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,gweddell}@uwaterloo.ca

REFERING EXPRESSIONS

(INTRO AND 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 center of analytic philosophy for over a century now.

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 Referrences:

"The Morning Star" vs. "The Evening Star" [Fregge]

(multiple distinct references to the same object)

. . .

Tutorial Outline

REFERING 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/9202a8c04000641f800000002f9e349` (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/9202a8c04000641f800000002f9e349` (as of April, 2015.)

Referring Expressions

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

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

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

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

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

  . . . set of constants may *change* with evolution of the theory (updates!)

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

  . . . set of constants may *change* with evolution of the theory (updates!)
- similar issues with other *non-logical 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
  ... set of constants may *change* with evolution of the theory (updates!)
- similar issues with other *non-logical symbols*

$\Rightarrow$ even (standard) conatants don't quite satisfy Russell'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) *use rigid constants*, but
- 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) *use rigid constants*, but
- 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) *use rigid constants*, but
- 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*!

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 $\mathcal{I}$ model of $\mathcal{K}$.

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.

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

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.

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.

Ontology Based Data Access

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 $\Rightarrow \{\text{Bob}\}$ (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 el. (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

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 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 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 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 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 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 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 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 an unary formula that, when used as 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. PhoneNo(x, "+1(519) 888-4567x34447") holds; ✓
2. it is a phone x s.t. UWPhone(x) $\wedge$ PhoneExt(x, "x34447") holds; ✓
3. it is a phone x s.t. UWRoom(x, "DC3344") holds; ✓
4. it is a phone x s.t. UWPhone(x) $\wedge$ PhonePort(x, `0x0123abcd`) holds; ✓
5. it is a phone x s.t. UWCSPhone(x) $\wedge$ InvNo(x, "100034447") holds; ✓
6. it is a phone x s.t. IsOwner("David", x) holds; ×
7. it is the phone x s.t. Colour(x, "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 ψ 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 ψ 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 $\mathcal{K}$.

More Examples

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

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

More Examples

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

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

  query: $\text{Father}(x)$?

More Examples

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

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

  query: $\text{Father}(x)$?
  answer: $x = \text{fred}$

More Examples

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

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

  query: $\text{Father}(x)?$

  answer: $x = \text{fred}, \text{fatherof}(x, \text{mary})$

More Examples

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

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

  query: Father(x)?

  answer: $x =$ fred, fatherof$(x,$ mary$)$, $\exists y.$fatherof$(x, y) \wedge$ fatherof$(y,$ mary$)$, ...

More Examples

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

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

  query: $\text{Father}(x)$?
  answer: $x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y.\text{fatherof}(x, y) \land \text{fatherof}(y, \text{mary}), \ldots$

  query: $\text{Person}(x)$?

More Examples

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

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

  query: $\text{Father}(x)$?
  answer: $x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y.\text{fatherof}(x, y) \wedge \text{fatherof}(y, \text{mary}), \ldots$

  query: $\text{Person}(x)$?
  answer: $x = \text{mary}, x = \text{fred}$

More Examples

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

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

  query: Father(x)?
  answer: $x =$ fred, fatherof$(x$, mary), $\exists y.$fatherof$(x, y) \wedge$ fatherof$(y$, mary), ...

  query: Person(x)?
  answer: $x =$ mary, $x =$ fred, father-of(fred, x) (?!?)

More Examples

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

  query: $\text{Father}(x)$?
  answer: $x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y.\text{fatherof}(x,y) \wedge \text{fatherof}(y, \text{mary}), \ldots$

  query: $\text{Person}(x)$?
  answer: $x = \text{mary}$, $x = \text{fred}$, $\text{father-of}(\text{fred}, x)$ (?!?)

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

More Examples

- $\mathcal{T} = \{ \text{fatherof}(x, y) \to (\text{Father}(x) \land \text{Person}(y)), \text{Father}(x) \to \text{Person}(x),$
  $\text{Father}(x) \to \exists y.\text{fatherof}(x, y), \text{Person}(x) \to \exists y.\text{fatherof}(y, x)$
  $\text{fatherof}(x, z) \land \text{fatherof}(y, z) \to x = y \}$
  $\mathcal{A} = \{ \text{Father}(\text{fred}), \text{Person}(\text{mary}) \}$

  query: $\text{Father}(x)$?
  answer: $x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y.\text{fatherof}(x, y) \land \text{fatherof}(y, \text{mary}), \ldots$

  query: $\text{Person}(x)$?
  answer: $x = \text{mary}, x = \text{fred}, \text{father-of}(\text{fred}, x)$ (?!?)

- $\mathcal{T} = \{ \text{spouse}(x, y) \to \text{spouse}(y, x),$
  $\text{spouse}(x, z) \land \text{spouse}(y, z) \to x = y \}$
  $\mathcal{A} = \{ \text{spouse}(\text{mary}, \text{fred}) \}$

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

More Examples

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

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

  query: $\text{Father}(x)$?
  answer: $x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y.\text{fatherof}(x, y) \wedge \text{fatherof}(y, \text{mary}), \ldots$

  query: $\text{Person}(x)$?
  answer: $x = \text{mary}, x = \text{fred}, \text{father-of}(\text{fred}, x)$ (?!?)

- $\mathcal{T} = \{\ \text{spouse}(x, y) \rightarrow \text{spouse}(y, x),$
  $\text{spouse}(x, z) \wedge \text{spouse}(y, z) \rightarrow x = y\ \}$

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

  query: $\text{spouse}(x, \text{mary})$?
  answer: $x = \text{fred}$

More Examples

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

  query: $\text{Father}(x)$?
  answer: $x = \text{fred}, \text{fatherof}(x, \text{mary}), \exists y.\text{fatherof}(x,y) \wedge \text{fatherof}(y, \text{mary}), \ldots$

  query: $\text{Person}(x)$?
  answer: $x = \text{mary}, x = \text{fred}, \text{father-of}(\text{fred}, x)$ (?!?)

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

  query: $\text{spouse}(x, \text{mary})$?
  answer: $x = \text{fred}, \text{spouse}(x, \text{mary})$

More Examples

- $\mathcal{T} = \{$ fatherof$(x, y) \rightarrow ($Father$(x) \wedge$ Person$(y))$, Father$(x) \rightarrow$ Person(x),
  Father$(x) \rightarrow \exists y.$fatherof$(x, y)$, Person$(x) \rightarrow \exists y.$fatherof$(y, x)$
  fatherof$(x, z) \wedge$ fatherof$(y, z) \rightarrow x = y$ $\}$
  $\mathcal{A} = \{$ Father(fred), Person(mary) $\}$

  query: Father(x)?
  answer: $x =$ fred, fatherof$(x,$ mary$)$, $\exists y.$fatherof$(x, y) \wedge$ fatherof$(y,$ mary$)$, ...

  query: Person(x)?
  answer: $x =$ mary, $x =$ fred, father-of(fred, x) (?!?)

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

  query: spouse$(x,$ mary$)$?
  answer: $x =$ fred, spouse$(x,$ mary$)$, $\exists y.$spouse$(x, y) \wedge$ spouse$(y,$ fred$)$, ...

Generic Background Knowledge?

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

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

Generic Background Knowledge?

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

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

Desiderata (Referring Expression Types and Weak Identification)

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

Referring Expression Types

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

Referring Expression Types

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

Referring Expression Type 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** ψ

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

Referring Expression Types

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

Referring Expression Type and Typed Queries

Types: $Rt ::= Pd = \{?\} \mid Rt_1 \wedge Rt_2 \mid T \to Rt \mid Rt_1 ; Rt_2$

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

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

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

Theorem (Weak Identification; paraphrased)

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

"are all answers to ψ 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 $Person(x) \land phone\#(x, 1234567)$

Examples of Typed Queries

Reference via a Single-Attribute Key

Reference by a Multi-Attribute Key

"The title and publisher of any journals"

$$\textbf{select } x : \textit{title} = \{?\} \land \textit{publishedBy} = \{?\}$$
$$\textbf{where } \textit{Journal}(x)$$

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"

$$\begin{aligned}&\textbf{select } x : \mathit{Person} \rightarrow \mathit{ssn\#} = \{?\} \; ; \\ &\qquad\quad \mathit{Company} \rightarrow \mathit{tickerSymbol} = \{?\} \\ &\textbf{where } \mathit{LegalEntity}(x)\end{aligned}$$

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

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."

$$\begin{array}{l} \textbf{select } x : \textit{Journal} \rightarrow (\textit{title} = \{?\} \wedge \textit{publisher} = \{?\}); \\ \qquad\qquad \textit{EditedCollection} \rightarrow \textit{isbn\#} = \{?\} \,;\, \{?\} \\ \textbf{where } \textit{Publication}(x) \end{array}$$

answers: $\{x \mapsto \textit{Journal}(x) \wedge \textit{title}(x, \text{“AIJ”}) \wedge \textit{publisher}(x, \text{“Elsevier”})\}$
$\{x \mapsto \textit{EditedCollection}(x) \wedge \textit{isbn\#}(x, 123456789)\}$
$\{x \mapsto x = \texttt{/guid/9202a8c04000641f8000000...}\}$.

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
$$T_i \to Pd_{i,1} = \{?\} \wedge \ldots \wedge Pd_{i,k_i} = \{?\};$$
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

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

The Tractable (practical) Cases

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 an ABox)

The witnesses for anonymous objects (step (3))
$\longrightarrow$ *last* named individual on a path *towards* the anonymous object

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

RECORDING/REPRESENTING FACTUAL DATA

Referring Expressions for Ground Knowledge (CBox)

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

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

Referring Expressions for Ground Knowledge (CBox)

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 A objects (o_1, o_2) identified by their f value (such as an employee id) within A:

$$A \sqcap \exists f.\{123\} \text{ and } A \sqcap \exists f.\{345\}.$$

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

$$A \sqcap \exists f.\{123\} \sqcap \exists g.(A \sqcap \exists f.\{345\}).$$

Referring Expressions for Ground Knowledge (CBox)

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 A objects (o_1, o_2) identified by their f value (such as an employee id) within A:

$$A \sqcap \exists f.\{123\} \text{ and } A \sqcap \exists f.\{345\}.$$

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

$$A \sqcap \exists f.\{123\} \sqcap \exists g.(A \sqcap \exists f.\{345\}).$$

Issues:

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

Referring Expressions for Ground Knowledge (CBox)

Example

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

```
{"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

$$\text{PERSON} \sqcap (\exists \mathit{fname}.\{\text{``John''}\}) \sqcap (\exists \mathit{lname}.\{\text{``Smith''}\}) \sqcap \exists \mathit{age}.\{25\}$$
$$\sqcap \exists \mathit{phoneNumFor}^{-1}.((\exists \mathit{loc}.\{\text{``home''}\}) \sqcap (\exists \mathit{dialnum}.\{\text{``212 555-1234''}\}))$$
$$\sqcap \exists \mathit{phoneNumFor}^{-1}.((\exists \mathit{loc}.\{\text{``work''}\}) \sqcap (\exists \mathit{dialnum}.\{\text{``212 555-4567''}\}))$$

This assertion is admissible, e.g., whenever the combination of *fname* and *lname* identifies PERSONs.

Heterogeneous Data Integration (example)

Example

- TBox

$$\{ \quad A \sqsubseteq B,\ C \sqsubseteq B,\\ A \sqsubseteq A : f \rightarrow id,\ B \sqsubseteq B : f, g \rightarrow id,\ C \sqsubseteq C : g \rightarrow id\\ A \sqsubseteq B : f \rightarrow id, C \sqsubseteq B : g \rightarrow id \quad \},$$

- CBox

$$\{\ A \sqcap \exists f.\{3\},\quad B \sqcap \exists f.\{3\} \sqcap \exists g.\{5\},\quad C \sqcap \exists g.\{5\}\ \}.$$

Heterogeneous Data Integration (example)

Example

- TBox

$$\{ \quad A \sqsubseteq B,\ C \sqsubseteq B, \\ A \sqsubseteq A : f \rightarrow id,\ B \sqsubseteq B : f, g \rightarrow id,\ C \sqsubseteq C : g \rightarrow id \\ A \sqsubseteq B : f \rightarrow id, C \sqsubseteq B : g \rightarrow id \quad \},$$

- CBox

$$\{ \ A \sqcap \exists f.\{3\}, \quad B \sqcap \exists f.\{3\} \sqcap \exists g.\{5\}, \quad C \sqcap \exists g.\{5\} \ \}.$$

Heterogeneous Identification

"$A \sqcap \exists f.\{3\}$" identifies *the same object* as "$B \sqcap \exists f.\{3\} \sqcap \exists g.\{5\}$",
and in turn as "$C \sqcap \exists g.\{5\}$"

Heterogeneous Data Integration (example)

Example

- TBox

$$\{\ A \sqsubseteq B,\ C \sqsubseteq B,\ A \sqsubseteq A : f \rightarrow id,\ B \sqsubseteq B : f, g \rightarrow id,\ C \sqsubseteq C : g \rightarrow id\ A \sqsubseteq B : f \rightarrow id, C \sqsubseteq B : g \rightarrow id \ \},$$

- CBox

$$\{\ A \sqcap \exists f.\{3\},\quad B \sqcap \exists f.\{3\} \sqcap \exists g.\{5\},\quad C \sqcap \exists g.\{5\}\ \}.$$

Heterogeneous Identification

"$A \sqcap \exists f.\{3\}$" identifies *the same object* as "$B \sqcap \exists f.\{3\} \sqcap \exists g.\{5\}$", and in turn as "$C \sqcap \exists g.\{5\}$"

…and thus is an answer to $\{\ x \mid \exists y.A(x) \wedge C(y) \wedge x = y\ \}$

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 $\to$ variant minimal co-referring expressions.

Reasoning and QA with CBoxes [DL18]

Theorem (CBox Admissibility)

Let $\mathcal{T}$ be a $\mathcal{CFDI}_{nc}^{\forall}$ 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

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

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

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 if $(\mathcal{T}, \mathrm{Simp}(\mathcal{C}))$ 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 $(\mathcal{T}, \mathrm{Simp}(\mathcal{C}))$.

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 $\mathcal{C}_{\mathrm{AR}}$

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 $\mathcal{C}_{\mathsf{AR}}$: abstract attributes and attribute paths

Abstract Relational Queries

SQLP

(pretty) standard `select-from-where-union-except` SQL syntax
. . . with extensions to $\mathcal{C}_{\mathsf{AR}}$: abstract attributes 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
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 $\mathcal{C}_{\mathrm{AR}}$: abstract attributes 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
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 the Approach/Technology Succeed? [EKAW18]

1. ARM/SQLP Helps Users (User Study)
2. ARM/SQLP Can be Efficiently Implemented
   - Mapping to *standard relational model* with the help of *referring expressions*
   - 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

Experimental Design (HCI experiments)

Hypotheses

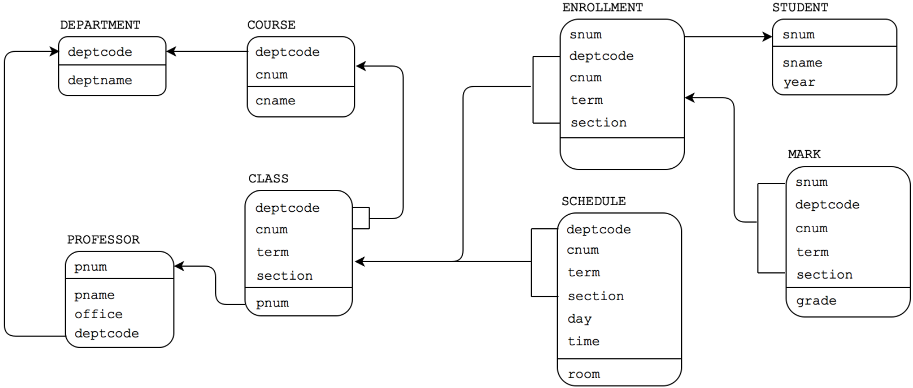
H_t: no difference between RM/SQL and ARM/SQLP in the mean time taken

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

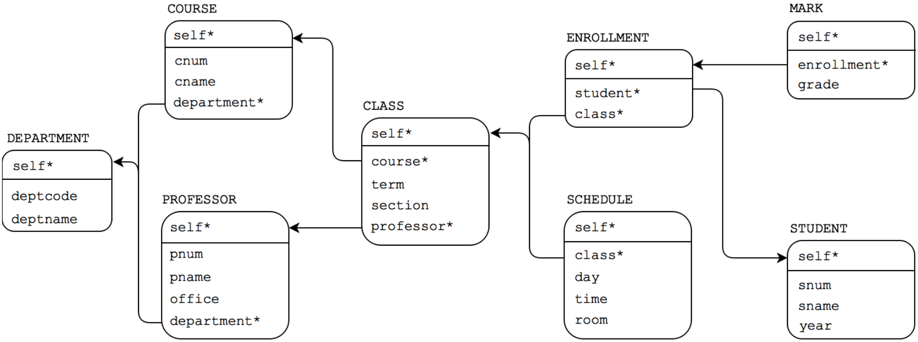
Methods

- 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

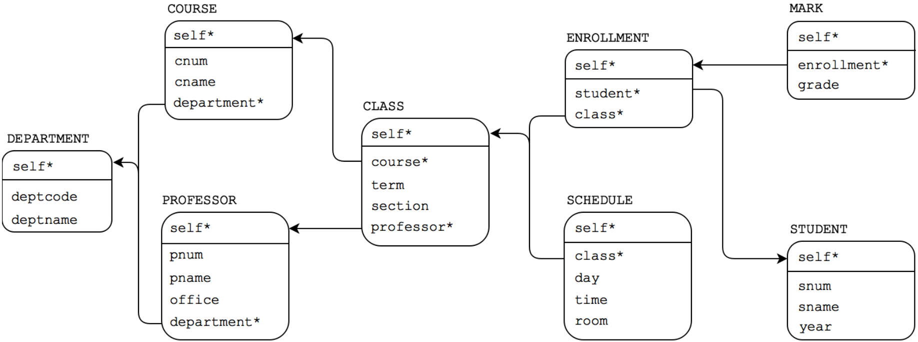
Course Enrollment as an RM Schema



Course Enrollment as an ARM Schema



Course Enrollment 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'
```

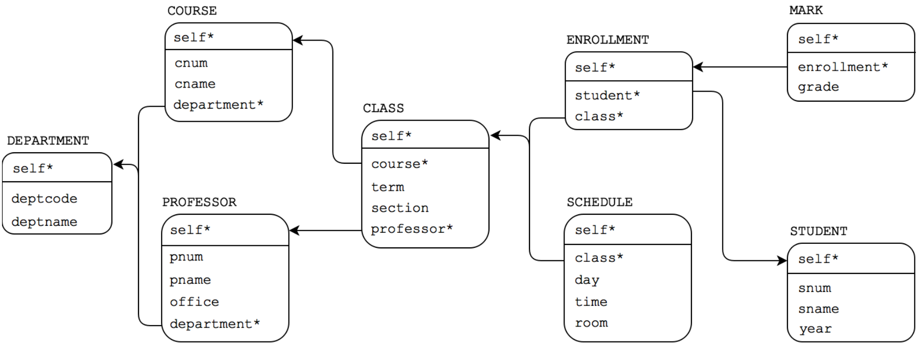
Domain expert needs to understand structure of PK/FKs: BAD!!

ARM/SQLP:

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

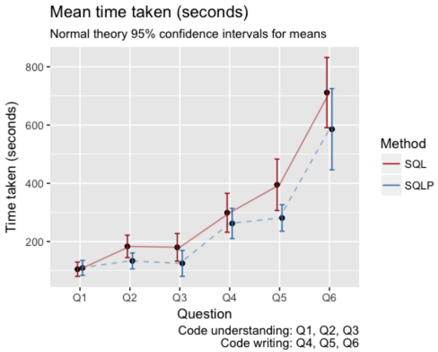
ARM Schema and Path Navigation

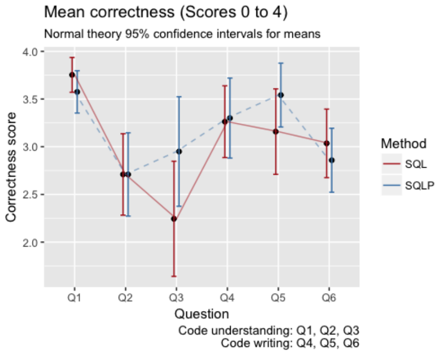
```
select distinct e.student.sname as name
from ENROLLMENT e
where e.class.professor.pname = 'Alan John'
```



Experiments: Results

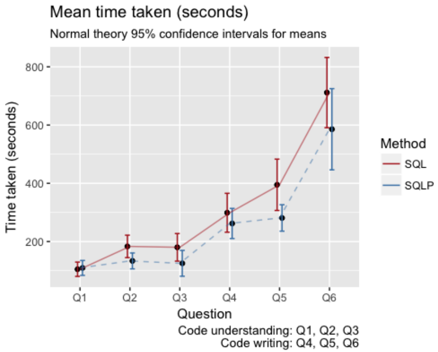
Mean performance for all subjects: SQL solid; SQLP dashed.

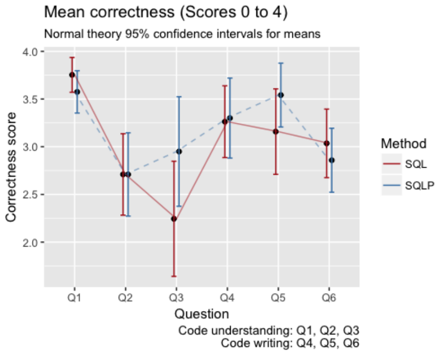




Experiments: Results

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





- SQLP outperforms SQL in time taken
- No significant difference in correctness (Q3, Q5 almost significant)

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`.

  ⇒ 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` $\rightarrow$ `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`.

  ⇒ but what happens to objects *that are both* a `PERSON` and a `COMPANY`??

  ⇒ 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 $\mathcal{C}_{\mathrm{AR}}$ to ones over concrete tables.

Assignment of Referring Types

IDEA

Assign a referring expression type RTA(T) to each table T in Σ.

Assignment of Referring Types

IDEA

Assign a referring expression type RTA(T) to each table T in Σ.

Example

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

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

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

   $\Rightarrow$ the ability to compare the $\texttt{OID}$ values is lost;

Assignment of Referring Types

IDEA

Assign a referring expression type RTA(T) to each table T in Σ.

Example

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

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

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

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

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

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

Assignment of Referring Types

IDEA

Assign a referring expression type RTA(T) to each table T in Σ.

Definition (Identity-resolving RTA(.))

Let Σ be a $\mathcal{C}_{\text{AR}}$ schema and RTA a referring type assignment for Σ. Given a linear order $\mathcal{O} = (T_{i_1}, \ldots, T_{i_n})$ on the set Tables(Σ), 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(Σ):

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

Assignment of Referring Types

IDEA

Assign a referring expression type RTA(T) to each table T in Σ.

Definition (Identity-resolving RTA(.))

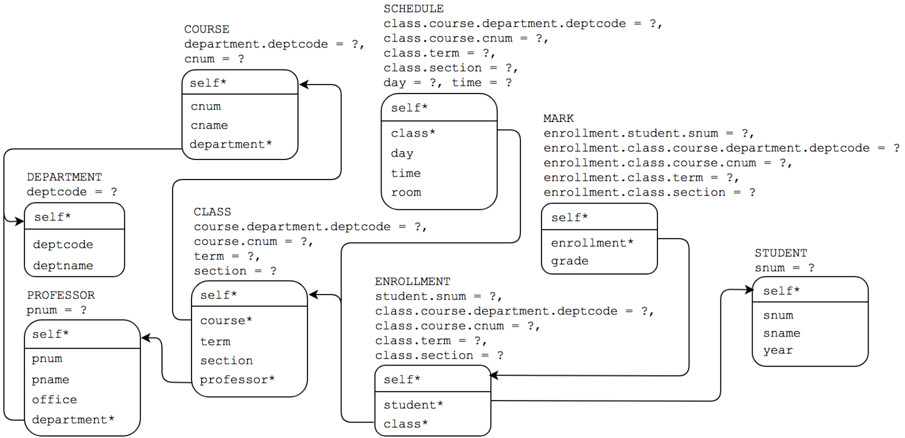
The definition achieves the following:

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



RM2ARM Algorithm (highlights)

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

RM2ARM Algorithm (highlights)

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?

RM2ARM Algorithm (highlights)

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?

RM2ARM Algorithm (highlights)

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

RM2ARM Algorithm (highlights)

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)

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:

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. (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:

Theorem

Let Σ *be a* $\mathcal{C}_{\mathrm{AR}}$ *schema and let* RTA *an identity resolving type assignment for* Σ*. For any SQLP query Q over* Σ

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

for every database instance I of Σ*.* □

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, 40+ graduate students, . . .
- 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 75+ professors, 300+ 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)