Reasoning with Data
• Names: Ontology Based Query Answering

• Sometimes OBDA (Ontology Based Data Access)

• Scenario:
  • data is incomplete
  • but is supplemented with additional knowledge
  • typically in the form of an ontology
  • query answering takes into account both
Ontology-based Query Answering

Answer = \{Alice\}
Ontology-based Query Answering

Answer = { }

Professor(Alice)
Teaches(Bob,CS100)

TeachingStaff(x)
Ontology-based Query Answering

Professors are teaching staff
Someone who teaches is teaching staff

Professor(Alice)
Teaches(Bob,CS100)

TeachingStaff(x)
Ontology-based Query Answering

\[ \forall x \ (\text{Professor}(x) \rightarrow \text{TeachingStaff}(x)) \]
\[ \forall x \forall y \ (\text{Teaches}(x, y) \rightarrow \text{TeachingStaff}(x)) \]

Professor(Alice)
Teaches(Bob, CS100)

Answer = \{Alice, Bob\}
Ontology-based Data Access: Architecture

- **Ontology**: provides a unified conceptual “global view” of the data
- **Data Sources**: external and independent (possibly multiple and heterogeneous)
- **Mapping**: semantically link data at the sources with the ontology
Query Answering in OBDA

- The sources and the mapping define a virtual data layer $M(D)$
Query Answering in OBDA

- The sources and the mapping define a virtual data layer $M(D)$
- Queries are answered against the knowledge base $\langle M(D), O \rangle$
Query Answering in OBDA

Ontology-Based Query Answering

OBDA

Ontology O

Virtual Data Layer

Query Q

M(D)
Ontology-based Query Answering (OBQA)

Certain-Answers($Q, \langle D, O \rangle$) = \bigcap_{M \in \text{models}(D \land O)} Q(M)

(formal definitions later - once we fix the languages)
Issues in Ontology-based Query Answering

**What is the right ontology language?**

- A wide spectrum of languages that differ in expressive power and computational complexity (e.g., description logics, existential rules)
- Data tractability is a key property to be useful in practice

**What is the right query language?**

- Well-known database query languages (e.g., conjunctive queries)
Few Words on Description Logics (DLs)

• DLs are well-behaved fragments of first-order logic
• Several DL-based languages exist (from lightweight to very expressive logics)
• Strongly influenced the W3C standard Web Ontology Language OWL

• Syntax: We start from a vocabulary with
  o Concept names: atomic classes or unary predicates - Parent, Person
  o Role names: atomic relations or binary predicates - HasParent

  and we build axioms
  o Person $\subseteq \exists \text{HasParent}.\text{Parent}$ - each person has a parent
  o Parent $\subseteq \text{Person}$ - each parent is a person

• Semantics: Standard first-order semantics
### DL-Lite Family

**DL-Lite**: Popular family of DLs - at the basis of the OWL 2 QL profile of OWL 2

<table>
<thead>
<tr>
<th>DL-Lite Axioms</th>
<th>First-order Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⊆ B</td>
<td>∀x (A(x) → B(x))</td>
</tr>
<tr>
<td>A ⊆ ∃R</td>
<td>∀x (A(x) → ∃y R(x,y))</td>
</tr>
<tr>
<td>∃R ⊆ A</td>
<td>∀x∀y (R(x,y) → A(x))</td>
</tr>
<tr>
<td>∃R ⊆ ∃P</td>
<td>∀x∀y (R(x,y) → ∃z P(x,z))</td>
</tr>
<tr>
<td>A ⊆ ∃R.B</td>
<td>∀x (A(x) → ∃y (R(x,y) ∧ B(y)))</td>
</tr>
<tr>
<td>R ⊆ P</td>
<td>∀x∀y (R(x,y) → P(x,y))</td>
</tr>
<tr>
<td>A ⊆ ¬B</td>
<td>∀x (A(x) ∧ B(x) → ⊥)</td>
</tr>
</tbody>
</table>
## The Description Logic EL

**EL**: Popular DL for biological applications - at the basis of the OWL 2 EL profile

<table>
<thead>
<tr>
<th>EL Axioms</th>
<th>First-order Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \sqsubseteq B$</td>
<td>$\forall x \ (A(x) \rightarrow B(x))$</td>
</tr>
<tr>
<td>$A \sqcap B \sqsubseteq C$</td>
<td>$\forall x \ (A(x) \land B(x) \rightarrow C(x))$</td>
</tr>
<tr>
<td>$A \sqsubseteq \exists R.B$</td>
<td>$\forall x \ (A(x) \rightarrow \exists y \ (R(x,y) \land B(y)))$</td>
</tr>
<tr>
<td>$\exists R.B \sqsubseteq A$</td>
<td>$\forall x \forall y \ (R(x,y) \land B(y) \rightarrow A(x))$</td>
</tr>
</tbody>
</table>

…several other, more expressive, description logics exist
A Simple Example

\[ \forall x \ (\text{Researcher}(x) \rightarrow \exists y \ (\text{WorksFor}(x,y) \land \text{Project}(y))) \]

\[ \forall x \ (\text{Project}(x) \rightarrow \exists y \ (\text{WorksFor}(y,x) \land \text{Researcher}(y))) \]

\[ \forall x \forall y \ (\text{WorksFor}(x,y) \rightarrow \text{Researcher}(x) \land \text{Project}(y)) \]

\[ \forall x \ (\text{Project}(x) \rightarrow \exists y \ (\text{ProjectName}(x,y))) \]
Some Terminology

• Our basic vocabulary:
  o A countable set $C$ of constants - domain of a database
  o A countable set $N$ of (labeled) nulls - globally $\exists$-quantified variables
  o A countable set $V$ of (regular) variables - used in rules and queries

• A term is a constant, null or variable

• An atom has the form $P(t_1,\ldots,t_n)$ - $P$ is an n-ary predicate and $t_i$’s are terms

• An instance is a (possibly infinite) set of atoms with constants and nulls

• A database is a finite instance with only constants
Syntax of Existential Rules

An existential rule is a first-order sentence

$$\forall x \forall y (\varphi(x,y) \rightarrow \exists z \psi(x,z))$$

- $x,y$ and $z$ are tuples of variables of $V$
- $\varphi(x,y)$ and $\psi(x,z)$ are conjunctions of atoms (possibly with constants)

...a.k.a. tuple-generating dependencies and Datalog$^\pm$ rules
Homomorphism

• Semantics of existential rules via the key notion of homomorphism

• A substitution from a set of symbols $S$ to a set of symbols $T$ is a function $h : S \rightarrow T$ - $h$ is a set of mappings of the form $s \mapsto t$, where $s \in S$ and $t \in T$

• A homomorphism from a set of atoms $A$ to a set of atoms $B$ is a substitution $h : C \cup N \cup V \rightarrow C \cup N \cup V$ such that:

  (i) $t \in C \Rightarrow h(t) = t$

  (ii) $P(t_1,\ldots, t_n) \in A \Rightarrow h(P(t_1,\ldots, t_n)) = P(h(t_1),\ldots, h(t_n)) \in B$

• Can be naturally extended to conjunctions of atoms
Semantics of Existential Rules

• An instance $J$ is a model of the existential rule

$$\rho = \forall x \forall y \ (\varphi(x, y) \rightarrow \exists z \ \psi(x, z))$$

written as $J \models \rho$, if the following holds:

whenever there exists a homomorphism $h$ such that $h(\varphi(x, y)) \subseteq J$,
then there exists $g \supseteq h|_x$ such that $g(\psi(x, z)) \subseteq J$

\[ \{ t \mapsto h(t) \mid t \in x \} - \text{the restriction of } h \text{ to } x \]

• Given a set $O$ of existential rules, $J$ is a model of $O$, written as $J \models O$, if the following holds: for each $\rho \in O$, $J \models \rho$
Ontology Based Query Answering (OBQA)

database

ontology

knowledge base

existential / Datalog\pm rules

\forall x \forall y (\varphi(x,y) \rightarrow \exists z \psi(x,z))
Query Languages

- The four most important query languages
  - Conjunctive Queries (CQ)
  - Unions of Conjunctive Queries (UCQ)
  - First-order Queries (FO)
  - Datalog
A conjunctive query (CQ) is an expression

\[ \exists y \ (\varphi(x,y)) \quad \text{or} \quad \text{Ans}(x) \leftarrow \varphi(x,y) \]

- \( x \) and \( y \) are tuples of variables of \( V \)
- \( \varphi(x,y) \) is a conjunction of atoms (possibly with constants)

The most important query language used in practice

Forms the SELECT-FROM-WHERE fragment of SQL
Semantics of Conjunctive Queries

• A match of a CQ $\exists y \ (\varphi(x,y))$ in an instance $J$ is a homomorphism $h$ such that $h(\varphi(x,y)) \subseteq J$ - all the atoms of the query are satisfied

• The answer to $Q(x) = \exists y \ (\varphi(x,y))$ over $J$ is the set of tuples $Q(J) = \{h(x) \in C \mid h$ is a match of $Q$ in $J\}$

• The answer consists of the witnesses for the free variables of the query
 Conjunctive Queries: Example

Find the researchers who work for the “VADA” project

\[ \exists y \ (\text{Researcher}(x) \land \text{WorksFor}(x, y) \land \text{Project}(y) \land \text{ProjectName}(y, \text{“VADA”})) \]

\[
\text{SELECT} \ R.\text{id} \\
\text{FROM} \ \text{Researcher} \ R, \ \text{WorksFor} \ W, \ \text{Project} \ P, \ \text{ProjectName} \ N \\
\text{WHERE} \ R.\text{id} = W.\text{rid} \ \text{AND} \\
\quad W.\text{pid} = P.\text{id} \ \text{AND} \\
\quad P.\text{id} = N.\text{pid} \ \text{AND} \\
\quad N.\text{name} = \text{“VADA”}
\]
Ontology-based Query Answering (OBQA)

- Database
- Knowledge base $\langle D, O \rangle$
- Ontology

Existential / Datalog± rules:
$$\forall x \forall y (\varphi(x, y) \rightarrow \exists z \psi(x, z))$$

Conjunctive queries:
$$\exists y (\varphi(x, y))$$
Ontology-based Query Answering (OBQA)

Certain-Answers(Q,⟨D,O⟩) = \bigcap_{M \in \text{models}(D \land O)} Q(M)

{J | J \supseteq D \text{ and } J \not\models O}
OBQA: Formal Definition

Input: database $D$, ontology $O \in L$, CQ $Q(x) = \exists y (\varphi(x,y))$, tuple $t \in \text{adom}(D)^{|x|}$

Question: $t \in \text{Certain-Answers}(Q,\langle D,O \rangle) = \bigcap_{M \in \text{models}(D \land O)} Q(M)$?
OBQA: Complexity Metrics

- **Combined complexity** - everything is part of the input

- **Data complexity** - only $D$ and $t$ are part of the input

\[
\text{OBQA}[O,Q]
\]

Input: database $D$, tuple $t \in \text{adom}(D)^{|x|}$

Question: $t \in \text{Certain-Answers}(Q,\langle D,O \rangle)$?

\[
\text{OBQA}(L) \text{ is } \mathcal{C}\text{-complete in data complexity if:}
\]

1. For every $O \in L$ and CQ $Q$, OBQA$[O,Q]$ is in $\mathcal{C}$

2. There exists $O \in L$ and CQ $Q$ such that OBQA$[O,Q]$ is $\mathcal{C}$-hard
OBQA: The Boolean Case

OBQA(L)

Input: database D, ontology O ∈ L, CQ Q(x) = ∃y (φ(x,y)), tuple t ∈ adom(D)|x|

Question: t ∈ Certain-Answers(Q,⟨D,O⟩) = \bigcap_{M ∈ \text{models}(D \land O)} Q(M)?

\[
t ∈ \text{Certain-Answers}(Q,⟨D,O⟩) \iff \forall M ∈ \text{models}(D \land O), M ⊨ ∃y (φ(t,y))

\iff D \land O ⊨ ∃y (φ(t,y))

Boolean CQ - no free variables
OBQA: The Boolean Case

OBQA(\textbf{L})
Input: database \textit{D}, ontology \textit{O} \in \textbf{L}, CQ \textit{Q(x)} = \exists y (\varphi(x,y))$, tuple $t \in \text{adom}(D)^{|X|}$

Question: $t \in \text{Certain-Answers}(Q,\langle D,O \rangle) = \bigcap_{M \in \text{models}(D \land O)} Q(M)$?

For understanding the complexity of OBQA(\textbf{L}), it suffices to focus on Boolean CQs

\begin{tabular}{|l|}
\hline
\textbf{OBQA(\textbf{L})} \\
\hline
Input: database \textit{D}, ontology \textit{O} \in \textbf{L}, Boolean CQ \textit{Q} \\
\hline
Question: $D \land O \vdash Q$? \\
\hline
\end{tabular}
Why is OBQA technically challenging?

What is the right tool for tackling this problem?
The Two Dimensions of Infinity

Consider the database $D$, and the ontology $O$

$D \land O$ admits infinitely many models, of possibly infinite size
The Two Dimensions of Infinity

\[ D = \{P(c)\} \quad O = \{\forall x \ (P(x) \rightarrow \exists y \ (R(x,y) \land P(y)))\} \]

\[ \text{model of } D \land O \]

\[ \text{size} \]

\[ z_1, z_2, z_3, \ldots \text{ are nulls of } N \]
Taming the First Dimension of Infinity

\[ D = \{P(c)\} \quad O = \{\forall x \ P(x) \rightarrow \exists y \ (R(x,y) \land P(y))\} \]

**Key Idea:** Focus on a representative, a model that is as general as possible

---

**Diagram:**

A visual representation of the models for \( D \) and \( O \) with specific instances demonstrating the focus on a general model.
Universal Models (a.k.a. Canonical Models)

An instance $U$ is a universal model of $D \land O$ if the following holds:

1. $U$ is a model of $D \land O$

2. $\forall J \in \text{models}(D \land O)$, there exists a homomorphism $h_J$ such that $h_J(U) \subseteq J$
Query Answering via Universal Models

**Theorem:** $D \land O \models Q$ iff $U \models Q$, where $U$ is a universal model of $D \land O$

**Proof:**

$(\Rightarrow)$ Trivial since, for every $J \in \text{models}(D \land O)$, $J \models Q$

$(\Leftarrow)$ By exploiting the universality of $U$

\[
\forall J \in \text{models}(D \land O), \exists h_J \text{ such that } h_J(g(Q)) \subseteq J \quad \Rightarrow \quad \forall J \in \text{models}(D \land O), J \models Q
\]

\[
\Rightarrow \quad D \land O \models Q
\]
The Chase Procedure

• Fundamental algorithmic tool used in databases

• It has been applied to a wide range of problems:
  o Checking containment of queries under constraints
  o Computing data exchange solutions
  o Computing certain answers in data integration settings
  o ...

... what’s the reason for the ubiquity of the chase in databases?

it constructs universal models
The Chase Procedure

\[ \forall x \ (\text{Person}(x) \rightarrow \exists y \ (\text{HasParent}(x,y) \land \text{Person}(y))) \]

\[ \text{chase}(D,O) = D \cup \]
The Chase Procedure

\[ \forall x \ (\text{Person}(x) \rightarrow \exists y \ (\text{HasParent}(x,y) \land \text{Person}(y))) \]

\[ \text{chase}(D,O) = D \cup \{\text{HasParent}(\text{John}, z_1), \text{Person}(z_1)\} \]
The Chase Procedure

\[ \forall x \ (\text{Person}(x) \rightarrow \exists y \ (\text{HasParent}(x,y) \land \text{Person}(y))) \]

\[ \text{chase}(D,O) = D \cup \{ \text{HasParent}(\text{John}, z_1), \text{Person}(z_1), \text{HasParent}(z_1, z_2), \text{Person}(z_2) \} \]
The Chase Procedure

\[
\forall x \ (\text{Person}(x) \rightarrow \exists y \ (\text{HasParent}(x,y) \land \text{Person}(y)))
\]

\[
\text{chase}(D,O) = D \cup \{\text{HasParent}(\text{John}, z_1), \text{Person}(z_1), \\
\text{HasParent}(z_1, z_2), \text{Person}(z_2), \\
\text{HasParent}(z_2, z_3), \text{Person}(z_3)\}
\]
The Chase Procedure

\[ \text{chase}(D,O) = D \cup \{ \text{HasParent}(\text{John}, z_1), \text{Person}(z_1), \text{HasParent}(z_1, z_2), \text{Person}(z_2), \text{HasParent}(z_2, z_3), \text{Person}(z_3), \ldots \} \]

infinite instance
The Chase Procedure: Formal Definition

- **Chase rule** - the building block of the chase procedure

- A rule $\rho = \forall x \forall y \ (\varphi(x,y) \rightarrow \exists z \ \psi(x,z))$ is applicable to instance $J$ if:
  1. There exists a homomorphism $h$ such that $h(\varphi(x,y)) \subseteq J$
  2. There is no $g \supseteq h_x$ such that $g(\psi(x,z)) \subseteq J$

```
J = \{R(a), P(a,b)\}

h = \{x \mapsto a\}
\forall x \ (R(x) \rightarrow \exists y \ P(x,y))

\xmark

\checkmark
```

```
J = \{R(a), P(b,a)\}

h = \{x \mapsto a\}
\forall x \ (R(x) \rightarrow \exists y \ P(x,y))

\xmark
```
The Chase Procedure: Formal Definition

• **Chase rule** - the building block of the chase procedure

• A rule $\rho = \forall x \forall y (\varphi(x,y) \rightarrow \exists z \psi(x,z))$ is applicable to instance $J$ if:
  1. There exists a homomorphism $h$ such that $h(\varphi(x,y)) \subseteq J$
  2. There is no $g \supseteq h_x$ such that $g(\psi(x,z)) \subseteq J$

• Let $J_+ = J \cup \{g(\psi(x,z))\}$, where $g \supseteq h_x$ and $g(z)$ are “fresh” nulls not in $J$

• The result of applying $\rho$ to $J$ is $J_+$, denoted $J(\rho,h)J_+ - \text{ single chase step}$
The Chase Procedure: Formal Definition

- A **finite chase** of $D$ w.r.t. $O$ is a finite sequence

$$D\langle \rho_1, h_1 \rangle J_1 \langle \rho_2, h_2 \rangle J_2 \langle \rho_3, h_3 \rangle J_3 \ldots \langle \rho_n, h_n \rangle J_n$$

and $\text{chase}(D, O)$ is defined as the instance $J_n$

- An **infinite chase** of $D$ w.r.t. $O$ is a **fair** infinite sequence

$$D\langle \rho_1, h_1 \rangle J_1 \langle \rho_2, h_2 \rangle J_2 \langle \rho_3, h_3 \rangle J_3 \ldots \langle \rho_n, h_n \rangle J_n \ldots$$

and $\text{chase}(D, O)$ is defined as the instance $\bigcup_{k \geq 0} J_k$ (with $J_0 = D$)

all applicable rules will eventually be applied

least fixpoint of a monotonic operator - the chase step
**Theorem:** chase(D,O) is a universal model of D \land O

**Proof (sketch):**
- By construction, chase(D,O) \in \text{models}(D \land O)
- It remains to show that chase(D,O) can be homomorphically embedded into every other model of D \land O
- Fix an arbitrary instance J \in \text{models}(D \land O). We need to show that there exists h such that h(chase(D,O)) \subseteq J
- By induction on the number of applications of the chase step, we show that for every k \geq 0, there exists h_k such that h_k(chase^{[k]}(D,O)) \subseteq J, and h_k is compatible with h_{k-1}
- Clearly, \bigcup_{k \geq 0} h_k is a well-defined homomorphism that maps chase(D,O) to J
- The claim follows with h = \bigcup_{k \geq 0} h_k

the result of the chase after k applications of the chase step
Chase: Uniqueness Property

• The result of the chase is **not unique** - depends on the order of rule application

D = \{P(a)\} \quad \rho_1 = \forall x \ (P(x) \rightarrow \exists y \ R(y)) \quad \rho_2 = \forall x \ (P(x) \rightarrow R(x))

\begin{align*}
\text{Result}_1 &= \{P(a), R(z), R(a)\} \\
\text{Result}_2 &= \{P(a), R(a)\} \\
\text{Result}_3 &= \text{Result}_1 \text{ after } \rho_1 \text{ then } \rho_2 \\
\text{Result}_4 &= \text{Result}_2 \text{ after } \rho_2 \text{ then } \rho_1
\end{align*}

• But, it is **unique up to homomorphically equivalent**

⇒ it is unique for query answering purposes
Query Answering via the Chase

**Theorem:** $D \land O \models Q$ iff $U \models Q$, where $U$ is a universal model of $D \land O$

&

**Theorem:** $\text{chase}(D,O)$ is a universal model of $D \land O$

\[\Downarrow\]

**Corollary:** $D \land O \models Q$ iff $\text{chase}(D,O) \models Q$

We can tame the first dimension of infinity by exploiting the chase procedure
Can we tame the second dimension of infinity?
Undecidability of OBQA

**Theorem:** OBQA(∃RULES) is undecidable

**Proof Idea:** By simulating a deterministic Turing machine with an empty tape.
Gaining Decidability

By restricting the database
- \{\text{Start}(c)\} \land O \vdash Q \iff \text{the Turing Machine } T \text{ accepts}
- The problem is undecidable already for singleton databases

By restricting the query language
- D \land O \vdash \exists x \text{ Accept}(x) \iff \text{the Turing Machine } T \text{ accepts}
- The problem is undecidable already for atomic queries

By restricting the ontology language
- Achieve a good trade-off between expressive power and complexity
- Field of intense research (Calabria, Dresden, Edinburgh, Montpellier, Oxford, Vienna)
Datalog± Nomenclature

- Extend Datalog by allowing in the head:
  - Existential quantification (∃)
  - Equality atoms (=)
  - Constant false (⊥)

Datalog[∃,=,⊥]

a highly expressive ontology language
Datalog\(\pm\) Nomenclature

- Extend Datalog by allowing in the head:
  - Existential quantification (\(\exists\))
  - Equality atoms (\(=\))
  - Constant false (\(\bot\))

- But, already Datalog[\(\exists\)] is undecidable

- Datalog[\(\exists,=,\bot\)] is syntactically restricted \(\rightarrow\) Datalog\(\pm\)
Gaining Decidability

By restricting the database

• \( \{ \text{Start}(c) \} \land O \models Q \iff \text{the DTM M accepts} \)
• The problem is undecidable already for singleton databases

By restricting the query language

• \( D \land O \models \exists x \text{ Accept}(x) \iff \text{the DTM M accepts} \)
• The problem is undecidable already for atomic queries

By restricting the ontology language

• Achieve a good trade-off between expressive power and complexity
• Field of intense research (Calabria, Dresden, Edinburgh, Montpellier, Oxford, Vienna)
What is the Source of Non-termination?

\[ \text{D} \]

\[ \text{Person(John)} \]

\[ \forall x \ (\text{Person}(x) \rightarrow \exists y \ (\text{HasParent}(x,y) \land \text{Person}(y))) \]

\[ \text{chase(D,O) = D} \cup \{\text{HasParent}(\text{John}, \text{z}_1), \text{Person(\text{z}_1)}, \text{HasParent(\text{z}_1, \text{z}_2)}, \text{Person(\text{z}_2)}, \text{HasParent(\text{z}_2, \text{z}_3)}, \text{Person(\text{z}_3)}, \ldots \} \]

1. Existential quantification
2. Recursive definitions
Termination of the Chase

- Drop existential quantification
  - We obtain the class of **full** existential rules
  - Very close to Datalog

- Drop recursive definitions
  - We obtain the class of **acyclic** existential rules
  - A.k.a. non-recursive existential rules
Recall our Example

\[ \forall x \ (\text{Person}(x) \rightarrow \exists y \ (\text{HasParent}(x,y) \land \text{Person}(y))) \]

\[ \text{chase}(D,O) = D \cup \{\text{HasParent}(\text{John}, z_1), \text{Person}(z_1), \]
\[ \text{HasParent}(z_1, z_2), \text{Person}(z_2), \]
\[ \text{HasParent}(z_2, z_3), \text{Person}(z_3), \ldots \] 

The above rule can be written as the DL-Lite axiom

\[ \text{Person} \sqsubseteq \exists \text{HasParent}.\text{Person} \]
Recall our Example

\[ \text{chase}(D, O) = D \cup \{\text{HasParent}(\text{John}, z_1), \text{Person}(z_1), \text{HasParent}(z_1, z_2), \text{Person}(z_2), \text{HasParent}(z_2, z_3), \text{Person}(z_3), \ldots \} \]

\[ \forall x (\text{Person}(x) \rightarrow \exists y (\text{HasParent}(x, y) \land \text{Person}(y))) \]

Existential quantification & recursive definitions are key features for modelling ontologies
Research Challenge

We need classes of existential rules such that:

1. Existential quantification and recursive definitions coexist
2. OBQA is decidable, and tractable in the data complexity

Tame the infinite chase:
Deal with infinite instances without explicitly building them
Linear Existential Rules

- A linear existential rule is an existential rule of the form

\[ \forall x \forall y \ (P(x,y) \rightarrow \exists z \ \psi(x,z)) \]

- We denote **LINEAR** the ontology language based on linear existential rules

- A local property - we can inspect one rule at a time

  \[ \Rightarrow \text{given } O, \text{ we can decide in linear time whether } O \in \text{LINEAR} \]

  \[ \Rightarrow \text{closed under union} \]

- But, is this a reasonable ontology language?
**LINEAR vs. DL-Lite**

**DL-Lite:** Popular family of DLs - at the basis of the OWL 2 QL profile of OWL 2

<table>
<thead>
<tr>
<th>DL-Lite Axioms</th>
<th>First-order Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⊆ B</td>
<td>∀x (A(x) → B(x))</td>
</tr>
<tr>
<td>A ⊆ ∃R</td>
<td>∀x (A(x) → ∃y R(x,y))</td>
</tr>
<tr>
<td>∃R ⊆ A</td>
<td>∀x∀y (R(x,y) → A(x))</td>
</tr>
<tr>
<td>∃R ⊆ ∃P</td>
<td>∀x∀y (R(x,y) → ∃z P(x,z))</td>
</tr>
<tr>
<td>A ⊆ ∃R.B</td>
<td>∀x (A(x) → ∃y (R(x,y) ∧ B(y)))</td>
</tr>
<tr>
<td>R ⊆ P</td>
<td>∀x∀y (R(x,y) → P(x,y))</td>
</tr>
<tr>
<td>A ⊆ ¬B</td>
<td>∀x (A(x) ∧ B(x) → ⊥)</td>
</tr>
</tbody>
</table>
Linear Existential Rules

• A linear existential rule is an existential rule of the form

\[ \forall x \forall y \ (P(x,y) \rightarrow \exists z \ \psi(x,z)) \]

• We denote **LINEAR** the ontology language based on linear existential rules

• A local property - we can inspect one rule at a time

  \[ \Rightarrow \text{given } O, \text{ we can decide in linear time whether } O \in \text{ LINEAR} \]

  \[ \Rightarrow \text{closed under union} \]

• But, is this a reasonable ontology language? **OWL 2 QL**
Chase Graph

The chase can be naturally seen as a graph - chase graph

$$D = \{R(a,b), S(b)\}$$

$$O = \{\forall x \forall y \ (R(x,y) \land S(y) \rightarrow \exists z \ R(z,x)) \land \forall x \forall y \ (R(x,y) \rightarrow S(x))\}$$

For **LINEAR** the chase graph is a **forest**
**Definition:** An ontology language $L$ enjoys the BDDP if:

for every ontology $O \in L$ and CQ $Q$, there exists $k \geq 0$ such that,

for every database $D$, $\text{chase}(D,O) \models Q \Rightarrow \text{chase}^k(D,O) \models Q$
**Bounded Derivation-depth Property (BDDP)**

**Definition:** An ontology language $L$ enjoys the BDDP if:

for every ontology $O \in L$ and CQ $Q$, there exists $k \geq 0$ such that,

for every database $D$, $\text{chase}(D, O) \models Q \Rightarrow \text{chase}^k(D, O) \models Q$

For **LINEAR**, $k = |Q| \cdot m$

with $m = |\text{sch}(O)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

predicates occurring in $O$
The Blocking Algorithm for **LINEAR**

The blocking algorithm shows that OBQA(LINEAR) is

- in 2EXPTIME in combined complexity
- in PTIME in data complexity

\[ k = |Q| \cdot |\text{sch}(O)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}} \]
Complexity of OBQA(\textbf{LINEAR})

…but, we can do better than the blocking algorithm

\textbf{Theorem}: OBQA(\textbf{LINEAR}) is

\begin{itemize}
  \item PSPACE-complete in combined complexity
  \item in LOGSPACE in data complexity
\end{itemize}
Key Observation

at most $|Q|$ atoms

non-deterministic, level-by-level construction
Combined Complexity of **LINEAR**

**Theorem:** $\text{OBQA}(\text{LINEAR})$ is in PSPACE

**Proof (high-level idea):**

$L_0 = D$
Combined Complexity of **LINEAR**

**Theorem:** OBQA**(LINEAR)** is in PSPACE

**Proof (high-level idea):**

\[
L_0 = D \\
L_1
\]
Combined Complexity of **LINEAR**

**Theorem:** $\text{OBQA(\text{LINEAR})}$ is in PSPACE

**Proof (high-level idea):**

\[ L_1 \]
\[ L_2 \]
Theorem: OBQA(LINEAR) is in PSPACE

Proof (high-level idea):
**Theorem:** OBQA(LINEAR) is in PSPACE

**Proof (high-level idea):**

\[
\vdots
\]

\[L_n\]
Theorem: \( \text{OBQA} (\text{LINEAR}) \) is in PSPACE

Proof (high-level idea):

- At each step we need to maintain
  - \( \mathcal{O}(|Q|) \) atoms
  - A counter \( \text{ctr} \leq |Q|^2 \cdot |\text{sch}(O)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}} \)

- Thus, we need polynomial space

- The claim follows since \( \text{NPSPACE} = \text{PSPACE} \)
Combined Complexity of **LINEAR**

We cannot do better than the previous algorithm

**Theorem:** $OBQA(\text{LINEAR})$ is PSPACE-hard

**Proof Idea:** By simulating a deterministic polynomial space Turing machine
Complexity of OBQA(\textbf{LINEAR})

\textbf{Theorem:} OBQA(\textbf{LINEAR}) is

- PSPACE-complete in combined complexity
- in LOGSPACE in data complexity
Query Rewriting

∀D : D ∧ O ⊨ Q ⇔ D ⊨ Q₀
Query Rewriting

**Theorem:** OBQA(L) is UCQ-rewritable

⇒ OBQA(L) is in LOGSPACE in data complexity

**Proof:** Fix $O \in L$ and CQ $Q$. We need to show that OBQA[$O,Q$] is in LOGSPACE:

1. Construct $Q_O$ in $O(1)$ time (due to UCQ rewritability)
2. Check whether $D \models Q_O$ in LOGSPACE (classical result)
Complexity of OBQA(\textbf{LINEAR})

\textbf{Theorem}: OBQA(\textbf{LINEAR}) is

- PSPACE-complete in combined complexity
- in LOGSPACE in data complexity

…it suffices to show that OBQA(\textbf{LINEAR}) is UCQ-rewritable
Bounded Derivation-depth Property (BDDP)

**Definition:** An ontology language \( L \) enjoys the BDDP if:

for every ontology \( O \in L \) and CQ \( Q \), there exists \( k \geq 0 \) such that,

for every database \( D \), \( \text{chase}(D, O) \models Q \Rightarrow \text{chase}^k(D, O) \models Q \)
Bounded Derivation-depth Property (BDDP)

**Proposition:** $L$ enjoys the BDDP $\Rightarrow$ OBQA($L$) is UCQ-rewritable

- Each atom is obtained by at most $\beta^k$ atoms.
- To entail a CQ $Q$ we need at most $|Q| \cdot \beta^k$ database atoms.
Given an ontology $O \in L$ and a CQ $Q$: 

• $D_{\beta,\delta,q}$ be the set of all possible databases of size at most $|Q| \cdot \beta^\delta$

• $C = \{ D \in D_{\beta,\delta,q} \mid \text{chase}(D,O) \models Q \}$

• Convert $C$ into a UCQ
Theorem: OBQA(LINEAR) is

- PSPACE-complete in combined complexity
- in LOGSPACE in data complexity
Recap

• Ontology-based query answering under existential rules

• Technical challenges and the right technical tool (the chase)

• Tame the infinite chase: linear existential rules - key properties and complexity

…but, is **LINEAR** the ultimate ontology language?
Research Challenge

We need classes of existential rules such that:

1. Existential quantification and recursive definitions coexist
2. OBQA is decidable, and tractable in the data complexity

Tame the infinite chase:
Deal with infinite structures without explicitly building them
Transitive Closure

\[ \forall x \forall y \ (\text{ParentOf}(x,y) \rightarrow \text{AncestorOf}(x,y)) \]

\[ \forall x \forall y \forall z \ (\text{ParentOf}(x,y) \land \text{AncestorOf}(y,z) \rightarrow \text{AncestorOf}(x,z)) \]
IDB-Linear Existential Rules

• A predicate that does not occur in the head of a rule is extensional (EDB); otherwise, is intensional (IDB).

• A set of existential rules is IDB-linear if every rule is of the form

\[ \forall x \forall y (\varphi(x,y) \rightarrow \exists z \psi(x,z)) \]

single occurrence of an IDB predicate

• We denote IDB-LINEAR the obtained ontology language.
Transitive Closure

\[ \forall x \forall y \ (\text{ParentOf}(x,y) \rightarrow \text{AncestorOf}(x,y)) \]

\[ \forall x \forall y \forall z \ (\text{ParentOf}(x,y) \land \text{AncestorOf}(y,z) \rightarrow \text{AncestorOf}(x,z)) \]
Complexity of OBQA(IDB-LINEAR)

**Theorem:** OBQA(IDB-LINEAR) is

- PSPACE-complete in combined complexity
- NLOGSPACE-complete in data complexity
Complexity of IDB-LINEAR

Proof (high-level idea):

\[ L_0 = D \]

\[ L_1 \]

\[ L_2 \]

\[ L_3 \]

\[ \vdots \]

\[ L_n \]

non-deterministic

level-by-level construction

\[ \forall x \forall y \ (R(x,y) \land \varphi(x,y) \rightarrow \exists z \ \psi(x,z)) \]

and then apply the linear rule

\[ \forall x \forall y \ (R(h(x),h(y)) \rightarrow \exists z \ \psi(h(x),z)) \]
Complexity of OBQA($\text{IDB-LINEAR}$)

**Theorem:** OBQA($\text{IDB-LINEAR}$) is

- PSPACE-complete in combined complexity
- NLOGSPACE-complete in data complexity
But

\[ \forall x \forall y (\varphi(x,y) \rightarrow \exists z \psi(x,z)) \]

single occurrence of an IDB predicate

- We cannot have joins over null values
- We cannot express “complex” recursive definitions

...we need more sophisticated restrictions at the level of variables
Restrict the Use of Body-variables

Classification of body-variables

- **Harmless**: one that can be satisfied only by constants
- **Harmful**: one that is not harmless
- **Dangerous**: one that is harmful, and also appears in the rule-head

\[
\forall x \forall y \forall z \ (P(x,y), S(y,z) \rightarrow \exists w \ T(y,x,w))
\]

\[
\forall x \forall y \forall z \ (T(x,y,z) \rightarrow \exists w \ P(w,z))
\]

\[
\forall x \forall y \ (P(x,y) \rightarrow \exists z \ Q(x,z))
\]
Restrict the Use of Body-variables

Classification of body-variables

- **Harmless**: one that can be satisfied only by constants
- **Harmful**: one that is not harmless
- **Dangerous**: one that is harmful, and also appears in the rule-head

\[
\forall x \forall y \forall z (P(x,y), S(y,z) \rightarrow \exists w T(y,x,w))
\]

**Existential Positions**

\[
\forall x \forall y \forall z (T(x,y,z) \rightarrow \exists w P(w,z))
\]

\[
\forall x \forall y (P(x,y) \rightarrow \exists z Q(x,z))
\]

T[3], P[1], Q[2]
Restrict the Use of Body-variables

Classification of body-variables

- **Harmless**: one that can be satisfied only by constants
- **Harmful**: one that is not harmless
- **Dangerous**: one that is harmful, and also appears in the rule-head

\[
\forall x \forall y \forall z \ (P(x, y), S(y, z) \rightarrow \exists w \ T(y, x, w))
\]

Existential Positions

\[
\forall x \forall y \forall z \ (T(x, y, z) \rightarrow \exists w \ P(w, z))
\]

T[3], P[1], Q[2]

\[
\forall x \forall y \ (P(x, y) \rightarrow \exists z \ Q(x, z))
\]

T[2]
Restrict the Use of Body-variables

Classification of body-variables

- **Harmless**: one that can be satisfied only by constants
- **Harmful**: one that is not harmless
- **Dangerous**: one that is harmful, and also appears in the rule-head

\[
\forall x \forall y \forall z \ (P(x,y), S(y,z) \rightarrow \exists w \ T(y,x,w)) \quad \text{Existential Positions}
\]

\[
\forall x \forall y \forall z \ (T(x,y,z) \rightarrow \exists w \ P(w,z)) \quad \text{T[3], P[1], Q[2]}
\]

\[
\forall x \forall y \ (P(x,y) \rightarrow \exists z \ Q(x,z)) \quad \text{T[2], P[2]}
\]
Restrict the Use of Body-variables

Classification of body-variables

- **Harmless**: one that can be satisfied only by constants
- **Harmful**: one that is not harmless
- **Dangerous**: one that is harmful, and also appears in the rule-head

\[
\forall x \forall y \forall z (P(x,y), S(y,z) \rightarrow \exists w T(y,x,w)) \quad \text{Existential Positions}
\]

\[
\forall x \forall y \forall z (T(x,y,z) \rightarrow \exists w P(w,z)) \quad \text{T[3], P[1], Q[2]}
\]

\[
\forall x \forall y (P(x,y) \rightarrow \exists z Q(x,z)) \quad \text{T[2], P[2], Q[1]}
\]
Restrict the Use of Body-variables

Classification of body-variables

- **Harmless**: one that can be satisfied only by constants
- **Harmful**: one that is not harmless
- **Dangerous**: one that is harmful, and also appears in the rule-head

\[
\forall x \forall y \forall z \ (P(x,y), S(y,z) \rightarrow \exists w \ T(y,x,w))
\]

Existential Positions

- \(\forall x \forall y \forall z \ (T(x,y,z) \rightarrow \exists w \ P(w,z))\)
- \(\forall x \forall y \ (P(x,y) \rightarrow \exists z \ Q(x,z))\)

T[3], P[1], Q[2]

T[2], P[2], Q[1]
Weakly-Frontier-Guarded (WFG)

• A set of existential rules is WFG if every rule is of the form

\[ \forall x \forall y (\varphi(x,y) \rightarrow \exists z \psi(x,z)) \]

there exists a guard atom that contains all the dangerous variables

• We denote WFG the obtained ontology language
Complexity of OBQA(WFG)

**Theorem:** OBQA(WFG) is

- 2EXPTIME-complete in combined complexity
- EXPTIME-complete in data complexity

Source of complexity: The guard and the rest of the body share harmful variables
A set of existential rules is *warded* if every rule is of the form

\[ \forall x \forall y \ (G(x,y) \land \varphi(x,y) \rightarrow \exists z \ \psi(x,z)) \]

contains all the dangerous variables, and shares with \( \varphi(x,y) \) only harmless variables.

We denote **WARDED** the obtained ontology language.
Complexity of OBQA(WARDED)

**Theorem:** OBQA(WARDED) is

- EXPTIME-complete in combined complexity
- PTIME-complete in data complexity

a “nearly” maximal fragment of WFG

at least one occurrence of a dangerous variable that appears in the guard, appears outside the guard ⇒ EXPTIME-complete in data complexity
Warded + Stratified Negation

RDF Query Language
+ SPARQL
+ OWL 2 QL
+ OWL 2 RL

Ontology Language
OWL 2 QL
OWL 2 RL

Modeling Language
(Fragments of) UML and ER

DB Query Language
Datalog[¬strat]

...and provides the logical core of VADALOG