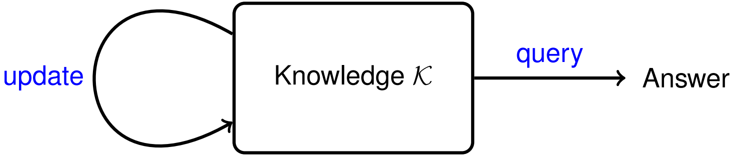
From Data Independence to Ontology Based Data Access (and back)

David Toman

D.R. Cheriton School of Computer Science
University of Waterloo

Joint work with Alexander Hudek and Grant Weddell

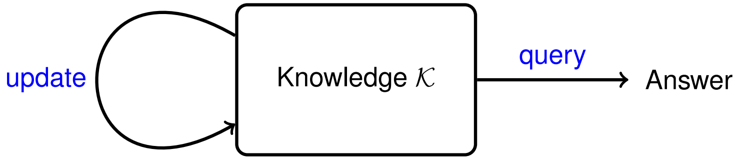
Knowledge Representation: a Big Picture



What is "Knowledge" (how is it represented, and does the user care?)

$\Rightarrow$ not really as long as the updates and queries "play nicely together"

Knowledge Representation: a Big Picture



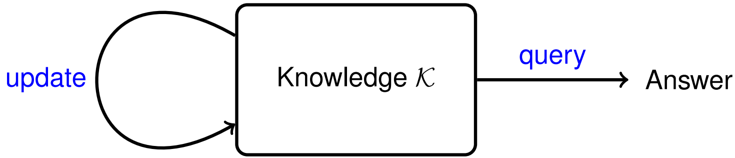
What is "Knowledge" (how is it represented, and does the user care?)

$\Rightarrow$ not really as long as the updates and queries "play nicely together"

Structured World:

- $\mathcal{K}$ is a (first order) theory,
- queries are (FO) formulæ with answers defined by entailment, and
- updates are (variations on) belief revision.

Knowledge Representation: a Big Picture



What is “Knowledge” (how is it represented, and does the user care?)

$\Rightarrow$ not really as long as the updates and queries “play nicely together”

Probabilistic World:

- $\mathcal{K}$ is a ML model (e.g., neural net),
- queries are inputs (e.g., photos) and answers are labels
- updates are pairs of, e.g., photos with their labels.

Ontology-based Data Access (OBDA) [Calvanese et al.: Mastro, 2011]

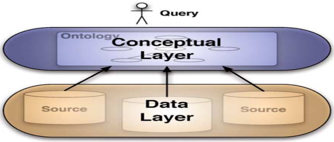
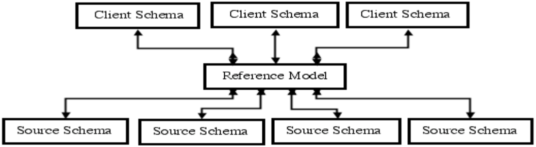


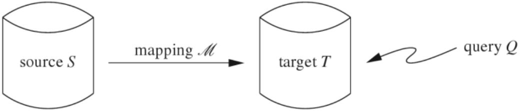
Fig. 1. Ontology-based data access.

Information Integration [Genesereth: Data Integration, 2010]

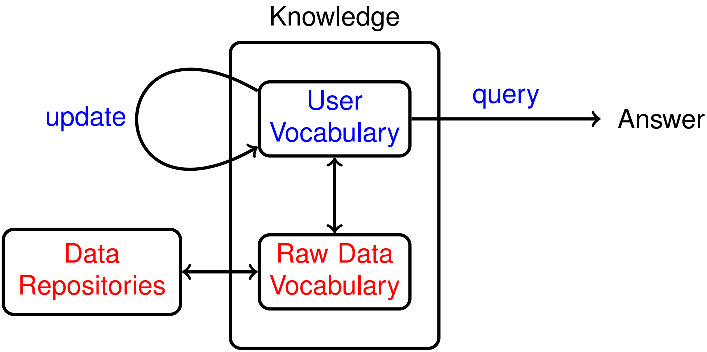


Data Exchange [Arenas et el.: Data Exchange, 2014]

The general setting of data exchange is this:



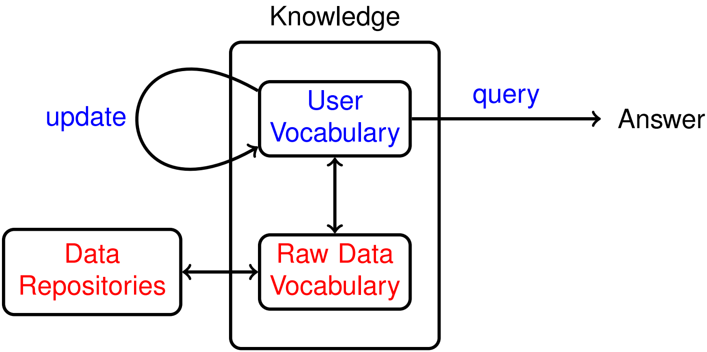
Data vs. Metadata



1. Metadata: constraints formulated in FOL (static) [called a TBox]
2. Data: ground tuples (can be "modified") [called an ABox]

$\Rightarrow$ user queries and updates only about data.

Data vs. Metadata



1. Metadata: constraints formulated in FOL (static) [called a TBox]
2. Data: ground tuples (can be "modified") [called an ABox]

$\Rightarrow$ user queries and updates only about data.

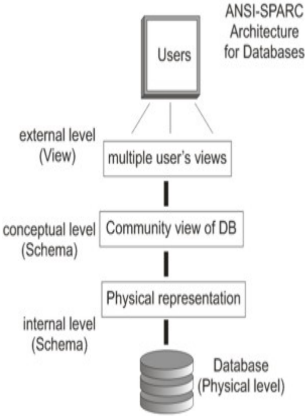
(Physical) Data Independence

IDEA:

Separate the users' view(s) of the data from the way it is physically represented.

- independent customized user views;
- changes to conceptual structure without affecting users;
- physical storage details hidden from users;
- changes to physical storage without affecting logical view;

Originally just two levels: physical and conceptual/logical [Codd1970].



[ANSI/X3/SPARC Standards Planning and Requirements Committee, Bachman, 1975]

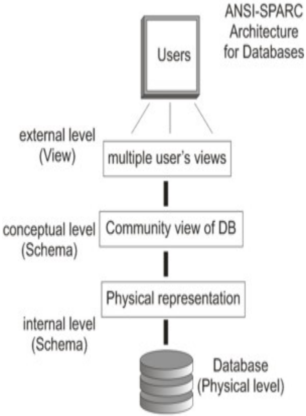
(Physical) Data Independence

IDEA:

Separate the users' view(s) of the data from the way it is physically represented.

- independent customized user views,
- changes to conceptual structure without affecting users,
- physical storage details hidden from users,
- changes to physical storage without affecting logical view,

Originally just two levels: physical and conceptual/logical [Codd1970].



[ANSI/X3/SPARC Standards Planning and Requirements Committee, Bachman, 1975]

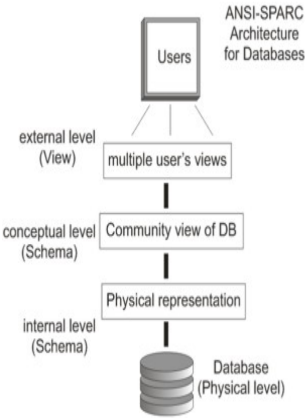
(Physical) Data Independence

IDEA:

Separate the users' view(s) of the data from the way it is physically represented.

- independent customized user views,
- changes to conceptual structure without affecting users,
- physical storage details hidden from users,
- changes to physical storage without affecting logical view,

Originally just two levels: physical and conceptual/logical [Codd1970].



[ANSI/X3/SPARC Standards Planning and Requirements Committee, Bachman, 1975]

Outline

1. Queries
2. Updates
3. How does it Work and (Performance) Bonus
4. Future Research/Open Issues

QUERIES AND QUERY COMPILATION

The Structured/Logical Way (via an OBDA example)

Queries and Ontologies

Queries are answered not only w.r.t. *explicit data* ($\mathcal{A}$)
but also w.r.t. *background knowledge* ($\mathcal{T}$)

$\Rightarrow$ Ontology-based Data Access (OBDA)

Example

- Socrates is a MAN (explicit data)
- Every MAN is MORTAL (ontology)

List all MORTALs $\Rightarrow$ {Socrates} (query)

How do we answer queries?

Using *logical implication* (to define *certain answers*):

$$\text{Ans}(\varphi, \mathcal{A}, \mathcal{T}) := \{\varphi(a_1, \ldots, a_k) \mid \mathcal{T} \cup \mathcal{A} \models \varphi(a_1, \ldots, a_k)\}$$

$\Rightarrow$ answers are *ground φ-atoms* logically implied by $\mathcal{A} \cup \mathcal{T}$.

The Structured/Logical Way (via an OBDA example)

Queries and Ontologies

Queries are answered not only w.r.t. *explicit data* ($\mathcal{A}$)
but also w.r.t. *background knowledge* ($\mathcal{T}$)

$\Rightarrow$ Ontology-based Data Access (OBDA)

Example

- Socrates is a MAN (explicit data)
- Every MAN is MORTAL (ontology)

List all MORTALs $\Rightarrow$ $\{$Socrates$\}$ (query)

How do we answer queries?

Using *logical implication* (to define *certain answers*):

$$\text{Ans}(\varphi, \mathcal{A}, \mathcal{T}) := \{\varphi(a_1, \ldots, a_k) \mid \mathcal{T} \cup \mathcal{A} \models \varphi(a_1, \ldots, a_k)\}$$

$\Rightarrow$ answers are *ground φ-atoms* logically implied by $\mathcal{A} \cup \mathcal{T}$.

The Logical Way: Complexity

The Good News

LOGSPACE/PTIME (data complexity) for query answering:

- (U)CQ and
- DL-Lite/$\mathcal{EL}_\bot$/$\mathcal{CFD}_{\mathrm{nc}}^{\forall}$/“rules”-lite (Horn), s-t dependencies,. . .

The Bad News

- no negative queries/sub-queries
- no negations in ABox
- no closed-world assumption
- counter-intuitive query answers

$\Rightarrow$ the same goes for *information integration, data exchange, etc.*

The Logical Way: Complexity

The Good News

LOGSPACE/PTIME (data complexity) for query answering:

- (U)CQ and
- DL-Lite/$\mathcal{EL}_{\perp}$/$\mathcal{CFD}^{\forall}_{\mathrm{nc}}$/“rules”-lite (Horn), s-t dependencies,. . .

The Bad News

- no negative queries/sub-queries
- no negations in ABox
- no closed-world assumption
- **counter-intuitive query answers**

⇒ the same goes for *information integration*, *data exchange*, etc.

The Logical Way: Complexity

The Good News

LOGSPACE/PTIME (data complexity) for query answering:

- (U)CQ and
- DL-Lite/$\mathcal{EL}_{\perp}$/$\mathcal{CFD}_{\text{nc}}^{\forall}$/"rules"-lite (Horn), s-t dependencies,...

The Bad News

- no negative queries/sub-queries
- no negations in ABox
- no closed-world assumption
- **counter-intuitive query answers**

$\Rightarrow$ the same goes for *information integration*, *data exchange*, etc.

Difficulties: Unintuitive Answers

Example

- $EMP(Sue)$
- $EMP \sqsubseteq \exists PHONENUM$ (or $\forall x.EMP(x) \rightarrow \exists y.PHONENUM(x, y)$)

Difficulties: Unintuitive Answers

Example

- $EMP(Sue)$
- $EMP \sqsubseteq \exists PHONENUM$ (or $\forall x.EMP(x) \to \exists y.PHONENUM(x, y)$)

User: *Does Sue have a phone number?*

Information System: *YES*

Difficulties: Unintuitive Answers

Example

- $EMP(Sue)$
- $EMP \sqsubseteq \exists PHONENUM$ (or $\forall x.EMP(x) \rightarrow \exists y.PHONENUM(x, y)$)

User: *Does Sue have a phone number?*

Information System: *YES*

User: *OK, tell me Sue's phone number!*

Information System: *(no answer)*

Difficulties: Unintuitive Answers

Example

- $EMP(Sue)$
- $EMP \sqsubseteq \exists PHONENUM$ (or $\forall x.EMP(x) \rightarrow \exists y.PHONENUM(x,y)$)

User: *Does Sue have a phone number?*

Information System: *YES*

User: *OK, tell me Sue's phone number!*

Information System: *(no answer)*

User: 

Rewritability and Definability

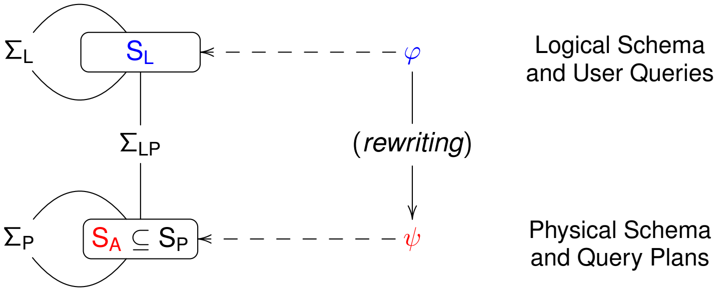
User and System Expectations

Queries	range-restricted FOL (a.k.a. SQL)
Ontology/Schema	range-restricted FOL $\Sigma := \Sigma_L \cup \Sigma_{LP} \cup \Sigma_P$
Data	CWA (complete information)

Rewritability and Definability

User and System Expectations

Queries	range-restricted FOL	over S_L *definable* w.r.t. Σ and S_A
Ontology/Schema	range-restricted FOL	$\Sigma := \Sigma_L \cup \Sigma_{LP} \cup \Sigma_P$
Data	CWA (complete information for S_A symbols)	



[Borgida, de Bruijn, Franconi, Seylan, Straccia, Toman, Weddell: On Finding Query Rewritings under Expressive Constraints. SEBD 2010: 426-437]

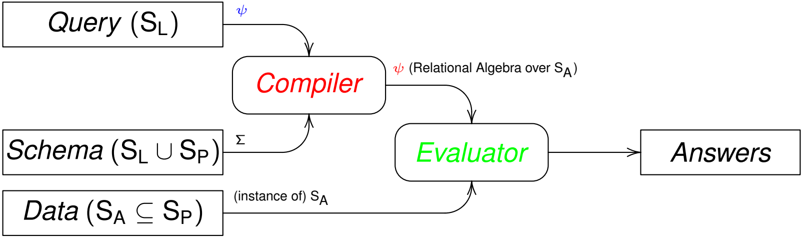
Rewritability and Definability

User and System Expectations

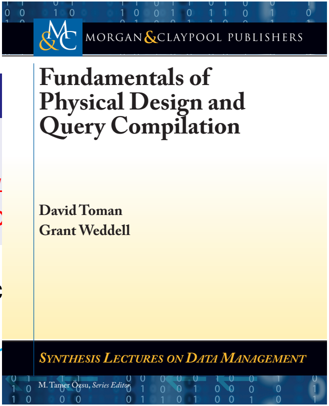
Queries	range-restricted FOL over S_L *definable* w.r.t. Σ and S_A
Ontology/Schema	range-restricted FOL $\Sigma := \Sigma_L \cup \Sigma_{LP} \cup \Sigma_P$
Data	CWA (complete information for S_A symbols)

- to users it looks like a *single model* (of the logical schema)
- implementation can pick from many models

  but *definable* queries answer the same in each of them



Rewritability and Definability



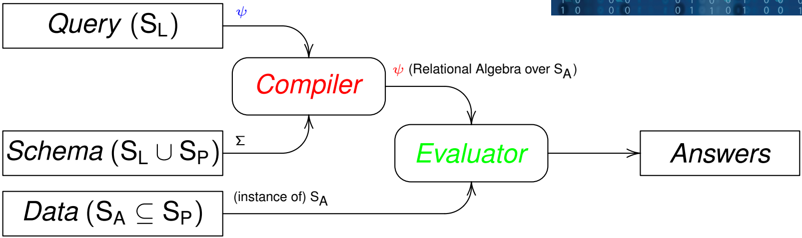
©2011

User and System Expectations

Queries	range-restricted FOL over S_L
Ontology/Schema	range-restricted FOL $\Sigma := \Sigma_L$
Data	CWA (complete information fo

- to users it looks like a *single model* (of the logic
- implementation can pick from many models

but *definable* queries answer



(First-order) Query Rewritability

Rewritability (Decision Problem)

Given

1. a TBox $\mathcal{T}$ and
2. a Query φ

decide whether there is a FO query ψ such that

$$\text{Ans}(\varphi, \mathcal{A}, \mathcal{T}) = \text{Ans}(\psi, \mathcal{A}, \emptyset)$$

for every ABox $\mathcal{A}$ (optionally where ψ is over a sub-vocabulary of $\mathcal{T}$).

[Bienvenu, Lutz, Wolter: First-Order Rewritability of Atomic Queries in Horn Description Logics. IJCAI 2013. (and many papers followed...)]

What can we do?

GOAL

Generate query plans *that compete with hand-written programs in C*

1. standard RDBMS physical designs
2. linked data structures, pointers, . . .
3. access to search structures (index access and selection),
4. hash-based access to data (including hash-joins),
5. multi-level storage (aka disk/remote/distributed files), . . .
6. materialized views (FO-definable),

. . . all without having to code (too much) in C/C++ !

Standard Physical Designs

1. scanning (flat) files
2. primary and secondary indices (via record ids/addresses)
3. horizontal partitioning/sharding
4. column store/index-only plans
5. (disjoint) generalizations

Example: disjoint subclasses

Query

```
undergrad(x,y) <-> ex(r,ustudent(r,x,y))
```

…with access paths `student` and `gstudent`

Example: disjoint subclasses

Query

```
undergrad(x,y) <-> ex(r,ustudent(r,x,y))
```

```
% coverage
student(r,x,y) -> (gstudent(r) or ustudent(r,x,y)),
ustudent(r,x,y) -> student(r,x,y),
gstudent(r) -> ex([x,y],student(r,x,y)),
% disjointness
gstudent(r) and ex([x,y],ustudent(r,x,y)) -> bot,
% key
student(r,x1,y1) and student(r,x2,y2) ->
                                  (x1=x2 and y1=y2)
```

...with access paths `student` and `gstudent`

Example: disjoint subclasses

Query

```
undergrad(x,y) <-> ex(r,ustudent(r,x,y))
```

…with access paths `student` and `gstudent`

```
david$ compile tests/848ex/subclass2.fol
query(undergrad,2,0,[var(0,0,1,int),var(0,0,2,int)]) <->
  ex(var(0,19,4),
    and (
      student(var(0,19,4),var(0,0,1),var(0,0,2))
      not (
        gstudent(var(0,19,4))
      ) ) )
```

Example: disjoint subclasses

Query

```
undergrad(x,y) <-> ex(r,ustudent(r,x,y))
```

. . . with access paths `student` and `gstudent`

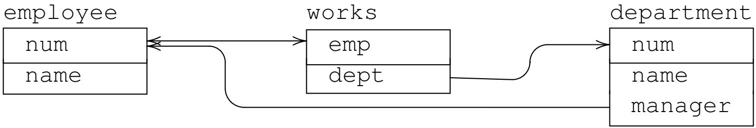
```
david$ compile tests/848ex/subclass2.fol
query(undergrad,2,0,[var(0,0,1,int),var(0,0,2,int)]) <->
  ex(var(0,19,4),
    and (
      student(var(0,19,4),var(0,0,1),var(0,0,2))
      not (
        gstudent(var(0,19,4))
      ) ) )
```

or, in C-like syntax:

for (r, x, y) in `student` do
        if r in `gstudent` skip else return (x, y);

Lists and Pointers

1. Logical Schema



2. Physical Design: a *linked list of `emp` records pointing to `dept` records*.

```
record emp of                      record dept of
       integer     num                    integer     num
       string      name                   string      name
       integer     salary                 reference   manager
       reference   dept
```

3. Access Paths: $\texttt{empfile}/1/0$, $\texttt{emp-num}/2/1$, … (but no `deptfile`)

4. Integrity Constraints (many), e.g.,

$$\forall x, y, z.\texttt{employee}(x, y, z) \rightarrow \exists w.\texttt{empfile}(w) \wedge \texttt{emp-num}(w, x),$$
$$\forall a, x.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \rightarrow \exists y, z.\texttt{employee}(x, y, z), \ldots$$

What can this do: navigating pointers

1. List all employee numbers and names (employee(x, y)):

$$\exists a.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \wedge \texttt{emp-name}(a, y)$$

What can this do: navigating pointers

1. List all employee numbers and names (employee(x, y)):

$$\exists a.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \wedge \texttt{emp-name}(a, y)$$

or, in C-like syntax:

```
for a in empfile do
        x := a->num;
        y := a->name;
```

What can this do: navigating pointers

1. List all employee numbers and names $(\texttt{employee}(x, y))$:

   $$\exists a.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \wedge \texttt{emp-name}(a, y)$$

2. List all department numbers and their names $(\exists z.\texttt{department}(x, y, z))$:

University of Waterloo

What can this do: navigating pointers

1. List all employee numbers and names $(\texttt{employee}(x, y))$:

$$\exists a.\texttt{empfile}(a) \land \texttt{emp-num}(a, x) \land \texttt{emp-name}(a, y)$$

2. List all department numbers and their names $(\exists z.\texttt{department}(x, y, z))$:

$$\exists a, d, e.\texttt{empfile}(a) \land \texttt{emp-dept}(a, d) \land \texttt{dept-num}(d, x) \land \texttt{dept-name}(d, y)$$

$\Rightarrow$ needs "departments have at least one employee".

What can this do: navigating pointers

1. List all employee numbers and names (employee(x, y)):

$$\exists a.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \wedge \texttt{emp-name}(a, y)$$

2. List all department numbers and their names ($\exists z.\texttt{department}(x, y, z)$):

$$\exists a, d, e.\texttt{empfile}(a) \wedge \texttt{emp-dept}(a, d) \wedge \texttt{dept-num}(d, x) \wedge \texttt{dept-name}(d, y)$$

$\Rightarrow$ needs "departments have at least one employee".

$$\exists a, b, d.\texttt{empfile}(a) \wedge \texttt{emp-dept}(a, d) \wedge \texttt{dept-num}(d, x) \wedge \texttt{dept-name}(d, y) \wedge \texttt{dept-mgr}(d, a)$$

$\Rightarrow$ needs "managers work in their own departments".

What can this do: navigating pointers

1. List all employee numbers and names ($\texttt{employee}(x, y)$):

$$\exists a.\texttt{empfile}(a) \land \texttt{emp-num}(a, x) \land \texttt{emp-name}(a, y)$$

2. List all department numbers and their names ($\exists z.\texttt{department}(x, y, z)$):

$$\exists a, d, e.\texttt{empfile}(a) \land \texttt{emp-dept}(a, d) \land \texttt{dept-num}(d, x) \land \texttt{dept-name}(d, y)$$

$\Rightarrow$ needs "departments have at least one employee".

... needs *duplicate elimination* during projection.

$$\exists a, b, d.\texttt{empfile}(a) \land \texttt{emp-dept}(a, d) \land \texttt{dept-num}(d, x) \land \texttt{dept-name}(d, y) \land \texttt{dept-mgr}(d, a)$$

$\Rightarrow$ needs "managers work in their own departments".

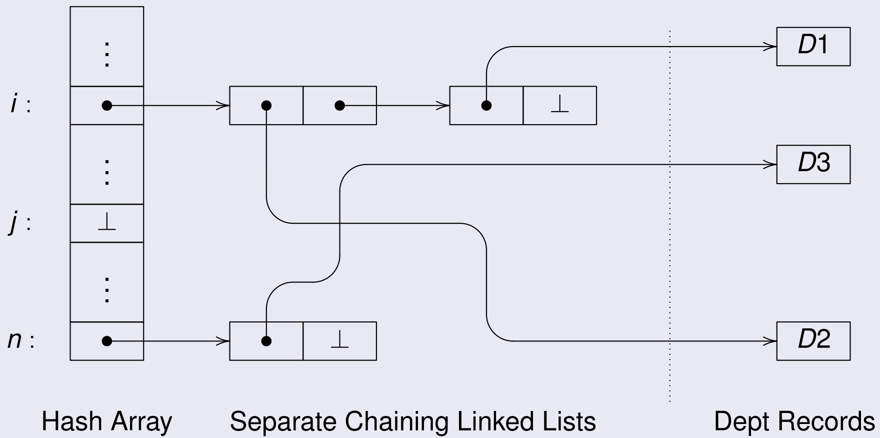
... NO *duplicate elimination* during projection.

. . . and we really can synthesize this!

```
david$ compile tests/new_fe/book-em-v4-new-query.fol
query(q0dept2,2,0,[var(0,0,1,int),var(0,0,2,int)]) <->
  ex(var(0,76,4),
    ex(var(0,81,5),
      and (
        and (
          empfile(var(0,76,4))
          emp_dept(var(0,76,4),var(0,81,5))
        )
        and (
          and (
            dept_num(var(0,81,5),var(0,0,1))
            dept_name(var(0,81,5),var(0,0,2))
          )
          dept_mgr(var(0,81,5),var(0,76,4))
        )
      )
    )
```

What can it do: Hashing, Lists, et al.

Hash Index with (list-based) Separate Chaining



What can it do: Hashing, Linked lists, et al.

Hash Index on `department`'s `name`:

Access paths:

$\mathsf{S}_\mathsf{A} \supseteq \{\mathtt{hash}/2/1, \mathtt{hasharraylookup}/2/1, \mathtt{listscan}/2/1\}$.

Physical Constraints:

$$\begin{aligned}\Sigma_{\mathsf{LP}} \supseteq \{ & \forall x,y.((\mathtt{deptfile}(x) \wedge \mathtt{dept\text{-}name}(x,y)) \to \exists z,w.(\mathtt{hash}(y,z) \\ & \qquad \wedge \mathtt{hasharraylookup}(z,w) \wedge \mathtt{listscan}(w,x))), \\ & \forall x,y.(\mathtt{hash}(x,y) \to \exists z.\mathtt{hasharraylookup}(y,z)), \\ & \forall x,y.(\mathtt{listscan}(x,y) \to \mathtt{deptfile}(y)) \qquad \}\end{aligned}$$

What can it do: Hashing, Linked lists, et al.

Hash Index on `department`'s `name`:

Access paths:

$S_A \supseteq \{\texttt{hash}/2/1, \texttt{hasharraylookup}/2/1, \texttt{listscan}/2/1\}$.

Physical Constraints:

$$\Sigma_{LP} \supseteq \{\forall x, y.((\texttt{deptfile}(x) \wedge \texttt{dept-name}(x, y)) \to \exists z, w.(\texttt{hash}(y, z) \wedge \texttt{hasharraylookup}(z, w) \wedge \texttt{listscan}(w, x))),$$
$$\forall x, y.(\texttt{hash}(x, y) \to \exists z.\texttt{hasharraylookup}(y, z)),$$
$$\forall x, y.(\texttt{listscan}(x, y) \to \texttt{deptfile}(y)) \}$$

Query:

$\exists y.(\texttt{department}(x_1, p, y) \wedge \texttt{employee}(y, x_2))\{p\}$.

$$\exists h, l, d, e.\texttt{hash}(p, h) \wedge \texttt{hasharraylookup}(h, l) \wedge \texttt{listscan}(l, d) \wedge \texttt{dept-name}(d, p) \wedge \texttt{dept-num}(d, x_1) \wedge \texttt{dept-mgr}(d, e) \wedge \texttt{emp-name}(e, x_2)$$

What can this do: two-level store

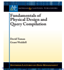
The access path empfile is refined by emppages/1/0 and emprecords/2/1:

emppages returns (sequentially) disk pages containing emp records, and
emprecords given a disc page, returns emp records in that page.

5. List all employees with the same name $(\exists z.\texttt{employee}(x_1, z) \wedge \texttt{employee}(x_2, z))$:

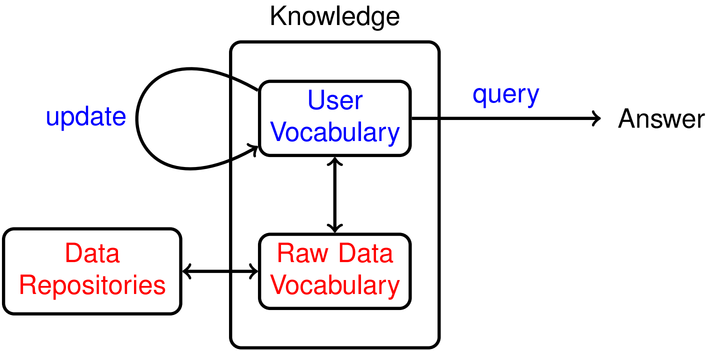
$$\exists y, z, w, v, p, q.\texttt{emppages}(p) \wedge \texttt{emppages}(q) \\ \wedge \texttt{emprecords}(p, y) \wedge \texttt{emp-num}(y, x_1) \wedge \texttt{emp-name}(y, w) \\ \wedge \texttt{emprecords}(q, z) \wedge \texttt{emp-num}(z, x_2) \wedge \texttt{emp-name}(z, v) \\ \wedge \texttt{compare}(w, v).$$

$\Rightarrow$ this plan implements the *block nested loops join* algorithm.

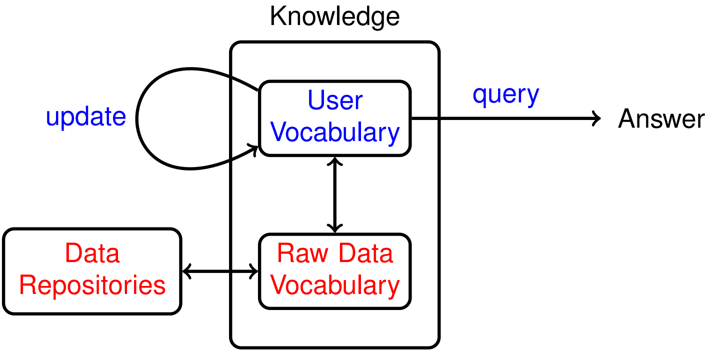
. . . more examples in .

UPDATES

Updates

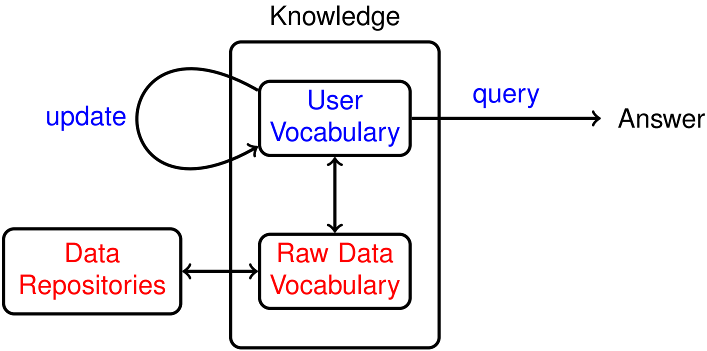

Knowledge
update
User Vocabulary
query
Answer
Data Repositories
Raw Data Vocabulary

Updates



1. Katsuno, Mendelzon: On the Difference between Updating a Knowledge Base and Revising It. KR 1991.
2. De Giacomo, Lenzerini, Poggi, Rosati: On Instance-level Update and Erasure in Description Logic Ontologies. J. Log. Comput. 19(5) 2009.

Updates



1. Katsuno, Mendelzon: On the Difference between Updating a Knowledge Base and Revising It. KR 1991.
2. De Giacomo, Lenzerini, Poggi, Rosati: On Instance-level Update and Erasure in Description Logic Ontologies. J. Log. Comput. 19(5) 2009.

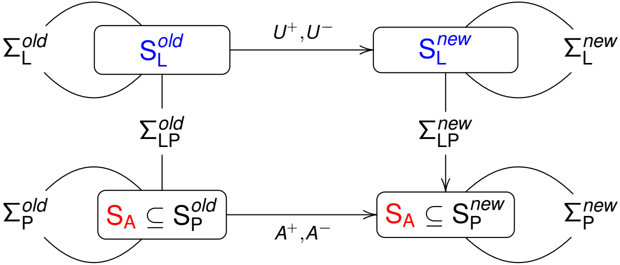
. . . we follow a *definable updates* approach here instead. . .

Updates and Definability

User updates *only through logical schema*:

⇒ supplying "delta" relations (sets of tuples)

- Two copies of the schema: Σ^{old} and Σ^{new};
- Delta relations: R^+ (insertions) and R^- (deletions);
- Constraints: $\forall\bar{x}.(R^{old}(\bar{x}) \vee R^+(\bar{x})) \equiv (R^{new}(\bar{x}) \vee R^-(\bar{x}))$,
  $\forall\bar{x}.(R^+(\bar{x}) \wedge R^-(\bar{x})) \to \bot$



Updates and Definability

User updates *only through logical schema*:
$\Rightarrow$ supplying "delta" relations (sets of tuples)

- Two copies of the schema: Σ^{old} and Σ^{new};
- Delta relations: R^{+} (insertions) and R^{-} (deletions);
- Constraints: $\forall \bar{x}.(R^{old}(\bar{x}) \vee R^{+}(\bar{x})) \equiv (R^{new}(\bar{x}) \vee R^{-}(\bar{x}))$,
  $\forall \bar{x}.(R^{+}(\bar{x}) \wedge R^{-}(\bar{x})) \rightarrow \bot$

Update turned into *definability* question

Is A^{new} (or A^{+}, A^{-}) definable in terms of $A_i^{old} \in \mathsf{S}_\mathsf{A}^{old}$ (old access paths) and U_j^{+}, U_j^{-} (user updates) for every access path $A \in \mathsf{S}_\mathsf{A}$?

Unknown/Anonymous Values?

Example (Add a new Undergraduate student)

```
INSERT into undergrad values (1234, 'Wilma');
```

⇒ the request then needs to be translated to

```
INSERT into student values (0xFE1234, 1234, 'Wilma');
```

⇒ but where did `0xFE1234` came from? (definability issue!)

Unknown/Anonymous Values?

Example (Add a new Undergraduate student)

```
INSERT into undergrad values (1234, 'Wilma');
```

⇒ the request then needs to be translated to

```
    INSERT into student values (0xFE1234, 1234, 'Wilma');
```

⇒ but where did `0xFE1234` came from? (definability issue!)

Constant Complement: [Bancilhon, Spyratos: Update semantics of relational views. ACM Trans. Database Syst. 6(4), 1981.]

additional access paths that *provide* such values:

⇒ in our case `student-addr(id,adress)`
⇒ and where $\texttt{undergrad}^{+} = \{(1234, \mathit{Vilma})\}$

$$\texttt{student}^{+}(x_1, x_2, x_3) = \texttt{undergrad}^{+}(x_1, x_3) \wedge \texttt{student-addr}(x_2, x_1)$$

Unknown/Anonymous Values?

Example (Add a new Undergraduate student)

`INSERT into undergrad values (1234, 'Wilma');`

⇒ the request then needs to be translated to

`INSERT into student values (0xFE1234, 1234, 'Wilma');`

⇒ but where did `0xFE1234` came from? (definability issue!)

Constant Complement: [Bancilhon, Spyratos: Update semantics of relational views. ACM Trans. Database Syst. 6(4), 1981.]

additional access paths that *provide* such values:

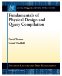
⇒ in our case `student-addr(id,adress)`

⇒ and where $\texttt{undergrad}^{+} = \{(1234, \mathit{Vilma})\}$

$$\texttt{student}^{+}(x_1, x_2, x_3) = \texttt{undergrad}^{+}(x_1, x_3) \land \texttt{student-addr}(x_2, x_1)$$

The additional access path(s) correspond to *space allocation*

. . . and cyclic dependencies are broken via *reification*.

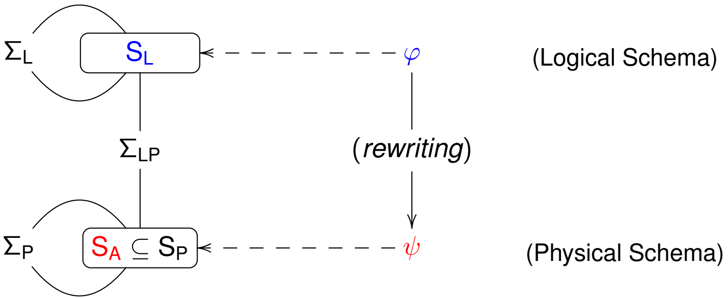
. . . more details and examples in .

HOW DOES IT ALL WORK?

The Plan

Definability and Rewriting

Queries	range-restricted FOL over S_L *definable* w.r.t. Σ and S_A
Ontology/Schema	range-restricted FOL
Data	CWA (complete information for S_A symbols)



Query Plans via Interpolation

IDEA #1: Plans as Formulas

Represent *query plans* as (annotated) range-restricted formulas ψ over S_A:

atomic formula	$\mapsto$	access path (`get-first`–`get-next` iterator)
conjunction	$\mapsto$	nested loops join
existential quantifier	$\mapsto$	projection (annotated w/duplicate info)
disjunction	$\mapsto$	concatenation
negation	$\mapsto$	simple complement

Query Plans via Interpolation

IDEA #1: Plans as Formulas

Represent *query plans* as (annotated) range-restricted formulas ψ over S_A:

atomic formula	$\mapsto$	access path (`get-first`–`get-next` iterator)
conjunction	$\mapsto$	nested loops join
existential quantifier	$\mapsto$	projection (annotated w/duplicate info)
disjunction	$\mapsto$	concatenation
negation	$\mapsto$	simple complement

$\Rightarrow$ reduces correctness of ψ to logical implication $\Sigma \models \varphi \leftrightarrow \psi$

Query Plans via Interpolation

IDEA #1: Plans as Formulas

Represent *query plans* as (annotated) range-restricted formulas ψ over S_A:

atomic formula	$\mapsto$	access path (`get-first`–`get-next` iterator)
conjunction	$\mapsto$	nested loops join
existential quantifier	$\mapsto$	projection (annotated w/duplicate info)
disjunction	$\mapsto$	concatenation
negation	$\mapsto$	simple complement

$\Rightarrow$ reduces correctness of ψ to logical implication $\Sigma \models \varphi \leftrightarrow \psi$

Non-logical (but necessary) Add-ons

1. Non-logical properties/operators
   - binding patterns
   - duplication of data and duplicate-preserving/eliminating projections
   - sortedness of data (with respect to the *iterator semantics*) and sorting
2. Cost model

Beth Definability and Craig Interpolation

IDEA #2: What Queries do we allow?

We only allow queries that have *the same answer* in every model of Σ

. . . for a fixed signature S_A (i.e., where the actual data is).

Beth Definability and Craig Interpolation

IDEA #2: What Queries do we allow?

We only allow queries that have *the same answer* in every model of Σ

. . . for a fixed signature S_A (i.e., where the actual data is).

How do we test for this?

φ is *Beth definable* [Beth'56] if

$$\Sigma \cup \Sigma' \models \varphi \rightarrow \varphi'$$

where Σ' (φ') is Σ (φ) in which symbols *NOT in* S_A are *primed*, respectively.

Beth Definability and Craig Interpolation

IDEA #2: What Queries do we allow?

We only allow queries that have *the same answer* in every model of Σ

…for a fixed signature S_A (i.e., where the actual data is).

How do we test for this?

φ is *Beth definable* [Beth'56] if

$$\Sigma \cup \Sigma' \models \varphi \to \varphi'$$

where Σ' (φ') is Σ (φ) in which symbols *NOT in* S_A are *primed*, respectively.

How do we find ψ?

If $\Sigma \cup \Sigma' \models \varphi \to \varphi'$ then there is ψ s.t. $\Sigma \cup \Sigma' \models \varphi \to \psi \to \varphi'$ with $\mathcal{L}(\psi) \subseteq \mathcal{L}(S_A)$.

… ψ is called the *Craig Interpolant* [Craig'57].

…we extract an *interpolant* ψ from a (TABLEAU) proof of $\Sigma \cup \Sigma' \models \varphi \to \varphi'$

Issues with TABLEAU

Dealing with the *subformula property* of Tableau

$\Rightarrow$ analytic tableau *explores* formulas *structurally*

$\Rightarrow$ (to large degree) the structure of interpolant depends on where access paths are present in queries/constraints.

Factoring *logical reasoning* from *plan enumeration*

$\Rightarrow$ backtracking tableau to get alternative plans: too slow, too few plans

Issues with TABLEAU

Dealing with the *subformula property* of Tableau

$\Rightarrow$ analytic tableau *explores* formulas *structurally*

$\Rightarrow$ (to large degree) the structure of interpolant depends on where access paths are present in queries/constraints.

IDEA #3:

Separate *general constraints* from *physical rules* in the formulation of the definability question (and the subsequent interpolant extraction):

$$\Sigma^L \cup \Sigma^R \cup \Sigma^{LR} \models \varphi^L \rightarrow \varphi^R \text{ where } \Sigma^{LR} = \{\forall \bar{x}.P^L \leftrightarrow P \leftrightarrow P^R \mid P \in \mathsf{S_A}\}$$

Factoring *logical reasoning* from *plan enumeration*

$\Rightarrow$ backtracking tableau to get alternative plans: too slow, too few plans

IDEA #4:

Define *conditional* tableau exploration (using general constraints) and separate it from plan generation (using physical rules)

Conditional Tableau and Closing Sets

1. Byte code generation for `q/2`

```
q(x,y) <-> ex(z,table(x,x,z) and table(z,y,y)
                              and not table(x,x,x))
```

2. Split Tableau Construction

```
L { -p0basetable(sl19:7,sl14:3,sl0:2,sl0:2) }
L { -p0basetable(sl19:5,sl0:1,sl0:1,sl14:3) }
L { +p0basetable(sr19:8,sl0:1,sl0:1,sl0:1) }
R { -p0basetable(sr19:8,sl0:1,sl0:1,sl0:1),
    +p0basetable(sl19:7,sl14:3,sl0:2,sl0:2),
    +p0basetable(sl19:5,sl0:1,sl0:1,sl14:3) }
```

3. Cost-based Optimization (A*)
4. C code Generation (+ compilation/linking w/runtime library)

[Hudek, Toman, Weddell: On Enumerating Query Plans Using Analytic Tableau. TABLEAUX 2015.]
[Toman, Weddell: An Interpolation-based Compiler and Optimizer for Relational Queries (System design Report). IWIL-LPAR 2017.]

Conditional Tableau: Result

```
query(q,2,0,[var(0,0,1,int),var(0,0,2,int)]) <->
  ex(var(0,14,3),
    ex(var(0,19,5),
      ex(var(0,19,7),
        and (
          and (
            p0basetable(var(0,19,7),var(0,14,3),
                        var(0,0,2),var(0,0,2))
            p0basetable(var(0,19,5),var(0,0,1),
                        var(0,0,1),var(0,14,3))
          )
          not (
            ex(var(1,19,8),
              p0basetable(var(1,19,8),var(0,0,1),
                          var(0,0,1),var(0,0,1))
            )
          ) ) ) ) )
```

Postprocessing: Duplicate Elimination Elimination

IDEA:

Separate the projection operation ($\exists \bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Postprocessing: Duplicate Elimination Elimination

IDEA:

Separate the projection operation ($\exists\bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Use the following rewrites to eliminate/minimize the use of $\{\cdot\}$:

$$
\begin{aligned}
Q[\{R(x_1,\ldots,x_k)\}] &\leftrightarrow Q[R(x_1,\ldots,x_k)] \\
Q[\{Q_1 \wedge Q_2\}] &\leftrightarrow Q[\{Q_1\} \wedge \{Q_2\}] \\
Q[\{\neg Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\neg\{Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\{Q_1 \vee Q_2\}] &\leftrightarrow Q[\{Q_1\} \vee \{Q_2\}] \quad \text{if } \Sigma \cup \{Q[]\} \models Q_1 \wedge Q_2 \to \bot \\
Q[\{\exists x.Q_1\}] &\leftrightarrow Q[\exists x.\{Q_1\}] \quad \text{if} \\
& \Sigma \cup \{Q[] \wedge (Q_1)[y_1/x] \wedge (Q_1)[y_2/x] \models y_1 \approx y_2
\end{aligned}
$$

Postprocessing: Duplicate Elimination Elimination

IDEA:

Separate the projection operation ($\exists\bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Use the following rewrites to eliminate/minimize the use of $\{\cdot\}$:

$$
\begin{aligned}
Q[\{R(x_1,\ldots,x_k)\}] &\leftrightarrow Q[R(x_1,\ldots,x_k)] \\
Q[\{Q_1 \wedge Q_2\}] &\leftrightarrow Q[\{Q_1\} \wedge \{Q_2\}] \\
Q[\{\neg Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\neg\{Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\{Q_1 \vee Q_2\}] &\leftrightarrow Q[\{Q_1\} \vee \{Q_2\}] \quad \text{if } \Sigma \cup \{Q[]\} \models Q_1 \wedge Q_2 \rightarrow \bot \\
Q[\{\exists x.Q_1\}] &\leftrightarrow Q[\exists x.\{Q_1\}] \quad \text{if} \\
&\qquad \Sigma \cup \{Q[] \wedge (Q_1)[y_1/x] \wedge (Q_1)[y_2/x] \models y_1 \approx y_2
\end{aligned}
$$

… reasoning abstracted: a DL $\mathcal{CFD}^{\forall -}_{nc}$ (a PTIME fragment)

[Toman, Weddell: Using Feature-Based Description Logics to avoid Duplicate Elimination in Object-Relational Query Languages. Künstliche Intell. 34(3): 2020]

Summary

Take Home

While in theory *interpolation* essentially solves the *query rewriting over FO schemas/views* problem, the devil is (as usual) in the details.

[Borgida, de Bruijn, Franconi, Seylan, Straccia, Toman, Weddell: On Finding Query Rewritings under Expressive Constraints. SEBD 2010: 426-437

. . . but an (almost) working system only this year.

1. FO tableau based interpolation algorithm
   - ⇒ enumeration of plans factored from of tableau reasoning
   - ⇒ extra-logical binding patterns and cost model
2. Post processing (using $\mathcal{CFDI}_{\text{nc}}$ approximation)
   - ⇒ duplicate elimination elimination
   - ⇒ cut insertion
3. Run time
   - ⇒ library of common data/legacy structures+schema constraints
   - ⇒ finger data structures to simulate merge joins et al.

Research Directions and Open Issues

1. Dealing with ordered data? (merge-joins etc.: we have a partial solution)
2. Decidable schema languages (decidable interpolation problem)?
3. More powerful schema languages (inductive types, etc.)?
4. Beyond FO Queries/Views (e.g., count/sum aggregates)?
5. Coding extra-logical bits (e.g., binding patterns, postprocessing, etc.) in the schema itself?
6. Standard Designs (a plan can always be found as in SQL)?
7. Explanation(s) of non-definability?
8. Fine(r)-grained updates?
9. ...

. . . and, as always, performance, performance, performance!