## the rest of this course

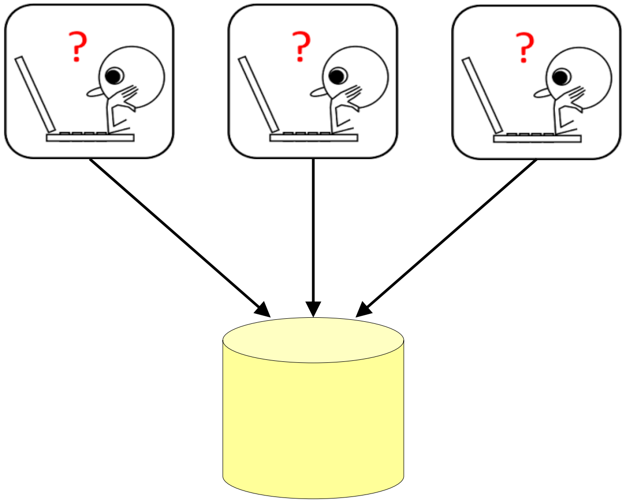
**Volume**
size does mattes
(thousands of TBs of data)
✓

**Veracity**
data is often
incomplete/inconsistent


**Variety**
many data formats
(structured, semi-structured, etc.)
✓

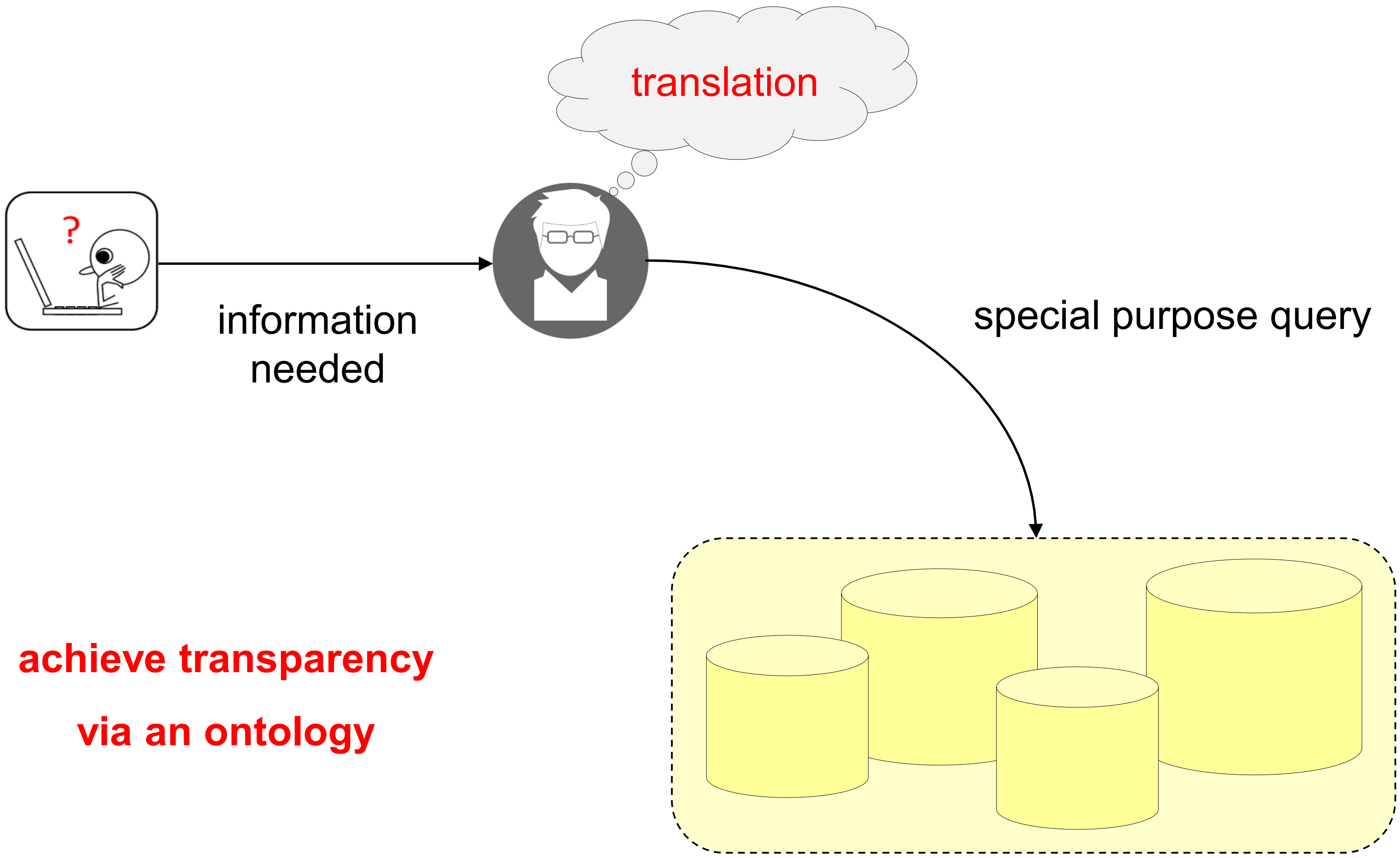
**Velocity**
data often arrives at fast speed
(updates are frequent)

# Ontology-Based Data Access



ideal information system


in many modern organizations




**achieve transparency via an ontology**

# What is an Ontology?

***An engineering artifact; its objective is to provide***

***an explicit specification of a conceptualization***

an abstract model of (some aspect of) the world

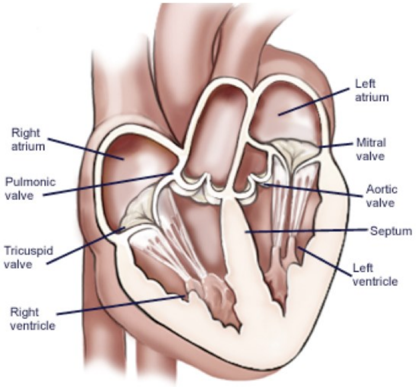
using unambiguous language, typically logic

# What is an Ontology?

1. Introduces vocabulary relevant to a domain

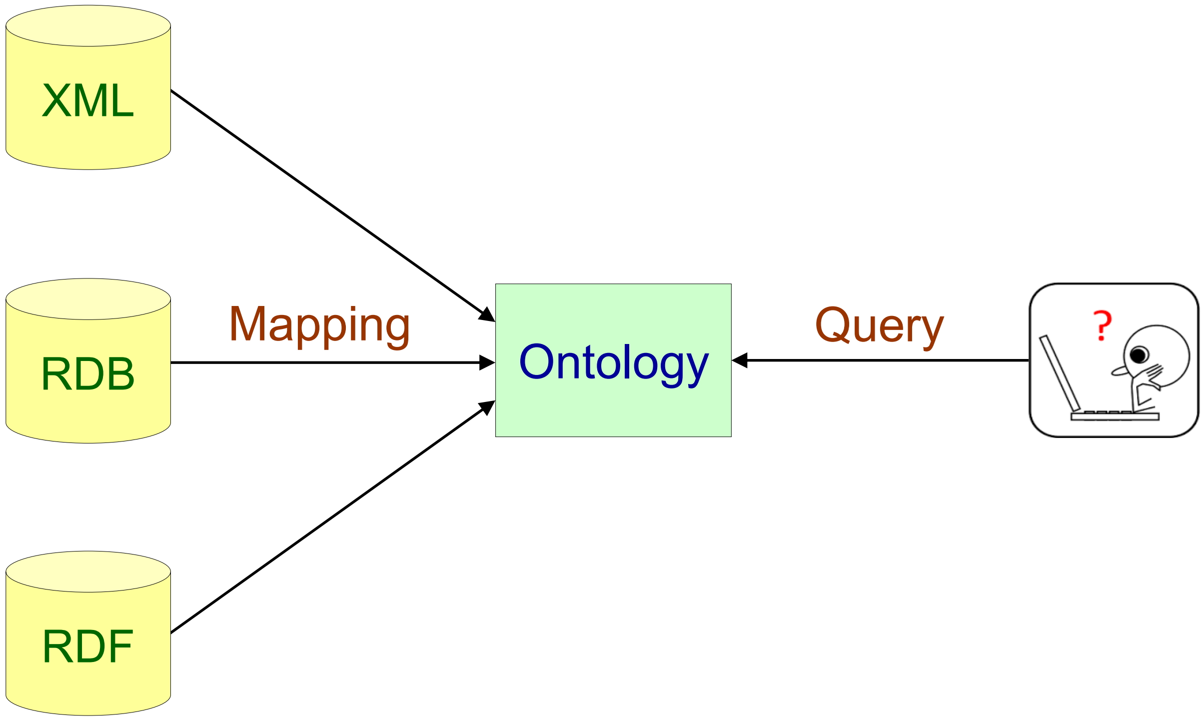
2. Specifies the meaning (semantics) of the terms

*Heart is a muscular organ that is part of the circulatory system*



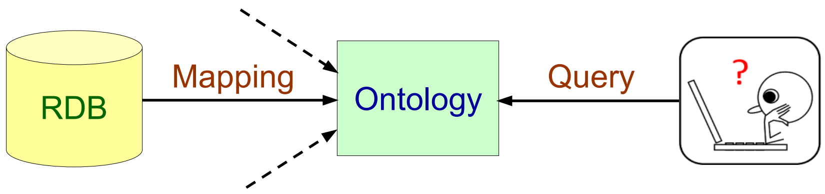

$$\forall x \, (Heart(x) \rightarrow MuscularOrgan(x) \wedge \exists y \, (isPartOf(x,y) \wedge CirculatorySystem(y)))$$

# Ontology-Based Data Access (OBDA)




**use an ontology as a mediator**

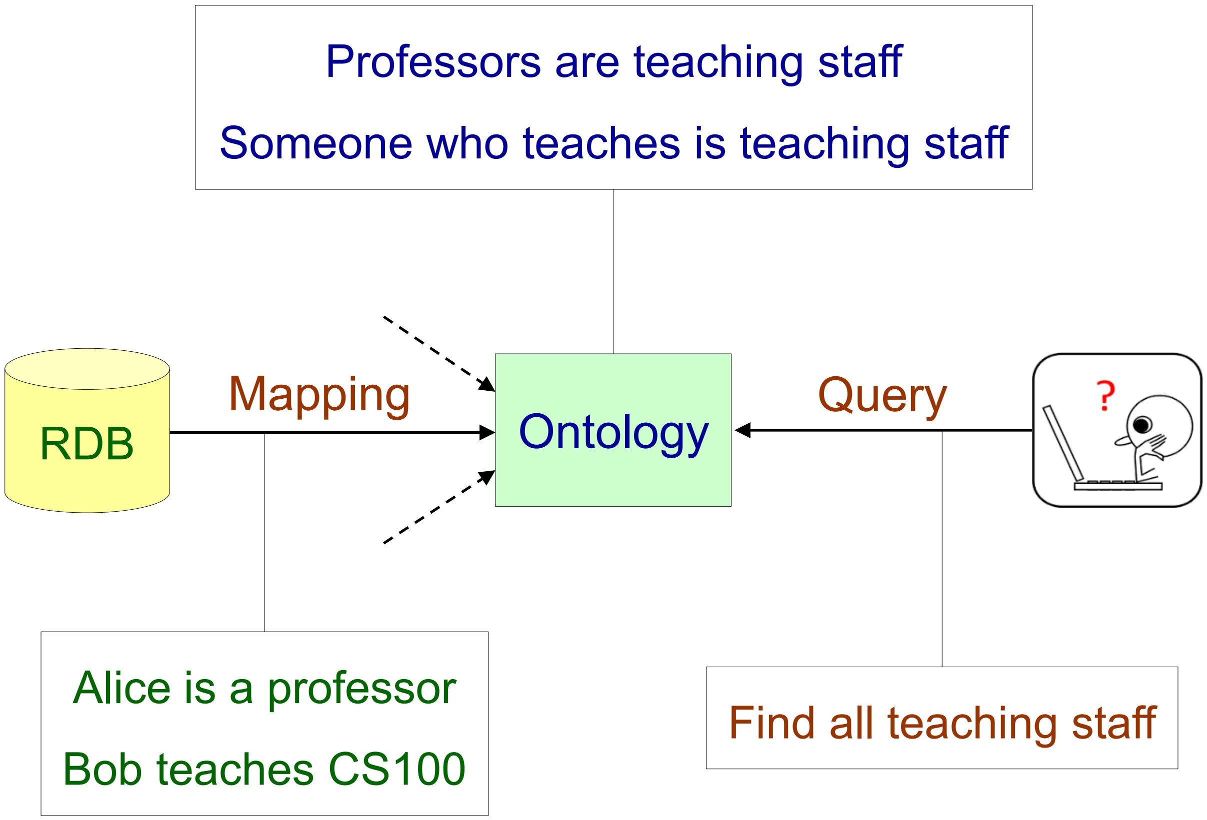
# What are Ontologies Good For?




1. **Integrate different data sources (variety)**
   - Conceptual “global view” of the data
   - Access data in a uniform and transparent way

2. **Support automated reasoning (incompleteness)**
   - Implicit consequences are taken into account
   - More complete answers

# Incomplete Data Sources




Expected answer = {Alice, Bob}

# Ontology-Based Data Access: Example

Ontology $\Sigma$ - high level representation of the domain of interest

$$\forall x\, (\text{Researcher}(x) \rightarrow \exists y\, (\text{worksFor}(x,y) \wedge \text{Project}(y)))$$

$$\forall x\, (\text{Project}(x) \rightarrow \exists y\, (\text{worksFor}(y,x) \wedge \text{Researcher}(y)))$$

$$\forall x \forall y\, (\text{worksFor}(x,y) \rightarrow \text{Researcher}(x) \wedge \text{Project}(y))$$

$$\forall x\, (\text{Project}(x) \rightarrow \exists y\, (\text{ProjectName}(x,y)))$$

# Ontology-Based Data Access: Example

Relational database *D* - a single database that represents the sources

| worksIn | SSN | Name |
|---|---|---|
| | 100 | AAA |
| | 200 | BBB |
| | 300 | CCC |

# Ontology-Based Data Access: Example

Relational database *D* - a single database that represents the sources

| worksIn | SSN | Name |
|---|---|---|
| | 100 | AAA |
| | 200 | BBB |
| | 300 | CCC |

the researcher with SSN 100 works for the project with name “AAA”

# Ontology-Based Data Access: Example

Mapping *M* - semantically link data at the sources with the ontology

$$
\begin{array}{l} \text{SELECT SSN, Name} \\ \text{FROM worksIn} \end{array}
\quad \subseteq \quad
\begin{array}{l}
\text{Researcher(person(SSN))} \land \\
\text{Project(proj(Name))} \land \\
\text{worksFor(person(SSN), proj(Name))} \land \\
\text{ProjectName(proj(Name), Name)}
\end{array}
$$

# Ontology-Based Data Access: Example

Mapping *M* - semantically link data at the sources with the ontology

SELECT SSN, Name
FROM worksIn

$\subseteq$

Researcher(person(SSN)) $\wedge$
Project(proj(Name)) $\wedge$
worksFor(person(SSN), proj(Name)) $\wedge$
ProjectName(proj(Name), Name)

- Constructors to create objects from tuples of values in the database
- The constructors are simply Skolem functions

# Ontology-Based Data Access: Example

Virtual data layer *M*(*D*)

| worksIn | SSN | Name |
|---|---|---|
| | 100 | AAA |
| | 200 | BBB |
| | 300 | CCC |

| | | |
|---|---|---|
| SELECT SSN, Name<br>FROM worksIn | $\subseteq$ | Researcher(person(SSN)) $\wedge$<br>Project(proj(Name)) $\wedge$<br>worksFor(person(SSN), proj(Name)) $\wedge$<br>ProjectName(proj(Name), Name) |

Researcher(person(100)), Project(proj(AAA)), worksFor(person(100), proj(AAA)),

ProjectName(proj(AAA), AAA),

# Ontology-Based Data Access: Example

Virtual data layer *M*(*D*)

| worksIn | SSN | Name |
|---|---|---|
| | 100 | AAA |
| | 200 | BBB |
| | 300 | CCC |

| | | |
|---|---|---|
| SELECT SSN, Name<br>FROM worksIn | $\subseteq$ | Researcher(person(SSN)) $\wedge$<br>Project(proj(Name)) $\wedge$<br>worksFor(person(SSN), proj(Name)) $\wedge$<br>ProjectName(proj(Name), Name) |

Researcher(person(100)), Project(proj(AAA)), worksFor(person(100), proj(AAA)),

ProjectName(proj(AAA), AAA),

Researcher(person(200)), Project(proj(BBB)), worksFor(person(200), proj(BBB)),

ProjectName(proj(BBB), BBB),

# Ontology-Based Data Access: Example

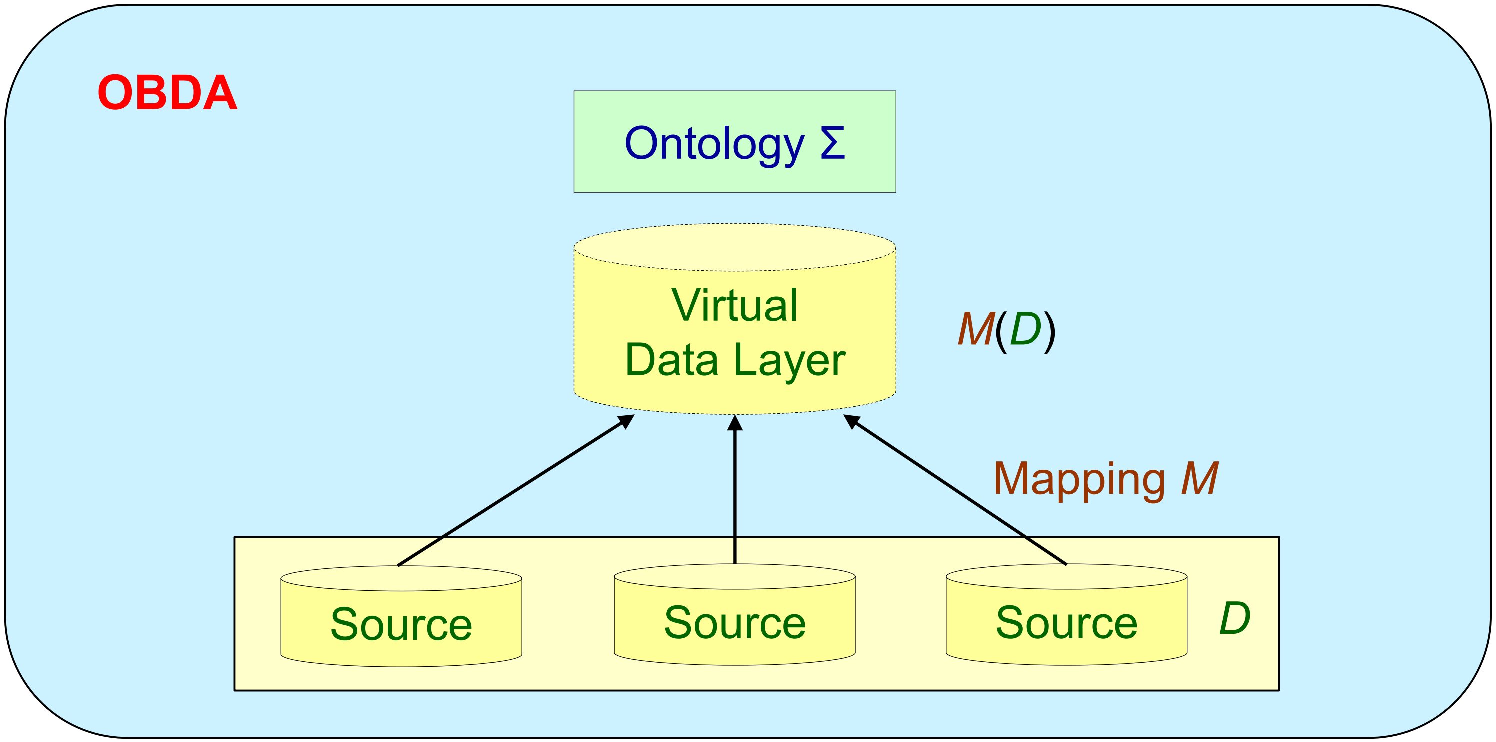
Virtual data layer *M*(*D*)

| worksIn | SSN | Name |
|---|---|---|
| | 100 | AAA |
| | 200 | BBB |
| | 300 | CCC |

SELECT SSN, Name
FROM worksIn
$\subseteq$
Researcher(person(SSN)) $\wedge$
Project(proj(Name)) $\wedge$
worksFor(person(SSN), proj(Name)) $\wedge$
ProjectName(proj(Name), Name)

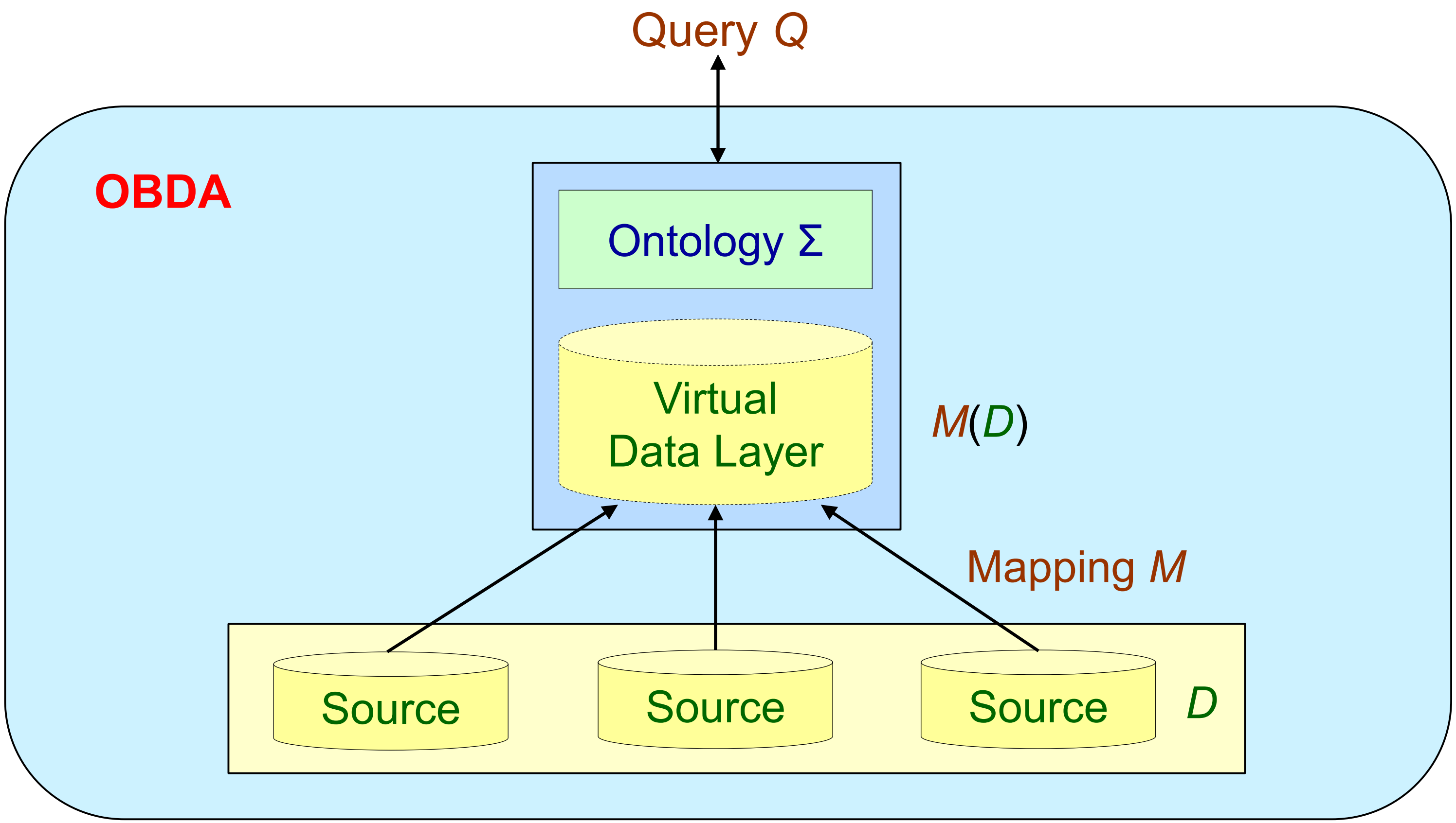
Researcher(person(100)), Project(proj(AAA)), worksFor(person(100), proj(AAA)),
ProjectName(proj(AAA), AAA),
Researcher(person(200)), Project(proj(BBB)), worksFor(person(200), proj(BBB)),
ProjectName(proj(BBB), BBB),
Researcher(person(300)), Project(proj(CCC)), worksFor(person(300), proj(CCC)),
ProjectName(proj(CCC), CCC)

# Query Answering in OBDA



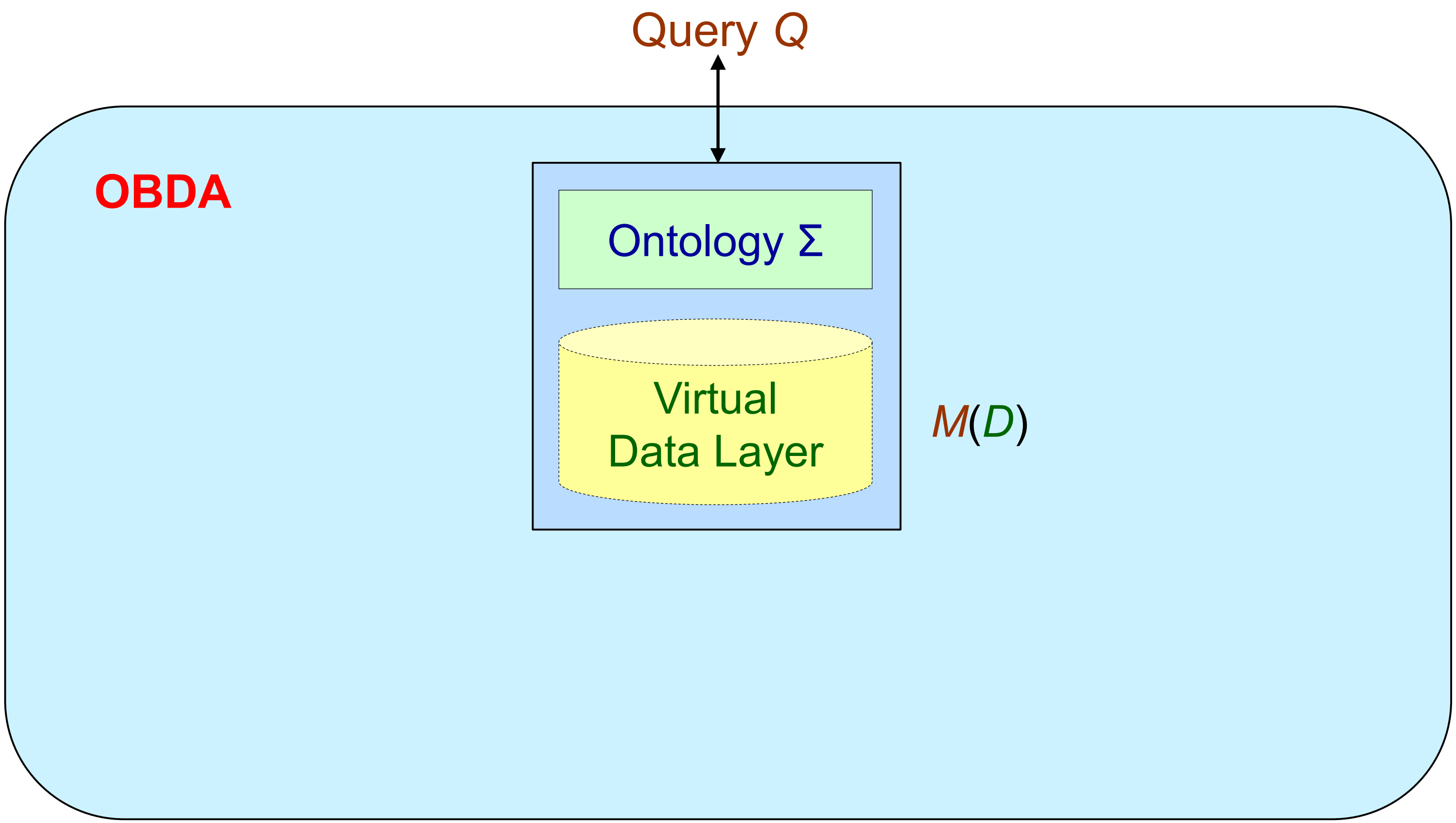

- The sources and the mapping define a virtual data layer $M(D)$

# Query Answering in OBDA



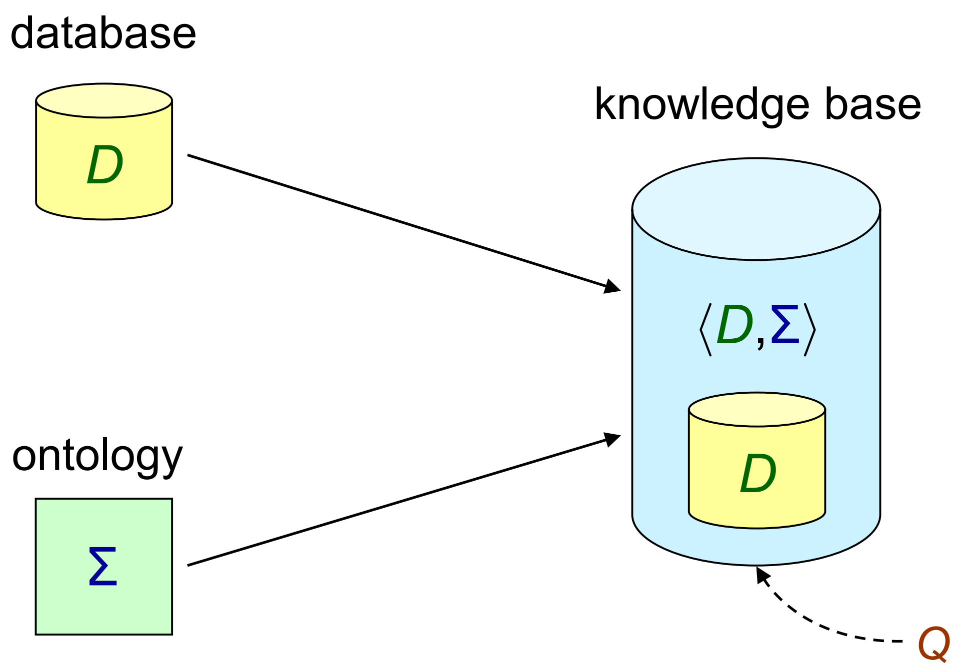

- The sources and the mapping define a virtual data layer $M(D)$
- Queries are answered against the knowledge base $\langle M(D), \Sigma \rangle$

# Query Answering in OBDA




**Ontology-Based Query Answering**

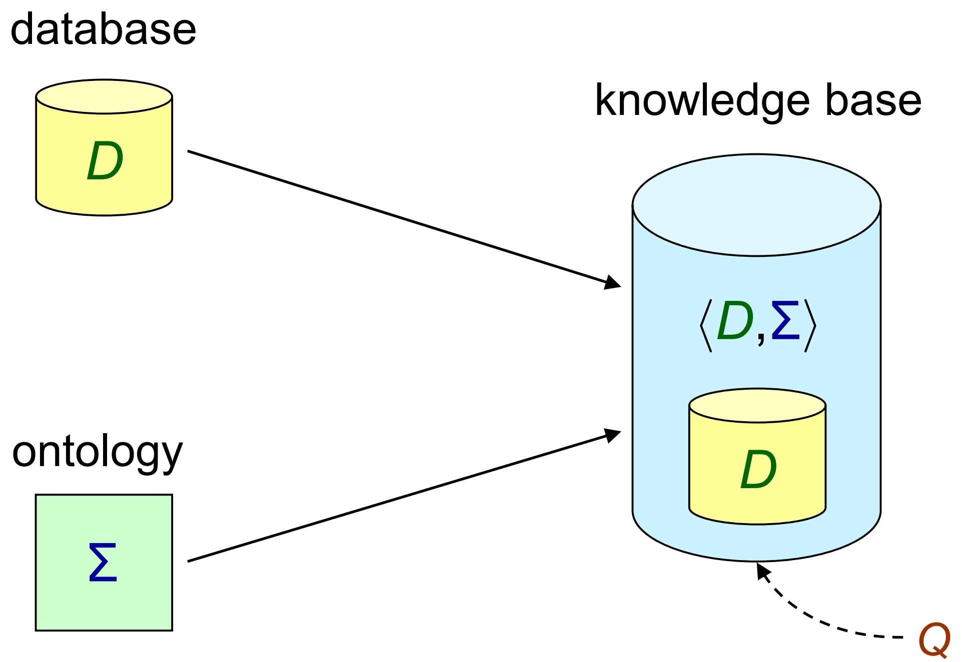
# Ontology-Based Query Answering (OBQA)




$$\text{certain-answers}(Q, \langle D, \Sigma \rangle) \quad = \quad \bigcap_{J \in \text{models}(D \wedge \Sigma)} Q(J)$$

(formal definitions later - once we fix the languages)

# Ontology-Based Query Answering (OBQA)




**NOTE:** OBQA is not OBDA, but a crucial task in OBDA

We should talk about OBDA only in the presence of external sources and mappings

# 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)
- Scalability to very large amounts of data is a key

**What is the right query language?**

- Well-known languages from database theory (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
  - Concept names: atomic classes or unary predicates, e.g., Parent, Person
  - Role names: atomic relations or binary predicates, e.g., hasParent

  and we build axioms
  - $\mathsf{Person} \sqsubseteq \exists \mathsf{hasParent}.\mathsf{Parent}$ - each person has a parent
  - $\mathsf{Parent} \sqsubseteq \mathsf{Person}$ - each parent is a person

- **Semantics:** Via first-order interpretations

# DL-Lite Family

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

| DL-Lite Axioms | First-order Representation |
|---|---|
| $A \sqsubseteq B$ | $\forall x\, (A(x) \rightarrow B(x))$ |
| $A \sqsubseteq \exists R$ | $\forall x\, (A(x) \rightarrow \exists y\, R(x,y))$ |
| $\exists R \sqsubseteq A$ | $\forall x \forall y\, (R(x,y) \rightarrow A(x))$ |
| $\exists R \sqsubseteq \exists P$ | $\forall x \forall y\, (R(x,y) \rightarrow \exists z\, P(x,z))$ |
| $A \sqsubseteq \exists R.B$ | $\forall x\, (A(x) \rightarrow \exists y\, (R(x,y) \wedge B(y)))$ |
| $R \sqsubseteq P$ | $\forall x \forall y\, (R(x,y) \rightarrow P(x,y))$ |
| $A \sqsubseteq \neg B$ | $\forall x\, (A(x) \wedge B(x) \rightarrow \bot)$ |

# The Description Logic EL

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

| EL Axioms | First-order Representation |
|---|---|
| $A \sqsubseteq B$ | $\forall x\ (A(x) \rightarrow B(x))$ |
| $A \sqcap B \sqsubseteq C$ | $\forall x\ (A(x) \wedge B(x) \rightarrow C(x))$ |
| $A \sqsubseteq \exists R.B$ | $\forall x\ (A(x) \rightarrow \exists y\ (R(x,y) \wedge B(y)))$ |
| $\exists R.B \sqsubseteq A$ | $\forall x \forall y\ (R(x,y) \wedge B(y) \rightarrow A(x))$ |

…several other, more expressive, description logics exist

…but, in what follows we focus on existential rules

an alternative way for representing ontologies

# A Simple Example

$$\forall x\, (\text{Researcher}(x) \rightarrow \exists y\, (\text{worksFor}(x,y) \wedge \text{Project}(y)))$$

$$\forall x\, (\text{Project}(x) \rightarrow \exists y\, (\text{worksFor}(y,x) \wedge \text{Researcher}(y)))$$

$$\forall x \forall y\, (\text{worksFor}(x,y) \rightarrow \text{Researcher}(x) \wedge \text{Project}(y))$$

$$\forall x\, (\text{Project}(x) \rightarrow \exists y\, (\text{ProjectName}(x,y)))$$

# Some Terminology

- Our basic vocabulary:
  - A countable set **C** of constants - domain of a database
  - A countable set **N** of (labeled) nulls - globally ∃-quantified variables
  - 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 $R(t_1, \ldots, t_n)$ - R is an n-ary relation 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 an expression

$$\forall \mathbf{x} \forall \mathbf{y}\ (\underbrace{\varphi(\mathbf{x},\mathbf{y})}_{\text{body}} \rightarrow \exists \mathbf{z}\ \underbrace{\psi(\mathbf{x},\mathbf{z})}_{\text{head}})$$

- $\mathbf{x}$,$\mathbf{y}$ and $\mathbf{z}$ are tuples of variables of $\mathbf{V}$
- $\varphi(\mathbf{x},\mathbf{y})$ and $\psi(\mathbf{x},\mathbf{z})$ are (constant-free) conjunctions of atoms

…a.k.a. tuple-generating dependencies and Datalog$^{\pm}$ rules

# Semantics of Existential Rules

- An instance $J$ is a model of the rule

$$\sigma \;=\; \forall \mathbf{x} \forall \mathbf{y}\; (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\; \psi(\mathbf{x},\mathbf{z}))$$

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

  whenever there exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$,

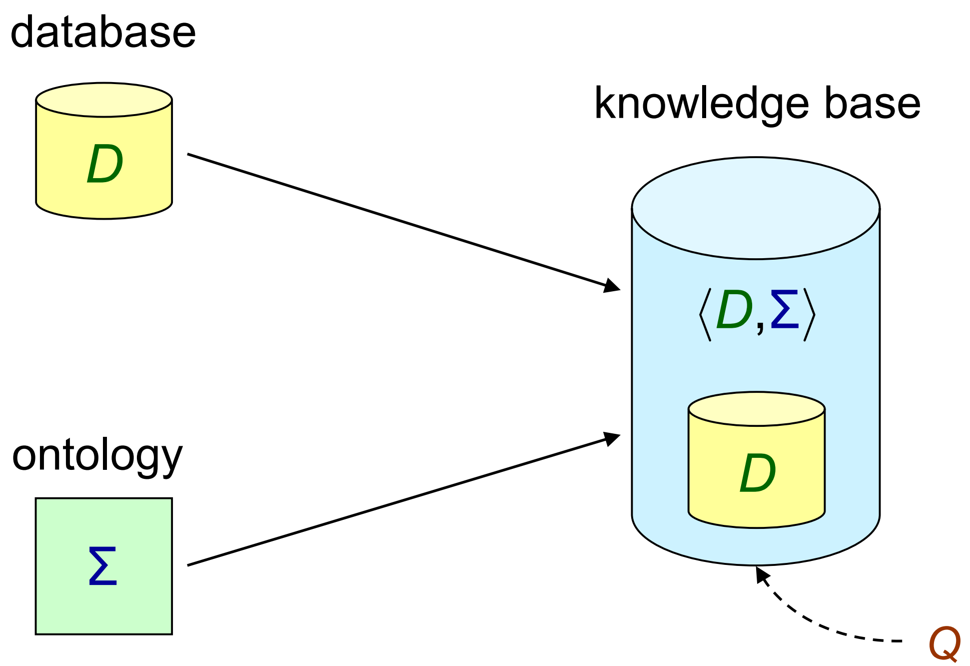
  then there exists $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

  $\{t \rightarrow h(t) \mid t \in \mathbf{x}\}$ - the restriction of h to **x**

- Given a set $\Sigma$ of existential rules, $J$ is a model of $\Sigma$, written as $J \models \Sigma$, if the following holds: for each $\sigma \in \Sigma$, $J \models \sigma$

- $J \models \Sigma$ iff $\mathfrak{J}$ is a model of the first-order theory $\bigwedge_{\sigma \in \Sigma} \sigma$

# Ontology-Based Query Answering (OBQA)



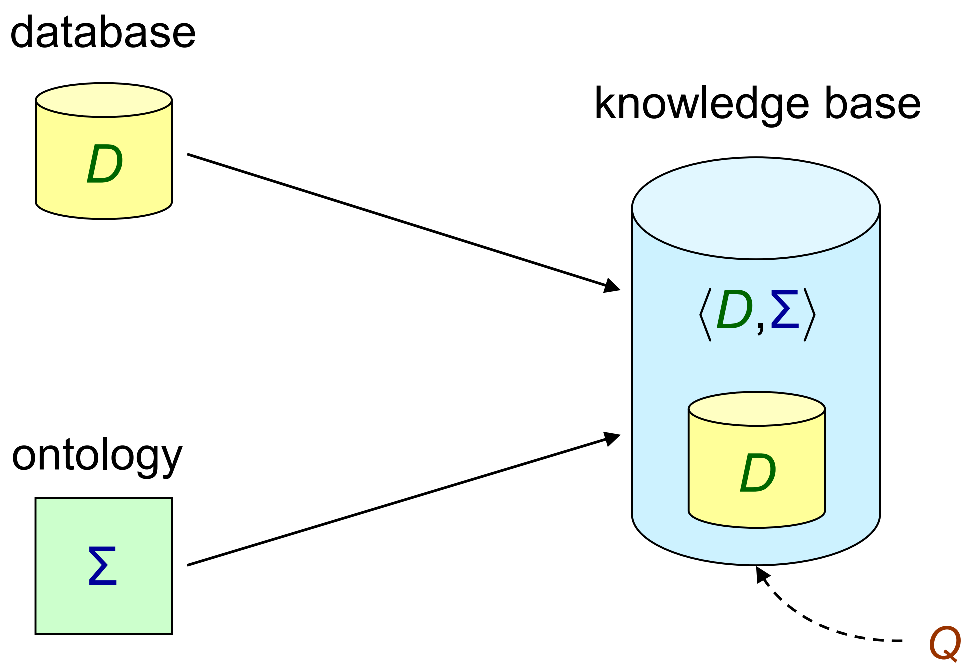

**existential rules**

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

**conjunctive queries**

$Q(\mathbf{x})\ \text{:-}\ R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})$

# Ontology-Based Query Answering (OBQA)




$$\text{certain-answers}(Q, \langle D, \Sigma \rangle) = \bigcap_{J \,\in\, \text{models}(D \,\wedge\, \Sigma)} Q(J)$$

$$\{J \mid J \supseteq D \text{ and } J \vDash \Sigma\}$$

# Exercise: Compute the Certain Answers

$D$ = {Person(john), Person(bob), Person(tom),

hasFather(john,bob), hasFather(bob,tom)}

$\Sigma = \{\forall x\, (\mathrm{Person}(x) \rightarrow \exists y\, \mathrm{hasFather}(x,y)),$

$\forall x \forall y\, (\mathrm{hasFather}(x,y) \rightarrow \mathrm{Person}(x) \wedge \mathrm{Person}(y))\}$

$Q_1(x,y)$ :- hasFather(x,y)

$Q_2(x)$ :- hasFather(x,y)

$Q_3(x)$ :- hasFather(x,y), hasFather(y,z), hasFather(z,w)

$Q_4(x,w)$ :- hasFather(x,y), hasFather(y,z), hasFather(z,w)

# Exercise: Compute the Certain Answers

$D$ = {Person(john), Person(bob), Person(tom),
hasFather(john,bob), hasFather(bob,tom)}

$$\Sigma = \{\forall x\, (\text{Person}(x) \rightarrow \exists y\, \text{hasFather}(x,y)),\ \forall x \forall y\, (\text{hasFather}(x,y) \rightarrow \text{Person}(x) \wedge \text{Person}(y))\}$$

$Q_1(x,y)$ :- hasFather(x,y)

{(john,bob), (bob,tom)}

# Exercise: Compute the Certain Answers

$D$ = {Person(john), Person(bob), Person(tom),
hasFather(john,bob), hasFather(bob,tom)}

$\Sigma = \{\forall x\, (\text{Person}(x) \rightarrow \exists y\, \text{hasFather}(x,y)),$
$\forall x \forall y\, (\text{hasFather}(x,y) \rightarrow \text{Person}(x) \wedge \text{Person}(y))\}$

$Q_2(x)$ :- hasFather(x,y)

{(john), (bob), (tom)}

# Exercise: Compute the Certain Answers

$D$ = {Person(john), Person(bob), Person(tom),
hasFather(john,bob), hasFather(bob,tom)}

$$\Sigma = \{\forall x\, (\text{Person}(x) \rightarrow \exists y\ \text{hasFather}(x,y)), \quad \forall x \forall y\, (\text{hasFather}(x,y) \rightarrow \text{Person}(x) \wedge \text{Person}(y))\}$$

$Q_3(x)$ :- hasFather(x,y), hasFather(y,z), hasFather(z,w)

{(john), (bob), (tom)}

# Exercise: Compute the Certain Answers

$D$ = {Person(john), Person(bob), Person(tom),

hasFather(john,bob), hasFather(bob,tom)}

$\Sigma = \{\forall x\,(\text{Person}(x) \rightarrow \exists y\ \text{hasFather}(x,y)),$

$\forall x \forall y\,(\text{hasFather}(x,y) \rightarrow \text{Person}(x) \wedge \text{Person}(y))\}$

$Q_4(x,w)$ :- hasFather(x,y), hasFather(y,z), hasFather(z,w)

{ }

# OBQA: Formal Definition

ontology language based on existential rules

OBQA($\mathbf{L}$)

Input: database $D$, existential rules $\Sigma \in \mathbf{L}$, CQ $Q(\mathbf{x})$, tuple $\mathbf{t} \in \text{adom}(D)^{|\mathbf{X}|}$

Question: $\mathbf{t} \in \text{certain-answers}(Q,\langle D,\Sigma\rangle) = \bigcap_{J \in \text{models}(D \wedge \Sigma)} Q(J)$?

$$\mathbf{t} \in \text{certain-answers}(Q,\langle D\ \Sigma\rangle) \Leftrightarrow \forall J \in \text{models}(D \wedge \Sigma), \mathbf{t} \in Q(J)$$

$$\Leftrightarrow \forall J \in \text{models}(D \wedge \Sigma), () \in Q_{\mathbf{t}}(J), \text{ where } Q_{\mathbf{t}} = Q(\mathbf{t})$$

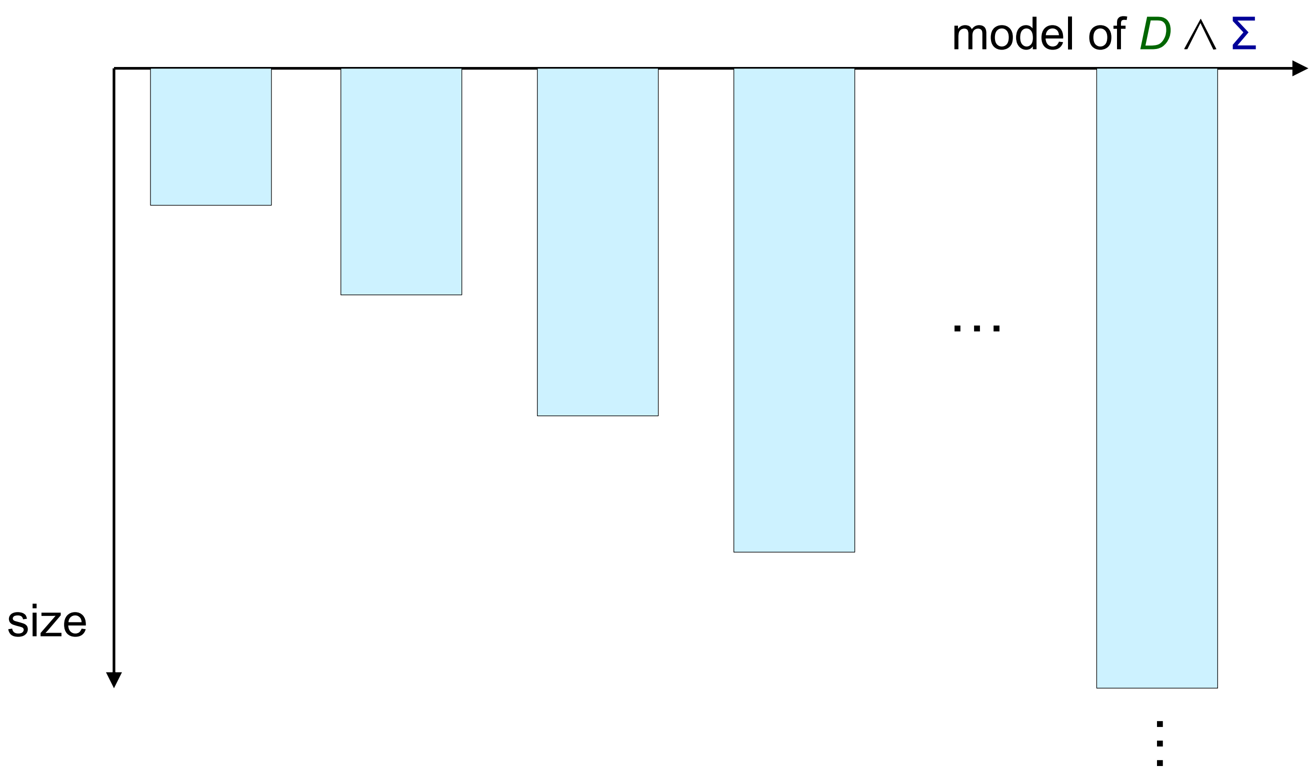
Boolean CQ - no output variables

Why is OBQA technically challenging?

What is the right tool for tackling this problem?

# The Two Dimensions of Infinity

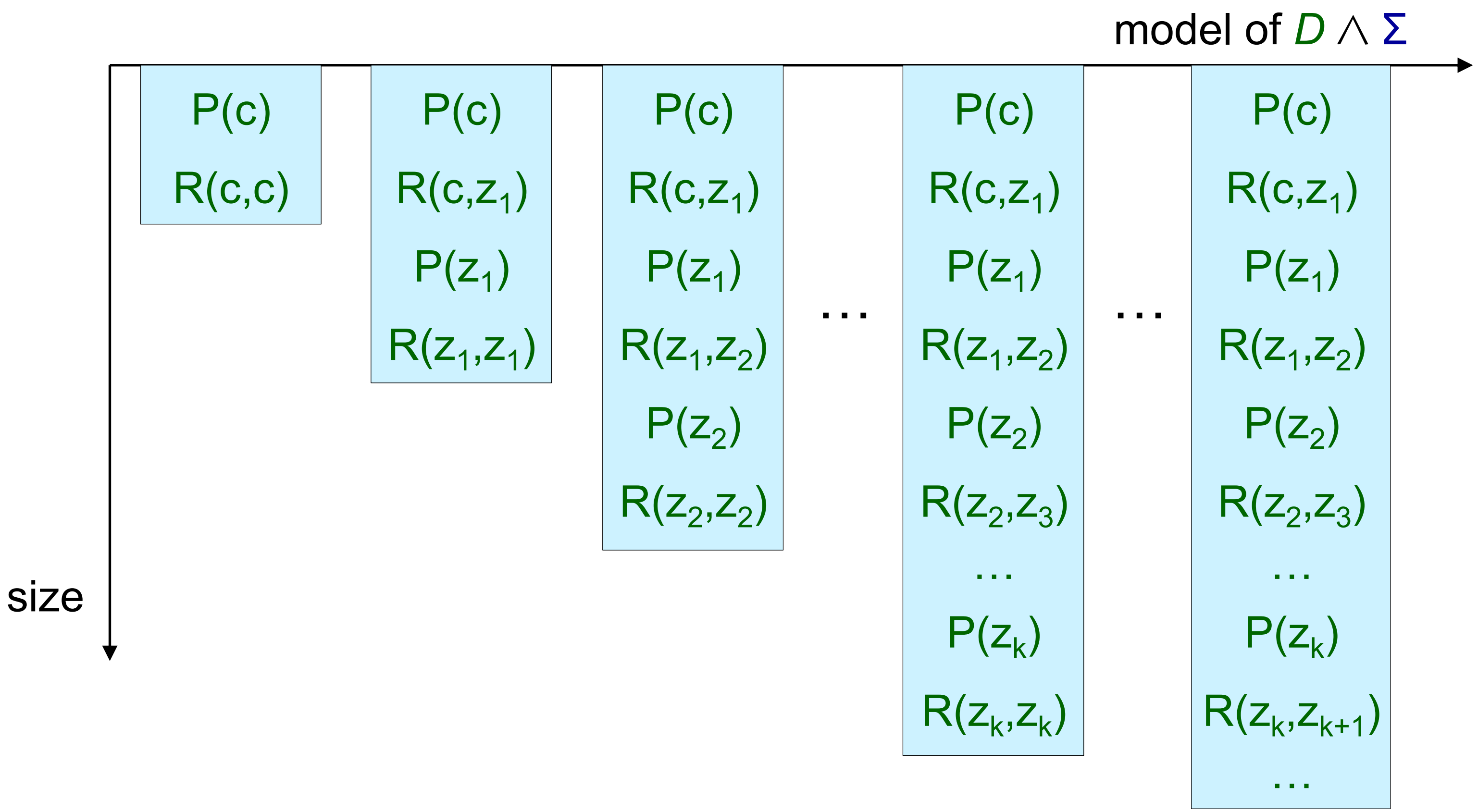
Consider the database *D*, and the set of existential rules Σ




$D \wedge \Sigma$ admits infinitely many models, of possibly infinite size

# The Two Dimensions of Infinity

$D$ = {P(c)} $\Sigma = \{\forall x\,(P(x) \rightarrow \exists y\,(R(x,y) \wedge P(y)))\}$




$z_1$, $z_2$, $z_3$, … are nulls of **N**

# Taming the First Dimension of Infinity

$D$ = {P(c)}  $\Sigma = \{\forall x\, (P(x) \rightarrow \exists y\, (R(x,y) \wedge P(y)))\}$

model of $D \wedge \Sigma$

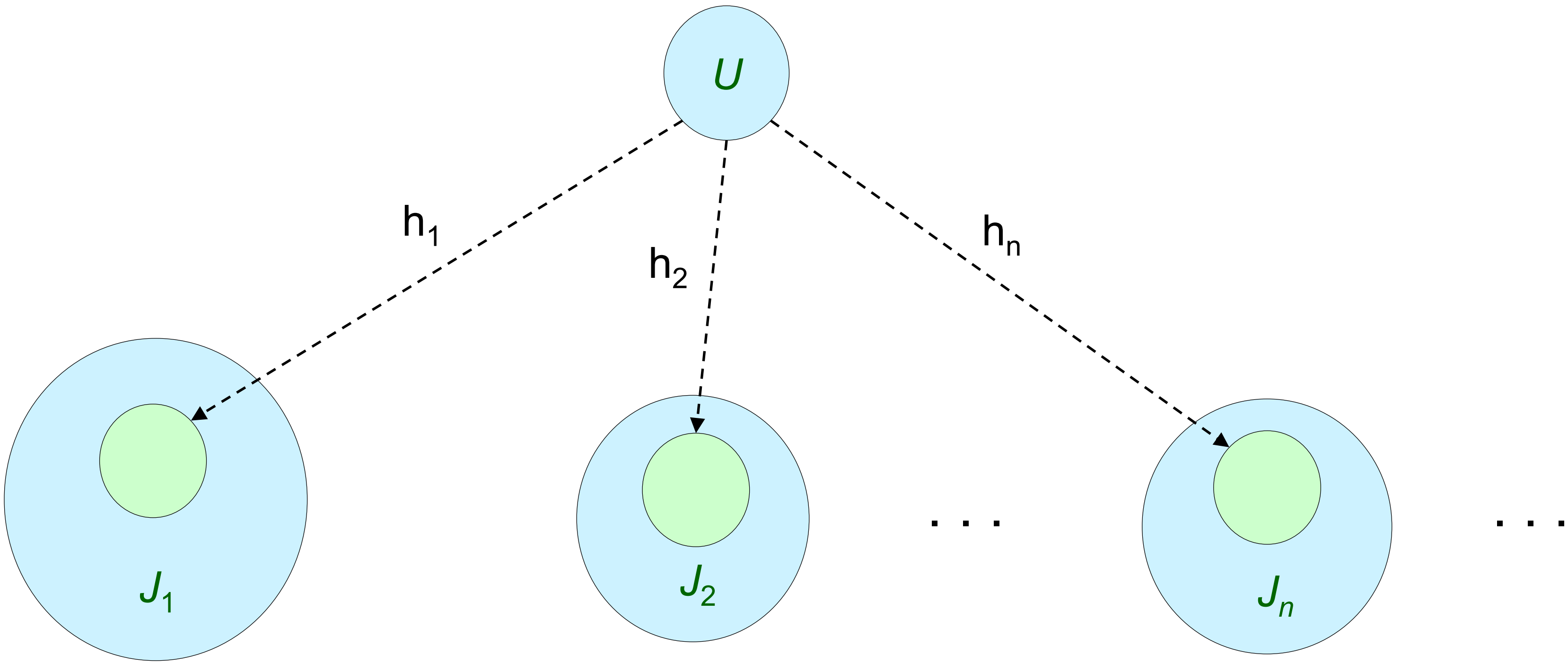
size

| | | | | | | |
|---|---|---|---|---|---|---|
| P(c) | P(c) | P(c) | … | P(c) | … | P(c) |
| R(c,c) | $R(c,z_1)$ | $R(c,z_1)$ | | $R(c,z_1)$ | | $R(c,z_1)$ |
| ✗ | $P(z_1)$ | $P(z_1)$ | | $P(z_1)$ | | $P(z_1)$ |
| | $R(z_1,z_1)$ | $R(z_1,z_2)$ | | $R(z_1,z_2)$ | | $R(z_1,z_2)$ |
| | ✗ | $P(z_2)$ | | $P(z_2)$ | | $P(z_2)$ |
| | | $R(z_2,z_2)$ | | $R(z_2,z_3)$ | | $R(z_2,z_3)$ |
| | | ✗ | | … | | … |
| | | | | $P(z_k)$ | | $P(z_k)$ |
| | | | | $R(z_k,z_k)$ | | $R(z_k,z_{k+1})$ |
| | | | | ✗ | | … |
| | | | | | | ✓ |

**Key Idea:** Focus on a representative,

a model that is as general as possible

# Universal Models (a.k.a. Canonical Models)




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

1. $U$ is a model of $D \wedge \Sigma$

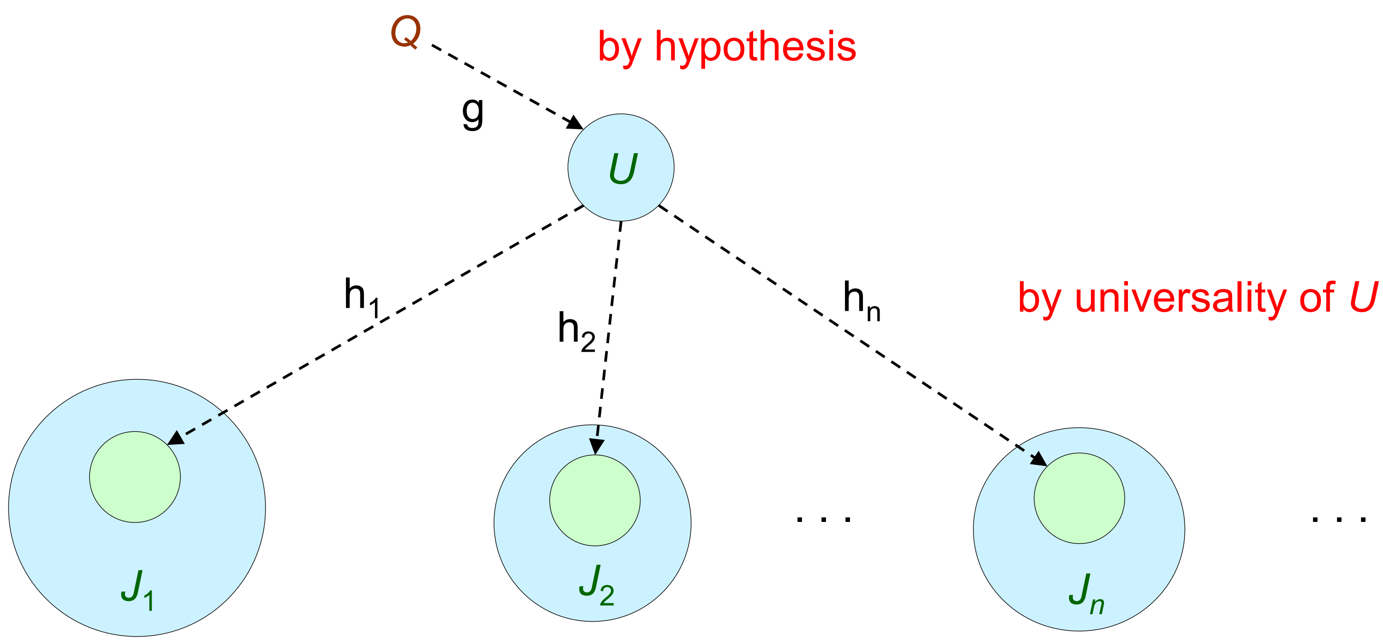
2. $\forall J \in$ models($D \wedge \Sigma$), there exists a homomorphism $h_J$ such that $h_J(U) \subseteq J$

# Query Answering via Universal Models

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

**Proof:** ($\Rightarrow$) Trivial since, for every $J \in$ models($D \wedge \Sigma$), $J \models Q$

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




$\forall J \in$ models($D \wedge \Sigma$), $\exists h_J$ such that $h_J(g(Q)) \subseteq J$ $\Rightarrow$ $\forall J \in$ models($D \wedge \Sigma$), $J \models Q$

$\Rightarrow$ $D \wedge \Sigma \models Q$

# The Chase Procedure

- Fundamental algorithmic tool used in databases

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

… what's the reason for the ubiquity of the chase in databases?

it constructs universal models

# The Chase Procedure

$D$

person(john)

$\Sigma$

$\forall x\ (\text{Person}(x) \rightarrow \exists y\ (\text{hasParent}(x,y) \land \text{Person}(y)))$

chase($D$,$\Sigma$) = $D$ $\cup$

# The Chase Procedure

*D*

person(john)

Σ

$\forall x\, (\mathrm{Person}(x) \rightarrow \exists y\, (\mathrm{hasParent}(x,y) \wedge \mathrm{Person}(y)))$

$\mathrm{chase}(D,\Sigma) = D \cup \{\mathrm{hasParent}(\mathrm{john}, z_1), \mathrm{Person}(z_1)$

# The Chase Procedure

$D$

person(john)

$\Sigma$

$\forall x\ (\text{Person}(x) \rightarrow \exists y\ (\text{hasParent}(x,y) \wedge \text{Person}(y)))$

$\text{chase}(D,\Sigma) = D \cup \{\text{hasParent}(\text{john}, z_1), \text{Person}(z_1),$

$\text{hasParent}(z_1, z_2), \text{Person}(z_2)$

# The Chase Procedure

$D$

person(john)

$\Sigma$

$\forall x\ (\mathrm{Person}(x) \rightarrow \exists y\ (\mathrm{hasParent}(x,y) \wedge \mathrm{Person}(y)))$

$$\begin{aligned}\mathrm{chase}(D,\Sigma) = D \cup \{&\mathrm{hasParent}(\mathrm{john}, z_1), \mathrm{Person}(z_1),\\ &\mathrm{hasParent}(z_1, z_2), \mathrm{Person}(z_2),\\ &\mathrm{hasParent}(z_2, z_3), \mathrm{Person}(z_3)\end{aligned}$$

# The Chase Procedure

$D$

person(john)

$\Sigma$

$\forall x\ (\mathrm{Person}(x) \rightarrow \exists y\ (\mathrm{hasParent}(x,y) \wedge \mathrm{Person}(y)))$

$\mathrm{chase}(D,\Sigma) = D \cup \{\mathrm{hasParent}(\mathrm{john}, z_1), \mathrm{Person}(z_1),$

$\mathrm{hasParent}(z_1, z_2), \mathrm{Person}(z_2),$

$\mathrm{hasParent}(z_2, z_3), \mathrm{Person}(z_3), \ldots$

**infinite instance**

# The Chase Procedure: Formal Definition

- Chase rule - the building block of the chase procedure

- A rule $\sigma = \forall \mathbf{x} \forall \mathbf{y} \, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z} \, \psi(\mathbf{x},\mathbf{z}))$ is applicable to instance $J$ if:

   1. There exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$
   2. There is no $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

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

$h = \{x \rightarrow a\}$ $\quad$ $g = \{x \rightarrow a, y \rightarrow b\}$

$\forall x \, (R(x) \rightarrow \exists y \, P(x,y))$

✗

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

$h = \{x \rightarrow a\}$ $\quad$ ×

$\forall x \, (R(x) \rightarrow \exists y \, P(x,y))$

✓

# The Chase Procedure: Formal Definition

- Chase rule - the building block of the chase procedure

- A rule $\sigma = \forall \mathbf{x} \forall \mathbf{y}\, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x},\mathbf{z}))$ is applicable to instance $J$ if:

  1. There exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$
  2. There is no $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

- Let $J_{+} = J \cup \{g(\psi(\mathbf{x},\mathbf{z}))\}$, where $g \supseteq h_{|\mathbf{z}}$ and $g(\mathbf{z})$ are "fresh" nulls not in $J$

- The result of applying $\sigma$ to $J$ is $J_{+}$, denoted $J\langle\sigma,h\rangle J_{+}$ - single chase step

# The Chase Procedure: Formal Definition

- A finite chase of $D$ w.r.t. $\Sigma$ is a finite sequence

$$D\langle\sigma_1,h_1\rangle J_1\langle\sigma_2,h_2\rangle J_2\langle\sigma_3,h_3\rangle J_3 \ldots \langle\sigma_n,h_n\rangle J_n$$

and chase($D$,$\Sigma$) is defined as the instance $J_n$

all applicable rules will eventually be applied

- An infinite chase of $D$ w.r.t. $\Sigma$ is a fair finite sequence

$$D\langle\sigma_1,h_1\rangle J_1\langle\sigma_2,h_2\rangle J_2\langle\sigma_3,h_3\rangle J_3 \ldots \langle\sigma_n,h_n\rangle J_n \ldots$$

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

least fixpoint of a monotonic operator - chase step

# Chase: A Universal Model

**Theorem:** chase($D$,$\Sigma$) is a universal model of $D \wedge \Sigma$

the result of the chase after k applications of the chase step

**Proof:**

- By construction, chase($D$,$\Sigma$) $\in$ models($D \wedge \Sigma$)
- It remains to show that chase($D$, $\Sigma$) can be homomorphically embedded into every other model of $D \wedge \Sigma$
- Fix an arbitrary instance $J \in$ models($D \wedge \Sigma$). We need to show that there exists h such that h(chase($D$,$\Sigma$)) $\subseteq J$
- **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 $h_k(\text{chase}^{[k]}(D,\Sigma)) \subseteq J$, and $h_k$ is compatible with $h_{k-1}$**
- Clearly, $\cup_{k \geq 0}\, h_k$ is a well-defined homomorphism that maps chase($D$,$\Sigma$) to $J$
- The claim follows with $h = \cup_{k \geq 0}\, h_k$

# Chase: Uniqueness Property

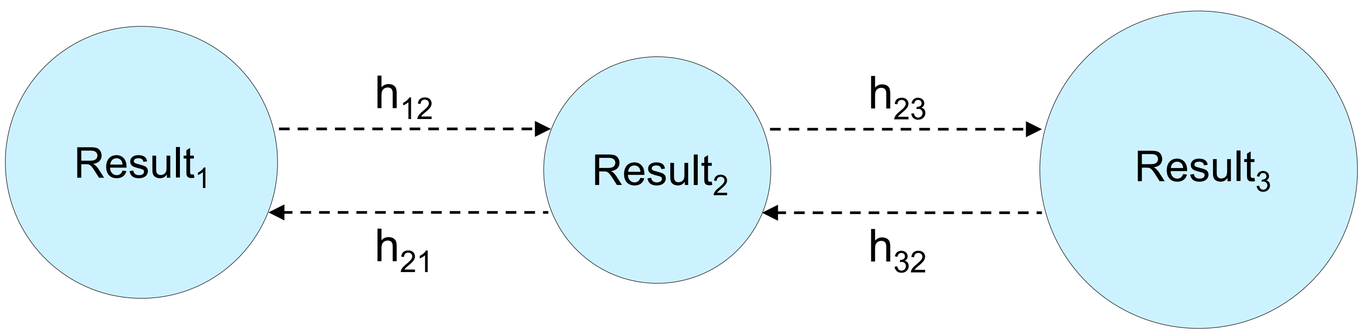
- The result of the chase is not unique - depends on the order of rule application

$D = \{P(a)\}$ $\quad\quad$ $\sigma_1 = \forall x\,(P(x) \rightarrow \exists y\, R(y))$ $\quad\quad$ $\sigma_2 = \forall x\,(P(x) \rightarrow R(x))$

$\text{Result}_1 = \{P(a), R(z), R(a)\}$ $\quad\quad$ $\sigma_1$ then $\sigma_2$

$\text{Result}_2 = \{P(a), R(a)\}$ $\quad\quad$ $\sigma_2$ then $\sigma_1$

- But, it is unique up to homomorphic equivalence




- Thus, it is unique for query answering purposes

# Query Answering via the Chase

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

&

**Theorem:** chase($D$, $\Sigma$) is a universal model of $D \wedge \Sigma$

$\Downarrow$

**Corollary:** $D \wedge \Sigma \models Q$ iff chase($D$,$\Sigma$) $\models Q$

- We can tame the first dimension of infinity by exploiting the chase procedure
- What about the second dimension of infinity? - the chase may be infinite

Can we tame the second dimension of infinity?

# Undecidability of OBQA

arbitrary existential rules

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

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

# Deterministic Turing Machine (DTM)

$S \backslash \{s_{acc}\} \times \Lambda \rightarrow S \times \Lambda \times \{-1,0,+1\}$

accepting state

$$M \;=\; (S, \Lambda, \sqcup, \delta, s_0, s_{acc})$$

states

tape symbols

blank symbol

initial state

$\delta(s_1, \alpha) = (s_2, \beta, +1)$

**IF** at some time instant $\tau$ the machine is in sate $s_1$, the cursor points to cell $\kappa$, and this cell contains $\alpha$

**THEN** at instant $\tau+1$ the machine is in state $s_2$, cell $\kappa$ contains $\beta$, and the cursor points to cell $\kappa+1$

# Undecidability of OBQA

arbitrary existential rules

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

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

Encode the computation of a DTM *M* with an empty tape using a database *D*,

a set $\Sigma$ of existential rules, and a BCQ *Q* such that $D \wedge \Sigma \vDash Q$ iff *M* accepts

How we ensure decidability of OBQA?

# Gaining Decidability

**By restricting the database**

- $\{\text{Start}(c)\} \wedge \Sigma \vDash Q$ iff the DTM $M$ accepts
- The problem is undecidable already for singleton databases
- No much to do in this direction

**By restricting the query language**

- $D \wedge \Sigma \vDash Q$ :- Accept(x) iff the DTM $M$ accepts
- The problem is undecidable already for atomic queries
- No much to do in this direction

**By restricting the ontology language**

- Achieve a good trade-off between expressive power and complexity
- Field of intense research
- Any ideas?

# What is the Source of Non-termination?

$D$

person(john)

$\Sigma$

$\forall x\,(\text{Person}(x) \rightarrow \exists y\,(\text{hasParent}(x,y) \wedge \text{Person}(y)))$

$\text{chase}(D,\Sigma) = D \cup \{\text{hasParent}(\text{john}, z_1), \text{Person}(z_1),$

$\text{hasParent}(z_1, z_2), \text{Person}(z_2),$

$\text{hasParent}(z_2, z_3), \text{Person}(z_3), \ldots$

1. **Existential quantification**
2. **Recursive definitions**

# Termination of the Chase

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

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

# Full Existential Rules

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

$$\forall \mathbf{x} \forall \mathbf{y}\, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \psi(\mathbf{x}))$$

- We denote **FULL** the class of full existential rules

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

  $\Rightarrow$ given $\Sigma$, we can decide in linear time whether $\Sigma \in$ **FULL**

  $\Rightarrow$ closed under union - $\Sigma_1 \in$ **FULL**, $\Sigma_2 \in$ **FULL** $\Rightarrow (\Sigma_1 \cup \Sigma_2) \in$ **FULL**

- But, is this a reasonable ontology language?

# **FULL** and OWL 2 RL

- The acronym RL reflects its relation to rules
- **FULL** captures OWL 2 RL

$$\mathrm{Parent} \sqcap \mathrm{Male} \sqsubseteq \mathrm{Father}$$

$$\forall x\, (\mathrm{Parent}(x) \land \mathrm{Male}(x) \rightarrow \mathrm{Father}(x))$$

$$\exists \mathrm{parentOf}.\exists \mathrm{parentOf}.\mathrm{T} \sqsubseteq \mathrm{Grandfather}$$

$$\forall x \forall y\, (\mathrm{parentOf}(x,y) \land \mathrm{parentOf}(y,z) \rightarrow \mathrm{Grandfather}(x))$$

$$\mathrm{MetalDevice} \sqsubseteq \forall \mathrm{hasPart}.\mathrm{Metal}$$

$$\forall x \forall y\, (\mathrm{MetalDevice}(x) \land \mathrm{hasPart}(x,y) \rightarrow \mathrm{Metal}(y))$$

# **FULL** and OWL 2 RL

- The acronym RL reflects its relation to rules
- **FULL** captures OWL 2 RL

$$\mathrm{childOf} \circ \mathrm{childOf} \sqsubseteq \mathrm{grandchildOf}$$

$$\forall x \forall y \forall z\, (\mathrm{childOf}(x,y) \land \mathrm{childOf}(y,z) \rightarrow \mathrm{grandchildOf}(x,z))$$

$$\mathrm{Person} \sqsubseteq \exists_{\leq 1}\, \mathrm{hasPassport.Valid}$$

$$\forall x \forall y \forall z\, (\mathrm{Person}(x) \land \mathrm{hasPassport}(x,y) \land \mathrm{Valid}(y) \land$$
$$\mathrm{hasPassport}(x,z) \land \mathrm{Valid}(z) \rightarrow y = z)$$

$$\mathbf{Disj}(\mathrm{childOf}, \mathrm{parentOf})$$

$$\forall x \forall y\, (\mathrm{childOf}(x,y) \land \mathrm{parentOf}(x,y) \rightarrow \bot)$$

# Full Existential Rules

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

$$\forall \mathbf{X} \forall \mathbf{Y} \, (\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \psi(\mathbf{X}))$$

- We denote **FULL** the class of full existential rules

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

  $\Rightarrow$ given $\Sigma$, we can decide in linear time whether $\Sigma \in$ **FULL**

  $\Rightarrow$ closed under union - $\Sigma_1 \in$ **FULL**, $\Sigma_2 \in$ **FULL** $\Rightarrow (\Sigma_1 \cup \Sigma_2) \in$ **FULL**

- But, is this a reasonable ontology language? **OWL 2 RL**

# Full Existential Rules

- Consider a database $D$ and a set $\Sigma \in$ **FULL**

- $\text{chase}(D,\Sigma) \subseteq \{P(c_1,\ldots,c_n) \mid (c_1,\ldots,c_n) \in \text{adom}(D)^n \text{ and } P \in \text{sch}(\Sigma)\}$

active domain - constants occurring in $D$

schema - predicates occurring in $\Sigma$

- $|\text{chase}(D,\Sigma)| \leq \underbrace{|\text{sch}(\Sigma)| \cdot \overbrace{(|\text{adom}(D)|)^{\text{maxarity}}}^{\text{maximum number of tuples with terms of adom}(D)}}_{\text{maximum number of atoms with predicates of sch}(\Sigma)\text{ and terms of adom}(D)}$  maxarity $= \max_{P \in \text{sch}(\Sigma)} \{\text{arity}(P)\}$

# Complexity Measures for OBQA

OBQA($\mathbf{L}$)

Input: database $D$, existential rules $\Sigma \in \mathbf{L}$, CQ $Q(\mathbf{x})$, tuple $\mathbf{t} \in \mathrm{adom}(D)^{|\mathbf{x}|}$

Question: $\mathbf{t} \in \text{certain-answers}(Q,\langle D,\Sigma\rangle) = \bigcap_{J \in \mathrm{models}(D \wedge \Sigma)} Q(J)$?

- Data complexity: is calculated by considering only the database as part of the input, while the ontology and the query are fixed - $\mathrm{OBQA}_{\Sigma,Q}(\mathbf{L})$

- Combined complexity: is calculated by considering, apart from the database, also the ontology and the query as part of the input

# Data Complexity of **FULL**

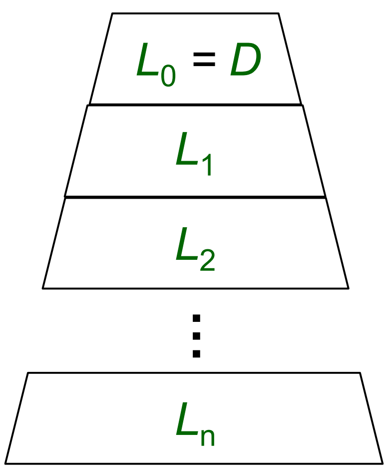
**Theorem:** $\mathrm{OBQA}_{\Sigma,Q}$(**FULL**) is in PTIME

**Proof:** Consider a database $D$, a set $\Sigma \in$ **FULL**, and a (Boolean) CQ $Q$

We apply the naïve algorithm:

1. Construct chase($D$,$\Sigma$)
2. Check for the existence of a homomorphism h such that h($Q$) $\subseteq$ chase($D$,$\Sigma$)

Step 1: We construct the chase level-by-level




- **From $L_k$ to $L_{k+1}$**: for each $\sigma \in \Sigma$, find all the homomorphisms h such that h(body($\sigma$)) $\subseteq L_k$, and add to $L_k$ the set of atoms h(head($\sigma$))
- **Stop** when $L_k = L_{k+1}$

$$|\Sigma| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxvariables}(\Sigma)} \cdot \mathrm{maxbody}(\Sigma) \cdot |L_k|$$

# Data Complexity of FULL

**Theorem:** $\text{OBQA}_{\Sigma,Q}$(**FULL**) is in PTIME

**Proof:** Consider a database $D$, a set $\Sigma \in$ **FULL**, and a (Boolean) CQ $Q$

We apply the naïve algorithm:

1. Construct chase($D$,$\Sigma$)
2. Check for the existence of a homomorphism h such that $h(Q) \subseteq \text{chase}(D,\Sigma)$

Step 1: We construct the chase level-by-level in time

$$(k\text{-}1) \cdot |\Sigma| \cdot (|\text{adom}(D)|)^{\text{maxvariables}(\Sigma)} \cdot \text{maxbody}(\Sigma) \cdot |L|$$

$$\text{where } \; k, |L| \leq |\text{chase}(D,\Sigma)| \leq |\text{sch}(\Sigma)| \cdot (|\text{adom}(D)|)^{\text{maxarity}}$$

# Data Complexity of FULL

**Theorem:** $\text{OBQA}_{\Sigma,Q}$(**FULL**) is in PTIME

**Proof:** Consider a database $D$, a set $\Sigma \in$ **FULL**, and a (Boolean) CQ $Q$

We apply the naïve algorithm:

1. Construct chase($D$,$\Sigma$)
2. Check for the existence of a homomorphism h such that $h(Q) \subseteq$ chase($D$,$\Sigma$)

Step 2: By applying similar analysis, we can show that the existence of h can be checked in time

$$(|\text{adom}(D)|)^{\#\text{variables}(Q)} \cdot |Q| \cdot |\text{chase}(D,\Sigma)|$$

$$\text{where} \quad |\text{chase}(D,\Sigma)| \leq |\text{sch}(\Sigma)| \cdot (|\text{adom}(D)|)^{\text{maxarity}}$$

# Data Complexity of **FULL**

**Theorem:** $\text{OBQA}_{\Sigma,Q}$(**FULL**) is in PTIME

**Proof:** Consider a database $D$, a set $\Sigma \in$ **FULL**, and a (Boolean) CQ $Q$

We apply the naïve algorithm:

1. Construct chase($D$,$\Sigma$)
2. Check for the existence of a homomorphism h such that h($Q$) $\subseteq$ chase($D$,$\Sigma$)

Consequently, in the worst case, the naïve algorithm runs in time

$$(|\text{sch}(\Sigma)| \cdot (|\text{adom}(D)|)^{\text{maxarity}})^2 \cdot |\Sigma| \cdot (|\text{adom}(D)|)^{\text{maxvariables}(\Sigma)} \cdot \text{maxbody}(\Sigma)$$

$$+$$

$$(|\text{adom}(D)|)^{\#\text{variables}(Q)} \cdot |Q| \cdot |\text{sch}(\Sigma)| \cdot (|\text{adom}(D)|)^{\text{maxarity}}$$

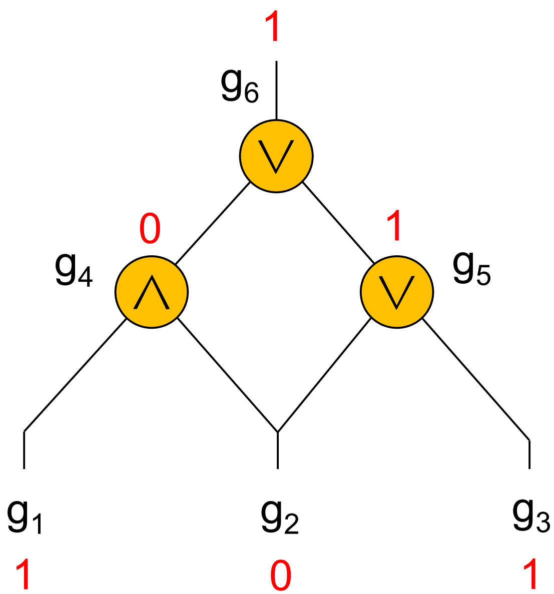
# Data Complexity of **FULL**

**We cannot do better than the naïve algorithm**

**Theorem:** $\mathrm{OBQA}_{\Sigma, Q}$(**FULL**) is PTIME-hard

**Proof :** By a LOGSPACE reduction from Monotone Circuit Value problem

# Data Complexity of **FULL**




Does the circuit evaluate to *true*?

encoding of the circuit as a database *D*

$T(g_1)$ $T(g_3)$

$AND(g_4,g_1,g_2)$ $OR(g_5,g_2,g_3)$ $OR(g_6,g_4,g_5)$

evaluation of the circuit via a *fixed* set $\Sigma$

$\forall x \forall y \forall z\, (T(x) \wedge OR(z,x,y) \rightarrow T(z))$

$\forall x \forall y \forall z\, (T(y) \wedge OR(z,x,y) \rightarrow T(z))$

$\forall x \forall y \forall z\, (T(x) \wedge T(y) \wedge AND(z,x,y) \rightarrow T(z))$

Circuit evaluates to *true* iff $D \wedge \Sigma \vDash T(g_6)$

# Combined Complexity of FULL

**Theorem:** OBQA(**FULL**) is in EXPTIME

**Proof:** Consider a database $D$, a set $\Sigma \in$ **FULL**, and a BCQ $Q$

We apply the naïve algorithm:

1. Construct chase($D$,$\Sigma$)
2. Check for the existence of a homomorphism h such that h($Q$) $\subseteq$ chase($D$,$\Sigma$)

Consequently, in the worst case, the naïve algorithm runs in time

$$(|\mathrm{sch}(\Sigma)| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxarity}})^2 \cdot |\Sigma| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxvariables}(\Sigma)} \cdot \mathrm{maxbody}(\Sigma)$$

$$+$$

$$(|\mathrm{adom}(D)|)^{\#\mathrm{variables}(Q)} \cdot |Q| \cdot |\mathrm{sch}(\Sigma)| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxarity}}$$

# Combined Complexity of **FULL**

**We cannot do better than the naïve algorithm**

**Theorem:** OBQA(**FULL**) is in EXPTIME-hard

**Proof :** By simulating a deterministic exponential time Turing machine

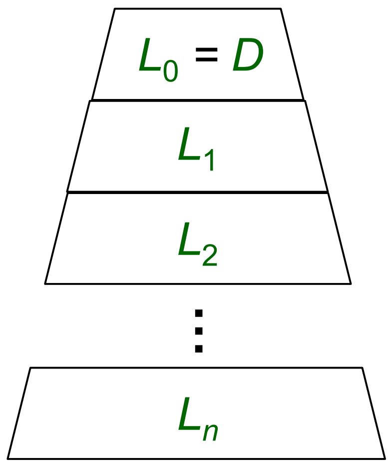
# Termination of the Chase

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

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

…the naïve algorithm is not clever enough

# The Naïve Algorithm for **ACYCLIC**




$|L_0| = 2$

$|L_1| = (|L_0|)^2$

| | | $L_1$ |
|---|---|---|
| 0 | 0 | $z_{00}$ |
| 0 | 1 | $z_{01}$ |
| 1 | 0 | $z_{10}$ |
| 1 | 1 | $z_{11}$ |

$D = \{P_0(0), P_0(1)\}$

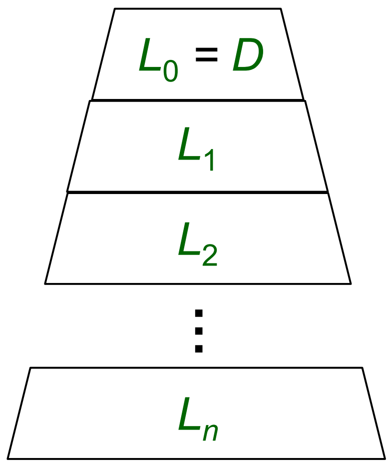
$$\Sigma = \{\forall x \forall y\ (P_0(x) \wedge P_0(y) \rightarrow \exists z\ (S_1(x,y,z) \wedge P_1(z)))$$

$$\forall x \forall y\ (P_1(x) \wedge P_1(y) \rightarrow \exists z\ (S_2(x,y,z) \wedge P_2(z)))$$

$$\ldots$$

$$\forall x \forall y\ (P_{n-1}(x) \wedge P_{n-1}(y) \rightarrow \exists z\ (S_n(x,y,z) \wedge P_n(z)))\}$$

# The Naïve Algorithm for **ACYCLIC**




$|L_0| = 2$

$|L_1| = (|L_0|)^2$

$|L_2| = (|L_1|)^2$

$D = \{P_0(0), P_0(1)\}$

$\Sigma = \{\forall x \forall y\ (P_0(x) \wedge P_0(y) \rightarrow \exists z\ (S_1(x,y,z) \wedge P_1(z)))$

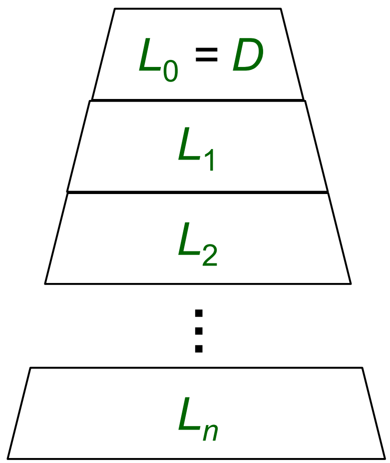
$\forall x \forall y\ (P_1(x) \wedge P_1(y) \rightarrow \exists z\ (S_2(x,y,z) \wedge P_2(z)))$

…

$\forall x \forall y\ (P_{n-1}(x) \wedge P_{n-1}(y) \rightarrow \exists z\ (S_n(x,y,z) \wedge P_n(z)))\}$

| | | $L_2$ |
|---|---|---|
| $z_{00}$ | $z_{00}$ | $z_{0000}$ |
| $z_{00}$ | $z_{01}$ | $z_{0001}$ |
| $z_{00}$ | $z_{10}$ | $z_{0010}$ |
| $z_{00}$ | $z_{11}$ | $z_{0011}$ |
| $z_{01}$ | $z_{00}$ | $z_{0100}$ |
| $z_{01}$ | $z_{01}$ | $z_{0101}$ |
| $z_{01}$ | $z_{10}$ | $z_{0110}$ |
| $z_{01}$ | $z_{11}$ | $z_{0111}$ |
| $z_{10}$ | $z_{00}$ | $z_{1000}$ |
| $z_{10}$ | $z_{01}$ | $z_{1001}$ |
| $z_{10}$ | $z_{10}$ | $z_{1010}$ |
| $z_{10}$ | $z_{11}$ | $z_{1011}$ |
| $z_{11}$ | $z_{00}$ | $z_{1100}$ |
| $z_{11}$ | $z_{01}$ | $z_{1101}$ |
| $z_{11}$ | $z_{10}$ | $z_{1110}$ |
| $z_{11}$ | $z_{11}$ | $z_{1111}$ |

# The Naïve Algorithm for **ACYCLIC**




$|L_0| = 2$

$|L_1| = (|L_0|)^2$

$|L_2| = (|L_1|)^2$

$|L_n| = (|L_{n-1}|)^2$

| | | $L_n$ |
|---|---|---|
| $z_{0\ldots0}$ | $z_{0\ldots0}$ | $z_{0\ldots00\ldots0}$ |
| … | … | … |
| $z_{1\ldots1}$ | $z_{1\ldots1}$ | $z_{1\ldots11\ldots1}$ |

$D = \{P_0(0), P_0(1)\}$

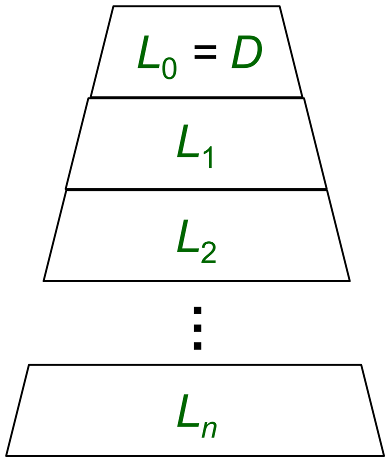
$\Sigma = \{\forall x \forall y\, (P_0(x) \wedge P_0(y) \rightarrow \exists z\, (S_1(x,y,z) \wedge P_1(z)))$

$\forall x \forall y\, (P_1(x) \wedge P_1(y) \rightarrow \exists z\, (S_2(x,y,z) \wedge P_2(z)))$

…

$\forall x \forall y\, (P_{n-1}(x) \wedge P_{n-1}(y) \rightarrow \exists z\, (S_n(x,y,z) \wedge P_n(z)))\}$

# The Naïve Algorithm for **ACYCLIC**




$|L_0| = 2$

$|L_1| = (|L_0|)^2$

$|L_2| = (|L_1|)^2$

$|L_n| = (|L_{n-1}|)^2$

$$\boldsymbol{|L_n| = 2^{(2\wedge n)}}$$

$D = \{P_0(0), P_0(1)\}$

$\Sigma = \{\forall x \forall y\ (P_0(x) \wedge P_0(y) \rightarrow \exists z\ (S_1(x,y,z) \wedge P_1(z)))$

$\forall x \forall y\ (P_1(x) \wedge P_1(y) \rightarrow \exists z\ (S_2(x,y,z) \wedge P_2(z)))$

$\ldots$

$\forall x \forall y\ (P_{n-1}(x) \wedge P_{n-1}(y) \rightarrow \exists z\ (S_n(x,y,z) \wedge P_n(z)))\}$

# Complexity of **ACYCLIC**

- The naïve algorithm shows OBQA(**ACYCLIC**) is
  - in PTIME w.r.t. the data complexity
  - in 2EXPTIME w.r.t. the combined complexity

…however, we can do better than the naïve algorithm

**Theorem:** It holds that

- $\text{OBQA}_{\Sigma,Q}$(**FULL**) is in LOGSPACE (data complexity)
- OBQA(**FULL**) is NEXPTIME-complete (combined complexity)

# Our Simple Example

*D*

person(john)

Σ

$\forall x\, (Person(x) \rightarrow \exists y\, (hasParent(x,y) \wedge Person(y)))$

$chase(D,\Sigma) = D \cup \{hasParent(john, z_1), Person(z_1),$

$hasParent(z_1, z_2), Person(z_2),$

$hasParent(z_2, z_3), Person(z_3), \ldots$

**Existential quantification & recursive definitions**

**are key features for modelling ontologies**

# Research Challenge

We need classes of existential rules such that

- Existential quantification and recursive definition coexist

  $\Rightarrow$ the chase may be infinite

- OBQA is decidable, and tractable w.r.t. the data complexity

$$\Downarrow$$

**Tame the infinite chase:**

**Deal with infinite structures without explicitly building them**

# Linear Existential Rules

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

$$\forall \mathbf{x} \forall \mathbf{y}\, (P(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x},\mathbf{z}))$$

single atom

- We denote **LINEAR** the class of linear existential rules

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

  $\Rightarrow$ given $\Sigma$, we can decide in linear time whether $\Sigma \in$ **LINEAR**

  $\Rightarrow$ 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

| DL-Lite Axioms | First-order Representation |
|---|---|
| $A \sqsubseteq B$ | $\forall x\, (A(x) \rightarrow B(x))$ |
| $A \sqsubseteq \exists R$ | $\forall x\, (A(x) \rightarrow \exists y\, R(x,y))$ |
| $\exists R \sqsubseteq A$ | $\forall x \forall y\, (R(x,y) \rightarrow A(x))$ |
| $\exists R \sqsubseteq \exists P$ | $\forall x \forall y\, (R(x,y) \rightarrow \exists z\, P(x,z))$ |
| $A \sqsubseteq \exists R.B$ | $\forall x\, (A(x) \rightarrow \exists y\, (R(x,y) \wedge B(y)))$ |
| $R \sqsubseteq P$ | $\forall x \forall y\, (R(x,y) \rightarrow P(x,y))$ |
| $A \sqsubseteq \neg B$ | $\forall x\, (A(x) \wedge B(x) \rightarrow \bot)$ |

# Linear Existential Rules

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

$$\forall \mathbf{X} \forall \mathbf{Y} \, (P(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z} \, \psi(\mathbf{X},\mathbf{Z}))$$

single atom

- We denote **LINEAR** the class of linear existential rules

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

  $\Rightarrow$ given $\Sigma$, we can decide in linear time whether $\Sigma \in$ **LINEAR**

  $\Rightarrow$ closed under union

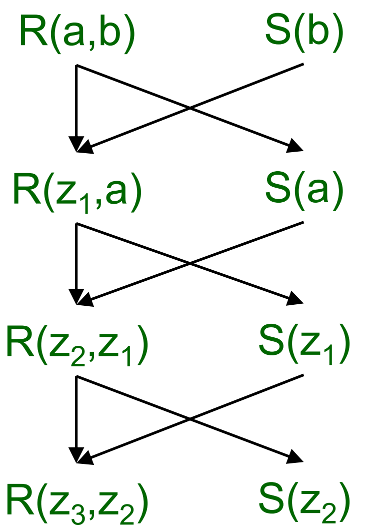
- But, is this a reasonable ontology language? **OWL 2 QL**

# Chase Graph

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

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

$$\Sigma = \begin{cases} \forall x \forall y\ (R(x,y) \wedge S(y) \rightarrow \exists z\ R(z,x)) \\ \forall x \forall y\ (R(x,y) \rightarrow S(x)) \end{cases}$$

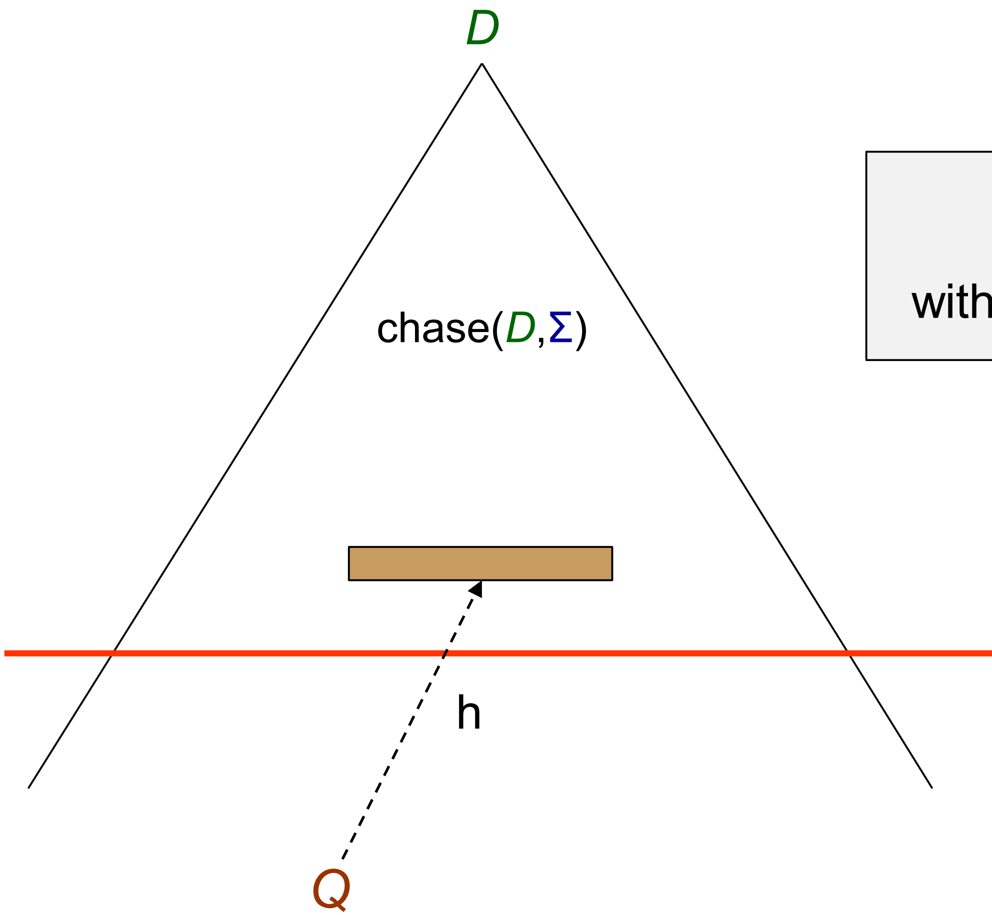



For **LINEAR** the chase graph is a forest

# Bounded Derivation-Depth Property

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

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



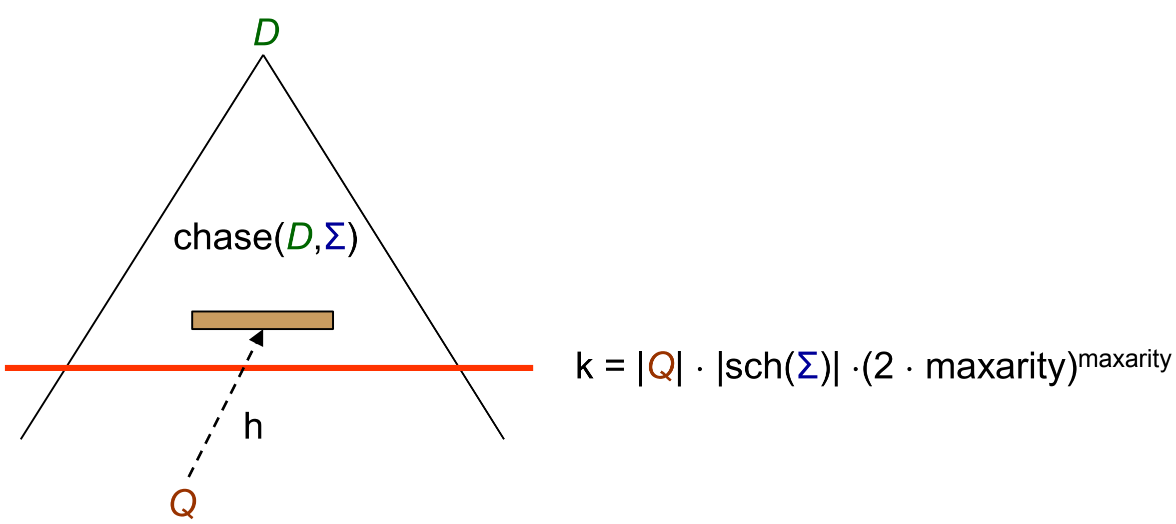

depth k that does not depend on *D*

chase graph up to depth k

$$\text{chase}(D,\Sigma) \models Q \quad \Rightarrow \quad \text{chase}^k(D,\Sigma) \models Q$$

# The Blocking Algorithm for **LINEAR**

- The blocking algorithm shows that OBQA(**LINEAR**) is
  - in PTIME w.r.t. the data complexity
  - in 2EXPTIME w.r.t. the combined complexity



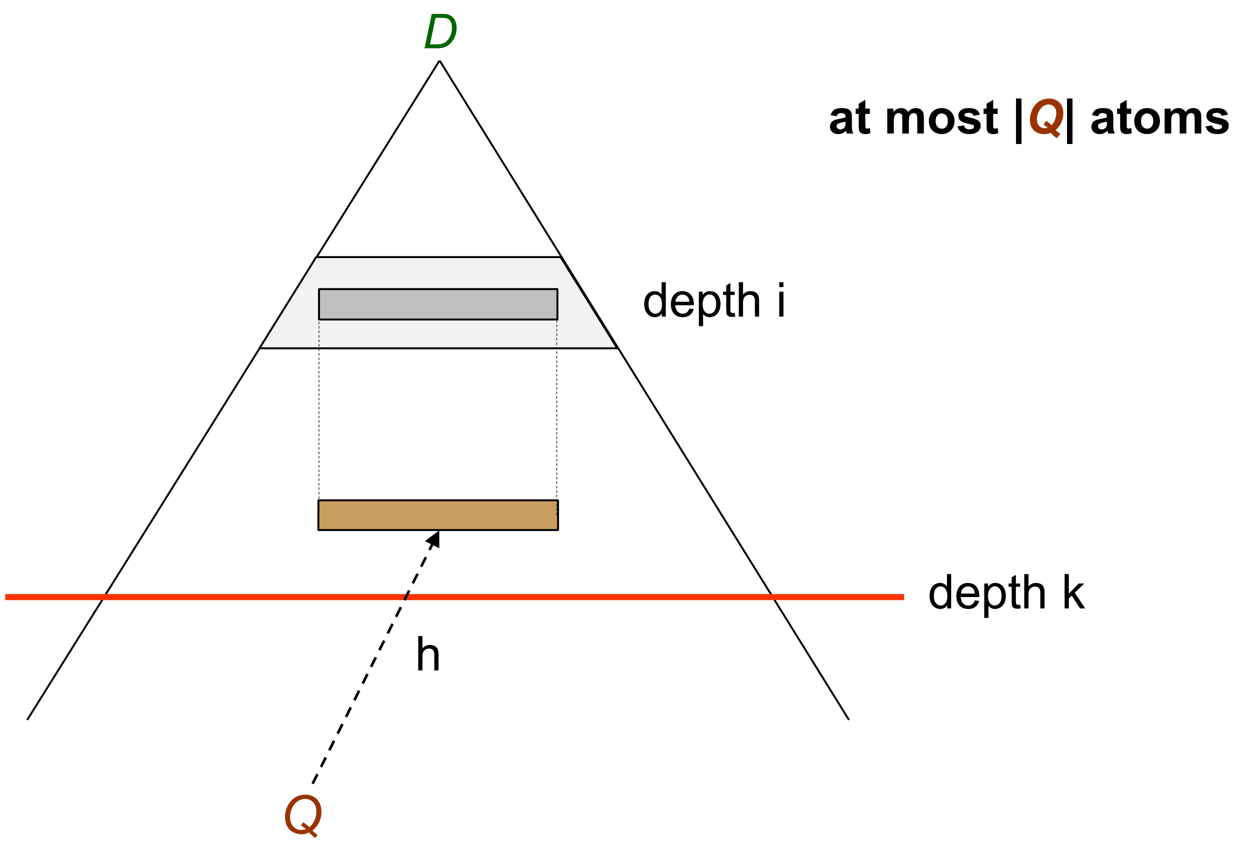
# Complexity of **LINEAR**

…but, we can do better than the blocking algorithm

**Theorem:** It holds that

- $\mathrm{OBQA}_{\Sigma,Q}$(**LINEAR**) is in LOGSPACE (data complexity)
- OBQA(**LINEAR**) is PSPACE-complete (combined complexity)

# Key Observation

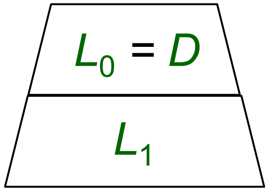



**non-deterministic, level-by-level construction**

# Combined Complexity of **LINEAR**

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

**Proof Idea:**



# Combined Complexity of **LINEAR**

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

**Proof Idea:**

# Combined Complexity of **LINEAR**

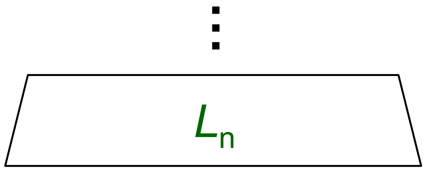
**Theorem:** OBQA(**LINEAR**) is in PSPACE

**Proof Idea:**

# Combined Complexity of **LINEAR**

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

**Proof Idea:**



# Combined Complexity of **LINEAR**

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

**Proof Idea:**

- At each step we need to maintain
  - $O(|Q|)$ atoms
  - A counter ctr $\leq |Q|^2 \cdot |\text{sch}(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$
- Thus, we need polynomial space
- The claim follows since NPSPACE = PSPACE

# Combined Complexity of **LINEAR**

**We cannot do better than the previous algorithm**

**Theorem:** OBQA(**LINEAR**) is PSPACE-hard

**Proof :** By simulating a deterministic polynomial space Turing machine

# PSPACE-hardness of **LINEAR**

**Our Goal:** Encode the polynomial space computation of a DTM $M$ on input string $I$

using a database $D$, a set $\Sigma \in$ **LINEAR**, and a (Boolean) CQ $Q$ such that

$D \wedge \Sigma \models Q$ iff $M$ accepts $I$ using at most $n = |I|^k$ cells

# PSPACE-hardness of **LINEAR**

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that *M* halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

Initial configuration - the database *D*

$$\mathit{Config}(s_{init},\alpha_1,\ldots,\alpha_m,\underbrace{\sqcup,\ldots,\sqcup}_{n - m},1,\underbrace{0,\ldots,0}_{n - 1})$$

# PSPACE-hardness of LINEAR

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that $M$ halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

Transition rule - $\delta(s_1,\alpha) = (s_2,\beta,+1)$

for each $i \in \{1,\ldots,n\}$:

$$\forall \mathbf{x}\ (\mathrm{Config}(s_1,x_1,\ldots,x_{i-1},\alpha,x_{i+1},\ldots,x_n,\overbrace{0,\ldots,0}^{i-1},1,\overbrace{0,\ldots,0}^{n-i}) \rightarrow$$

$$\mathrm{Config}(s_2,x_1,\ldots,x_{i-1},\beta,x_{i+1},\ldots,x_n,\underbrace{0,\ldots,0}_{i},1,\underbrace{0,\ldots,0}_{n-i-1}))$$

# PSPACE-hardness of **LINEAR**

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that *M* halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

$$D \wedge \Sigma \models Q :- \mathrm{Config}(s_{acc}, \mathbf{X}) \quad \text{iff} \quad M \text{ accepts } I$$

…but, the rules are not constant-free

we can eliminate the constants by applying a simple trick

# PSPACE-hardness of **LINEAR**

Initial configuration - the database *D*

auxiliary constants for the states

and the tape alphabet

$$\text{Config}(s_{init},\alpha_1,\ldots,\alpha_m,\underbrace{\sqcup,\ldots,\sqcup}_{n-m},1,\underbrace{0,\ldots,0}_{n-1},s_1,\ldots s_\ell,0,1,\sqcup)$$

# PSPACE-hardness of **LINEAR**

Transition rule - $\delta(s_1,0) = (s_2,\sqcup,+1)$

for each $i \in \{1,\ldots,n\}$:

$$\mathrm{Config}(s_1,x_1,\ldots,x_{i-1},z,x_{i+1},\ldots,x_n,\overbrace{z,\ldots,z}^{i-1},o,\overbrace{z,\ldots,z}^{n-i},s_1,\ldots,s_\ell,z,o,b) \rightarrow$$

$$\mathrm{Config}(s_2,x_1,\ldots,x_{i-1},b,x_{i+1},\ldots,x_n,\underbrace{z,\ldots,z}_{i},o,\underbrace{z,\ldots,z}_{n-i-1},s_1,\ldots,s_\ell,z,o,b)$$

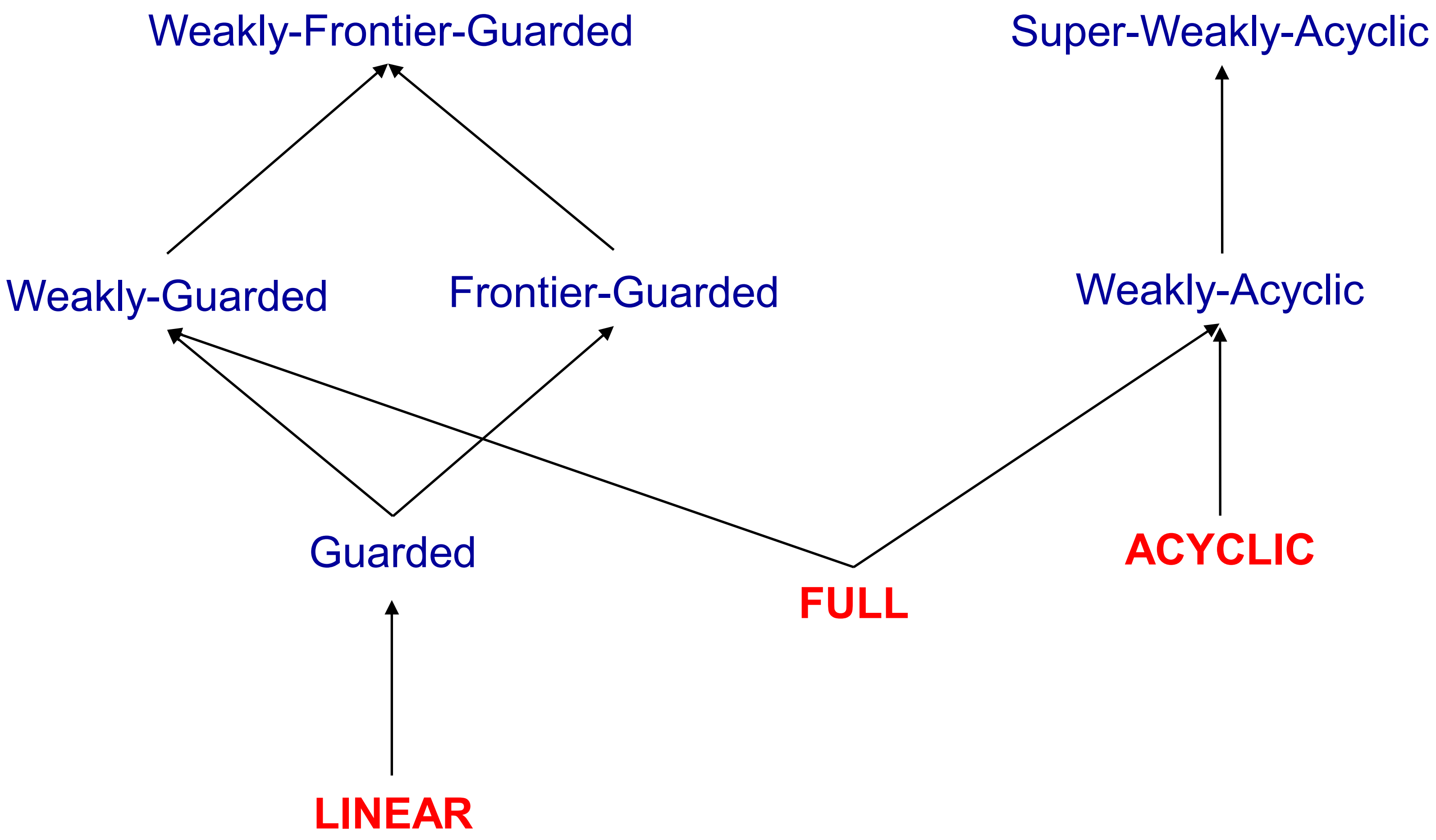
($\forall$-quantifiers are omitted)

# Sum Up

| | Data Complexity | |
|---|---|---|
| **FULL** | PTIME-c | Naïve algorithm |
| | | Reduction from Monotone Circuit Value problem |
| **ACYCLIC** | in LOGSPACE | **Query rewriting** |
| **LINEAR** | | |

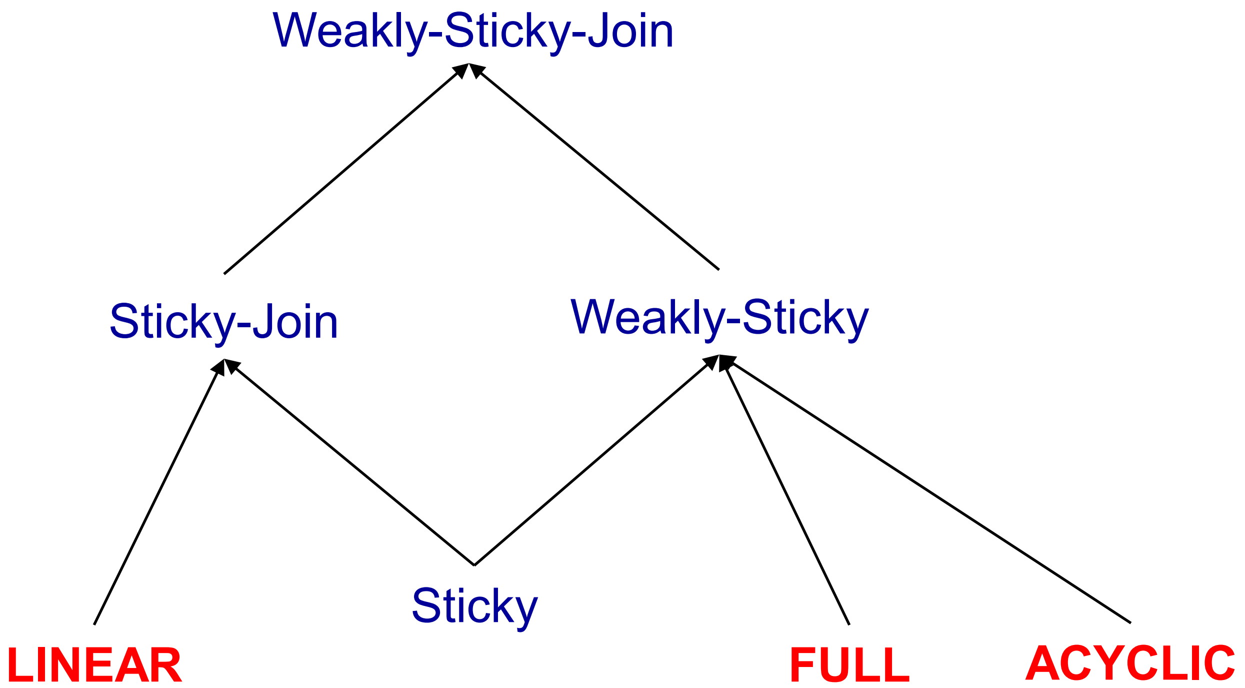
| | Combined Complexity | |
|---|---|---|
| **FULL** | EXPTIME-c | Naïve algorithm |
| | | Simulation of a deterministic exponential time TM |
| **ACYCLIC** | NEXPTIME-c | Small witness property |
| | | Reduction from a Tiling problem |
| **LINEAR** | PSPACE-c | Level-by-level non-deterministic algorithm |
| | | Simulation of a deterministic polynomial space TM |

# Several Other Languages Exist




**Field of intense research**

# Several Other Languages Exist




**Field of intense research**

# Additional Modelling Features

- Counting quantifiers - very little is known

$$\forall x\ (\mathrm{Professor}(x) \rightarrow \exists_{\leq 4} y\ (\mathrm{supervisorOf}(x,y) \wedge \mathrm{Student}(y))$$

- Default negation (or negation as failure) - relatively well-understood

$$\forall x\ (\mathrm{Number}(x) \rightarrow \exists y\ (\mathrm{hasSucc}(x,y) \wedge \mathrm{Number}(y))$$

$$\forall x\ (\mathrm{Number}(x) \wedge \mathrm{not}\ \mathrm{Even}(x) \rightarrow \mathrm{Odd}(x))$$

$$\forall x\ (\mathrm{Number}(x) \wedge \mathrm{not}\ \mathrm{Odd}(x) \rightarrow \mathrm{Even}(x))$$

- Disjunction - relatively well-understood

$$\forall x\ (\mathrm{Number}(x) \rightarrow \mathrm{Even}(x) \vee \mathrm{Odd}(x))$$