

Relationship to other classes:

$$\textsc{LogSpace} \subseteq \mathrm{NL} \subseteq \mathrm{LOGCFL} \subseteq \mathrm{NC}^2 \subseteq \mathrm{P} \subseteq \mathrm{NP}$$

Considered highly parallelizable

Devise procedure that can be implemented by non-deterministic polytime logspace-bounded TM augmented with a stack

Devise procedure that can be implemented by non-deterministic polytime logspace-bounded TM augmented with a stack

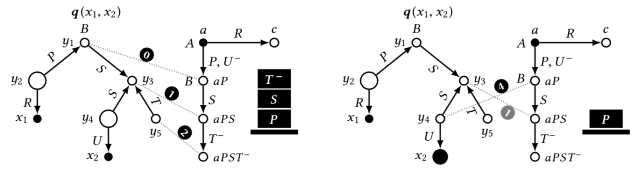
Idea: guess homomorphism into canonical model, use stack to store word part w of domain element aw in canonical model

Devise procedure that can be implemented by non-deterministic polytime logspace-bounded TM augmented with a stack

Idea: guess homomorphism into canonical model, use stack to store word part w of domain element aw in canonical model

Difficulty: need to store several words, but have only one stack!

Solution: 'synchronize' traversal of different branches

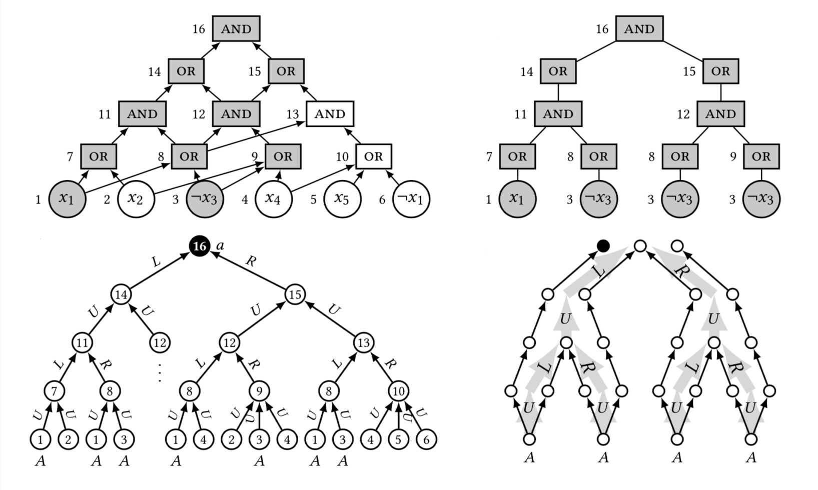


Reduction from SAC^1 acceptance problem:
decide whether an input of length n is accepted by the nth circuit of a logspace-uniform family of SAC^1 circuits

Use characterization of acceptance in terms of proof trees:

- **associate skeleton proof tree $Skel_C$ to each circuit C**
- label each node in skeleton with gate from C
- **circuit C accepts input $\sigma \Leftrightarrow$ valid labelling of $Skel_C$**
  - labelling respects the structure of C
  - leaves in $Skel_C$ mapped to input gates which are 1 under σ

EXAMPLE: SAC[1] CIRCUIT AND SKELETON PROOF TREE



Reduction from SAC^1 acceptance problem: decide whether an input of length n is accepted by the nth circuit of a logspace-uniform family of SAC^1 circuits

Use characterization of acceptance in terms of proof trees:

- associate **skeleton proof tree** $Skel_C$ to each circuit C
- label each node in skeleton with gate from C
- **circuit C accepts input σ $\Leftrightarrow$ valid labelling of $Skel_C$**
  - labelling respects the structure of C
  - leaves in $Skel_C$ mapped to input gates which are 1 under σ

Sketch of reduction:

- TBox generates tree-unfolding of circuit C, input gates marked 1, 0
- linear query corresponds to depth-first traversal of $Skel_C$
- query holds $\Leftrightarrow$ valid labelling of $Skel_C$

Upper bound on time needed to evaluate our NDL-rewritings:

- depth d / number of leaves ℓ occur in the exponent

Upper bound on time needed to evaluate our NDL-rewritings:

- depth d / number of leaves ℓ occur in the exponent

Is it possible to do better?

- formally: fixed-parameter tractable (FPT)? $f(d, \ell) \cdot p(|q|, |\mathcal{T}|, |\mathcal{A}|)$

Upper bound on time needed to evaluate our NDL-rewritings:

- depth d / number of leaves ℓ occur in the exponent

Is it possible to do better?

- formally: fixed-parameter tractable (FPT)? $f(d, \ell) \cdot p(|q|, |\mathcal{T}|, |\mathcal{A}|)$

Parameterized complexity of answering tree-shaped OMQs $(\mathcal{T}, q)$:

- parameters: depth d of $\mathcal{T}$, number ℓ of leaves in CQs

Upper bound on time needed to evaluate our NDL-rewritings:

- depth d / number of leaves ℓ **occur in the exponent**

Is it **possible to do better**?

- formally: **fixed-parameter tractable (FPT)**? $f(d,\ell) \cdot p(|q|, |\mathcal{T}|, |\mathcal{A}|)$

Parameterized complexity of answering tree-shaped OMQs $(\mathcal{T}, q)$:

- parameters: depth d of $\mathcal{T}$, number ℓ of leaves in CQs
- **not FPT** if **depth** d taken as parameter W[2]-hard
- **not FPT** if number of **leaves** ℓ taken as parameter W[1]-hard

Message: for good performance, **want d and ℓ small**