# Referring Expressions in Articial Intelligence and Knowledge Representation Systems

David Toman[‡]

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

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

University of Waterloo

[‡]Cheriton School of Computer Science
University of Waterloo, Canada
{david,gweddell}@uwaterloo.ca

# Identifying and Communicating References

## (to objects/entities)

# (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?

# (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?

Typical solutions:

1. OIDs (proxying entity *identity* by a number uniformly in the whole system)
   $\Rightarrow$ typically managed by *The System* (OO languages), or
2. Keys (proxying entity *identity* by a unique combination of values (local))
   $\Rightarrow$ typically declared/managed by user (Relational DBMS).

# Object IDs: the Horror Stories

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

# Object IDs: the Horror Stories

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

## Performance: The PROTEL2 Case

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

$\Rightarrow$ storage/performance overhead (need to be generated/managed)

can we proxy by (storage) *address*?

what about memory/storage reuse and/or garbage collection??

what about data replication??

# Object IDs: the Horror Stories

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*??

$\Rightarrow$ what does `CORBA::Object::is_equivalent(in Object)` do??

# Object IDs: the Horror Stories

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

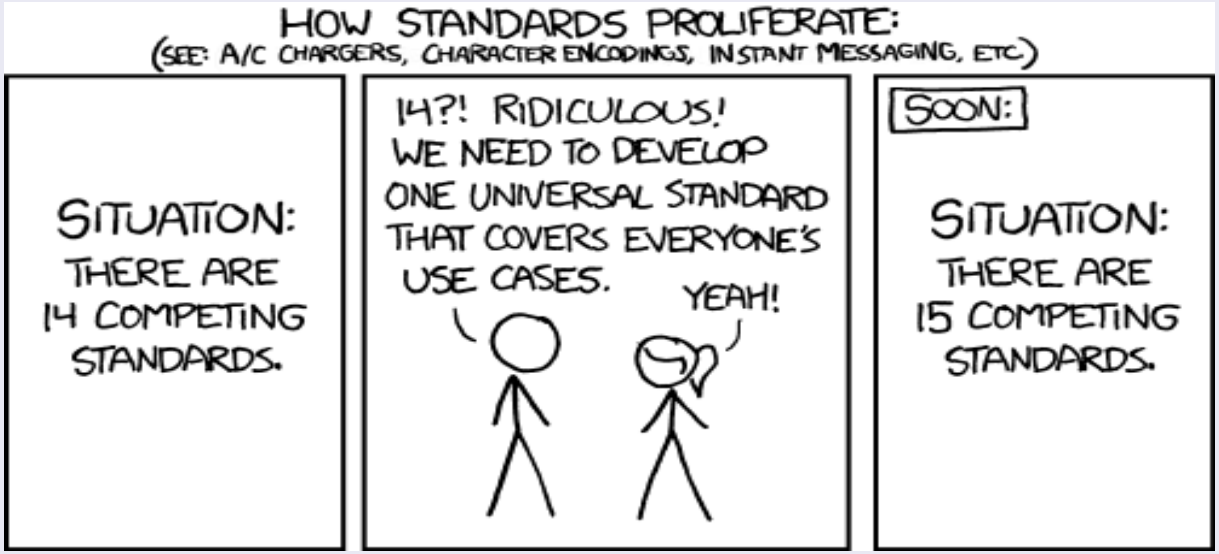
## Performance: The PROTEL2 Case

## 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:



# Object IDs: the Horror Stories

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/9202a8c04000641f8000000002f9e349`
(as of April, 2015.)

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

$\Rightarrow$ and RDF *introduces* additional internal identifiers!

# Object IDs: the Horror Stories

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)

# Object IDs: the Horror Stories

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

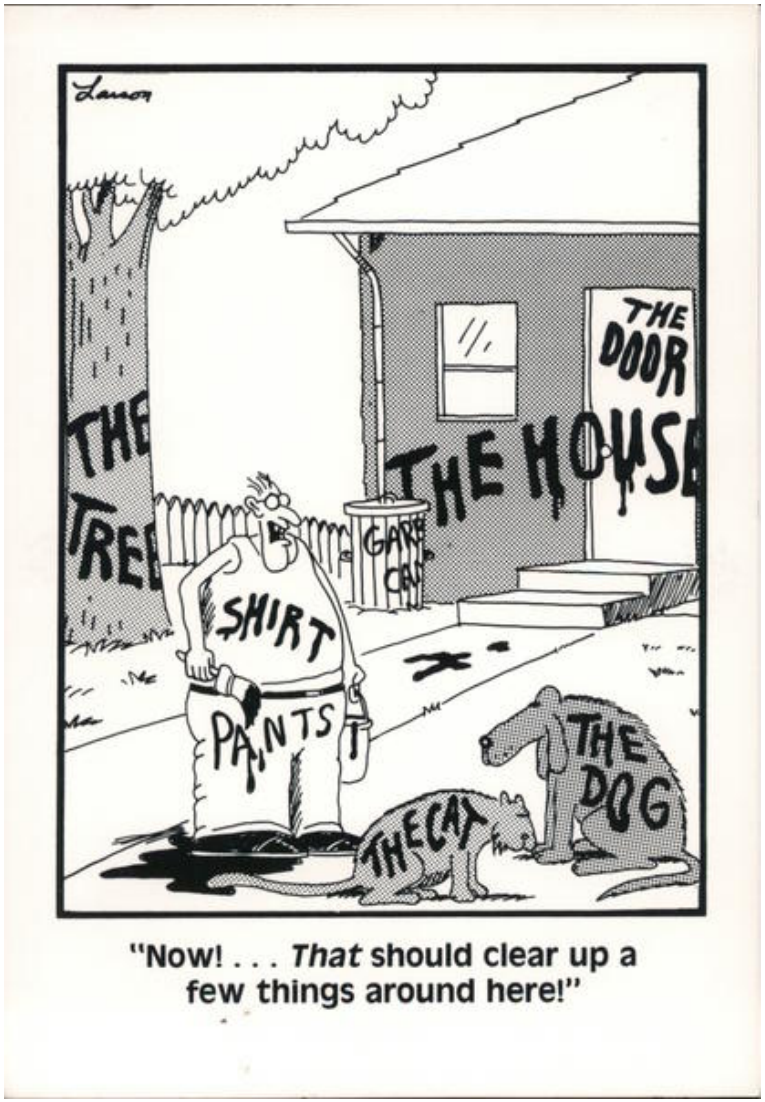
## Alternative Preferred Answers

Internal (computer) addresses vs. physical locations of equipment

$\Rightarrow$ programs need electronic address (to route the electric signals)

$\Rightarrow$ technicians need physical location (to find the equipmant)

# Relational Keys




"Now! . . . *That* should clear up a few things around here!"

# Goal of the Tutorial

**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 recording Ground Knowledge
  3. Referring Expressions in Conceptual Design
- Summary and Open Problems

# Referring Expressions

## (Background)

# What is an Referring Expression?

## Referring Expression

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

# What is an Referring Expression?

## Referring Expression

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

## Russell: "On Denoting," Mind, New Series, Vol.14, No.56, pp. 479–493, 1905.

A *definite description* "the $F$ is a $G$" is understood to have the form

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

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?

## Referring Expression

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

## Russell: "On Denoting," Mind, New Series, Vol.14, No.56, pp. 479–493, 1905.

A *definite description* "the $F$ is a $G$" is understood to have the form

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

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/9202a8c04000641f8000000002f9e349` (as of April, 2015.)

## Referring Expressions

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

$$ALBUM(x) \wedge (title(x) = \text{“Synchronicity”}) \wedge (band(x) = \text{“The Police”})$$

# 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 Expressions

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

$$ALBUM \sqcap (title = \text{“Synchronicity”}) \sqcap (band = \text{“The Police”})$$

# Bottom Line

## 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 $\mathcal{K}$.

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

# Bottom Line

## 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 $\mathcal{K}$.

$\Rightarrow$ 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*

$\Rightarrow$ must denote *something* in *every* interpretation

# Single Interpretations/Models

## Generating Referring Expressions (GRE)

Task: given an *interpretation*, find formulæ (referring expressions) that denote (selected) single objects.

Carlos Areces, Santiago Figueira, Daniel Gorín: Using Logic in the Generation of Referring Expressions. Logical Aspects of Computational Linguistics 2011.

Carlos Areces, Alexander Koller, Kristina Striegnitz: Referring Expressions as Formulas of Description Logic. International Natural Language Generation Conference 2008.

# Logical Theories and Knowledge Bases

## 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

# Logical Theories and Knowledge Bases

## 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
- similar issues with other *non-logical symbols*

# Logical Theories and Knowledge Bases

**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
- similar issues with other *non-logical symbols*

$\Rightarrow$ (standard) constants don't quite satisfy Russell's/Kripke's requirements

# Rigidity and Genericity: DB Theory Way

Why not require constants to be *rigid designators*?

$\Rightarrow$ symbols interpreted identically in all models

# Rigidity and Genericity: DB Theory Way

Why not require constants to be *rigid designators*?

$\Rightarrow$ symbols interpreted identically in all models

## Database (theory) Approach

- Database Instances (aka models) *expect constants* to be rigid

  $\Rightarrow$ but constraints/queries do not *know*
- Database Queries are required to be *generic*

  $\Rightarrow$ invariant under permutations of the underlying domain

# Rigidity and Genericity: DB Theory Way

Why not require constants to be *rigid designators*?

$\Rightarrow$ symbols interpreted identically in all models

## Database (theory) Approach

- Database Instances (aka models) *expect constants* to be rigid

  $\Rightarrow$ but constraints/queries do not *know*
- Database Queries are required to be *generic*

  $\Rightarrow$ 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: $\bigcap_{I \models K}\{a \mid \mathcal{I}, [x \mapsto a] \models \varphi\}$

   (conflates constants with domain elements)

# Rigidity and Genericity: DB Theory Way

Why not require constants to be *rigid designators*?

$\Rightarrow$ symbols interpreted identically in all models

## Database (theory) Approach

- Database Instances (aka models) *expect constants* to be rigid

  $\Rightarrow$ but constraints/queries do not *know*
- Database Queries are required to be *generic*

  $\Rightarrow$ 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: $\bigcap_{I \models K}\{a \mid \mathcal{I}, [x \mapsto a] \models \varphi\}$

   (conflates constants with domain elements)

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

# Referring to Objects (fine print)

## The rest of the presentation is based on


KR16 Alexander Borgida, David Toman, and Grant E. Weddell: On Referring Expressions in Query Answering over First Order Knowledge Bases. Proc. *International Conference on Principles of Knowledge Representation and Reasoning* KR 2016, 319-328, 2016.

ER16 Alexander Borgida, David Toman, and Grant Weddell: On Referring Expressions in Information Systems Derived from Conceptual Modelling. Proc, *International Conference on Conceptual Modeling* ER 2016, 183-197, 2016.

AI16 David Toman, and Grant Weddell: Ontology Based Data Access with Referring Expressions for Logics with the Tree Model Property. Proc. *Australasian Joint Conference on Artificial Intelligence*, 2016.

EKAW18 Weicong Ma, C. Maria Keet, Wayne Oldford, David Toman, and Grant Weddell: The Utility of the Abstract Relational Model and Attribute Paths in SQL. Proc. *International Conference on Knowledge Engineering and Knowledge Management*, 195-211, EKAW 2018.

DL18 David Toman and Grant E. Weddell: Identity Resolution in Conjunctive Querying over DL-based Knowledge Bases. Proc. *Description Logics* DL 2018, 2018 (to appear in PRICAI 2019).

DL19 David Toman, Grant E. Weddell: Exhaustive Query Answering via Referring Expressions. Proc. *Description Logics* DL 2019, 2019.

DL22 Alexander Borgida, Enrico Franconi, David Toman, and Grant E. Weddell: Accessing Document Data Sources using Referring Expression Types. Proc. *Description Logics* DL 2022, 2022 (next week).

# 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 (explicit data)
- Every BOSS is an EMPloyee (ontology)

*List all EMPloyees* $\Rightarrow$ {Bob} (query)

## Goal: compute all *certain answers*

$\Rightarrow$ answers *common* in all models of KB (aka. answers *logically implied* by KB)

# Approaches to Ontology-based Data Access

## Main Task

INPUT: $\underbrace{\text{Ontology } (\mathcal{T}), \text{ Data } (\mathcal{A})}_{\text{Knowledge Base}(\mathcal{K})}$, and a Query ($Q$)

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 ($\mathcal{CFDI}$ or Horn-$\mathcal{ALC}*$ 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

# 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.

# 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"

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

# From Query Answers to Referring Expressions [KR16]

## (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 $\mathcal{K} \models \psi\theta$.

# From Query Answers to Referring Expressions [KR16]

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

. . . are logically implied by $\mathcal{K}$.

# Controlling the number of Answers

## Example KB

$$\mathcal{T} = \{\ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \wedge \text{Person}(y)),$$
$$\text{Father}(x) \rightarrow \text{Person}(x),$$
$$\text{Father}(x) \rightarrow \exists y.\text{fatherof}(x, y),$$
$$\text{Person}(x) \rightarrow \exists y.\text{fatherof}(y, x)$$
$$\}$$

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

# Controlling the number of Answers

## Example KB

$$\mathcal{T} = \{\ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \wedge \text{Person}(y)),$$
$$\text{Father}(x) \rightarrow \text{Person}(x),$$
$$\text{Father}(x) \rightarrow \exists y.\text{fatherof}(x, y),$$
$$\text{Person}(x) \rightarrow \exists y.\text{fatherof}(y, x)$$
$$\}$$

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

## Query: $\text{Father}(x)$?

Answers: $x = \text{fred}$

# Controlling the number of Answers

## Example KB

$$\mathcal{T} = \{\ \text{fatherof}(x, y) \rightarrow (\text{Father}(x) \wedge \text{Person}(y)),$$
$$\text{Father}(x) \rightarrow \text{Person}(x),$$
$$\text{Father}(x) \rightarrow \exists y.\text{fatherof}(x, y),$$
$$\text{Person}(x) \rightarrow \exists y.\text{fatherof}(y, x)$$
$$\}$$

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

## Query: $\text{Father}(x)$?

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

# Controlling the number of Answers

## Example KB

$$\begin{aligned}\mathcal{T} = \{\ & \text{fatherof}(x,y) \rightarrow (\text{Father}(x) \wedge \text{Person}(y)),\\ & \text{Father}(x) \rightarrow \text{Person}(x),\\ & \text{Father}(x) \rightarrow \exists y.\text{fatherof}(x,y),\\ & \text{Person}(x) \rightarrow \exists y.\text{fatherof}(y,x)\\ & \text{fatherof}(x,z) \wedge \text{fatherof}(y,z) \rightarrow x = y \quad \}\end{aligned}$$

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

## Query: $\text{Father}(x)$?

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

# Controlling the number of Answers

## Example KB

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

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

## Query: $\text{Father}(x)$?

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

## Query: $\text{Person}(x)$?

Answers: $x = \text{mary}$, $x = \text{fred}$, $\text{fatherof}(\text{fred}, x)$ (NO!)
$\text{fatherof}(x, \text{mary})$, $\text{fatherof}(x, \text{fred})$, ...

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

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

Answers: $x = \text{fred}$

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

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

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

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

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

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

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

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

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

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

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

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

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

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

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

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

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

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

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

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

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

$\text{fred} = \text{spouse}(x, \text{mary}) = \exists y.\text{spouse}(x, y) \wedge \text{spouse}(y, \text{fred}) = \ldots$
$\text{mary} = \text{spouse}(x, \text{fred}) = \exists y.\text{spouse}(x, y) \wedge \text{spouse}(y, \text{mary}) = \ldots$
$\text{mary} \neq \text{fred}$ (last constraint!)

# Controlling the number of Answers II

## Example KB

$$\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$$
$$\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y$$
$$\text{spouse}(x, y) \rightarrow x \neq y \qquad \}$$
$$\mathcal{A} = \{\ \text{spouse}(\text{mary}, \text{fred})\ \}$$

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

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

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

$\text{fred} = \text{spouse}(x, \text{mary}) = \exists y.\text{spouse}(x, y) \wedge \text{spouse}(y, \text{fred}) = \ldots$
$\text{mary} = \text{spouse}(x, \text{fred}) = \exists y.\text{spouse}(x, y) \wedge \text{spouse}(y, \text{mary}) = \ldots$
$\text{mary} \neq \text{fred}$ (last constraint!) $\Rightarrow$ exactly 2 distinct objects

# Controlling the 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

# Controlling the 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 $\mathcal{K}$.

# Controlling the 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 $\mathcal{K}$.

$\Rightarrow$ but is there a *finite representation*?

# Example: Horn Logics with Tree Models [DL19]

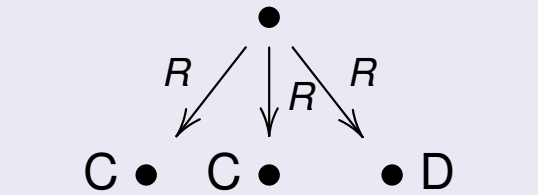
## What to do $\mathcal{EL}^{\bot}$ (and Horn-$\mathcal{ALC}$)?

- *singularity* requires *role functionality* (not expressible in $\mathcal{EL}^{\bot}$/Horn-$\mathcal{ALC}$)

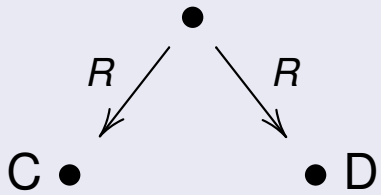
# Example: Horn Logics with Tree Models [DL19]

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

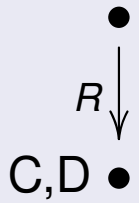
- *singularity* requires *role functionality* (not expressible in $\mathcal{EL}^{\bot}$/Horn-$\mathcal{ALC}$)
- (Tree) Models of $a : \exists R.C \sqcap \exists R.D$:




spurious $R$-successor $C$




just right (canonical)




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

$\Rightarrow$ only non-redundant successors in matching tuples

$\Rightarrow$ preserves complexity bounds for both logics

# How Does it Work?

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

Looping automaton-like construction

$\Rightarrow$ only non-redundant successors in matching tuples

$\Rightarrow$ preserves complexity bounds for both logics

## Generalizations&Limitations

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

# Controlling the 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: $Rt ::= Pd = \{?\} \mid Rt_1 \wedge Rt_2 \mid T \rightarrow Rt \mid Rt_1; Rt_2$

$\Rightarrow$ each type induces a set of unary formulæ;

Queries: **select** $x_1 : Rt_1, \ldots, x_k : Rt_k$ **where** $\psi$

$\Rightarrow$ $x_1 : Rt_1, \ldots, x_k : Rt_k$ is called the head, $\psi$ is the body.

# Referring Expression Types

## Desiderata: only Referring Expressions that are Singular

Given
1. a KB $\mathcal{K}$ (the "background knowledge"),
2. a query $\psi\{x_1, \ldots, x_k\}$, and
3. types $Rt_1, \ldots, Rt_k$ for 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*.

# Referring Expression Types

## Desiderata: only Referring Expressions that are Singular

Given
1. a KB $\mathcal{K}$ (the "background knowledge"),
2. a query $\psi\{x_1, \ldots, x_k\}$, and
3. types $Rt_1, \ldots, Rt_k$ for 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* $\mathcal{K}$.

# Examples of Typed Queries

## Reference via a Single-Attribute Key

*"The ssn# of any person with phone 1234567"*

**select** $x : ssn\# = \{?\}$
**where** $Person(x) \wedge phone\#(x, 1234567)$

# Examples of Typed Queries

## Reference via a Single-Attribute Key

## Reference by a Multi-Attribute Key

*"The title and publisher of any journals"*

**select** $x : \mathit{title} = \{?\} \wedge \mathit{publishedBy} = \{?\}$
**where** $\mathit{Journal}(x)$

# 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"*

**select** $x$ : $Person \rightarrow ssn\# = \{?\}$ ;
$Company \rightarrow tickerSymbol = \{?\}$
**where** $LegalEntity(x)$

answers: $\{x \mapsto Person(x) \wedge ssn\#(x, 7654)\}$
$\{x \mapsto Company(x) \wedge tickerSymbol(x, \text{“IBM”})\}$.

# 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."*

**select** $x$ : *Journal* $\rightarrow$ (*title* $= \{?\} \wedge$ *publisher* $= \{?\}$);
*EditedCollection* $\rightarrow$ *isbn#* $= \{?\}$ ; $\{?\}$
**where** *Publication*($x$)

answers: $\{x \mapsto$ *Journal*($x$) $\wedge$ *title*($x$, "AIJ") $\wedge$ *publisher*($x$, "Elsevier")$\}$
$\{x \mapsto$ *EditedCollection*($x$) $\wedge$ *isbn#*($x$, 123456789)$\}$
$\{x \mapsto x =$ `/guid/9202a8c04000641f8000000...`$\}$.

# 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
$$T_i \rightarrow Pd_{i,1} = \{?\} \wedge \ldots \wedge Pd_{i,k_i} = \{?\};$$
2. Create queries $\psi_i\{x, y_1, \ldots, y_{k_i}\}$ as
$$\psi \wedge T_i(x) \wedge Pd_{i,1}(x, y_1) \wedge \ldots \wedge Pd_{i,k_i}(x, y_{k_i});$$
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

## Lite Description Logics

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

- 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

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

The witnesses for anonymous objects (step (3))

$\longrightarrow$ *last* named individual on a path *towards* the anonymous object

# Recording/Representing Factual Data

# Referring Expressions for Ground Knowledge

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

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

# 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:

$$\text{PERSON} \sqcap \exists ssn.\{123\} \quad \text{and} \quad \text{PERSON} \sqcap \exists ssn.\{456\}.$$

- Role (feature) assertions of the form $mother(o_1) = o_2$ can then be captured as:

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

# 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:

$$\text{PERSON} \sqcap \exists ssn.\{123\} \quad \text{and} \quad \text{PERSON} \sqcap \exists ssn.\{456\}.$$

- Role (feature) assertions of the form $mother(o_1) = o_2$ can then be captured as:

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

Issues:

- admissibility: what descriptions qualify here? $\Rightarrow$ singularity!
- minimality: is the description succinct? (similar to keys/superkeys issues)

# 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 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 $\mathcal{T} = \{$ $\text{FRIEND} \sqsubseteq \text{PERSON}$,
  $\text{FRIEND} \sqsubseteq \text{PERSON} : \mathit{fname} \rightarrow \mathit{id}$,
  $\text{MATRIARCH} \sqsubseteq \text{PERSON}$,
  $\text{MATRIARCH} \sqsubseteq \text{PERSON} : \mathit{lname} \rightarrow \mathit{id}$,
  $\text{PERSON} \sqsubseteq \text{PERSON} : \mathit{fname}, \mathit{lname} \rightarrow \mathit{id}, \ldots \}$
- CBox $\mathcal{C} = \{$ $\text{FRIEND} \sqcap \exists \mathit{fname}.\{\text{“Mary”}\}$,
  $\text{PERSON} \sqcap (\exists \mathit{fname}.\{\text{“Mary”}\}) \sqcap (\exists \mathit{lname}.\{\text{“Smith”}\})$,
  $\text{MATRIARCH} \sqcap \exists \mathit{lname}.\{\text{“Smith”}\}, \ldots \}$

## Heterogeneous Identification

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

. . . and thus is an answer to $\{x \mid \text{MATRIARCH}(x)\}$.

# Minimality

## 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 $C$.

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

# Minimality

## 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 $C$.

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

$\Rightarrow$ computes a syntactically-minimal co-referring expression for $C$.
$\Rightarrow$ order of enumeration $\rightarrow$ variant minimal co-referring expressions.

# Reasoning and QA with CBoxes [DL18]

## Theorem (CBox Admissibility)

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

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

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

## Theorem (Satisfiability of KBs with CBoxes)

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

## Theorem (Query Answering)

*Let $\mathcal{K} = (\mathcal{T}, \mathcal{C})$ 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 $(\mathcal{T}, \mathsf{Simp}(\mathcal{C}))$.*

# Documents and Ontologies

## Ontologies for Documents: Goals

1. to capture *class mambership* of entities captured in a document, and
2. to establish *how entities are identified* in a document.

# Documents and Ontologies

## Ontologies for Documents: Goals

1. to capture *class mambership* of entities captured in a document, and
2. to establish *how entities are identified* in a document.

## IDEA: Documents as Concepts, Semantics as Ontology

1. Syntactical document structure captured as a concept in FunDL

   $\Rightarrow$ similar to the IBM IMS hierarchical data model
2. Ontology adds meaning to this concept and its subconcepts
   - identifies class membership of entities described by subdocuments,
   - discovers subdocuments pertaining to the same entity, and
   - drives document normalization.

# Example: JSON Document

```
{ "collection": "person",
  "data" : [
    { "fname": "John", "lname": "Smith", "age": 25,
      "wife": { "fname" : "Mary" },
      "phone": [
         {"colour": "red", "dnum": "212 555-1234"}
      ]
    },
    { "fname": "Mary", "lname": "Jones",
      "salary": "$150,000 (CAD)",
      "spouse": { "fname": "John" },
      "phone": [
         {"loc": "home", "dnum": "212 555-1234"},
         {"loc": "work", "dnum": "212 666-4567"}
      ]
    }
  ]
}
```

# Example: JSON as a FunDL Concept

$$
\begin{aligned}
&\exists \texttt{collection}.\{\texttt{"person"}\} \sqcap \\
&\exists \texttt{data}.\exists \texttt{dom}^{-}.\exists \texttt{ran}( \\
&\quad \exists \texttt{fname}.\{\texttt{"John"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Smith"}\} \sqcap \\
&\quad \exists \texttt{age}.\{\texttt{"25"}\} \sqcap \exists \texttt{wife}.\exists \texttt{fname}.\{\texttt{"Mary"}\} \sqcap \\
&\quad \exists \texttt{phone}(\exists \texttt{dom}^{-}.\exists \texttt{ran}( \\
&\qquad \exists \texttt{colour}.\{\texttt{"red"}\} \sqcap \exists \texttt{dnum}.\{\texttt{"212 555-1234"}\}))) \sqcap \\
&\quad \exists \texttt{dom}^{-}.\exists \texttt{ran}( \\
&\quad \exists \texttt{fname}.\{\texttt{"Mary"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Jones"}\} \sqcap \\
&\quad \exists \texttt{salary}.\{\texttt{"\$150000CAD"}\} \sqcap \exists \texttt{spouse}.\exists \texttt{fname}.\{\texttt{"John"}\} \sqcap \\
&\quad \exists \texttt{phone}(\exists \texttt{dom}^{-}.\exists \texttt{ran}( \\
&\qquad \exists \texttt{loc}.\{\texttt{"home"}\} \sqcap \exists \texttt{dnum}.\{\texttt{"212 555-1234"}\}) \sqcap \\
&\qquad \exists \texttt{dom}^{-}.\exists \texttt{ran}( \\
&\qquad \exists \texttt{loc}.\{\texttt{"work"}\} \sqcap \exists \texttt{dnum}.\{\texttt{"212 666-4567"}\})))
\end{aligned}
$$

# Example: Ontology

1. The TBox:

$$
\begin{aligned}
(\exists \texttt{collection}.\top) \sqcap (\exists \texttt{data}.\top) &\sqsubseteq \mathit{DOCUMENT} \\
(\exists \texttt{fname}.\top) \sqcap (\exists \texttt{lname}.\top) &\sqsubseteq \mathit{PERSON} \\
\exists \texttt{dnum}.\top &\sqsubseteq \mathit{PHONE} \\
\\
\mathit{DOCUMENT} &\sqsubseteq \mathit{DOCUMENT} : \texttt{collection} \rightarrow \mathit{id} \\
\mathit{PERSON} &\sqsubseteq \mathit{PERSON} : \texttt{fname}, \texttt{lname} \rightarrow \mathit{id} \\
\mathit{PHONE} &\sqsubseteq \mathit{PHONE} : \texttt{dnum} \rightarrow \mathit{id} \\
\mathit{PERSON} &\sqsubseteq \exists \texttt{wife}.\mathit{PERSON}
\end{aligned}
$$

2. The Referring Expression Type Assignment:

$$
\begin{aligned}
\mathrm{RTA}(\mathit{DOCUMENT}) &= \mathit{DOCUMENT} \sqcap \exists \texttt{collection}.\{?\} \\
\mathrm{RTA}(\mathit{PERSON}) &= \mathit{PERSON} \sqcap \exists \texttt{lname}.\{?\} \sqcap \exists \texttt{fname}.\{?\} \\
\mathrm{RTA}(\mathit{PHONE}) &= \mathit{PHONE} \sqcap \exists \texttt{dnum}.\{?\}
\end{aligned}
$$

# Example: Normalized CBox/Document

$$\underline{\mathit{DOCUMENT} \sqcap \exists \texttt{collection}.\{\texttt{"person"}\} \sqcap \exists \texttt{data}(}$$
$$\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"John"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Smith"}\}}) \sqcap$$
$$\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"Mary"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Jones"}\}}))$$

$$\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"John"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Smith"}\} \sqcap}$$
$$\exists \texttt{age}.\{\texttt{"25"}\} \sqcap \exists \texttt{wife}.\exists \texttt{fname}\{\texttt{"Mary"}\} \sqcap$$
$$\exists \texttt{phone}(\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-1234"}\}}))$$

$$\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"Mary"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Jones"}\} \sqcap}$$
$$\exists \texttt{salary}.\{\texttt{"\$150000CAD"}\} \sqcap \exists \texttt{spouse}.\exists \texttt{fname}\{\texttt{"John"}\} \sqcap$$
$$\exists \texttt{phone}(\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-1234"}\}}) \sqcap$$
$$\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 666-4567"}\}}))$$

$$\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-1234"}\} \sqcap \exists \texttt{loc}.\{\texttt{"home"}\} \sqcap \exists \texttt{colour}.\{\texttt{"red"}\}}$$

$$\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-4567"}\} \sqcap \exists \texttt{loc}.\{\texttt{"work"}\}}$$

# Example: Normalized CBox/Document

$$\underline{\mathit{DOCUMENT} \sqcap \exists \texttt{collection}.\{\texttt{"person"}\} \sqcap \exists \texttt{data}(}$$
$$\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"John"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Smith"}\}) \sqcap}$$
$$\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"Mary"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Jones"}\}}))$$

$$\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"John"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Smith"}\} \sqcap}$$
$$\exists \texttt{age}.\{\texttt{"25"}\} \sqcap \exists \texttt{wife}.\exists \texttt{fname}\{\texttt{"Mary"}\} \sqcap$$
$$\exists \texttt{phone}(\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-1234"}\}}))$$

$$\underline{\mathit{PERSON} \sqcap \exists \texttt{fname}.\{\texttt{"Mary"}\} \sqcap \exists \texttt{lname}.\{\texttt{"Jones"}\} \sqcap}$$
$$\exists \texttt{salary}.\{\texttt{"\$150000CAD"}\} \sqcap \exists \texttt{spouse}.\exists \texttt{fname}\{\texttt{"John"}\} \sqcap$$
$$\exists \texttt{phone}(\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-1234"}\}) \sqcap}$$
$$\exists \texttt{dom}^{-}.\exists \texttt{ran}(\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 666-4567"}\}}))$$

$$\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-1234"}\} \sqcap \exists \texttt{loc}.\{\texttt{"home"}\} \sqcap \exists \texttt{colour}.\{\texttt{"red"}\}}$$

$$\underline{\mathit{PHONE} \sqcap \exists \texttt{dnum}\{\texttt{"212 555-4567"}\} \sqcap \exists \texttt{loc}.\{\texttt{"work"}\}}$$

Borgida et al. Accessing Document Data Sources using Referring Expression Types
Aug 10 at 11:25 in DL 2022, Session 112C (Taub 9)

# CONCEPTUAL MODELLING

(Decoupling *modelling* from *identification* issues)

# Conceptual Modeling and Identification [ER16]

**Thesis:**

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

# 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

**Example (`ROOM` within `BUILDING`)**

For the entity set `ROOM` with attributes `room-number` and `capacity`

$\Rightarrow$ natural attributes are insufficient to identify `ROOM`s

$\Rightarrow$ 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`

$\Rightarrow$ choice of key (`ssn`) for `PERSON` forces *the same* key for `FAMOUS-PERSON`

$\Rightarrow$ we may *prefer* to use `name` in this case (e.g., *Eric Clapton* or *The Edge*)

# 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

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

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

$\Rightarrow$ commonly *required* to create an *artificial* attribute `le-num`

$\Rightarrow$ 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 $\mathcal{C}$

Common features of so-called "attribute-based" semantic models

$\Rightarrow$ 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

$\Rightarrow$ 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)
```

# Abstract Relational Queries

## 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*:

```
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*:

```
select v.owned-by from VEHICLE v
where v.make = 'Mitsubishi'
```

  retrieving *abstract attributes* may yield
  *heterogeneous results* (`PERSON`s and `COMPAN`ies)

# 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*:

```
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*:

```
select v.owned-by from VEHICLE v
where v.make = 'Mitsubishi'
```

  retrieving *abstract attributes* may yield
  *heterogeneous results* (`PERSON`s and `COMPAN`ies)

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*
     $\Rightarrow$ and WITHOUT introducing *explicit, material* OIDs
   - Reverse-Engineering ARM from Legacy Relational Schemata

# Experimental Design (HCI experiments)

## Hypotheses

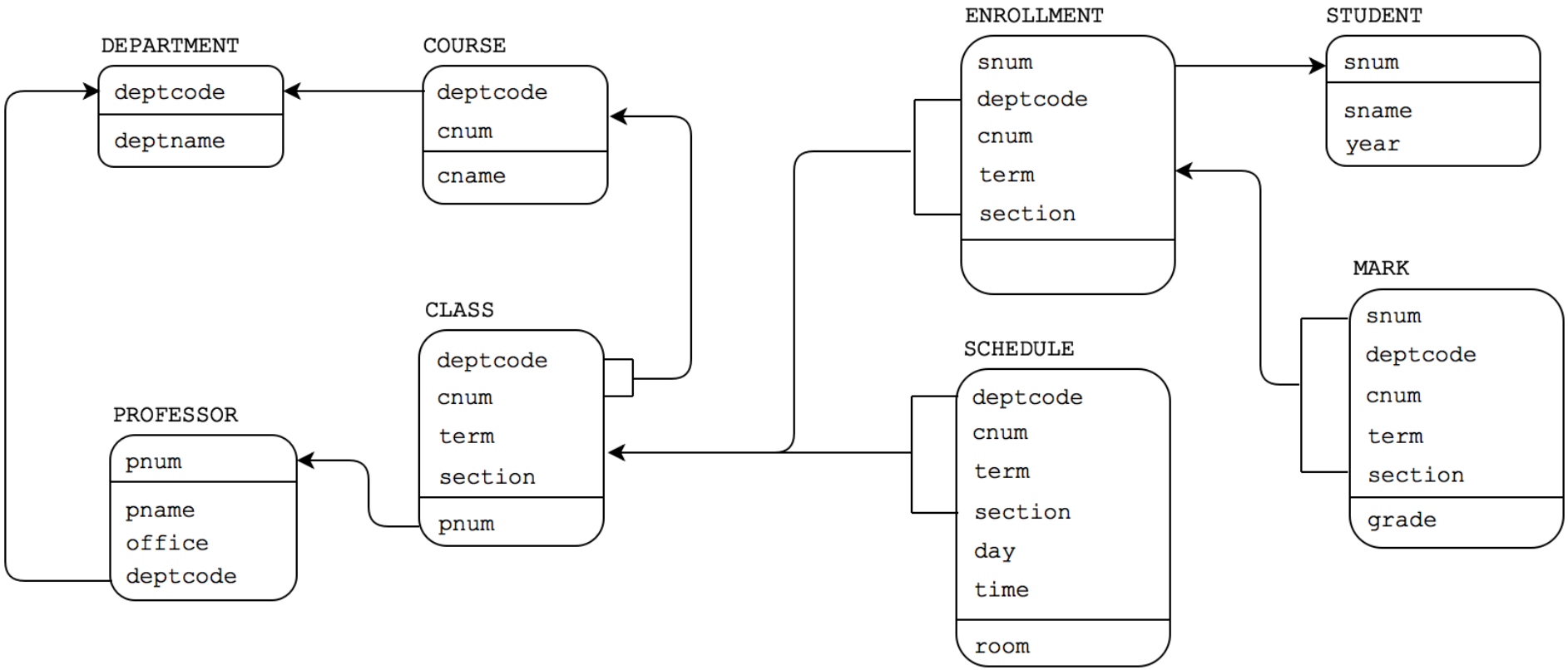
$H_t$: no difference between RM/SQL and ARM/SQLP in the mean time taken

$H_c$: no difference between RM/SQL and ARM/SQLP in the mean correctness

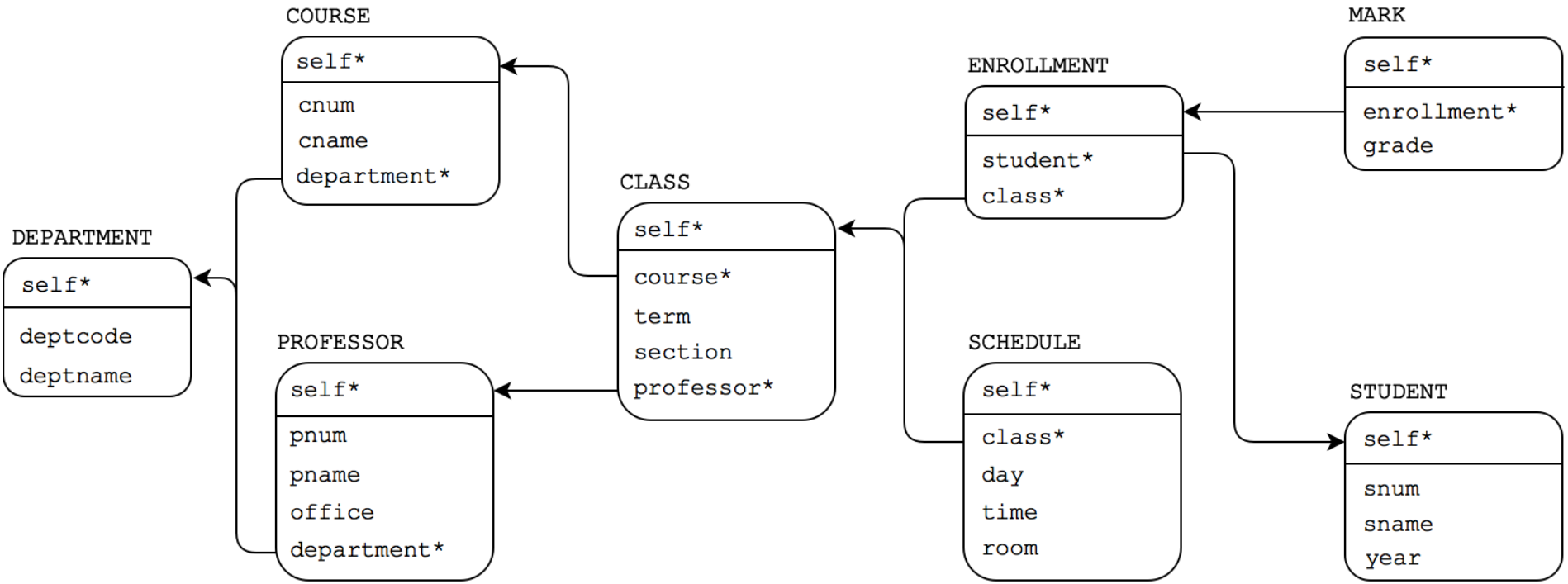
## Methods

- 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

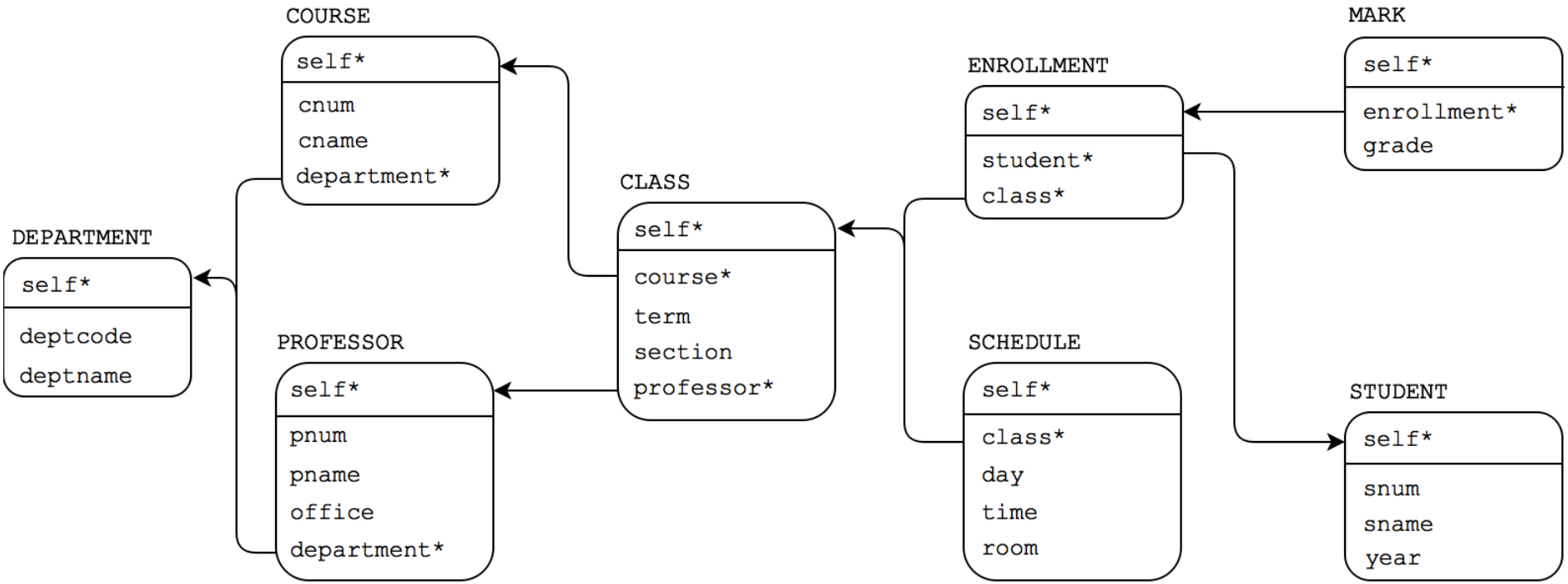
# Course Enrollment as an RM Schema



# Course Enrolment as an ARM Schema



# 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.

# Example Queries

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

RM/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:

```
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:

```
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'
```

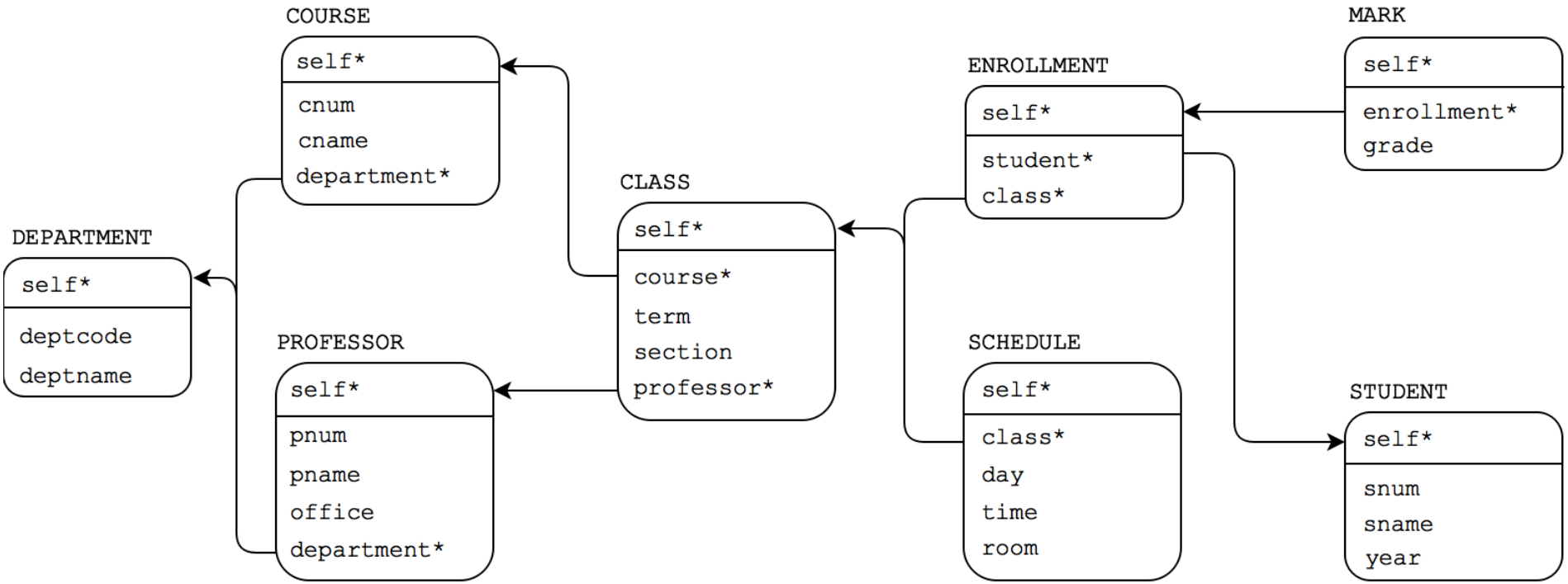
Domain expert needs to understand structure of PK/FKs: BAD!!

## ARM/SQLP:

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

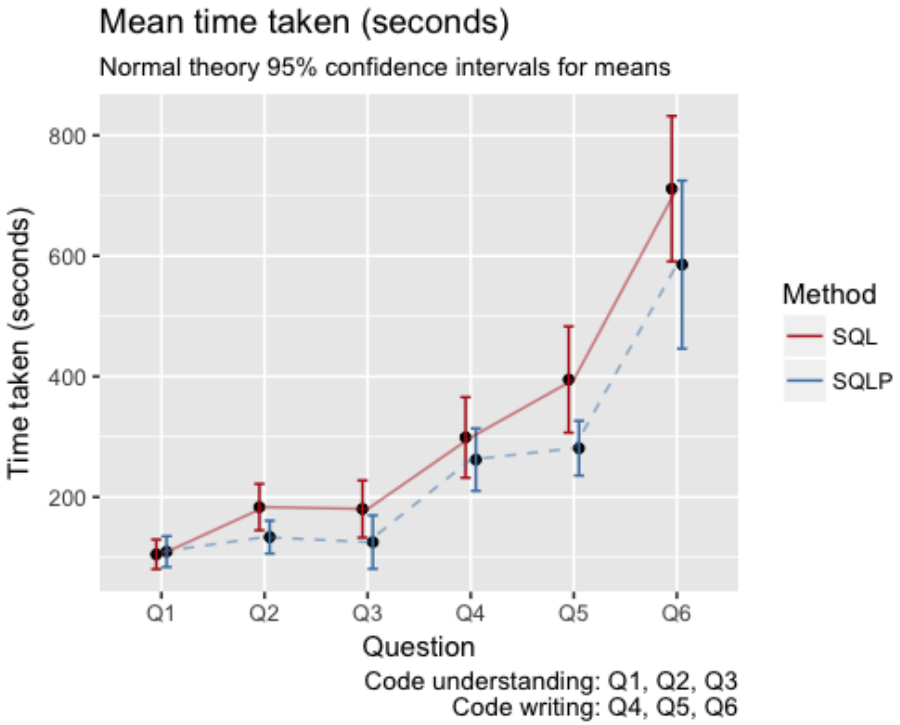
# ARM Schema and Path Navigation

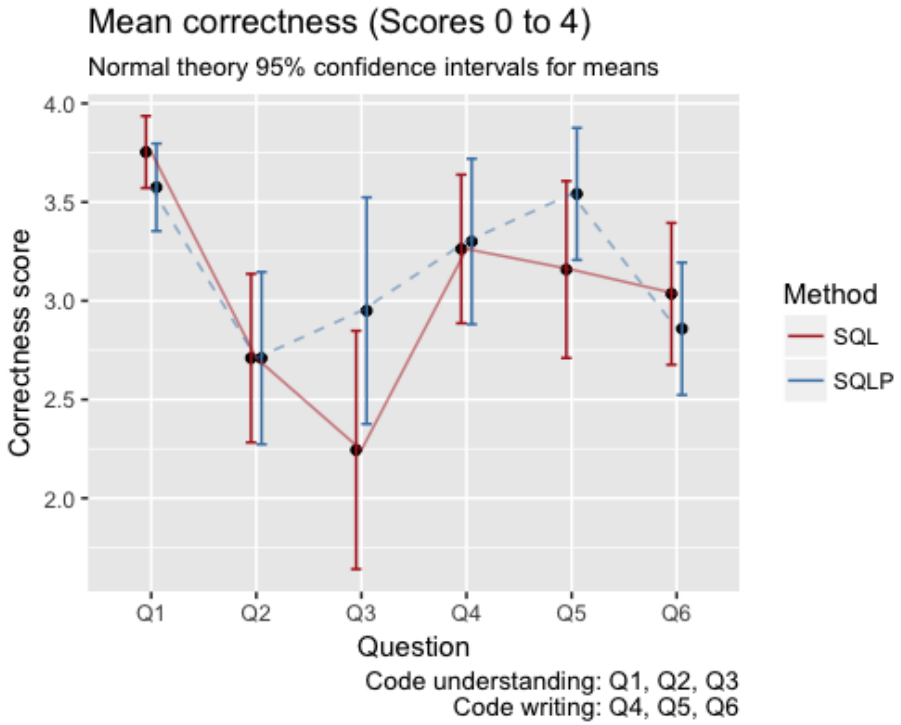
```
select distinct e.student.sname as name
from ENROLLMENT e
where e.class.professor.pname = 'Alan John'
```



# Experiments: Results

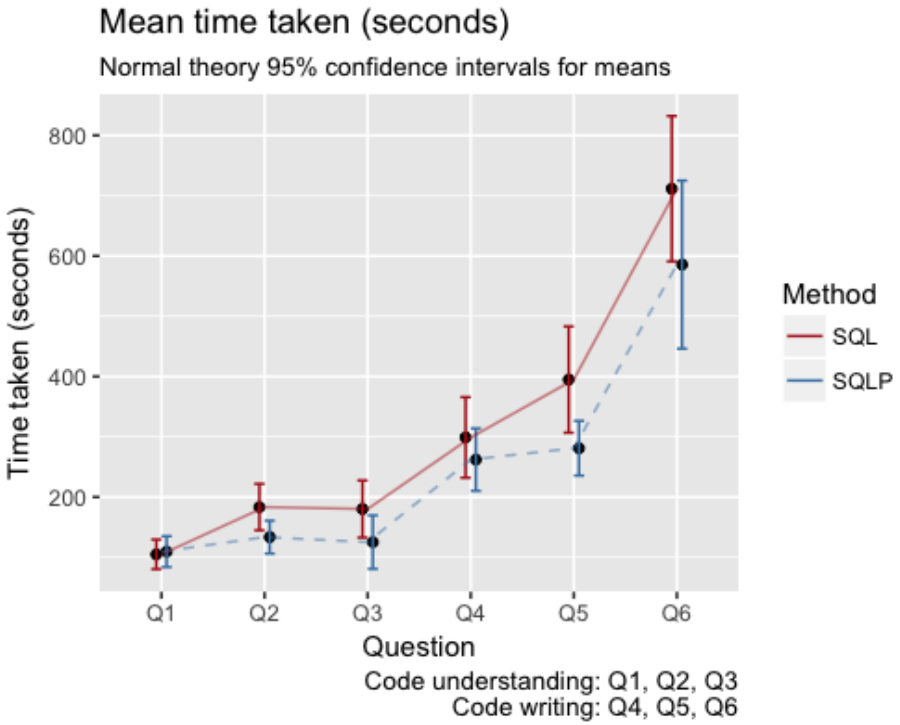
Mean performance for all subjects: SQL solid; SQLP dashed.

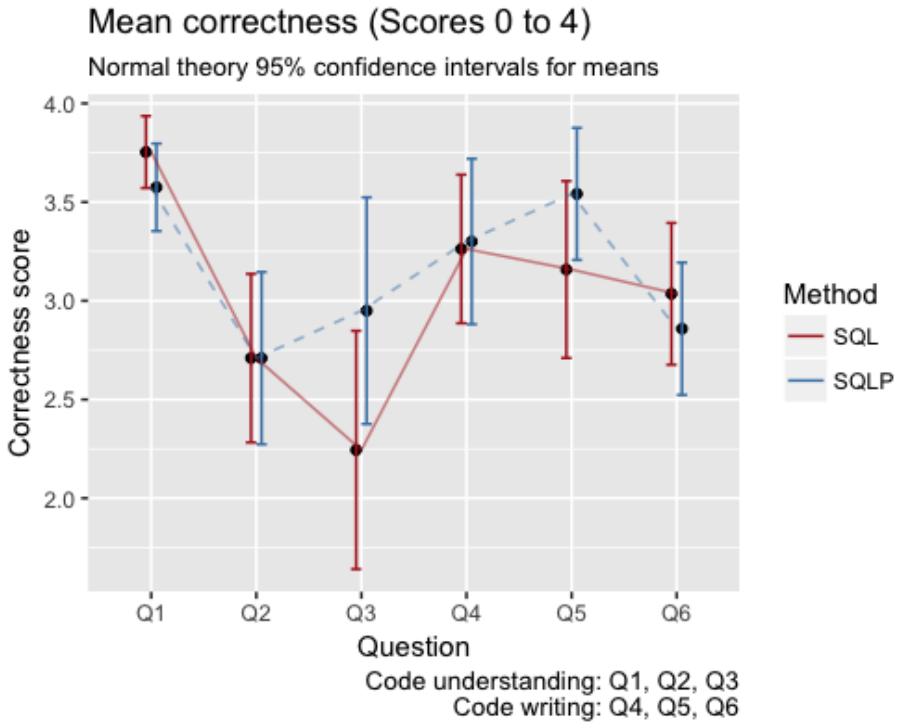


# Experiments: Results

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 LEGAL-ENTITY)

- invent a *new attribute for this purpose* (will be *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.,

  `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.,

  `ssn` for `PERSON` and `(name, city)` for `COMPANY`.

  $\Rightarrow$ 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`.

  $\Rightarrow$ but what happens to objects *that are both* a `PERSON` and a `COMPANY`??

  $\Rightarrow$ we need to resolve the *preferred* identification:

  `PERSON` $\rightarrow$ `ssn=?; COMPANY` $\rightarrow$ `(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`.

  $\Rightarrow$ but what happens to objects *that are both* a `PERSON` and a `COMPANY`??

  $\Rightarrow$ we need to resolve the *preferred* identification:

  `PERSON` $\rightarrow$ `ssn=?; COMPANY` $\rightarrow$ `(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 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

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

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

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

# 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

`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!;

2. (modified) assignment:
   RTA(`COMPANY`) $=$ (`PERSON` $\rightarrow$ `ssn` $=$ ?); (`name` $=$ ?, `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 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 Tables($\Sigma$), define $\mathcal{O}$(RTA) as the referring expression type RTA($T_{i_1}$); ...; 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 $T \in$ Tables($\Sigma$):

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

# Referring Type Assignment (RTA)

## IDEA

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

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

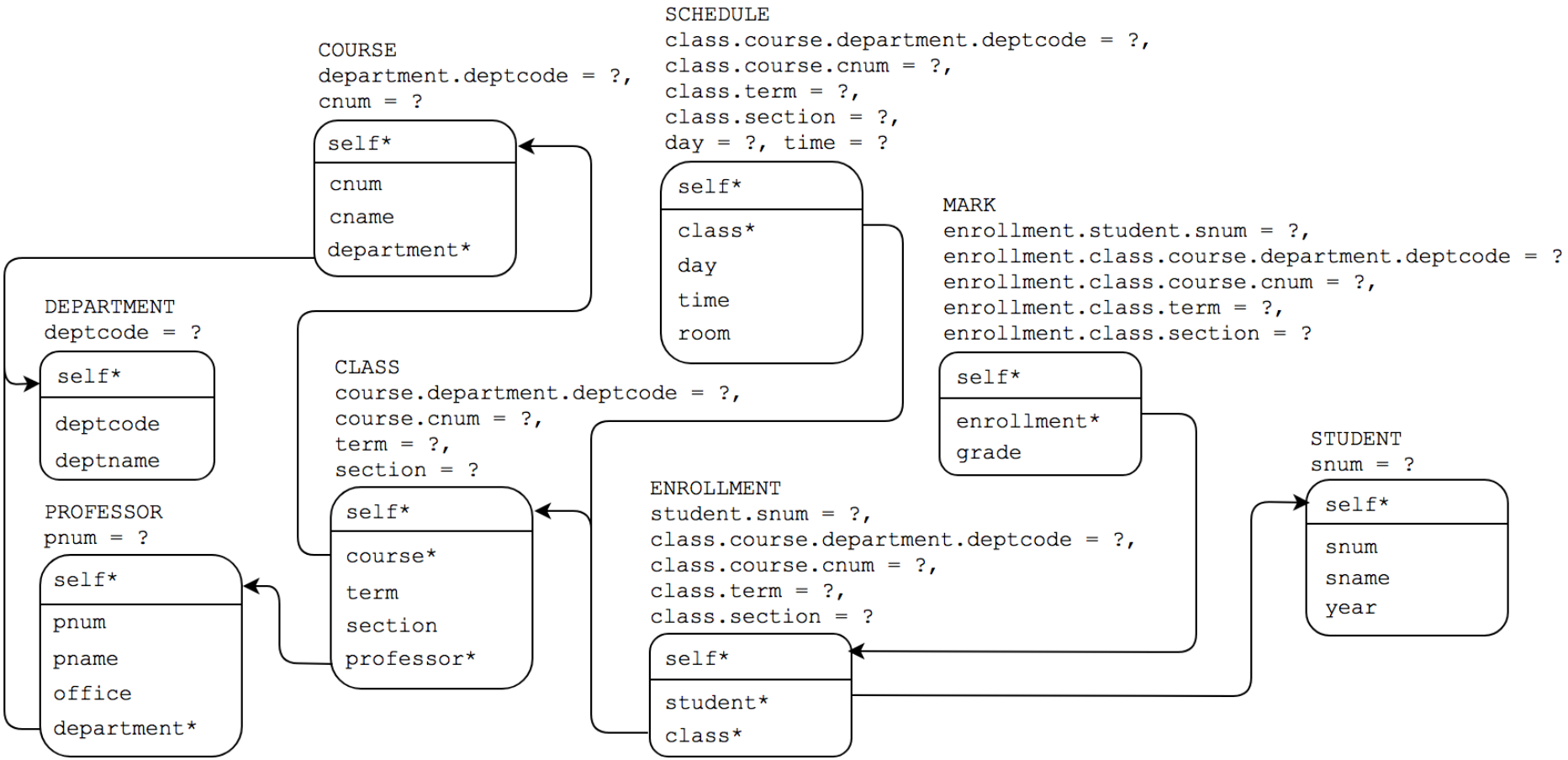
The definition achieves the following:

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



# Concrete Relational Back-end

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

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

*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

# Summary

## Contributions

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

1. General approach to OBDA-style query answering
   $\Rightarrow$ Ability to refer to implicit individuals/entities
2. General approach to representing data
   $\Rightarrow$ without need for *object id invention*;
3. Methodology that allows decoupling identification from modeling
   $\Rightarrow$ Referring Expressions resolve identity issues and
   $\Rightarrow$ Compilation 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, . . .

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