Managing and Communicating Object Identities in Knowledge Representation and Information Systems

David Toman†

(joint work with Alexander Borgida† and Grant Weddell‡)

†Department of Computer Science
Rutgers University, New Brunswick, USA
borgida@cs.rutgers.edu

‡Cheriton School of Computer Science
University of Waterloo, Canada
{david,gweddeell}@uwaterloo.ca
IDENTIFYING AND COMMUNICATING REFERENCES
(TO OBJECTS/ENTITIES)
Problem

- Information systems store information about *entities*
- Computers store (arrays of) *ints* and *strings*

How do we bridge the GAP?
Problem

- Information systems store information about *entities*
- Computers store (arrays of) *ints* and *strings*

How do we bridge the GAP?

1. **OIDs** (proxying entity *identity* by a number uniformly in the whole system)
   - typically managed by *The System* (OO languages), or
(Real world) Entities vs. (Computer) Representation(s)

Problem

- Information systems store information about *entities*
- Computers store (arrays of) *ints* and *strings*

How do we bridge the GAP?

1. **OIDs** (proxying entity *identity* by a number uniformly in the whole system)
   ⇒ typically managed by *The System* (OO languages), or
2. **Keys** (proxying entity *identity* by a unique combination of values (local))
   ⇒ typically declared/managed by user (Relational DBMS).
What is Wrong with (explicit) Object IDs?

a.k.a. proxying identities by values in a data type (say \texttt{int})
What is Wrong with (explicit) Object IDs?

a.k.a. proxying identities by values in a data type (say \texttt{int})

### Performance: The PROTEL2 Case

*every object WILL have an OID (say 64 bits)*

⇒ storage/performance overhead (need to be generated/managed)

- can we proxy by (storage) \textit{address}?
- what about memory/storage reuse and/or garbage collection??
- what about data replication??
What is Wrong with (explicit) Object IDs?

a.k.a. proxying identities by *values in a data type* (say int)

Performance: The PROTEL2 Case

Information Integration: The CORBA Case

What happens to an *object* stored in *different ORBs*??

⇒ what does `CORBA::Object::is_equivalent(in Object)` do??
What is Wrong with (explicit) Object IDs?

a.k.a. proxying identities by values in a data type (say `int`)

Performance: The PROTEL2 Case

Information Integration: The CORBA Case

What happens to an object stored in different ORBs??

⇒ what does `CORBA::Object::is_equivalent(in Object)` do??

... and before someone mentions URL/URI/IRIs:
What is Wrong with (explicit) Object IDs?

a.k.a. proxying identities by values in a data type (say int)

Performance: The PROTEL2 Case

Information Integration: The CORBA Case

Unintuitive Answers: RDF/Freebase/... Cases

Freebase  The (object id of the) “Synchronicity” album by “The Police” is /guid/9202a8c04000641f800000002f9e349 (as of April, 2015.)

W3C  URI/IRI/... do not improve the situation

⇒ and RDF introduces additional internal identifiers!
What is Wrong with (explicit) Object IDs?

a.k.a. proxying identities by *values in a data type* (say `int`)

Performance: The PROTEL2 Case

Information Integration: The CORBA Case

Unintuitive Answers: RDF/Freebase/… Cases

Missing (implied) Answers: The OBDA Case

In the presence of *background knowledge* we may *know* that certain objects exist, but we cannot identify/report them due to lack of an *explicit identifier* (example later)
What is Wrong with (explicit) Object IDs?

a.k.a. proxying identities by values in a data type (say \texttt{int})

Performance: The PROTEL2 Case

Information Integration: The CORBA Case

Unintuitive Answers: RDF/Freebase/… Cases

Missing (implied) Answers: The OBDA Case

Alternative Preferred Answers

Internal (computer) addresses vs. physical locations of equipment
  ⇒ programs need electronic address (to route the electric signals)
  ⇒ technicians need physical address (to find the equipment in a data centre)
Goal

Introduce *referring expressions* as an uniform approach to identification of entities in information systems.
Goal of the Tutorial

Goal

Introduce *referring expressions* as an uniform approach to identification of entities in information systems.

Outline

- Referring Expressions in Philosophy/Linguistics
- Logical Foundations: Single Interpretations vs. Models of Theories
- Use of Referring Expressions in Information Systems
  1. Referring Expressions in Answers to Queries over Knowledge Bases
  2. Referring Expressions for Ground Knowledge
  3. Referring Expressions in Conceptual Design
- Summary and Open Problems
Referring Expressions
(Background)
**What is an Referring Expression?**

<table>
<thead>
<tr>
<th>Referring Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>A referring expression in linguistics is any noun phrase identifying an object in a way that will be useful to interlocutors.</td>
</tr>
</tbody>
</table>


A definite description "the $F$ is a $G$" is understood to have the form $\exists x. F(x) \land \forall y. (F(y) \rightarrow x = y) \land G(x)$.

A definite description is a denoting phrase in the form of "the $F$" where $F$ is a noun-phrase or a singular common noun. The definite description is proper if $F$ applies to a unique individual or object.

The discussion of definite and indefinite descriptions (in English, phrases of the form 'the $F$' and 'an $F$') has been at the centre of analytic philosophy for over a century (so we won’t go there today!).
What is an Referring Expression?

A referring expression in linguistics is any noun phrase identifying an object in a way that will be useful to interlocutors.


A definite description “the $F$ is a $G$” is understood to have the form

$$\exists x. F(x) \land \forall y (F(y) \rightarrow x = y) \land G(x)$$

A definite description is a denoting phrase in the form of “the $F$” where $F$ is a noun-phrase or a singular common noun. The definite description is proper if $F$ applies to a unique individual or object.
What is an Referring Expression?

A referring expression in linguistics is any noun phrase identifying an object in a way that will be useful to interlocutors.


A **definite description** “the *F* is a *G*” is understood to have the form

\[ \exists x. F(x) \land \forall y (F(y) \rightarrow x = y) \land G(x) \]

A definite description is a denoting phrase in the form of “the *F*” where *F* is a noun-phrase or a singular common noun. The definite description is proper if *F* applies to a unique individual or object.

The discussion of *definite* and *indefinite* descriptions (in English, phrases of the form ‘the *F*’ and ‘an *F*’) has been at the centre of analytic philosophy for over a century *(so we won’t go there today!)*.
Issues and Criticisms

Referring to Non-existing Object:

“The King of Kentucky (is...)” [Strawson]
(object does NOT exist in this interpretation? or in principle?)

Referring to Object in Context:

“The table (is covered with books)”
(non-unique reference without assuming additional context)

Multiple Reference:

“The Morning Star” vs. “The Evening Star” [Frege]
(multiple distinct references to the same object)

Rigidity:

Should referring expressions identify the same object in all possible worlds? [Kripke, S.: Identity and Necessity, In Identity and Individuation. NYU Press, pp. 135-164 (1971)]
Referring Expressions and (Logical) Theories
Referring to Objects

How do we *communicate* Results of Queries?

Typical solution: tuples of *constant symbols* that, when substituted for free variables, make a query *logically implied* by the Knowledge Base.
Referring to Objects

How do we communicate Results of Queries?

Typical solution: tuples of constant symbols that, when substituted for free variables, make a query logically implied by the Knowledge Base.

1. only explicitly named objects are returned as certain answers
2. often system-generated ids (that aren’t too user-friendly)
Referring to Objects

How do we *communicate* Results of Queries?

Typical solution: tuples of *constant symbols* that, when substituted for free variables, make a query *logically implied* by the Knowledge Base.

1. only *explicitly named* objects are returned as certain answers
2. often *system-generated* ids (that aren’t too user-friendly)

Example (Freebase)

The (object id of the) “Synchronicity” album by “The Police” is /guid/9202a8c04000641f8000000002f9e349 (as of April, 2015.)
Referring to Objects

How do we communicate Results of Queries?

Typical solution: tuples of constant symbols that, when substituted for free variables, make a query logically implied by the Knowledge Base.

1. only explicitly named objects are returned as certain answers
2. often system-generated ids (that aren’t too user-friendly)

Example (Freebase)

The (object id of the) “Synchronicity” album by “The Police” is /guid/9202a8c04000641f80000000002f9e349 (as of April, 2015.)

Referring Expressions

More answers (e.g., objects without explicit name), and/or more informative/preferred answers, e.g.:

\[ \text{ALBUM} \sqcap (\text{title} = \text{“Synchronicity”}) \sqcap (\text{band} = \text{“The Police”}) \]
Russell’s *Definite Descriptions* ... denote exactly one object

What happens if we consider *logical theories* rather than a *particular model*?

- constant symbols
Single Interpretations vs. (non-trivial) Logical Theories

Russell’s *Definite Descriptions* . . . denote exactly one object

What happens if we consider *logical theories* rather than a *particular model*?

- constant symbols
  - . . . can be interpreted by *different individuals* in different models
Russell’s *Definite Descriptions* . . . denote exactly one object

What happens if we consider *logical theories* rather than a *particular model*?

- constant symbols
  - . . . can be interpreted by *different individuals* in different models
  - . . . set of constants may *change* with evolution of the theory (updates!)
Russell’s *Definite Descriptions* ... denote exactly *one* object

What happens if we consider *logical theories* rather than a *particular model*?

- constant symbols
  - ... can be interpreted by *different individuals* in different models
  - ... set of constants may *change* with evolution of the theory (updates!)
- similar issues with other *non-logical symbols*
Russell’s *Definite Descriptions* ... denote exactly one object

What happens if we consider *logical theories* rather than a *particular model*?

- constant symbols
  - ... can be interpreted by *different individuals* in different models
  - ... set of constants may *change* with evolution of the theory (updates!)
- similar issues with other *non-logical symbols*

⇒ (standard) constants don’t quite satisfy Russell’s/Kripke’s requirements
Rigidity and Genericity: DB Theory Way

Why not require constants to be \textit{rigid designators}?

\[
\Rightarrow \text{symbols interpreted identically in all models}
\]
Rigidity and Genericity: DB Theory Way

Why not require constants to be *rigid designators*?
⇒ symbols interpreted identically in all models

**Database (theory) Approach**

- Database Instances (aka models) *use rigid constants*, but
- Database Queries are required to be *generic*
  ⇒ invariant under permutations of the underlying domain

---

**Logical Definition:**

\[
\{ a | K = \varphi[a/x] \}
\]

**DB Definition:**

\[
\bigcap I = K \{ a | I, [x] = a \} = \varphi
\]

(conflates constants with domain elements)

. . . for generic (and domain-independent) queries the result is the same!
Rigidity and Genericity: DB Theory Way

Why not require constants to be *rigid designators*?
⇒ symbols interpreted identically in all models

**Database (theory) Approach**

- Database Instances (aka models) *use rigid constants*, but
- Database Queries are required to be *generic*
  ⇒ invariant under permutations of the underlying domain

**Certain Answers (to $\varphi\{x\}$ in $\mathcal{K}$)**

1. Logical Definition: $\{a \mid \mathcal{K} \models \varphi[a/x]\}$
2. DB Definition: $\bigcap_{I \models \mathcal{K}} \{a \mid I, [x \mapsto a] \models \varphi\}$
   (conflates constants with domain elements)
Rigidity andGenericity:DB Theory Way

Why not require constants to be *rigid designators*?  
⇒ symbols interpreted identically in all models

Database (theory) Approach

- Database Instances (aka models) *use rigid constants*, but  
- Database Queries are required to be *generic*  
  ⇒ invariant under permutations of the underlying domain

Certain Answers (to $\varphi\{x\}$ in $\mathcal{K}$)

1. Logical Definition:  
   $\{a \mid \mathcal{K} \models \varphi[a/x]\}$

2. DB Definition:  
   $\bigcap_{I \models \mathcal{K}} \{a \mid I, [x \mapsto a] \models \varphi\}$
   (conflates constants with domain elements)

...for generic (and domain-independent) queries the result is *the same*!
Referring Expressions

Formulæ $\phi\{x\}$ (in the language of the Knowledge Base)

1. with exactly one free variable ($x$) that are
2. singular with respect to a Knowledge Base $K$, i.e.,

$$|\{o \mid I, [x \mapsto o] \models \phi\}| = 1$$

for all models $I$ of $K$.

$\Rightarrow$ this intuition may be refined w.r.t. queries (e.g., singular among answers)
Referring Expressions

Formulæ $\phi\{x\}$ (in the language of the Knowledge Base)

1. with **exactly one free variable** ($x$) that are
2. **singular** with respect to a Knowledge Base $\mathcal{K}$, i.e.,

$$|\{o \mid \mathcal{I}, [x \mapsto o] \models \phi\}| = 1$$

for all models $\mathcal{I}$ of $\mathcal{K}$.

⇒ this intuition may be refined w.r.t. queries (e.g., singular **among answers**)

Why not **terms**?

Terms (with the standard FO semantics) suffer from **totality**

⇒ must denote **something** in **every** interpretation
Referring to Objects (fine print)

The rest of the presentation is based on


Ontology Based Data Access

(Better query answers when querying Knowledge Bases)
Queries and Ontologies

Ontology-based Data Access

Enriches (query answers over) *explicitly represented data* using *background knowledge* (captured using an *ontology*.)
Queries and Ontologies

Ontology-based Data Access

Enriches (query answers over) *explicitly represented data* using *background knowledge* (captured using an *ontology.*)

Example

- Bob is a BOSS  
- Every BOSS is an EMPloyee

*List all EMPloyees* ⇒ \{Bob\}

Goal: compute all *certain answers*

⇒ answers *common* in all models of KB (aka. answers *logically implied* by KB)
Approaches to Ontology-based Data Access

Main Task

INPUT: Ontology ($\mathcal{T}$), Data ($\mathcal{A}$), and a Query ($Q$)

Knowledge Base ($\mathcal{K}$)

OUTPUT: $\{a \mid \mathcal{K} \models Q[a]\}$

1. Reduction to standard reasoning (e.g., satisfiability)
2. Reduction to querying a relational database

$\Rightarrow$ very good at $\{a \mid \mathcal{A} \models Q[a]\}$ for range restricted $Q$

$\Rightarrow$ what to do with $\mathcal{T}$??

1. incorporate into $Q$ (perfect rewriting for DL-Lite et al. ($AC^0$ logics)); or
2. incorporate into $\mathcal{A}$ (combined approach for $\mathcal{EL}$ (PTIME-complete logics));
   or sometimes both ($\mathcal{CFDI}$ logics).
Issues with the Standard Definition of Answers

“David is a UWaterloo Employee” and “every Employee has a Phone”

**Question:** Does David have a Phone?

**Answer:** YES

Better Answers (possibly)
1. it is a phone with phone # +1(519) 888-4567x34447;
2. it is a UWaterloo phone with an extension x34447;
3. it is a phone in the Davis Centre, Office 3344;
4. it is a Waterloo phone attached to port 0x0123abcd;
5. it is a Waterloo CS phone with inventory # 100034447;
6. it is David's phone (??)
Issues with the Standard Definition of Answers

“David is a UWaterloo Employee” and “every Employee has a Phone”

Question: Does David have a Phone?
Answer: YES

Question: OK, tell me about David’s Phone!
Answer: {}
Issues with the Standard Definition of Answers

“David is a UWaterloo Employee” and “every Employee has a Phone”

Question: Does David have a Phone?
Answer: YES

Question: OK, tell me about David’s Phone!
Answer: {}
Issues with the Standard Definition of Answers

“David is a UWaterloo Employee” and “every Employee has a Phone”

Question: Does David have a Phone?
Answer: YES

Question: OK, tell me about David’s Phone!
Answer: {}

Better Answers (possibly)

1. it is a phone with phone # +1(519) 888-4567x34447;
2. it is a UWaterloo phone with an extension x34447;
3. it is a phone in the Davis Centre, Office 3344;
4. it is a Waterloo phone attached to port 0x0123abcd;
5. it is a Waterloo CS phone with inventory # 100034447;
6. it is David’s phone (??)
Referring Expressions (revisited)

Definition (Singular Referring Expression)

... is a noun phrase that, when used as a query answer, identifies *a particular* object in this query answer.
Definition (Singular Referring Expression)

... is a noun phrase that, when used as a query answer, identifies a particular object in this query answer.

“David is a UWaterloo Employee” and “every Employee has a Phone”

1. it is a phone with phone # "+1(519) 888-4567x34447";
2. it is a UWaterloo phone with extension x34447;
3. it is a phone in the Davis Centre, Office 3344;
4. it is a Waterloo phone attached to port 0x0123abcd;
5. it is a Waterloo CS phone with inventory # 100034447;
6. it is David’s phone;
7. it is the red phone;
Referring Expressions (revisited)

Definition (Singular Referring Expression)

… is a noun phrase that, when used as a query answer, identifies a particular object in this query answer.

“David is a UWaterloo Employee” and “every Employee has a Phone”

1. it is a phone with phone # "+1(519) 888-4567x34447" ; ✓
2. it is a UWaterloo phone with extension x34447 ; ✓
3. it is a phone in the Davis Centre, Office 3344 ; ✓
4. it is a Waterloo phone attached to port 0x0123abcd ;
5. it is a Waterloo CS phone with inventory # 100034447 ;
6. it is David’s phone ;
7. it is the red phone ;
Referring Expressions (revisited)

Definition (Singular Referring Expression)

... is a noun phrase that, when used as a query answer, identifies a particular object in this query answer.

“David is a UWaterloo Employee” and “every Employee has a Phone”

1. it is a phone with phone # "+1(519) 888-4567x34447" ; ✓
2. it is a UWaterloo phone with extension x34447 ; ✓
3. it is a phone in the Davis Centre, Office 3344 ; ✓
4. it is a Waterloo phone attached to port 0x0123abcd ; ✓
5. it is a Waterloo CS phone with inventory # 100034447 ; ✓
6. it is David’s phone ;
7. it is the red phone ;
Referring Expressions (revisited)

**Definition (Singular Referring Expression)**

... is a noun phrase that, when used as a query answer, identifies a particular object in this query answer.

### “David is a UWaterloo Employee” and “every Employee has a Phone”

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>it is a phone <em>with phone # &quot;+1(519) 888-4567x34447&quot;</em>;</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>it is a <em>UWaterloo</em> phone <em>with extension x34447</em>;</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>it is a phone <em>in the Davis Centre, Office 3344</em>;</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>it is a <em>Waterloo</em> phone <em>attached to port 0x0123abcd</em>;</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>it is a <em>Waterloo CS</em> phone <em>with inventory # 100034447</em>;</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>it is <em>David’s</em> phone;</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>it is the <em>red phone</em>;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Referring Expressions (revisited)

Definition (Singular Referring Expression)

... is a noun phrase that, when used as a query answer, identifies a particular object in this query answer.

"David is a UWaterloo Employee" and "every Employee has a Phone"

1. it is a phone *with phone # "+1(519) 888-4567x34447"* ; ✓
2. it is a *UWaterloo* phone *with extension x34447* ; ✓
3. it is a phone *in the Davis Centre, Office 3344* ; ✓
4. it is a *Waterloo* phone *attached to port 0x0123abcd* ; ✓
5. it is a *Waterloo CS* phone *with inventory # 100034447* ; ✓
6. it is *David’s* phone ; ×
7. it is the *red phone* ; ×
Referring Expressions (revisited)

Definition (Singular Referring Expression)

... is a unary formula that, when used as a query answer, identifies *a particular* object in this query answer.

“David is a UWaterloo Employee” and “every Employee has a Phone”

1. it is a phone $x$ s.t. $\text{PhoneNo}(x, "+1(519) 888-4567x34447")$ holds; ✓
2. it is a phone $x$ s.t. $\text{UWPhone}(x) \land \text{PhoneExt}(x, "x34447")$ holds; ✓
3. it is a phone $x$ s.t. $\text{UWRoom}(x, "DC3344")$ holds; ✓
4. it is a phone $x$ s.t. $\text{UWPhone}(x) \land \text{PhonePort}(x, 0x0123abcd)$ holds; ✓
5. it is a phone $x$ s.t. $\text{UWCSPhone}(x) \land \text{InvNo}(x, "100034447")$ holds; ✓
6. it is a phone $x$ s.t. $\text{IsOwner}("David", x)$ holds; ×
7. it is the phone $x$ s.t. $\text{Colour}(x, "red")$ holds; ×
(Certain) Query Answers

Given a query \( \psi\{x_1, \ldots, x_k\} \) and a KB \( \mathcal{K} \):

- Classical answers: substitutions

\[
\theta = \{x_1 \mapsto a_1, \ldots, x_k \mapsto a_k\}
\]

that map free variables of \( \psi \) to constants that appear in \( \mathcal{K} \) and \( \mathcal{K} \models \psi \theta \).
(Certain) Query Answers

Given a query $\psi\{x_1, \ldots, x_k\}$ and a KB $\mathcal{K}$;

- Classical answers: substitutions

$$\theta = \{x_1 \mapsto a_1, \ldots, x_k \mapsto a_k\}$$

that map free variables of $\psi$ to constants that appear in $\mathcal{K}$ and $\mathcal{K} \models \psi\theta$.

- Referring Expression-based answers: $R$-substitutions

$$\theta = \{x_1 \mapsto \phi_1\{x_1\}, \ldots, x_k \mapsto \phi_k\{x_k\}\}$$

where $\phi_i\{x_i\}$ are unary formulæ in the language of $\mathcal{K}$ such that

1. $\forall x_1, \ldots, x_k.(\phi_1 \land \ldots \land \phi_k) \rightarrow \psi$ (soundness)
2. $\exists x_1, \ldots, x_k. (\phi_1 \land \ldots \land \phi_k \land \psi)$ (existence)
3. $\forall x_1, \ldots, x_k, y_i. \phi_1 \land \ldots \land \phi_k \land \psi \land \phi_i[x_i/y_i] \land \psi[x_i/y_i] \rightarrow x_i = y_i$ (singularity)

... are logically implied by $\mathcal{K}$. 
Infinite number of Answers

\[ \mathcal{T} = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \text{Father}(x) \rightarrow \exists y.\text{fatherof}(x, y), \]
\[ \text{Person}(x) \rightarrow \exists y.\text{fatherof}(y, x) \}

\[ \mathcal{A} = \{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \} \]
Infinite number of Answers

\[ \mathcal{T} = \left\{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \right. \\
\left. \text{Father}(x) \rightarrow \text{Person}(x), \right. \\
\left. \text{Father}(x) \rightarrow \exists y. \text{fatherof}(x, y), \right. \\
\left. \text{Person}(x) \rightarrow \exists y. \text{fatherof}(y, x) \right\} \]

\[ \mathcal{A} = \left\{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \right\} \]

Query: Father(x)?
Infinite number of Answers

\[ T = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \text{Father}(x) \rightarrow \exists y. \text{fatherof}(x, y), \]
\[ \text{Person}(x) \rightarrow \exists y. \text{fatherof}(y, x) \}\]

\[ A = \{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \} \]

Query: \text{Father}(x)\

Answers: \( x = \text{fred} \)
Infinite number of Answers

\[ \mathcal{T} = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \quad \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \quad \text{Father}(x) \rightarrow \exists y.\text{fatherof}(x, y), \]
\[ \quad \text{Person}(x) \rightarrow \exists y.\text{fatherof}(y, x) \]
\[ \text{fatherof}(x, z) \land \text{fatherof}(y, z) \rightarrow x = y \} \]

\[ \mathcal{A} = \{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \} \]

Query: Father(x)?

Answers: \( x = \text{fred}, \text{fatherof}(x, \text{mary}) \)
Infinite number of Answers

\[ \mathcal{T} = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \text{Father}(x) \rightarrow \exists y. \text{fatherof}(x, y), \]
\[ \text{Person}(x) \rightarrow \exists y. \text{fatherof}(y, x) \]
\[ \text{fatherof}(x, z) \land \text{fatherof}(y, z) \rightarrow x = y \} \]

\[ \mathcal{A} = \{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \} \]

Query: Father(x)?

Answers: \( x = \text{fred}, \) fatherof(\( x, \) mary), \( \exists y. \) fatherof(\( x, y \) \) \( \land \) fatherof(\( y, \) mary), \ldots
Infinite number of Answers

\[ \mathcal{T} = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \text{Father}(x) \rightarrow \exists y.\text{fatherof}(x, y), \]
\[ \text{Person}(x) \rightarrow \exists y.\text{fatherof}(y, x) \]
\[ \text{fatherof}(x, z) \land \text{fatherof}(y, z) \rightarrow x = y \} \]

\[ \mathcal{A} = \{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \} \]

Query: Father(x)?

Answers: \( x = \text{fred}, \) fatherof(x, mary), \( \exists y.\text{fatherof}(x, y) \land \text{fatherof}(y, \text{mary}), \ldots \)
fatherof(x, fred), \( \exists y.\text{fatherof}(x, y) \land \text{fatherof}(y, \text{fred}), \ldots \)
Infinite number of Answers

\[ \mathcal{T} = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \text{Father}(x) \rightarrow \exists y.\text{fatherof}(x, y), \]
\[ \text{Person}(x) \rightarrow \exists y.\text{fatherof}(y, x) \]
\[ \text{fatherof}(x, z) \land \text{fatherof}(y, z) \rightarrow x = y \} \]

\[ \mathcal{A} = \{ \text{Father}(fred), \text{Person}(mary) \} \]

**Query: Father(x)?**

**Answers:** \( x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y.\text{fatherof}(x, y) \land \text{fatherof}(y, \text{mary}), \ldots \)
\[ \text{fatherof}(x, \text{fred}), \exists y.\text{fatherof}(x, y) \land \text{fatherof}(y, \text{fred}), \ldots \]

**Query: Person(x)?**
Infinite number of Answers

\[ T = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \text{Father}(x) \rightarrow \exists y. \text{fatherof}(x, y), \]
\[ \text{Person}(x) \rightarrow \exists y. \text{fatherof}(y, x) \]
\[ \text{fatherof}(x, z) \land \text{fatherof}(y, z) \rightarrow x = y \} \]

\[ A = \{ \text{Father}(fred), \text{Person}(mary) \} \]

Query: \text{Father}(x)\

Answers: \[ x = fred, \text{fatherof}(x, mary), \exists y. \text{fatherof}(x, y) \land \text{fatherof}(y, mary), \ldots \]
\[ \text{fatherof}(x, fred), \exists y. \text{fatherof}(x, y) \land \text{fatherof}(y, fred), \ldots \]

Query: \text{Person}(x)\

Answers: \[ x = mary, x = fred \]
Infinite number of Answers

\[ \mathcal{T} = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \]
\[ \text{Father}(x) \rightarrow \text{Person}(x), \]
\[ \text{Father}(x) \rightarrow \exists y. \text{fatherof}(x, y), \]
\[ \text{Person}(x) \rightarrow \exists y. \text{fatherof}(y, x) \]
\[ \text{fatherof}(x, z) \land \text{fatherof}(y, z) \rightarrow x = y \} \]

\[ \mathcal{A} = \{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \} \]

Query: Father(x)?

Answers: \( x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y. \text{fatherof}(x, y) \land \text{fatherof}(y, \text{mary}), \ldots \)
\[ \text{fatherof}(x, \text{fred}), \exists y. \text{fatherof}(x, y) \land \text{fatherof}(y, \text{fred}), \ldots \]

Query: Person(x)?

Answers: \( x = \text{mary}, x = \text{fred}, \text{fatherof}(\text{fred}, x) \) (NO!)
Infinite number of Answers

\[
\mathcal{T} = \{ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \land \text{Person}(y)), \\
\text{Father}(x) \rightarrow \text{Person}(x), \\
\text{Father}(x) \rightarrow \exists y. \text{fatherof}(x, y), \\
\text{Person}(x) \rightarrow \exists y. \text{fatherof}(y, x) \\
\text{fatherof}(x, z) \land \text{fatherof}(y, z) \rightarrow x = y \} 
\]

\[
\mathcal{A} = \{ \text{Father}(fred), \text{Person}(mary) \} 
\]

Query: Father(x)?

Answers: 
\( x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y. \text{fatherof}(x, y) \land \text{fatherof}(y, \text{mary}), \ldots \)
\( \text{fatherof}(x, \text{fred}), \exists y. \text{fatherof}(x, y) \land \text{fatherof}(y, \text{fred}), \ldots \)

Query: Person(x)?

Answers: 
\( x = \text{mary}, x = \text{fred}, \text{fatherof}(\text{fred}, x) \) (NO!)
\( \text{fatherof}(x, \text{mary}), \text{fatherof}(x, \text{fred}), \ldots \)
Infinite number of Answers II

\[ T = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ A = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

Query: \text{spouse}(x, \text{mary})?
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

Query: \( \text{spouse}(x, \text{mary})? \)

Answers: \( x = \text{fred} \)
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

Query: \text{spouse}(x, \text{mary})? 

Answers: \( x = \text{fred}, \text{spouse}(x, \text{mary}) \)
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

Query: \( \text{spouse}(x, \text{mary})? \)

Answers: \( x = \text{fred}, \text{spouse}(x, \text{mary}), \exists y.\text{spouse}(x, y) \land \text{spouse}(y, \text{fred}) \), \ldots
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

Query: \text{spouse}(x, \text{mary})?

Answers: \( x = \text{fred}, \text{spouse}(x, \text{mary}), \exists y. \text{spouse}(x, y) \land \text{spouse}(y, \text{fred}), \ldots \)

How many \textit{distinct} answers to \( \exists y. \text{spouse}(x, y) ? \)
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]
\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

Query: \text{spouse}(x, \text{mary})?

Answers: \( x = \text{fred}, \) \( \text{spouse}(x, \text{mary}) \), \( \exists y. \text{spouse}(x, y) \land \text{spouse}(y, \text{fred}) \), \ldots

How many distinct answers to \( \exists y. \text{spouse}(x, y) \)?

\( \text{fred} = \text{spouse}(x, \text{mary}) = \exists y. \text{spouse}(x, y) \land \text{spouse}(y, \text{fred}) = \ldots \)
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

**Query:** \( \text{spouse}(x, \text{mary})? \)

**Answers:** \( x = \text{fred}, \text{spouse}(x, \text{mary}), \exists y . \text{spouse}(x, y) \land \text{spouse}(y, \text{fred}), \ldots \)

**How many distinct answers to \( \exists y . \text{spouse}(x, y)? \)**

\( \text{fred} = \text{spouse}(x, \text{mary}) = \exists y . \text{spouse}(x, y) \land \text{spouse}(y, \text{fred}) = \ldots \)

\( \text{mary} = \text{spouse}(x, \text{fred}) = \exists y . \text{spouse}(x, y) \land \text{spouse}(y, \text{mary}) = \ldots \)
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

---

Query: \( \text{spouse}(x, \text{mary})? \)

Answers: \( x = \text{fred}, \text{spouse}(x, \text{mary}), \exists y . \text{spouse}(x, y) \land \text{spouse}(y, \text{fred}), \ldots \)

---

How many **distinct** answers to \( \exists y . \text{spouse}(x, y)? \)

\[ \text{fred} = \text{spouse}(x, \text{mary}) = \exists y . \text{spouse}(x, y) \land \text{spouse}(y, \text{fred}) = \ldots \]
\[ \text{mary} = \text{spouse}(x, \text{fred}) = \exists y . \text{spouse}(x, y) \land \text{spouse}(y, \text{mary}) = \ldots \]
\[ \text{mary} \neq \text{fred} \text{ (last constraint!)} \]
Infinite number of Answers II

\[ \mathcal{T} = \{ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x), \]
\[ \text{spouse}(x, z) \land \text{spouse}(y, z) \rightarrow x = y \]
\[ \text{spouse}(x, y) \rightarrow x \neq y \} \]

\[ \mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \} \]

Query: \text{spouse}(x, \text{mary})?

Answers: \( x = \text{fred}, \text{spouse}(x, \text{mary}), \exists y.\text{spouse}(x, y) \land \text{spouse}(y, \text{fred}), \ldots \)

How many distinct answers to \( \exists y.\text{spouse}(x, y)? \)

fred = \text{spouse}(x, \text{mary}) = \exists y.\text{spouse}(x, y) \land \text{spouse}(y, \text{fred}) = \ldots

mary = \text{spouse}(x, \text{fred}) = \exists y.\text{spouse}(x, y) \land \text{spouse}(y, \text{mary}) = \ldots

mary \neq \text{fred} (last constraint!)

\Rightarrow \text{exactly 2 distinct certain answers}
Infinite number of Answers: Finite Representation

How do we deal with multiple referring expression answers/preferences/...?

- potentially too many implied answers (infinitely many!)
- potentially too many ways to refer to the same object
### Infinite number of Answers: Finite Representation

**How do we deal with multiple referring expression answers/preferences/...?**

- potentially too many implied answers (infinitely many!)
- potentially too many ways to refer to the same object

---

**Can we (somehow) get ALL answers to Q over \( \mathcal{K} \)?**

Yes (for logics with *recursively enumerable* logical consequence):

for all (tuples of) unary formulas \( \varphi(x) \)

do test if \( \varphi(x) \) is a singular certain answer to \( Q \) in \( \mathcal{K} \).
Infinite number of Answers: Finite Representation

How do we deal with multiple referring expression answers/preferences/...?

- potentially too many implied answers (infinitely many!)
- potentially too many ways to refer to the same object

Can we (somehow) get ALL answers to Q over K?

Yes (for logics with *recursively enumerable* logical consequence):

- for all (tuples of) unary formulas $\varphi(x)$
- do test if $\varphi(x)$ is a singular certain answer to Q in K.

$\Rightarrow$ but is there a *finite representation*?
Example: Horn Logics with Tree Models [DL19]

What to do $\mathcal{EL}^\perp$ (and Horn-$\mathcal{ALC}$)?

- *singularity* requires *role functionality* (not expressible in $\mathcal{EL}^\perp$/Horn-$\mathcal{ALC}$)
Example: Horn Logics with Tree Models [DL19]

What to do $\mathcal{EL}^\perp$ (and Horn-$\mathcal{ALC}$)?

- **singularity** requires *role functionality* (not expressible in $\mathcal{EL}^\perp$/Horn-$\mathcal{ALC}$)
- (Tree) Models of $a : \exists R.C \sqcap \exists R.D$:

  ▶️ $\bullet$
  ▶️ $\bullet$
  ▶️ $\bullet$

  $C \bullet$  $C \bullet$  $\bullet D$

  spurious $R$-successor $C$

  $C \bullet$  $\bullet D$

  just right (canonical)

  $C, D \bullet$

  spurious equality between $R$-successors

$\Rightarrow$ **singular certain answers**: singular in a canonical model
How Does it Work?

Base Case: Instance Retrieval $B(x)$ over $\mathcal{T}$ and $\mathcal{A} = \{a : A\}$

Looping automaton-like construction

⇒ only non-redundant successors in matching tuples
⇒ preserves complexity bounds for both logics
How Does it Work?

Base Case: Instance Retrieval $B(x)$ over $T$ and $A = \{a : A\}$

Looping automaton-like construction

⇒ only non-redundant successors in matching tuples
⇒ preserves complexity bounds for both logics

Generalizations & Limitations

1. General ABoxes and Conjunctive Queries
   ⇒ lots of case analysis followed by existing approaches
2. Finite representation of answers (succinctness??)
3. More Expressive Logics
   ⇒ this will NOT work with at-least restrictions (functionality is fine)
4. Non-Horn Logics
   ⇒ non-unique canonical models
   ⇒ disjunctions in referring expressions (questionable)
Infinite number of Answers: Typing Restrictions

How do we deal with multiple referring expression answers/preferences/...?

- potentially too many implied answers (infinitely many!)
- potentially too many ways to refer to the same object

Referring Expression Types and Typed Queries

Types: \( Rt ::= Pd = \{?\} \mid Rt_1 \land Rt_2 \mid T \to Rt \mid Rt_1; Rt_2 \)

\( \Rightarrow \) each type induces a set of unary formulæ;

Queries: \textbf{select} \( x_1 : Rt_1, \ldots, x_k : Rt_k \) \textbf{where} \( \psi \)

\( \Rightarrow \) \( x_1 : Rt_1, \ldots, x_k : Rt_k \) is called the head, \( \psi \) is the body.
Examples of Typed Queries

Reference via a Single-Attribute Key

“The ssn# of any person with phone 1234567”

\[
\text{select } x : \text{ssn#} = \{?\} \\
\text{where } \text{Person}(x) \land \text{phone#}(x, 1234567)
\]
Examples of Typed Queries

Reference via a Single-Attribute Key

Reference by a Multi-Attribute Key

“The title and publisher of any journals”

```
select x : title = {?} ∧ publishedBy = {?}
where Journal(x)
```
Examples of Typed Queries

Reference via a Single-Attribute Key

Reference by a Multi-Attribute Key

Choice of Identification in a Heterogeneous Set

“Any legal entity”

\[
\text{select } x : \text{Person} \rightarrow \text{ssn#} = \{?\} ;
\text{Company} \rightarrow \text{tickerSymbol} = \{?\}
\text{where LegalEntity}(x)
\]

answers: \{x \mapsto \text{Person}(x) \land \text{ssn#}(x, 7654)\}
\{x \mapsto \text{Company}(x) \land \text{tickerSymbol}(x, \text{"IBM")}\}
Examples of Typed Queries

Reference via a Single-Attribute Key

Reference by a Multi-Attribute Key

Choice of Identification in a Heterogeneous Set

Preferred Identification

“Any publication, identified by its most specific identifier, when available.”

\[
\begin{align*}
\textbf{select} \ x : & \ Journal \rightarrow (\text{title} = \{?\} \land \text{publisher} = \{?\}) ; \\
& \quad \text{EditedCollection} \rightarrow \text{isbn#} = \{?\} ; \{?\} \\
\textbf{where} \ & \ Publication(x)
\end{align*}
\]

answers: \[
\begin{align*}
\{ \ x & \mapsto Journal(x) \land \text{title}(x, \text{“AIJ”}) \land \text{publisher}(x, \text{“Elsevier”}) \} \\
\{ \ x & \mapsto \text{EditedCollection}(x) \land \text{isbn#}(x, 123456789) \} \\
\{ \ x & \mapsto x = /\text{guid}/9202a8c04000641f8000000... \}
\end{align*}
\]
How do we deal with multiple referring expression answers/preferences/...?
How do we deal with multiple referring expression answers/preferences/...?

Desiderata: only Referring Expressions that Conform to a certain

Given

1. a KB $\mathcal{K}$ (the “background knowledge”),
2. a query $\psi\{x_1, \ldots, x_k\}$, and
3. (specifications of) sets of unary formulæ $S_1, \ldots, S_k$

We ask whether, for every $\mathcal{K}'$ (the “data”) consistent with $\mathcal{K}$ and an answer

$$\theta = \{x_1 \mapsto \phi_1\{x_1\}, \ldots, x_k \mapsto \phi_k\{x_k\}\}$$

to $\psi$ with respect to $\mathcal{K} \cup \mathcal{K}'$ such that $\phi_i \in S_i$, it is the case that $\theta$ is singular.
How do we deal with multiple referring expression answers/preferences/...?

Desiderata: only Referring Expressions that Conform to a certain

Given 1 a KB $\mathcal{K}$ (the “background knowledge”),
2 a query $\psi\{x_1, \ldots, x_k\}$, and
3 (specifications of) sets of unary formulæ $S_1, \ldots, S_k$

We ask whether, for every $\mathcal{K}'$ (the “data”) consistent with $\mathcal{K}$ and an answer

$$\theta = \{x_1 \mapsto \phi_1\{x_1\}, \ldots, x_k \mapsto \phi_k\{x_k\}\}$$

to $\psi$ with respect to $\mathcal{K} \cup \mathcal{K}'$ such that $\phi_i \in S_i$, it is the case that $\theta$ is singular.

Theorem (Weak Identification; paraphrased)

Given a query $\psi$ with a head $H$ and a KB $\mathcal{K}$, the question

“are all answers to $\psi$ conforming to $H$ over any $\mathcal{K} \cup \mathcal{K}'$ singular?”

reduces to logical implication in the underlying logic of $\mathcal{K}$. 
REQA (Referring Expression-based QA)

GOAL: reduce REQA to standard OBDA (used as an oracle)
REQA (outline, unary queries only)

GOAL: reduce REQA to standard OBDA (used as an oracle)

Input: \(\mathcal{K}\) (background knowledge), \(\mathcal{K}'\) (data), \(\psi\{x\}\) (query), \(H\) (query head)

1. Normalize \(H\) to \(H_1; \ldots; H_\ell\), each of the form
   \[ T_i \rightarrow Pd_{i,1} = \{?\} \land \ldots \land Pd_{i,k_i} = \{?\}; \]

2. Create queries \(\psi_i\{x, y_1, \ldots, y_{k_i}\}\) as
   \[ \psi \land T_i(x) \land Pd_{i,1}(x, y_1) \land \ldots \land Pd_{i,k_i}(x, y_{k_i}); \]

3. Create \(\mathcal{K}_i\) with a witnesses for \(x\) when no such witness exists;

4. Evaluate \(\mathcal{K} \cup \mathcal{K}' \cup \mathcal{K}_i \models \psi_i\) (OBDA oracle);

5. Resolve preferences (based on value of \(x\)); and

6. Reconstruct a referring expression from the values of \(y_1, \ldots, y_{k_i}\).

... extends naturally to higher arity queries: (more) messy
The Tractable (practical) Cases

\( \text{DL-Lite}^{\mathcal{F}}_{\text{core}}(idc) : \)

- Weak identification \( \longrightarrow \) sequence of KB consistency tests
- Query answering \( \longrightarrow \) REQA
  + Witnesses for \( x \) w.r.t. \( H \) + Perfect Reformulation

\( \mathcal{CFDI}^\forall_{nc} : \)

- Weak identification \( \longrightarrow \) sequence of logical implications
- Query answering \( \longrightarrow \) REQA
  + Combined Combined Approach
The Tractable (practical) Cases

$DL\text{-}Lite^F_{core}(idc)$:

- Weak identification $\rightarrow$ sequence of KB consistency tests
- Query answering $\rightarrow$ REQA
  + Witnesses for $x$ w.r.t. $H$ + Perfect Reformulation

$CFD\mbox{I}_{nc}$:

- Weak identification $\rightarrow$ sequence of logical implications
- Query answering $\rightarrow$ REQA
  + Combined Combined Approach

Logics with Tree Models (outside an ABox) [AI16]

The witnesses for anonymous objects (step (3)) $\rightarrow$ last named individual on a path towards the anonymous object
RECORDING/REPRESENTING FACTUAL DATA
Referring Expressions for Ground Knowledge

Standard approach: constant symbols ~ objects (and values!)

⇒ needs a constant symbol for *every individual* (Skolems?)

How are external objects identified in a KB?

Two PERSON objects, \( o_1 \) and \( o_2 \), identified by their ssn value:

\[
\text{PERSON} \sqcap \exists \text{ssn}. \{ 123 \} \text{ and PERSON} \sqcap \exists \text{ssn}. \{ 456 \}.
\]

Role (feature) assertions of the form \( \text{mother}(o_1) = o_2 \) can then be captured as:

\[
\text{PERSON} \sqcap \exists \text{ssn}. \{ 123 \} \sqcap \exists \text{mother}. (\text{PERSON} \sqcap \exists \text{ssn}. \{ 345 \}).
\]

Issues:

admissibility: what descriptions qualify here?

⇒ singularity!

minimality: is the description succinct? (similar to keys/superkeys issues)
Referring Expressions for Ground Knowledge

Standard approach: constant symbols $\sim$ objects (and values!)

$\Rightarrow$ needs a constant symbol for *every individual* (Skolems?)

How are *external* objects identified in a KB?

- Two PERSON objects, $o_1$ and $o_2$, identified by their ssn value:
  \[
  \text{PERSON} \sqsubseteq \exists ssn.\{123\} \quad \text{and} \quad \text{PERSON} \sqsubseteq \exists ssn.\{456\}.
  \]

- Role (feature) assertions of the form $mother(o_1) = o_2$ can then be captured as:
  \[
  \text{PERSON} \sqsubseteq \exists ssn.\{123\} \sqcap \exists mother.\left(\text{PERSON} \sqsubseteq \exists ssn.\{345\}\right).
  \]
Referring Expressions for Ground Knowledge

Standard approach: constant symbols \( \sim \) objects (and values!)

\[ \Rightarrow \text{needs a constant symbol for } \textit{every individual} \text{ (Skolems?)} \]

How are \textit{external} objects identified in a KB?

- Two \texttt{PERSON} objects, \( o_1 \) and \( o_2 \), identified by their \texttt{ssn} value:
  
  \[
  \texttt{PERSON} \sqcap \exists \texttt{ssn}.\{123\} \quad \text{and} \quad \texttt{PERSON} \sqcap \exists \texttt{ssn}.\{456\}.
  \]

- Role (feature) assertions of the form \( \text{mother}(o_1) = o_2 \) can then be captured as:
  
  \[
  \texttt{PERSON} \sqcap \exists \texttt{ssn}.\{123\} \sqcap \exists \texttt{mother}.(\texttt{PERSON} \sqcap \exists \texttt{ssn}.\{345\}).
  \]

Issues:

- admissibility: what descriptions qualify here? \( \Rightarrow \text{singularity!} \)
- minimality: is the description succinct? (similar to keys/superkeys issues)
Referring Expressions for Ground Knowledge

Example

JSON fragment describing persons, hypothetically occurring in a MongoDB document source:

```json
{
    "fname" : "John",  "lname" : "Smith",  "age" : 25,
    "phoneNum" : [
        {
            "loc" : "home",  "dialnum" : "212 555-1234"},
        {
            "loc" : "work",  "dialnum" : "212 555-4567"}
    ]
}
```

can be naturally and directly represented as a CBox assertion of the form

```
PERSON ⊓ (∃fname.{"John"}) ⊓ (∃lname.{"Smith"}) ⊓ ∃age.{25}
    ⊓ ∃phoneNumFor⁻¹.((∃loc.{"home"}) ⊓ (∃dialnum.{"212 555-1234"})))
    ⊓ ∃phoneNumFor⁻¹.((∃loc.{"work"}) ⊓ (∃dialnum.{"212 555-4567"})))
```

This assertion is admissible, e.g., whenever the combination of *fname* and *lname* identifies PERSONs.
Heterogeneous Data Integration (example)

Example

- **TBox** $\mathcal{T} = \{ \text{FRIEND} \sqsubseteq \text{PERSON}, \\
\text{FRIEND} \sqsubseteq \text{PERSON} : \text{fname} \rightarrow \text{id}, \\
\text{MATRIARCH} \sqsubseteq \text{PERSON}, \\
\text{MATRIARCH} \sqsubseteq \text{PERSON} : \text{lname} \rightarrow \text{id}, \\
\text{PERSON} \sqsubseteq \text{PERSON} : \text{fname}, \text{lname} \rightarrow \text{id}, \ldots \}$

- **CBox** $\mathcal{C} = \{ \text{FRIEND} \sqcap \exists \text{fname}.\{\text{“Mary”}\}, \\
\text{PERSON} \sqcap (\exists \text{fname}.\{\text{“Mary”}\}) \sqcap (\exists \text{lname}.\{\text{“Smith”}\}), \\
\text{MATRIARCH} \sqcap \exists \text{lname}.\{\text{“Smith”}\}, \ldots \}$
Heterogeneous Data Integration (example)

Example

**TBox**  \( \mathcal{T} = \{ \) FRIEND \( \sqsubseteq \) PERSON,  
FRIEND \( \sqsubseteq \) PERSON : fname \( \rightarrow \) id,  
MATRIARCH \( \sqsubseteq \) PERSON,  
MATRIARCH \( \sqsubseteq \) PERSON : lname \( \rightarrow \) id,  
PERSON \( \sqsubseteq \) PERSON : fname, lname \( \rightarrow \) id, \ldots  \} \)

**CBox**  \( \mathcal{C} = \{ \) FRIEND \( \sqcap \exists \) fname.\{“Mary”\},  
PERSON \( \sqcap ( \exists \) fname.\{“Mary”\}) \( \sqcap ( \exists \) lname.\{“Smith”\}),  
MATRIARCH \( \sqcap \exists \) lname.\{“Smith”\}, \ldots  \} \)

Heterogeneous Identification

“FRIEND \( \sqcap \exists \) fname.\{“Mary”\}” identifies the same object as “PERSON \( \sqcap ( \exists \) fname.\{“Mary”\}) \( \sqcap ( \exists \) lname.\{“Smith”\})” and in turn as “MATRIARCH \( \sqcap \exists \) lname.\{“Smith”\}”
Heterogeneous Data Integration (example)

Example

- **TBox** \( \mathcal{T} = \{ \)  
  FRIEND \( \sqsubseteq \) PERSON,  
  FRIEND \( \sqsubseteq \) PERSON : \( \text{fname} \rightarrow \text{id} \),  
  Matriarch \( \sqsubseteq \) PERSON,  
  Matriarch \( \sqsubseteq \) PERSON : \( \text{lname} \rightarrow \text{id} \),  
  PERSON \( \sqsubseteq \) PERSON : \( \text{fname}, \text{lname} \rightarrow \text{id} \), \ldots \)

- **CBox** \( \mathcal{C} = \{ \)  
  FRIEND \( \sqcap \exists \text{fname}. \{ \text{“Mary”} \}, \)  
  PERSON \( \sqcap ( \exists \text{fname}. \{ \text{“Mary”} \} ) \sqcap ( \exists \text{lname}. \{ \text{“Smith”} \} ), \)  
  Matriarch \( \sqcap \exists \text{lname}. \{ \text{“Smith”} \}, \ldots \)

Heterogeneous Identification

“FRIEND \( \sqcap \exists \text{fname}. \{ \text{“Mary”} \} ” \) identifies the same object as  
“PERSON \( \sqcap ( \exists \text{fname}. \{ \text{“Mary”} \} ) \sqcap ( \exists \text{lname}. \{ \text{“Smith”} \} ) ” \) and in turn as  
“Matriarch \( \sqcap \exists \text{lname}. \{ \text{“Smith”} \} ” \)

\( \ldots \) and thus is an answer to \( \{ x \mid \text{Matriarch}(x) \} \).
IDEA: minimal referring expressions (ala Candidate Keys)

\( C \) is a referring expression singular w.r.t. a TBox \( \mathcal{T} \) (e.g., a superkey)

- \( C \)'s subconcepts \( A, \{a\}, \exists f . \top, \exists f^{-1} . \top \), and \( \top \sqcap \top \) are leaves of \( C \).
- \( C[L \mapsto \top] \) is a description \( C \) in which a leaf \( L \) was replaced by \( \top \).
- “first-leaf” and “next-leaf” successively enumerate all leaves of \( C \).

\[
\begin{align*}
1. \quad & L := \text{first-leaf}(C); \\
2. \quad & \text{while } C[L \mapsto \top] \text{ is singular w.r.t. } \mathcal{T} \text{ do} \\
3. \quad & C := C[L \mapsto \top]; L := \text{next-leaf}(C); \\
4. \quad & \text{done} \\
5. \quad & \text{return } C;
\end{align*}
\]
C is a referring expression singular w.r.t. a TBox $T$ (e.g., a superkey)

- $C$’s subconcepts $A$, $\{a\}$, $\exists f. \top$, $\exists f^{-1}. \top$, and $\top \sqcap \top$ are leaves of $C$.
- $C[L \mapsto \top]$ is a description $C$ in which a leaf $L$ was replaced by $\top$.
- “first-leaf” and “next-leaf” successively enumerate all leaves of $C$.

1. $L := \text{first-leaf}(C)$;
2. while $C[L \mapsto \top]$ is singular w.r.t. $T$ do
3. $C := C[L \mapsto \top]; \quad L := \text{next-leaf}(C)$;
4. done
5. return $C$;

⇒ computes a syntactically-minimal co-referring expression for $C$.
⇒ order of enumeration → variant minimal co-referring expressions.
Reasoning and QA with CBoxes [DL18]

Theorem (CBox Admissibility)

Let $T$ be a $\mathcal{CFD}^{\forall}_{nc}$ TBox and $C$ a concept description. Then $C$ is a singular referring expression w.r.t. $T$ if and only if the knowledge base

$$(T \cup \{A \sqsubseteq \neg B\}, \operatorname{Simp}(a : C) \cup \operatorname{Simp}(b : C) \cup \{a : A, b : B\})$$

is inconsistent, where $a$ and $b$ are distinct constant symbols, and $A$ and $B$ are primitive concepts not occurring in $T$ and $C$.

Theorem (Satisfiability of KBs with CBoxes)

Let $\mathcal{K} = (T, C)$ be a knowledge base with an admissible CBox $C$. Then $\mathcal{K}$ is consistent iff $(T, \operatorname{Simp}(C))$ is consistent.

Theorem (Query Answering)

Let $\mathcal{K} = (T, C)$ be a consistent knowledge base and $Q = \{(x_1, \ldots, x_k) : \varphi\}$ a conjunctive query over $\mathcal{K}$. Then $(C_1, \ldots, C_k)$ is a certain answer to $Q$ in $\mathcal{K}$ if and only if $(a_{C_1}, \ldots, a_{C_k})$ is a certain answer to $Q$ over $(T, \operatorname{Simp}(C))$. 
CONCEPTUAL MODELLING

(Decoupling *modelling* from *identification* issues)
Modeling of *Entities* and their *Relationships* **should be decoupled** from issues of *managing the identity* of such entities.
Thesis:

Modeling of *Entities* and their *Relationships* **should be decoupled** from issues of *managing the identity* of such entities.

**Weak Entities** and dominant entity identification

Example (**ROOM within BUILDING**)  

For the entity set **ROOM** with attributes **room-number** and **capacity**  
⇒ natural attributes are **insufficient** to identify **ROOMs**  
⇒ need for a *key* of dominant set, such as **BUILDING**
Conceptual Modeling and Identification [ER16]

Thesis:
Modeling of *Entities* and their *Relationships* should be decoupled from issues of *managing the identity* of such entities.

**Weak Entities** and dominant entity identification

**Preferred Identification** in sub/super-classes

**Example** (*PERSON* and *FAMOUS-PERSON*)

For the entity set *FAMOUS-PERSON* a sub-entity of *PERSON*

⇒ choice of key (*ssn*) for *PERSON* forces *the same* key for *FAMOUS-PERSON*

⇒ we may *prefer* to use *name* in this case (e.g., *Eric Clapton* or *The Edge*)
Thesis:

Modeling of *Entities* and their *Relationships* **should be decoupled** from issues of *managing the identity* of such entities.

Weak Entities and dominant entity identification

Preferred Identification in sub/super-classes

Generalizations and heterogeneity

Example (*LEGAL-ENTITY*: PERSON or COMPANY)

For the entity set *LEGAL-ENTITY* a generalization of PERSON and COMPANY

⇒ **commonly required** to create an *artificial* attribute le-num

⇒ despite the fact that all entities are already identified by the (more) natural ssn and (name, city) identifiers.
Conceptual Modeling and Identification [ER16]

Thesis:
Modeling of *Entities* and their *Relationships* should be *decoupled* from issues of *managing the identity* of such entities.

**Weak Entities** and dominant entity identification

**Preferred Identification** in sub/super-classes

**Generalizations** and heterogeneity

**Contributions**

1. Methodology that allows decoupling identification from modeling;
2. Referring Expressions that subsequently resolve identity issues; and
3. Compilation-based technology that makes further translation to a *pure relational model* seamless.
Abstract (Relational) Model ARM

A simple conceptual model \( C \)

Common features of so-called “attribute-based” semantic models

⇒ class hierarchies, disjointness, coverage, attributes and typing, functional dependencies, …

Example (DMV)

class PERSON (ssn: INT, name: STRING,
    isa LEGAL-ENTITY, disjoint with VEHICLE)
class COMPANY (name: STRING, city: STRING,
    isa LEGAL-ENTITY)
class LEGAL-ENTITY (covered by PERSON, COMPANY)
class VEHICLE (vin: INT, make: STRING,
    owned-by: LEGAL-ENTITY)
class CAN-DRIVE (driver: PERSON, driven: VEHICLE)
Abstract (Relational) Model ARM

A simple conceptual model ARM

Common features of so-called “attribute-based” semantic models
⇒ class hierarchies, disjointness, coverage, attributes and typing, functional dependencies, …

Example (DMV and Relational Understanding)

table PERSON (self: OID, ssn: INT, name: STRING, isa LEGAL-ENTITY, disjoint with VEHICLE)
table COMPANY (self: OID, name: STRING, city: STRING, isa LEGAL-ENTITY)
table LEGAL-ENTITY (covered by PERSON, COMPANY)
table VEHICLE (self: OID, vin: INT, make: STRING, owned-by: LEGAL-ENTITY)
table CAN-DRIVE (self: OID, driver: PERSON, driven: VEHICLE)
SQLP

(pretty) standard select-from-where-union-except SQL syntax

...with extensions to ARM: abstract attributes (OID) and attribute paths
Abstract Relational Queries

SQLP

(pretty) standard select-from-where-union-except SQL syntax
...with extensions to ARM: abstract attributes (OID) and attribute paths

- The name of anyone who can drive a vehicle made by Honda:
  ```sql
  select d.driver.name from CAN-DRIVE d
  where d.driven.make = 'Honda'
  ```

- attribute paths in the select and where clauses

- The owners of Mitsubishi vehicles:
  ```sql
  select v.owned-by from VEHICLE v
  where v.make = 'Mitsubishi'
  ```

retrieving abstract attributes may yield heterogeneous results (PERSONS and COMPANIES)
Abstract Relational Queries

SQLP

(pretty) standard select-from-where-union-except SQL syntax
...with extensions to ARM: abstract attributes (OID) and attribute paths

- **The name of anyone who can drive a vehicle made by Honda:**
  ```sql
  select d.driver.name from CAN-DRIVE d
  where d.driven.make = 'Honda'
  ```

  *attribute paths in the select and where clauses*

- **The owners of Mitsubishi vehicles:**
  ```sql
  select v.owned-by from VEHICLE v
  where v.make = 'Mitsubishi'
  ```

  retrieving *abstract attributes* may yield
  *heterogeneous results* (**persons** and **companies**)

Note that queries do **NOT** rely on *(external) identification* of entities/objects.
How to Make this Technology Succeed?

1. ARM/SQLP Helps Users (User Study) [EKAW18]

2. ARM/SQLP Can be Efficiently Implemented [ER16]
   - Mapping to *standard relational model* with the help of *referring expressions*
   - Reverse-Engineering ARM from Legacy Relational Schemata
Experimental Design (HCI experiments)

Hypotheses

\( H_f \): no difference between RM/SQL and ARM/SQLP in the mean time taken
\( H_c \): no difference between RM/SQL and ARM/SQLP in the mean correctness
Experimental Design (HCI experiments)

Hypotheses

\( H_t \): no difference between RM/SQL and ARM/SQLP in the mean time taken
\( H_c \): no difference between RM/SQL and ARM/SQLP in the mean correctness

Methods

- Undergraduate (9) and Graduate (15) UW students
- Protocol
  1. Instructions (5") and Examples of SQL/SQLP (10")
  2. Six Questions (Q1–Q6), no time limit
  3. Subjects recorded start/end times for each Question
- Performance Assessment
  1. 3 assessors
  2. agreed upon grading scale

Borgida, Toman, and Weddell
Referring Expressions and Information Systems
Experiments 42 / 56
Course Enrollment as an RM Schema

Diagram:

- **DEPARTMENT**
  - deptcode
  - deptname

- **COURSE**
  - deptcode
  - cnum
  - cname

- **CLASS**
  - deptcode
  - cnum
  - term
  - section
  - pnum

- **ENROLLMENT**
  - snum
  - deptcode
  - cnum
  - term
  - section

- **STUDENT**
  - snum
  - sname
  - year

- **MARK**
  - snum
  - deptcode
  - cnum
  - term
  - section
  - day
  - time
  - room
  - grade
Course Enrolment as an ARM Schema

ARM completely frees domain experts/users from the need to understand how entities are identified in an information system.
ARM \textit{completely frees} domain experts/users from the need to understand how entities are \textit{identified} in an information system.
Example Queries

Query: *Names of students who have been taught by ‘Prof. Alan John’*

**RM/SQL:**

```sql
select distinct s.sname as name
from STUDENT s, ENROLLMENT e, CLASS c, PROFESSOR p
where e.snum = s.snum
and e.deptcode = c.deptcode and e.cnum = c.cnum
and e.term = c.term and e.section = c.section
and c.pnum = p.pnum and p.pname = 'Alan John'
```
Example Queries

Query: *Names of students who have been taught by ‘Prof. Alan John’*

**RM/SQL:**

```sql
select distinct s.sname as name
from STUDENT s, ENROLLMENT e, CLASS c, PROFESSOR p
where e.snum = s.snum
and e.deptcode = c.deptcode and e.cnum = c.cnum
and e.term = c.term and e.section = c.section
and c.pnum = p.pnum and p.pname = 'Alan John'
```

Domain expert needs to understand structure of PK/FKs: **BAD!!**
Example Queries

Query: *Names of students who have been taught by ‘Prof. Alan John’*

**RM/SQL:**

```sql
select distinct s.sname as name
from STUDENT s, ENROLLMENT e, CLASS c, PROFESSOR p
where e.snum = s.snum
and e.deptcode = c.deptcode and e.cnum = c.cnum
and e.term = c.term and e.section = c.section
and c.pnum = p.pnum and p.pname = 'Alan John'
```

Domain expert needs to understand structure of PK/FKs: BAD!!

**ARM/SQLP:**

```sql
select distinct e.student.sname as name
from ENROLLMENT e
where e.class.professor.pname = 'Alan John'
```
select distinct e.student.sname as name
from ENROLLMENT e
where e.class.professor.pname = 'Alan John'
Experiments: Results

Mean performance for all subjects: SQL solid; SQLP dashed.

Mean time taken (seconds)
Normal theory 95% confidence intervals for means

Mean correctness (Scores 0 to 4)
Normal theory 95% confidence intervals for means

Question
Code understanding: Q1, Q2, Q3
Code writing: Q4, Q5, Q6

Method
- SQL
- SQLP
Experiments: Results

Mean performance for all subjects: **SQL solid; SQLP dashed.**

- SQLP outperforms SQL in time taken
- No significant difference in correctness (Q3, Q5 almost significant)
How to make the Technology Succeed?

1. ARM/SQLP Helps Users (User Study)

2. ARM/SQLP Can be Efficiently Implemented [ER16]
   - Mapping to *standard relational model* with the help of *referring expressions*
   - Reverse-Engineering ARM from Legacy Relational Schemata
Referring to Abstract Entities

Example (How to refer to \texttt{LEGAL-ENTITY})

- invent a \textit{new attribute for this purpose} (will be \textit{inherited} by subclasses)
Referring to Abstract Entities

Example (How to refer to `LEGAL-ENTITY`)

- invent a **new attribute** for this purpose (will be **inherited** by subclasses)
- use (a combination of) the identities of **generalized** entities, e.g.,
  
  \[ \text{ssn for PERSON and (name, city) for COMPANY.} \]
Referring to Abstract Entities

Example (How to refer to **LEGAL-ENTITY**)

- invent a *new attribute for this purpose* (will be *inherited* by subclasses)
- use (a combination of) the identities of *generalized* entities, e.g.,
  
  \[\text{ssn for PERSON and (name, city) for COMPANY.}\]

⇒ but what happens to objects *that are both a PERSON and a COMPANY*??
Referring to Abstract Entities

Example (How to refer to **LEGAL-ENTITY**)

- Invent a new attribute for this purpose (will be inherited by subclasses).
- Use (a combination of) the identities of generalized entities, e.g.,
  
  ssn for PERSON and (name, city) for COMPANY.

⇒ but what happens to objects that are both a PERSON and a COMPANY??
⇒ We need to resolve the preferred identification:

  PERSON → ssn=??; COMPANY → (name=??, city=??).
Referring to Abstract Entities

Example (How to refer to LEGAL-ENTITY)

- invent a new attribute for this purpose (will be inherited by subclasses)
- use (a combination of) the identities of generalized entities, e.g.,
  ssn for PERSON and (name, city) for COMPANY.
  ⇒ but what happens to objects that are both a PERSON and a COMPANY??
  ⇒ we need to resolve the preferred identification:
  PERSON → ssn=?; COMPANY → (name=?, city=?).

Goal(s)

1. Flexible assignment of Referring Expression Types to classes,
2. Automatic check(s) for sanity of such an assignment, and
3. Compilation of queries (updates) over ARM to ones over concrete tables.
Referring Type Assignment (RTA)

**IDEA**

Assign a referring expression type $\text{RTA}(T)$ to each table $T$ in $\Sigma$. 
Referring Type Assignment (RTA)

**IDEA**

Assign a referring expression type RTA($T$) to each table $T$ in $\Sigma$.

**Example**

Is every RTA(.) assignment “good”? Consider the SQLP query

```sql
select x.self from PERSON x, COMPANY y where x.self = y.self
```

1 assignment:  
RTA(PERSON) = (ssn = ?),
RTA(COMPANY) = (name = ?, city = ?)

$\Rightarrow$ the ability to compare the OID values is lost $\Rightarrow$ BAD RTA!
Referring Type Assignment (RTA)

**IDEA**

Assign a referring expression type $\text{RTA}(T)$ to each table $T$ in $\Sigma$.

**Example**

Is every $\text{RTA}(.)$ assignment “good”? Consider the SQLP query

$$\text{select } x.\text{self } \text{from } \text{PERSON } x, \text{COMPANY } y \text{ where } x.\text{self} = y.\text{self}$$

1. assignment: $\text{RTA(PERSON)} = (\text{ssn} = ?),$
   $\text{RTA(COMPANY)} = (\text{name} = ?, \text{city} = ?)$

   $\Rightarrow$ the ability to compare the OID values is lost $\Rightarrow$ BAD RTA!

2. (modified) assignment:
   $\text{RTA(COMPANY)} = (\text{PERSON } \rightarrow \text{ssn} = ?); (\text{name} = ?, \text{city} = ?)$

   $\Rightarrow$ the ability to compare the OID values is preserved as COMPANY objects are *identified by* ssn values when also residing in PERSON.
Referring Type Assignment (RTA)

**IDEA**

Assign a referring expression type \( \text{RTA}(T) \) to each table \( T \) in \( \Sigma \).

**Definition (Identity-resolving RTA(.))**

Let \( \Sigma \) be a ARM schema and RTA a referring type assignment for \( \Sigma \). Given a linear order \( \mathcal{O} = (T_{i_1}, \ldots, T_{i_n}) \) on the set \( \text{Tables}(\Sigma) \), define \( \mathcal{O}(\text{RTA}) \) as the referring expression type \( \text{RTA}(T_{i_1}); \ldots; \text{RTA}(T_{i_k}) \).

We say that RTA is *identity resolving* if there is some linear order \( \mathcal{O} \) such that the following conditions hold for each \( T \in \text{Tables}(\Sigma) \):

1. \( \text{RTA}(T) = \text{Prune}(\mathcal{O}(\text{RTA}), T) \),
2. \( \Sigma \models (\text{covered by } \{T_1, \ldots, T_n\}) \in T \), and
3. for each component \( T_j \rightarrow (Pf_{j,1} = ?, \ldots, Pf_{j,k_j} = ?) \) of \( \text{RTA}(T) \), the following also holds:
   - (i) \( Pf_{j,i} \) is well defined for \( T_j \), for \( 1 \leq i \leq k_j \), and
   - (ii) \( \Sigma \models (\text{pathfd } Pf_{j,1}, \ldots, Pf_{j,k_j} \rightarrow \text{id}) \in T_j \).
Referring Type Assignment (RTA)

**IDEA**

Assign a referring expression type $\text{RTA}(T)$ to each table $T$ in $\Sigma$.

**Definition (Identity-resolving RTA(.))**

The definition achieves the following:

1. Referring expression types assigned to classes (tables) that can share objects must guarantee that a particular object is *uniquely* identified;
2. Referring expression types for disjoint classes/tables can be assigned *independently*;

**Consequences:**

- Referring expressions serve as a sound&complete proxy for entity/object (OID) equality;
- Referring expression can be *coerced* to a least common supertype.
Course Enrollment as an ARM Schema

### COURSE
- `department.deptcode = ?`
- `cnum = ?`
  - `self*`
  - `cnum`
  - `cname`
  - `department*`

### SCHEDULE
- `class.course.deptcode = ?`
- `class.cnum = ?`
- `class.term = ?`
- `class.section = ?`
- `day = ?`
- `time = ?`
  - `self*`
  - `class*`
  - `day`
  - `time`
  - `room`

### MARK
- `enrollment.student.snum = ?`
- `enrollment.class.course.department.deptcode = ?`
- `enrollment.class.cnum = ?`
- `enrollment.class.term = ?`
- `enrollment.class.section = ?`
  - `self*`
  - `enrollment*`
  - `grade`

### ENROLLMENT
- `student.snum = ?`
- `class.course.deptcode = ?`
- `class.cnum = ?`
- `class.term = ?`
- `class.section = ?`
  - `self*`
  - `student*`
  - `class*`

### DEPARTMENT
- `deptcode = ?`
  - `self*`
  - `deptcode`
  - `deptname`

### PROFESSOR
- `pnum = ?`
  - `self*`
  - `pnum`
  - `pname`
  - `office`
  - `department*`
Concrete Relational Back-end

1. Every *abstract attribute* and its referring expression type
   \[\Rightarrow\] a *concrete relational representation* (denoted by \(\text{Rep}(.)\)):
   essentially a discriminated variant record;

2. (distinct) Representations can be *coerced* to a common supertype
   \[\Rightarrow\] the ability to *compare the representations*
   a sound and complete proxy for comparing *object ids*;

3. A SQLP query is then compiled to a standard SQL query over the
   concrete representation of an abstract instance in such a way that:
Concrete Relational Back-end

1. Every *abstract attribute* and its referring expression type
   ⇒ a *concrete relational representation* (denoted by Rep(.)): essentially a discriminated variant record;

2. (distinct) Representations can be *coerced* to a common supertype
   ⇒ the ability to *compare the representations*
      a sound and complete proxy for comparing *object ids*;

3. A SQLP query is then compiled to a standard SQL query over the concrete representation of an abstract instance in such a way that:

---

**Theorem**

Let $\Sigma$ be a ARM schema and let RTA an identity resolving type assignment for $\Sigma$. For any SQLP query $Q$ over $\Sigma$

$$\text{Rep}(Q(I), \Sigma) = (C^{\Sigma, \text{RTA}}(Q))(\text{Rep}(I, \Sigma))$$

for every database instance $I$ of $\Sigma$. □
Obtaining an Initial ARM Schema (legacy setting)

RM2ARM Algorithm (highlights; see [EKAW18])

For every table in RM:

1. add “self OID” (as a new primary key)

2. replace foreign keys with unary ones and discard original FK attributes
   ⇒ what if original FK overlaps with primary key attributes?
   ⇒ how about cycles between (overlapping) PKs and FKs?

3. add ISA constraints (and remove corresponding FKs)
   ⇒ from PK to PK foreign keys in RM

4. add disjointness constraints
   ⇒ for tables with different PKs
Obtaining an Initial ARM Schema (legacy setting)

RM2ARM Algorithm (highlights; see [EKAW18])

For every table in RM:

1. add "self OID" (as a new primary key)

2. replace foreign keys with unary ones and discard original FK attributes
   ⇒ what if original FK overlaps with primary key attributes?
   ⇒ how about cycles between (overlapping) PKs and FKs?

3. add ISA constraints (and remove corresponding FKs)
   ⇒ from PK to PK foreign keys in RM

4. add disjointness constraints
   ⇒ for tables with different PKs

5. generate referring expressions (so the ARM2RM mapping works)
Summary

Contributions

Referring expressions allow one to get more/better (certain) answers . . .

1. General approach to OBDA-style query answering;
2. Methodology that allows decoupling identification from modeling;
3. Referring Expressions that subsequently resolve identity issues; and
4. Compilation-based technology translation to pure relational model.
Future work & Extensions

1. Strong Identification (distinct referring expr’s refer to distinct objects);
2. More complex referring expression types;
3. Replacing types by other preferred way to chose among referring expressions (e.g., length/formula complexity/... measure);
4. Alternatives to concrete representations;
5. More general/axiomatic definition of identity resolving RTA(.s);
Message from our Sponsors

Data Systems Group at the University of Waterloo

- ~10 professors, affiliated faculty, postdocs, ~45 grads, ...
- Wide range of research interests
  - Advanced query processing/Knowledge representation
  - System aspects of database systems and Distributed data management
  - Data quality-Managing uncertain data/Data mining
  - Information Retrieval and “big data”
  - New(-ish) domains (text, streaming, graph data/RDF, OLAP)
- Research sponsored by governments, and local/global companies
  - NSERC/CFI/OIT and Google, IBM, SAP, OpenText, ...
- Part of a School of CS with 85+ professors, 350+ grad students, etc.
  - AI&ML, Algorithms&Data Structures, PL, Theory, Systems, ...

Cheriton School of Computer Science has been ranked #18 in CS by the world by US News and World Report (#1 in Canada).

... and we are always looking for good graduate students (MMath/PhD)