KRDB Research Centre
for Knowledge and Data

Ontologies with Data
or:
the death of the AI dream

Enrico Franconi

Free University of Bozen-Bolzano, Italy

http://www.inf.unibz.it/∼franconi

São Paulo, September 2015

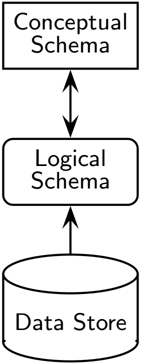
Summary

- Defining the problem of dealing with data together with an ontology
- The problem is not trivial
- A novel framework to solve the problem
- The *purpose* of ontologies...

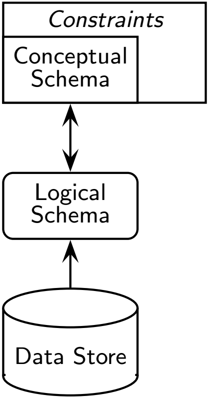
Ontologies and Constraints

- An ontology is a formal conceptualisation of the world: a conceptual schema.
- An ontology specifies a set of constraints, which declare what should necessarily hold in any possible world.
- Any possible world should conform to the constraints expressed by the ontology.
- Given an ontology, a *legal world description* (or legal database instance) is a finite possible world satisfying the constraints.

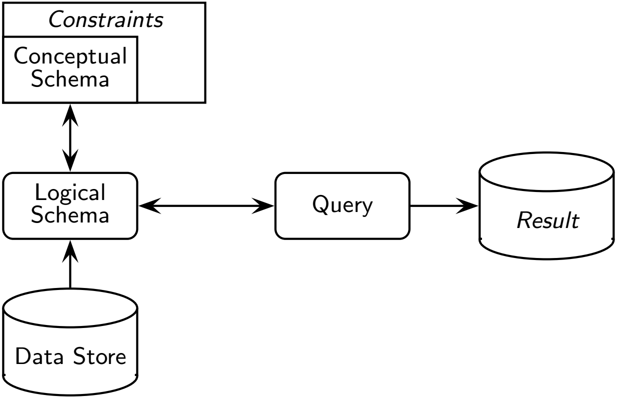
The role of a Conceptual Schema



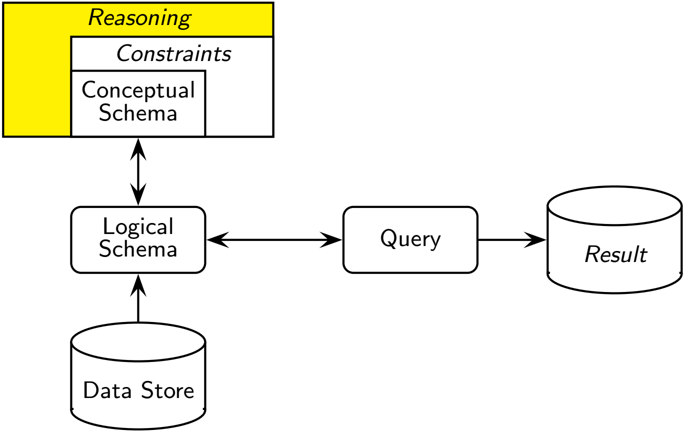
The role of a Conceptual Schema



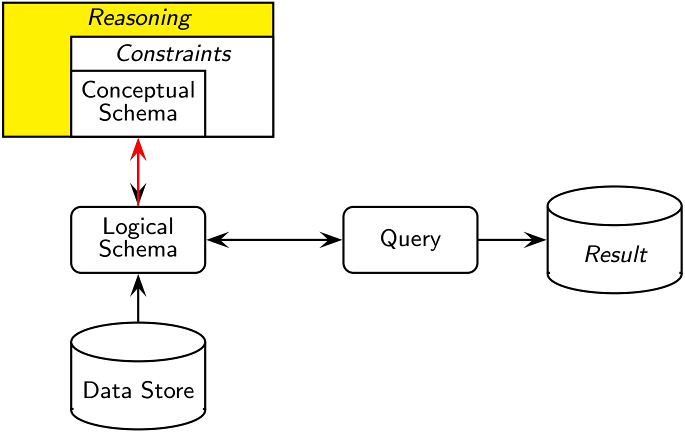
The role of a Conceptual Schema



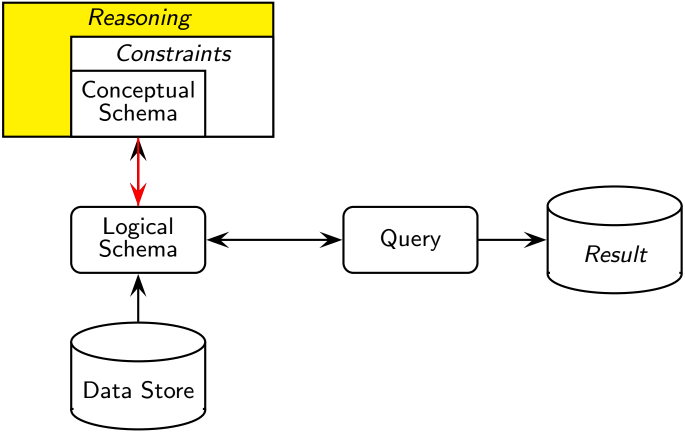
The role of a Conceptual Schema



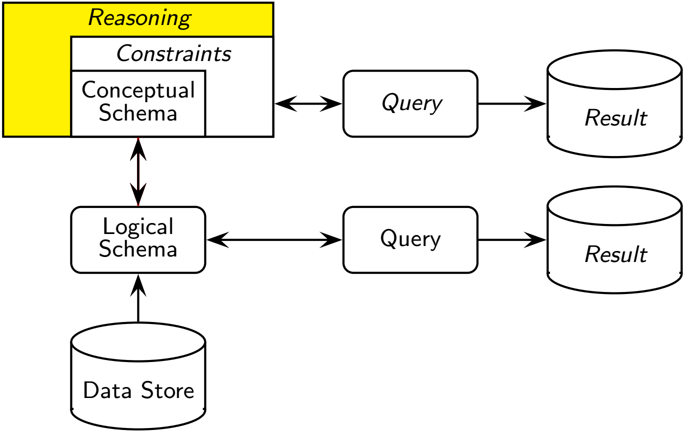
The role of a Conceptual Schema



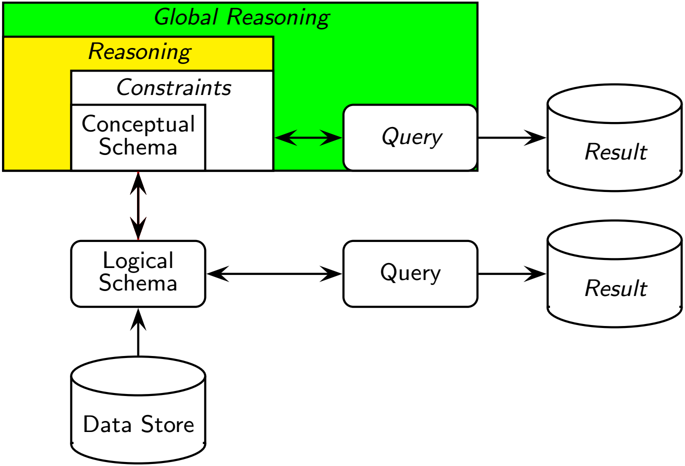
The role of a Conceptual Schema



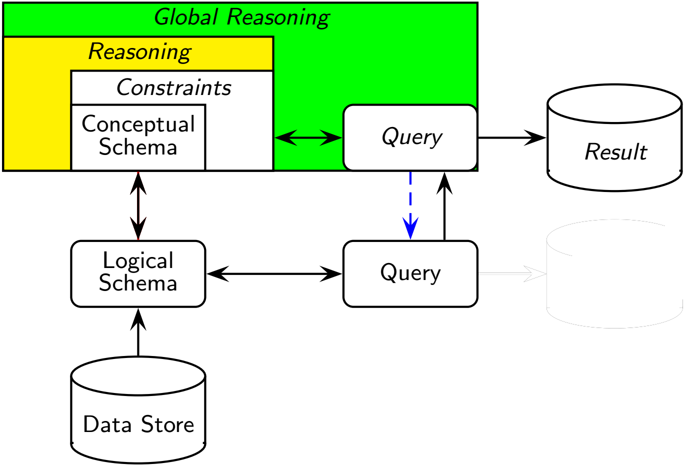
The role of a Conceptual Schema



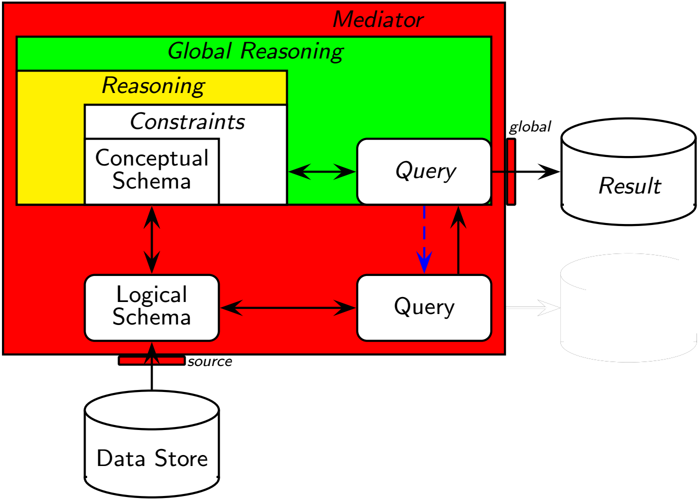
The role of a Conceptual Schema



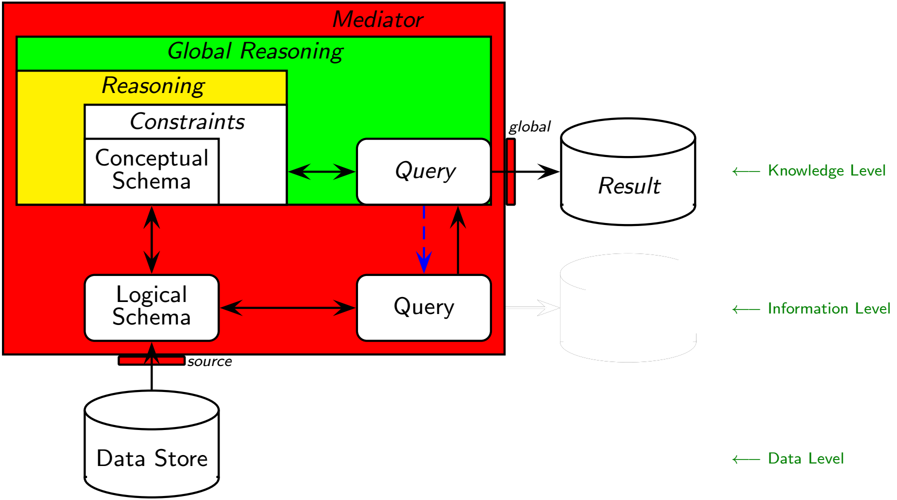
The role of a Conceptual Schema



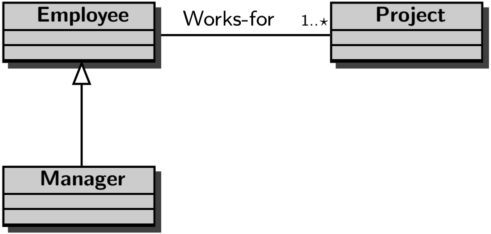
The role of a Conceptual Schema



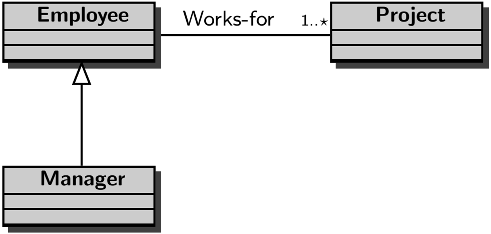
The role of a Conceptual Schema



Queries via Conceptual Schemas: the classical DB case

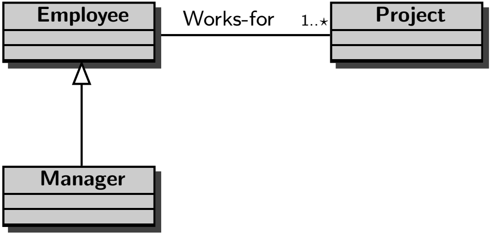


Queries via Conceptual Schemas: the classical DB case



Employee = { John, Mary, Paul }
Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }

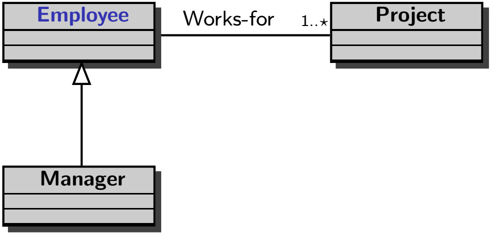
Queries via Conceptual Schemas: the classical DB case



```
Employee = { John, Mary, Paul }
Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }
```

```
Q(X) :- Manager(X), Works-for(X,Y), Project(Y)
⟹ { John }
```

Queries via Conceptual Schemas: the DBox case

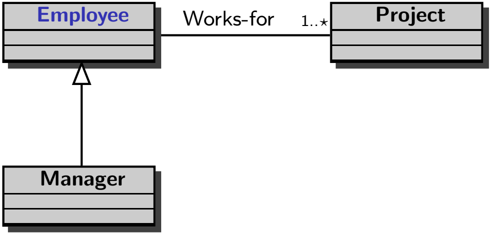


Manager = { John, Paul }

Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }

Project = { Prj-A, Prj-B }

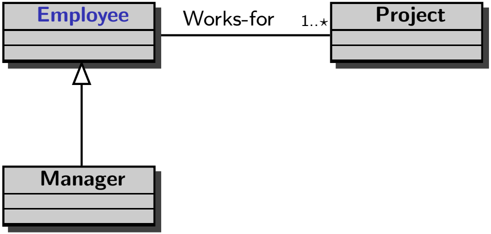
Queries via Conceptual Schemas: the DBox case



Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }

Q(X) :- Employee(X)

Queries via Conceptual Schemas: the DBox case



Manager = { John, Paul }

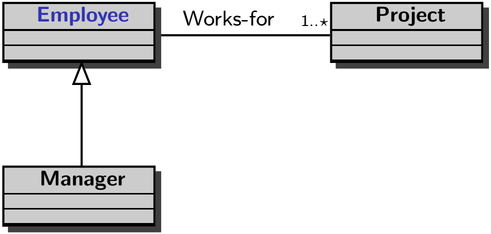
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }

Project = { Prj-A, Prj-B }

```
Q(X) :- Employee(X)
```

$\Longrightarrow$ { John, Paul, Mary }

Queries via Conceptual Schemas: the DBox case

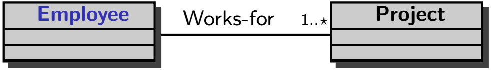


Manager = { John, Paul }
Works-for = { ⟨John,Prj-A⟩, ⟨Mary,Prj-B⟩ }
Project = { Prj-A, Prj-B }

Q(X) :- Employee(X)
⟹ { John, Paul, Mary }

⟹ Q'(X) :- Manager(X) ∪ Works-for(X,Y)

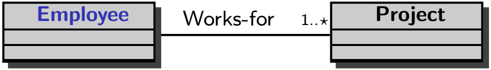
Queries via Conceptual Schemas: the ABox case



$\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle, \langle \texttt{Mary}, \texttt{Prj-A} \rangle \}$

$\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

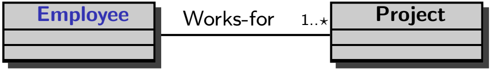
Queries via Conceptual Schemas: the ABox case



$\texttt{Works-for} \supseteq \{\ \langle\texttt{John},\texttt{Prj-A}\rangle,\ \langle\texttt{Mary},\texttt{Prj-A}\rangle\ \}$
$\texttt{Project} \supseteq \{\ \texttt{Prj-A},\ \texttt{Prj-B}\ \}$

`Q(X) :- Works-for(Y,X)`

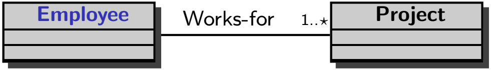
Queries via Conceptual Schemas: the ABox case



Works-for $\supseteq$ { ⟨John,Prj-A⟩, ⟨Mary,Prj-A⟩ }
Project $\supseteq$ { Prj-A, Prj-B }

Q(X) :- Works-for(Y,X)
$\Longrightarrow$ { Prj-A, Prj-B }

Queries via Conceptual Schemas: the ABox case



$\texttt{Works-for} \supseteq \{\ \langle\texttt{John},\texttt{Prj-A}\rangle,\ \langle\texttt{Mary},\texttt{Prj-A}\rangle\ \}$

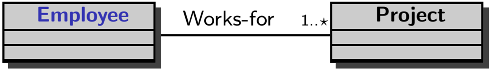
$\texttt{Project} \supseteq \{\ \texttt{Prj-A},\ \texttt{Prj-B}\ \}$

```
Q(X) :- Works-for(Y,X)
```

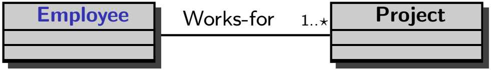
$\Longrightarrow \{\ \texttt{Prj-A},\ \texttt{Prj-B}\ \}$

$\Longrightarrow$ `Q'(X) :- Project(X)` $\cup$ `Works-for(Y,X)`

A DBox is stronger than an ABox



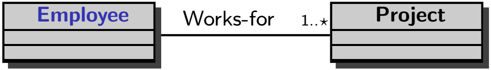
A DBox is stronger than an ABox



ABox:

$\texttt{Works-for} \supseteq \{ \langle \texttt{John},\texttt{Prj-A} \rangle, \langle \texttt{Mary},\texttt{Prj-A} \rangle \}$
$\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

A DBox is stronger than an ABox



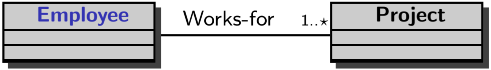
ABox:

$\texttt{Works-for} \supseteq \{\ \langle \texttt{John},\texttt{Prj-A}\rangle,\ \langle \texttt{Mary},\texttt{Prj-A}\rangle\ \}$
$\texttt{Project} \supseteq \{\ \texttt{Prj-A},\ \texttt{Prj-B}\ \}$

DBox:

$\texttt{Works-for} = \{\ \langle \texttt{John},\texttt{Prj-A}\rangle,\ \langle \texttt{Mary},\texttt{Prj-A}\rangle\ \}$
$\texttt{Project} = \{\ \texttt{Prj-A},\ \texttt{Prj-B}\ \}$

A DBox is stronger than an ABox



ABox:

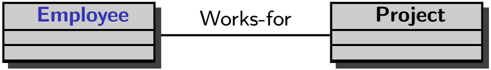
$\mathtt{Works\text{-}for} \supseteq \{ \langle \mathtt{John}, \mathtt{Prj\text{-}A} \rangle, \langle \mathtt{Mary}, \mathtt{Prj\text{-}A} \rangle \}$
$\mathtt{Project} \supseteq \{ \mathtt{Prj\text{-}A}, \mathtt{Prj\text{-}B} \}$

DBox:

$\mathtt{Works\text{-}for} = \{ \langle \mathtt{John}, \mathtt{Prj\text{-}A} \rangle, \langle \mathtt{Mary}, \mathtt{Prj\text{-}A} \rangle \}$
$\mathtt{Project} = \{ \mathtt{Prj\text{-}A}, \mathtt{Prj\text{-}B} \}$

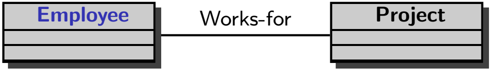
$\Longrightarrow$ INCONSISTENT

An ABox is not faithful to the DB



- Additional constraint as a standard view over the data:
  $\texttt{Bad-Project} = \texttt{Project} \setminus \pi_2\texttt{Works-for}$
  $\forall\texttt{x}.\ \texttt{Bad-Project(x)} \leftrightarrow \texttt{Project(x)} \land \neg\exists \texttt{y}.\texttt{Works-for(y,x)}$
  $\texttt{Bad-Project} = \texttt{Project} \sqcap \neg\exists \texttt{Works-for}^{-}.\top$

An ABox is not faithful to the DB



- Additional constraint as a standard view over the data:

  $\texttt{Bad-Project} = \texttt{Project} \setminus \pi_2\texttt{Works-for}$

  $\forall \texttt{x}.\ \texttt{Bad-Project(x)} \leftrightarrow \texttt{Project(x)} \land \neg \exists \texttt{y.Works-for(y,x)}$

  $\texttt{Bad-Project} = \texttt{Project} \sqcap \neg \exists \texttt{Works-for}^{-}.\top$

- DBox:

  $\texttt{Works-for} = \{ \langle \texttt{John},\texttt{Prj-A} \rangle, \langle \texttt{Mary},\texttt{Prj-A} \rangle \}$

  $\texttt{Project} = \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

- `Q(X) :- Bad-Project(X)`

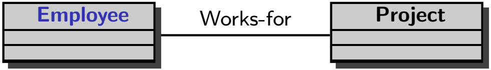
- ABox:

  $\texttt{Works-for} \supseteq \{ \langle \texttt{John},\texttt{Prj-A} \rangle, \langle \texttt{Mary},\texttt{Prj-A} \rangle \}$

  $\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

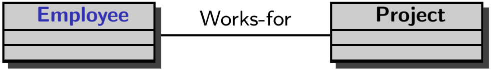
- `Q(X) :- Bad-Project(X)`

An ABox is not faithful to the DB



- Additional constraint as a standard view over the data:
  $\text{Bad-Project} = \text{Project} \setminus \pi_2\text{Works-for}$
  $\forall x.\ \text{Bad-Project}(x) \leftrightarrow \text{Project}(x) \wedge \neg\exists y.\text{Works-for}(y,x)$
  $\text{Bad-Project} = \text{Project} \sqcap \neg\exists\text{Works-for}^{-}.\top$
- DBox:
  $\text{Works-for} = \{\ \langle\text{John},\text{Prj-A}\rangle,\ \langle\text{Mary},\text{Prj-A}\rangle\ \}$
  $\text{Project} = \{\ \text{Prj-A},\ \text{Prj-B}\ \}$
- Q(X) :- Bad-Project(X)
  $\Longrightarrow \{\ \text{Prj-B}\ \}$
- ABox:
  $\text{Works-for} \supseteq \{\ \langle\text{John},\text{Prj-A}\rangle,\ \langle\text{Mary},\text{Prj-A}\rangle\ \}$
  $\text{Project} \supseteq \{\ \text{Prj-A},\ \text{Prj-B}\ \}$
- Q(X) :- Bad-Project(X)

An ABox is not faithful to the DB



- Additional constraint as a standard view over the data:

  $\text{Bad-Project} = \text{Project} \setminus \pi_2\text{Works-for}$

  $\forall x.\ \text{Bad-Project}(x) \leftrightarrow \text{Project}(x) \land \neg\exists y.\text{Works-for}(y,x)$

  $\text{Bad-Project} = \text{Project} \sqcap \neg\exists\text{Works-for}^{-}.\top$
- DBox:

  $\text{Works-for} = \{ \langle \text{John},\text{Prj-A} \rangle, \langle \text{Mary},\text{Prj-A} \rangle \}$

  $\text{Project} = \{ \text{Prj-A}, \text{Prj-B} \}$
- Q(X) :- Bad-Project(X)

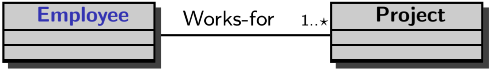
  $\Longrightarrow \{ \text{Prj-B} \}$
- ABox:

  $\text{Works-for} \supseteq \{ \langle \text{John},\text{Prj-A} \rangle, \langle \text{Mary},\text{Prj-A} \rangle \}$

  $\text{Project} \supseteq \{ \text{Prj-A}, \text{Prj-B} \}$
- Q(X) :- Bad-Project(X)

  $\Longrightarrow \{ \}$ *does not scale down to standard DB answer!*

An ABox doesn't preserve compositionality

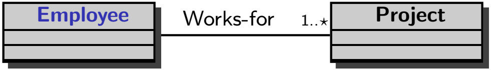


- ABox:

  $\texttt{Works-for} \supseteq \{\ \langle \texttt{John}, \texttt{Prj-A} \rangle\ \}$

  $\texttt{Project} \supseteq \{\ \texttt{Prj-A},\ \texttt{Prj-B}\ \}$

An ABox doesn't preserve compositionality



- ABox:

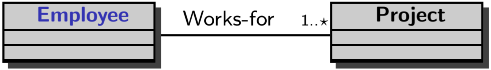
  $\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle \}$

  $\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

- Query as a standard view over the data:

  `Q(X) :- Works-for(Y,X)` $\texttt{Q} = \pi_2 \texttt{Works-for}$

An ABox doesn't preserve compositionality



- ABox:

  $\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle \}$

  $\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

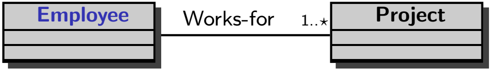
- Query as a standard view over the data:

  `Q(X) :- Works-for(Y,X)` $\texttt{Q} = \pi_2 \texttt{Works-for}$

  - $\texttt{Q} = \texttt{EVAL}(\pi_2 \texttt{Works-for})$

  - $\texttt{Q} = \pi_2(\texttt{EVAL}(\texttt{Works-for}))$

An ABox doesn't preserve compositionality



- ABox:

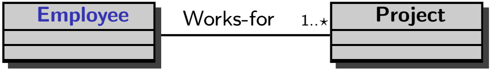
  $\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle \}$
  $\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

- Query as a standard view over the data:

  `Q(X) :- Works-for(Y,X)` $\quad \texttt{Q} = \pi_2 \texttt{Works-for}$

  - $\texttt{Q} = \texttt{EVAL}(\pi_2 \texttt{Works-for})$
    $\Longrightarrow \{ \texttt{Prj-A}, \texttt{Prj-B} \}$
  - $\texttt{Q} = \pi_2(\texttt{EVAL}(\texttt{Works-for}))$

An ABox doesn't preserve compositionality



- ABox:

  $\texttt{Works-for} \supseteq \{ \langle \texttt{John}, \texttt{Prj-A} \rangle \}$
  $\texttt{Project} \supseteq \{ \texttt{Prj-A}, \texttt{Prj-B} \}$

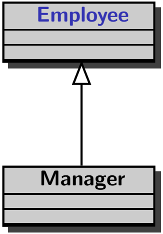
- Query as a standard view over the data:

  $\texttt{Q(X) :- Works-for(Y,X)} \qquad \texttt{Q} = \pi_2 \texttt{Works-for}$

  - $\texttt{Q} = \texttt{EVAL}(\pi_2 \texttt{Works-for})$
    $\Longrightarrow \{ \texttt{Prj-A}, \texttt{Prj-B} \}$
  - $\texttt{Q} = \pi_2(\texttt{EVAL}(\texttt{Works-for}))$
    $\Longrightarrow \{ \texttt{Prj-A} \}$

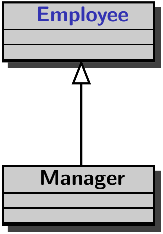
Queries are not compositional with an ABox

Queries with an incomplete answer



`Manager = { John, Paul }`

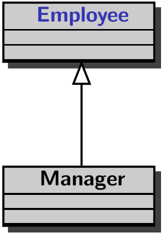
Queries with an incomplete answer



```
Manager = { John, Paul }

Q(X) :- Employee(X)
```

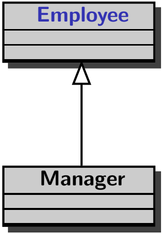
Queries with an incomplete answer



`Manager = { John, Paul }`

`Q(X) :- Employee(X)` $\Longrightarrow$ `{ John, Paul }`

Queries with an incomplete answer

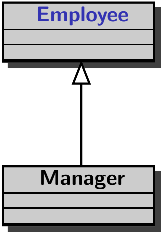


Manager = { John, Paul }

Q(X) :- Employee(X) ⟹ { John, Paul }

So are Managers and Employees the same?

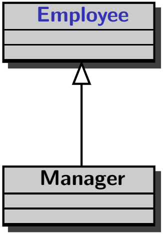
Queries with an incomplete answer



Manager = { John, Paul }

Q(X) :- Employee(X) $\Longrightarrow$ { John, Paul}

Queries with an incomplete answer



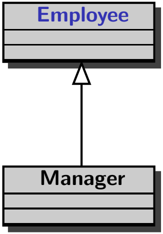
Manager = { John, Paul }

Q(X) :- Employee(X) ⟹ { John, Paul}

Q^1 :- ¬Manager(George)

Q^2 :- ¬Employee(George)

Queries with an incomplete answer



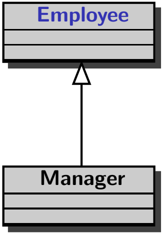
Manager = { John, Paul }

Q(X) :- Employee(X) $\Longrightarrow$ { John, Paul}

$\mathtt{Q}^1$:- ¬Manager(George) $\Longrightarrow$ TRUE

$\mathtt{Q}^2$:- ¬Employee(George)

Queries with an incomplete answer



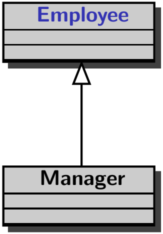
Manager = { John, Paul }

Q(X) :- Employee(X) $\Longrightarrow$ { John, Paul}

Q^1 :- ¬Manager(George) $\Longrightarrow$ TRUE

Q^2 :- ¬Employee(George) $\Longrightarrow$ FALSE

Queries with an incomplete answer



Manager = { John, Paul }

Q(X) :- Employee(X) $\Longrightarrow$ { John, Paul}

Q^1 :- ¬Manager(George) $\Longrightarrow$ TRUE

Q^2 :- ¬Employee(George) $\Longrightarrow$ FALSE

The result of the query can not be stored (as a view)
$\Longrightarrow$ *queries with an incomplete answer are not compositional*

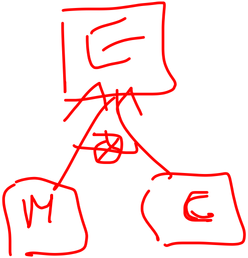
Determinacy - or implicit definability

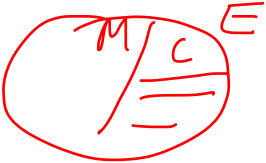
Given a conceptual schema and a DBox, a query which only gives complete answers is called implicitly definable or determined by the DBox.

If a query is implicitly definable, then its evaluation depends only on the database, and vice-versa; therefore implicitly definable queries characterise exactly views.

Beth (1953) constructively proved that there is a way to check whether an arbitrary query is determined by a DBox given a conceptual schema.

Example: implicit definability





$M = \{ j \}$

$E = \{ j, m \}$

C

Rewriting - or explicit definability

If a query is implicitly definable, Beth (1953) constructively proved that possible to rewrite the query using only the vocabulary of the DBox (and therefore independent on the conceptual schema, since the extension of the DBox is fixed).

This is its explicit definition.

Example: explicit definability

$$C \equiv E \setminus M$$

Task: query rewriting

For simplicity, let's focus on atomic queries.
Our goals are

1. to check whether the answers to a given query under a conceptual schema are *solely* determined by the DBox and, if so,
2. to find an equivalent (first-order/SQL) rewriting of the query in terms of the DBox predicates to allow the use of standard database technology for answering the query.
3. Indeed, this means we benefit from the low computational complexity – logarithmic space in the size of the data – of answering first-order queries on relational databases.
4. It is possible to pre-compute all the rewritings of all the determined predicates as relational views, and to allow arbitrary SQL queries on top of them.

Problem: “unsafe” rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$

DBox: $\texttt{Female} = \{\texttt{mary}\}$

Query: $\texttt{Q :- } \neg\exists x.\texttt{Male}(x)$

Rewriting of the query: $\forall x.\texttt{Female}(x)$

Problem: “unsafe” rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$

DBox: $\texttt{Female} = \{\texttt{mary}\}$

Query: $\texttt{Q :-} \neg\exists x.\texttt{Male}(x)$

Rewriting of the query: $\forall x.\texttt{Female}(x)$

- Answer to the query: NO

Problem: “unsafe” rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$

DBox: $\texttt{Female} = \{\texttt{mary}\}$

Query: $\texttt{Q :-} \neg\exists x.\texttt{Male}(x)$

Rewriting of the query: $\forall x.\texttt{Female}(x)$

- Answer to the query: NO
- But if we fix the domain to include *only* the individual `mary` (namely, the active domain), then the answer would be YES

Problem: “unsafe” rewritings

Conceptual schema: $\texttt{Male} \doteq \neg\texttt{Female}$
DBox: $\texttt{Female} = \{\texttt{mary}\}$
Query: $\texttt{Q :-}\ \neg\exists x.\texttt{Male}(x)$

Rewriting of the query: $\forall x.\texttt{Female}(x)$

- Answer to the query: NO
- But if we fix the domain to include *only* the individual `mary` (namely, the active domain), then the answer would be YES

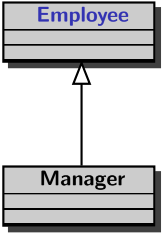
The query is not domain independent.

Domain independent rewritings

Theorem (- et al.)

If a conceptual schema is domain independent, then the rewriting of determined domain independent queries is also domain independent.

Abduction

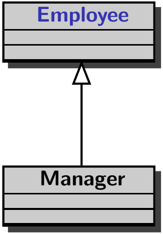


```
Manager = { John, Paul }
```

```
Q(X) :- Employee(X)
```

Rewriting of `Q`:

Abduction

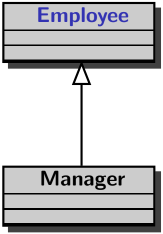


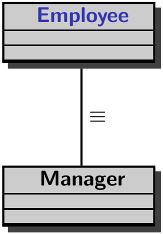
`Manager = { John, Paul }`

`Q(X) :- Employee(X)`

Rewriting of `Q`: impossible.

Abduction





`Manager = { John, Paul }`

`Q(X) :- Employee(X)`

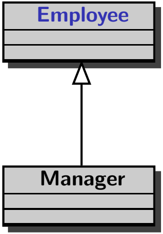
Rewriting of `Q`: impossible.

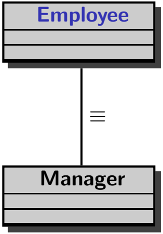
`Manager = { John, Paul }`

`Q(X) :- Employee(X)`

Rewriting of `Q`:

Abduction





Manager = { John, Paul }

Q(X) :- Employee(X)

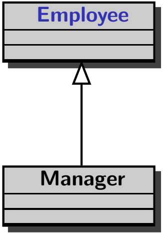
Rewriting of Q: impossible.

Manager = { John, Paul }

Q(X) :- Employee(X)

Rewriting of Q: Manager(X)

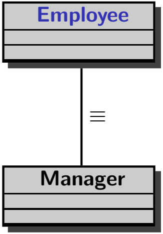
Abduction



Manager = { John, Paul }

Q(X) :- Employee(X)

Rewriting of Q: impossible.



Manager = { John, Paul }

Q(X) :- Employee(X)

Rewriting of Q: Manager(X)

The abduction is characterised by being the *least committing* extension of the ontology such that the query becomes implicitly definable.

Philosophical issue

with data

Do reusable ontologies exist?

NO

Conclusions

Conclusions

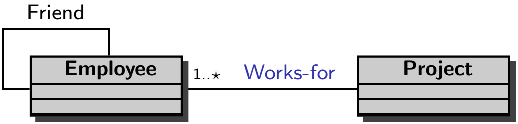
Do you want to exploit conceptual schema knowledge
(i.e., constraints or an ontology)
in your data intensive application?

Conclusions

Do you want to exploit conceptual schema knowledge
(i.e., constraints or an ontology)
in your data intensive application?

Pay attention!

Complexity of DBoxes

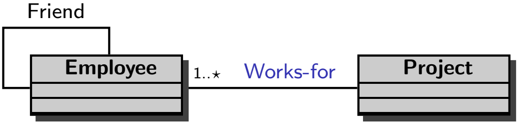


- DBox:

  $\texttt{Friend} = \{\langle \texttt{John}, \texttt{Mary} \rangle, \ldots\}$; $\texttt{Employee} = \{\texttt{John}, \texttt{Mary}, \ldots\}$

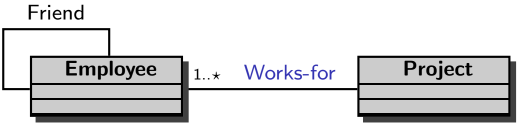
  $\texttt{Project} = \{ \texttt{Prj-A}, \texttt{Prj-B}, \texttt{Prj-C} \}$

Complexity of DBoxes



- DBox:
  Friend = {⟨John,Mary⟩,...}; Employee = {John,Mary,...}
  Project = { Prj-A, Prj-B, Prj-C }
- Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).
  Is it unavoidable that there are two friends working for the same project?

Complexity of DBoxes

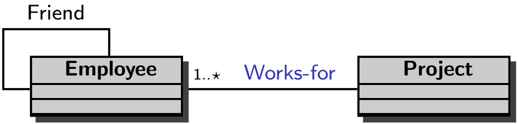


- DBox:

  $\texttt{Friend} = \{\langle \texttt{John},\texttt{Mary}\rangle,\ldots\}$; $\texttt{Employee} = \{\texttt{John},\texttt{Mary},\ldots\}$
  $\texttt{Project} = \{\ \texttt{Prj-A},\ \texttt{Prj-B},\ \texttt{Prj-C}\ \}$
- `Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).`

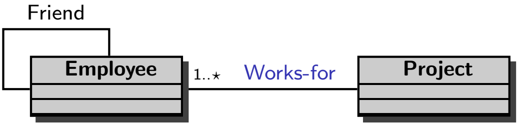
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.

Complexity of DBoxes



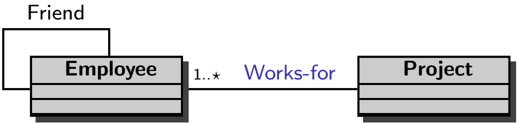
- DBox:
  Friend = {⟨John,Mary⟩,...}; Employee = {John,Mary,...}
  Project = { Prj-A, Prj-B, Prj-C }
- Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.
  - NO: there is at least a legal database instance in which no two friends work for the same project.

Complexity of DBoxes



- DBox:
  `Friend` = {⟨`John`,`Mary`⟩,...}; `Employee` = {`John`,`Mary`,...}
  `Project` = { `Prj-A`, `Prj-B`, `Prj-C` }
- `Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).`
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.
  - NO: there is at least a legal database instance in which no two friends work for the same project.
  - With *ABox semantics* the answer is always NO, since there is at least a legal database instance with *enough* distinct projects so that no two friends work for the same project.

Complexity of DBoxes



- DBox:
  `Friend` = {⟨`John`,`Mary`⟩,...}; `Employee` = {`John`,`Mary`,...}
  `Project` = { `Prj-A`, `Prj-B`, `Prj-C` }
- `Q :- Works-for(E1,P), Works-for(E2,P), Friend(E1,E2).`
  Is it unavoidable that there are two friends working for the same project?
  - YES: in any legal database instance, there are at least two friends working for the same project.
  - NO: there is at least a legal database instance in which no two friends work for the same project.
  - With *ABox semantics* the answer is always NO, since there is at least a legal database instance with *enough* distinct projects so that no two friends work for the same project.

Query answering with DBoxes has exponential complexity (3-colorability of maps), and it is harder than with ABoxes

Encoding a DBox in a $\mathcal{DL}$ with nominals

$$\mathcal{DL} \cup DBox \quad \simeq \quad \mathcal{DL} \cup nominals$$

($\Rightarrow$) Add completion and closure axioms:

- for every assertions involving A in the DBox $A(a_1), \ldots, A(a_n)$, add the axiom $A \equiv (\{a_1\} \sqcup \cdots \sqcup \{a_n\})$
- for every assertions involving R and a in the DBox $R(a, b_1), \ldots, R(a, b_n)$, add the axioms $\{a\} \sqsubseteq \exists R.\{b_1\} \sqcap \cdots \sqcap \exists R.\{b_n\}$ and $\{a\} \sqsubseteq \forall R.(\{b_1\} \sqcup \cdots \sqcup \{b_n\})$
- $\ldots$

($\Leftarrow$) For every nominal $\{a\}$, add to the DBox the assertion $A(a)$, where A is a new DBox concept name, and replace the occurrences of $\{a\}$ with A.