Data Integration

Local as View: View-based Query Processing

Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica

Università di Roma “La Sapienza”

Seminari di Ingnegneria del Software: Integrazione di Dati e Servizi
Corso di Laurea Specialistica in Ingegneria Informatica
Univesità degli Studi di Roma “La Sapenza”
A.A. 2006/07

View-based query processing

Basic Idea: Computing the answer to a query based on a set of views, rather than on the raw data in the database.

Significance: Relevant problem in query optimization, query answering with incomplete information, data warehousing, data integration, etc.

Two approaches:

- View-based query rewriting (indirect)
- View-based query answering (direct)

Part 1

View-based query processing: Computing the answer to a query based on a set of views, rather than on the raw data in the database.

- View-based query rewriting (indirect)
- View-based query answering (direct)
- Relationship between query answering and query rewriting

We study view-based query processing within the **relational data model**, focusing on **conjunctive queries** (CQs).

LAV: formal framework

The integrated database (DB) DB is simply a set of structures (relations, in the relational model), one for each symbol in an alphabet $\mathcal{A}^{\mathcal{G}}$

- the structure of the global view is specified in the schema language $\mathcal{L}_{\mathcal{G}}$ over $\mathcal{A}^{\mathcal{G}}$
- each source structure is modeled as a view over the global view, expressed in the view language $\mathcal{L}_{\mathcal{V}}$ over $\mathcal{A}^{\mathcal{G}}$
- queries are issued over the global view, and are expressed in the query language $\mathcal{L}_{\mathcal{Q}}$ over $\mathcal{A}^{\mathcal{G}}$

LAV: formal framework

- The global view $\mathcal{G}$ is specified as a set of constraints in $\mathcal{L}_{\mathcal{G}}$, and associated to each source structure we have a set (its extension)
- We have a view V_i for each source structure, with
  - extension $ext(V_i)$,
  - definition $def(V_i)$, i.e., a query $V_i(\vec{\mathbf{x}})$:- $v_i(\vec{\mathbf{x}}, \vec{\mathbf{y}})$, where $v_i(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ is expressed in the language $\mathcal{L}_{\mathcal{V}}$ over $\mathcal{A}^{\mathcal{G}}$
  - assumption $as(V_i)$, i.e., how to interpret $ext(V_i)$ wrt the tuples satisfying V_i
- A query Q is expressed in the language $\mathcal{L}_{\mathcal{Q}}$ over $\mathcal{A}^{\mathcal{G}}$. If DB satisfies $\mathcal{G}$, $ans(Q, DB)$ is the set of objects in DB that satisfy Q

LAV: formal framework

The specification $as(V_i)$ determines how accurate is the extension of the view with respect to the specification $def(V_i)$

- **Sound Views**: a database DB is *coherent with the sound view* V_i, if $ext(V_i) \subseteq ans(def(V_i), DB)$
- **Complete Views**: a database DB is *coherent with the complete view* V_i, if $ext(V_i) \supseteq ans(def(V_i), DB)$
- **Exact Views**: a database DB is *coherent with the exact view* V_i, if $ext(V_i) = ans(def(V_i), DB)$

LAV: formal framework

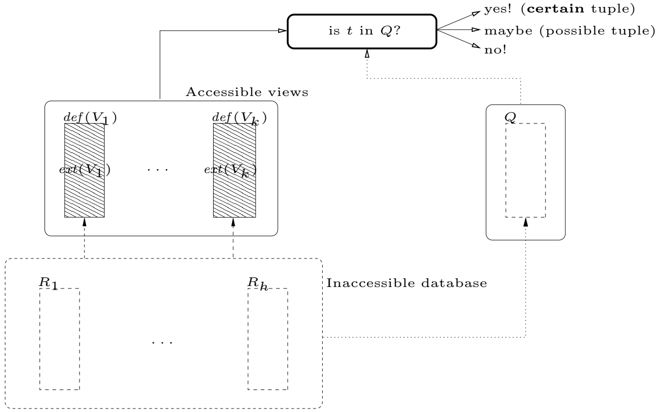
Suppose we have the extensions of the source structures. Let $\mathcal{G}$ be the specification (or, schema) of the global view, Q a query of arity n, and $\vec{\mathbf{d}} = (d_1, \ldots, d_n)$ a tuple of constants

Query answering is defined as follows:

> $\vec{\mathbf{d}} \in cert(Q, \mathcal{V})$ iff $(d_1, \ldots, d_n) \in ans(Q, DB)$, for each DB such that:
>
> - DB satisfies $\mathcal{G}$
> - DB is coherent with $V_1, \ldots, V_m$

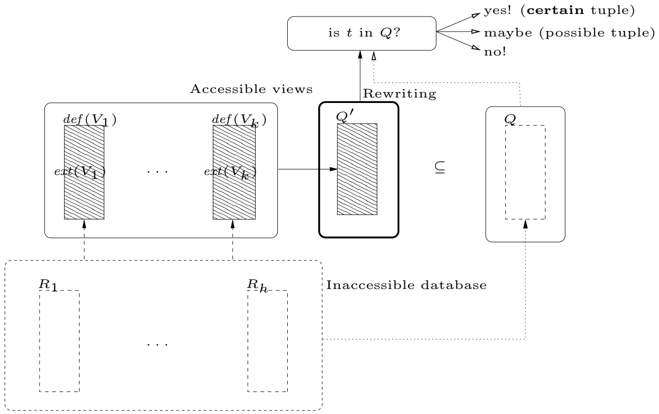
View-based query answering

Basic Idea: Given a query Q, a set of views $\mathcal{V}$ with definitions $def(\mathcal{V})$ and extensions $ext(\mathcal{V})$, compute the tuples t which are in the answer to Q in all databases consistent with the views (certain tuples).



View-based query rewriting

Basic Idea: Given a query Q and a set of views $\mathcal{V}$ with definitions $def(\mathcal{V})$, reformulate Q into a new query Q' expressed in some language on the alphabet of the view names $\mathcal{V}$.



Query rewriting

There are many results on view-based query rewriting.

- Conjunctive queries
- Limitations on binding patterns
- Queries with aggregates
- Under constraints (Functional dependencies, Inclusions dependencies, etc.)
- Description Logics queries
- Recursive queries
- Queries for semi-structured data

Query rewriting: the setting studied here

- **View definitions are CQs** over the global schema
- **View are all sound**
- **Queries are CQs**, or also **UCQs**, i.e., union of conjunctive queries, over the global schema
- **Rewritings are UCQs**, i.e., finite sets of CQs, in the alphabet of the views.

Query rewriting

Query answering by rewriting:

- Rewrite the query in the alphabet of the view names;
- Evaluate the rewriting on the view extension.

Typically people are interested in **rewritings** that are:

- **Contained** in the original query;
- Expressed in a given **query language**;
- **Maximal** for the given class of queries;
- **Exact**(??), i.e., rewritings that are logically equivalent to the query, if possible (observe that such rewritings may not exists).

Exact rewriting: example

Global schema:

```
nonstop(Airline, Number, From, To).
```

Views:

```
flights_by_United(From, To) -: nonstop(’UA’, Number, From, To)
flights_from_SFO(Airline, Number, To) -:
                                   nonstop(Airline, Number, ’SFO’, To)
```

Query:

```
q(Airline, Number) :- nonstop(Airline, Number, ’SFO’, ’LAX’)
```

An **exact rewriting** exists! Namely:

```
q(Airline, Number) :- flight_from_SFO(Airline, Number, ’LAX’)
```

Maximal rewriting: example

Global schema:

```
nonstop(Airline, Number, From, To).
```

Views:

```
flights_by_UA(Number, From, To) -: nonstop(’UA’, Number, From, To)
flights_from_SFO(From, To) -: nonstop(Flight, Number, ’SFO’, To)
```

Query:

```
q(Airline, Number) :- nonstop(Airline, Number, ’LAX’, ’PHX’)
```

A **maximal rewriting** (wrt UCQs) is:

```
q(’UA’, Number) :- flight_by_UA(Number, ’LAX’, ’PHX’)
```

Rewriting: extensions are not considered!

Observe that in computing the rewriting we are taking into account only the **view definitions**, not the view extensions!

How should such definitions be interpreted wrt to the possible extensions?

- A view definition gives properties that the **tuples** produced by the view must have.
- The view definition is **not** a guarantee that all such tuples are provided by the view (i.e., views are only **sound** in general).
- There is not even guarantee that the results produced by two views are consistent (they may need to be reconciled).

Example: a richer domain

Global schema:

`emp(E)` E is an employee
`phone(E,P)` P is E's phone
`office(E,O)` O is E's office
`mgr(E,M)` M is E's manager
`dept(E,D)` D is E's department

Views:

```
v1(E,P,M) -: emp(E), phone(E,P), mgr(E,M)
v2(E,O,D) -: emp(E), office(E,O), dept(E,D)
v3(E,P) -: emp(E), phone(E,P), dept(E,’ToyDept’)
```

- `v1` gives information about employees, their phones and managers.
- `v2` gives information about employees, their offices and department.
- `v3` gives information about employees and their phones but only of the Toy Department.

Example: a richer domain (cont.)

Query: "What are Sally's phone and office?"

```
q(P,O) :- phone('Sally',P), office('Sally',O)
```

Maximal rewriting (wrt UCQs):

```
q(P,O) :- v1('Sally',P,M), v2('Sally',O,D)
q(P,O) :- v3('Sally',P), v2('Sally',O,D)
```

Example: view expansion (cont.)

If we **expand the views** in the rewriting above, we can compare the rewriting with the original query:

```
q(P,O) :- emp('Sally'), phone('Sally',P), mgr('Sally',M),
          emp('Sally'), office('Sally',O), dept('Sally',D)
q(P,O) :- emp('Sally'), phone('Sally',P), dept('Sally','ToyDept'),
          emp('Sally'), office('Sally',O), dept('Sally',D);
```

Observe:

- Both CQs are **contained** in the original query.
- The original query is contained in neither of them, nor in their union, i.e., the rewriting is **not exact**.
- These are the CQs that come closest to the original query while still constructable from the views (their union is a **maximal rewriting** wrt UCQs).

Rewriting algorithms

Given a query and a collection of views that are both **CQs**, there is a **maximal rewriting** made up of finite set of CQs (i.e., a UCQ)!

We can use two algorithms to compute maximal rewritings:

- **Bucket algorithm**
- **Inverse-rules algorithm**

Bucket algorithm: basis

Theorem 1 *If R is a CQ rewriting for a query Q, and R has more atoms then Q, then there exists a CQ rewriting R' such that $R \subseteq R'$.*

Proof. $R^{expd} \subseteq Q$ (rewriting), hence there is a homomorphism μ from $\mathcal{I}_Q$ to $\mathcal{I}_{R^{expd}}$ (the canonical models of Q and R^{expd}, respectively).

If R has more atoms that Q, then there is an atom α such that no atoms of Q is mapped by μ to any atom that comes from α^{expd}.

If we delete α from R we get a new rewriting R' ($R'^{expd} \subseteq Q$, since μ is an homomorphism from Q to R'^{expd}).

Moreover $R \subseteq R'$ (identity mapping on atoms gives us the homomophism). □

Bucket algorithm: raw form

```
for all ( CQs on the alphabet of the views with a number of atoms
          that is less or equal to than those in $Q$) {
    if ($R^{expd} \subseteq Q$) add $R$ to the returned solution, else discard $R$
}
```

Bucket algorithm: refined form

```
/* bucket initialization */
```

Create a bucket for each atom α in the query Q, that will contain views that are relevant to answering the subgoal

Put a view V in the bucket for α if the definition of V contains an atom β that unifies with α

```
/* solution generation */
for all (CQs $R$ on the alphabet of the views
         formed by taking one atom from each bucket) {
    if ($R^{expd} \subseteq Q$) add $R$ to the returned solution, else discard $R$
}
```

Bucket algorithm: example

Views:

```
v1(E,P,M) -: emp(E), phone(E,P), mgr(E,M)
v2(E,O,D) -: emp(E), office(E,O), dept(E,D)
v3(E,P) -: emp(E), phone(E,P), dept(E,'ToyDept')
```

Query: “What are Sally’s phone and office?”

```
q(P,O) :- phone('Sally',P), office('Sally',O)
```

Buckets:

```
     B1                               B2
v1('Sally',P,M)                   v2('Sally',O,D)
v3('Sally', P)
```

Rewriting:

```
q(P,O) :- v1('Sally',P,M), v2('Sally',O,D)
q(P,O) :- v3('Sally',P), v2('Sally',O,D)
```

Bucket algorithm: main result

Theorem 2 *The rewriting generated by the bucket algorithm is the* **maximal rewriting wrt UCQs**.

Proof. For the raw form, immediate.

For the refined form [Grahne&Mendelzon, ICDT’99] □

Inverse-rule algorithm: basis

Obtain **inverse rules** by:

1. Replace existential variable in the body of each view definition by a **Skolem function**.

   Recall `v(X) -: a1(X,Y), a2(X,Y)` stands for:
   $\forall x.v(x) \Rightarrow \exists y.a_1(x,y) \wedge a_2(x,y)$.
   By applying Skolemization we get $\forall x.v(x) \Rightarrow a_1(x,f(x)) \wedge a_2(x,f(x))$.

2. **Split** the body of the rules.

   Recall that $\forall x.v(x) \Rightarrow a_1(x,f(x)) \wedge a_2(x,f(x))$ is equivalent to
   $(\forall x.v(x) \Rightarrow a_1(x,f(x))) \ \wedge \ (\forall x.v(x) \Rightarrow a_2(x,f(x)))$, thus we obtain:

   ```
   a1(X,f(X)) :- v(X)
   a2(X,f(X)) :- v(X)
   ```

Evaluate the query considering **inverse rules as IDB** and **views as EDB**.

Inverse-rule algorithm: observations

- Because all functions symbols are in the head of the inverse rules, we never introduce a function symbol within a function symbol, leading to a finite process.
- Bottom-up evaluation can produce tuples with function symbols, but these cannot be real answers to the query and need to be discarded.
- The algorithm works for datalog (recursive) queries (but CQ views) as well.
- For non recursive queries we can easily get an equivalent UCQs by evaluating the query a la prolog.
- If required, functional symbols can be polynomially eliminated by adding new predicates.

Inverse-rule algorithm: example

Views:

```
v1(E,P,M) -: emp(E), phone(E,P), mgr(E,M)
v2(E,O,D) -: emp(E), office(E,O), dept(E,D)
v3(E,P) -: emp(E), phone(E,P), dept(E,'ToyDept')
```

Inverse rules:

```
emp(E) :- v1(E,P,M)              mgr(E,M) :- v1(E,P,M)
emp(E) :- v2(E,O,D)
emp(E) :- v3(E,P)                office(E,O) :- v2(E,O,D)

phone(E,P) :- v1(E,P,M)          dept(E,D) :- v2(E, O, D)
phone(E,P) :- v3(E,P)            dept(E, 'ToyDept') :- v3(E,P)
```

Query: "What are Sally's phone and office?"

```
q(P,O) :- phone('Sally',P), office('Sally',O)
```

Inverse-rule algorithm: example (cont.)

Observe, if we unfold the query

```
q(P,O) :- phone('Sally',P), office('Sally',O)
```

using the inverse rules

```
emp(E) :- v1(E,P,M)              mgr(E,M) :- v1(E,P,M)
emp(E) :- v2(E,O,D)
emp(E) :- v3(E,P)                office(E,O) :- v2(E,O,D)

phone(E,P) :- v1(E,P,M)          dept(E,D) :- v2(E, O, D)
phone(E,P) :- v3(E,P)            dept(E, 'ToyDept') :- v3(E,P)
```

we get

```
q(P,O) :- v1('Sally',P,M), v2('Sally',O,D)
q(P,O) :- v3('Sally',P), v2('Sally',O,D)
```

Inverse-rule algorithm: main result

Theorem 3 *For CQs (and UCQs) the rewriting generated by the inverse-rule algorithm is the* **maximal rewriting wrt UCQs**.

For datalog (recursive) queries the rewriting generated by the inverse-rule algorithm is the maximal rewriting wrt datalog queries.

Proof. See [Duschka&Genesereth, PODS'97]. □

Maximal rewriting vs certain answers

Query answering (QA) is defined as follows:

$t \in cert(Q, \mathcal{V})$ **iff** $t \in ans(Q, DB)$**, for each** DB **such that:**

- DB **satisfies** $\mathcal{G}$
- DB **is coherent with** $V_1, \ldots, V_m$

For sound views, LAV mapping and no constraint on the global view, the definition of certain answers becomes as follows:

$$cert(Q, \mathcal{V}) = \{t \mid \forall DB.\ ext(\mathcal{V}) \subseteq ans(def(\mathcal{V}), DB) \Rightarrow t \in ans(Q, DB)\}$$

Next we show that for CQs the maximal rewriting wrt UCQs, coincides with the certain answers.

Maximal rewriting vs certain answers (cont.)

Theorem 4 *Every answer generated by evaluating the* **maximal rewriting** $Rmax_{UCQs}(Q, \mathcal{V})$ *is a* **certain answer** *in* $cert(Q, \mathcal{V})$.

Proof. Assume not. Let $R = Rmax_{UCQs}(Q, \mathcal{V})$, then there is a $t \in ans(R, ext(\mathcal{V}))$ such that $t \notin cert(Q, \mathcal{V})$.

Now for all DB such that $ext(\mathcal{V}) \subseteq ans(def(\mathcal{V}), DB)$ we have that $t \in ans(R^{expd}, DB)$.

Since R is a rewriting, we have that $R^{expd} \subseteq Q$, and hence $t \in ans(Q, DB)$.

That is, for all DB such that $ext(\mathcal{V}) \subseteq ans(def(\mathcal{V}), DB)$ we have that $t \in ans(Q, DB)$, i.e., $t \in cert(Q, \mathcal{V})$. Contradiction. □

Maximal rewriting vs certain answers (cont.)

Theorem 5 *Every* **certain answer** *in* $cert(Q, \mathcal{V})$ *is generated by evaluating the* **maximal rewriting** $Rmax_{UCQs}(Q, \mathcal{V})$.

Proof. Suppose not. Let $t \in cert(Q, \mathcal{V})$ such that $t \notin ans(Rmax_{UCQs}(Q, \mathcal{V}), ext(\mathcal{V}))$. Consider the CQ C_t on the alphabet of the views defined as:

$$C_t(x) \text{ :- } x = t, V_1(t_{11}), \cdots, V_1(t_{1k_1}), \cdots, V_n(t_{n1}), \cdots, V_n(t_{nk_n})$$

or simply,

$$C_t(t) \textbf{ :- } V_1(t_{11}), \cdots, V_1(t_{1k_1}), \cdots, V_n(t_{n1}), \cdots, V_n(t_{nk_n})$$

where $\mathcal{V} = \{V_1, \ldots, V_n\}$, and $ext(v_i) = \{t_{i,1} \ldots, t_{ik_i}\}$ for each $i = 1, \ldots, n$.

(cont...)

Then, $C_t^{expd} \subseteq Q$, indeed

- since $t \in cert(Q, \mathcal{V})$, we have that for all DB conforming with the views $ans(C_t^{expd}, DB) = \{t\} \subseteq ans(Q, DB)$
- while for those DB' not conforming with the views $ans(C_t^{expd}, DB') = \emptyset \subseteq ans(Q, DB')$.

Being C_t a CQ on the alphabet of the views and being $C_t^{expd} \subseteq Q$ it follows $C_t \subseteq Rmax_{UCQs}(Q, \mathcal{V})$. Contradiction! □

Part 1: conclusions

For the setting considered here, i.e,

- **View definitions are CQs** over the global schema
- **View are all sound**
- **Queries are CQs**, or also **UCQs**, i.e., union of conjunctive queries, over the global schema
- **Rewritings are UCQs**, i.e., finite sets of CQs, in the alphabet of the views

everything works fine:

- We can focus on **maximal rewriting** wrt UCQs.
- Maximal rewriting computes exactly the **certain answers**.
- Maximal rewriting is expressible in a simple (LOGSPACE in data complexity) query language.

Does these nice results extend to more general settings? See Part 2 and Part 3.

Part 2

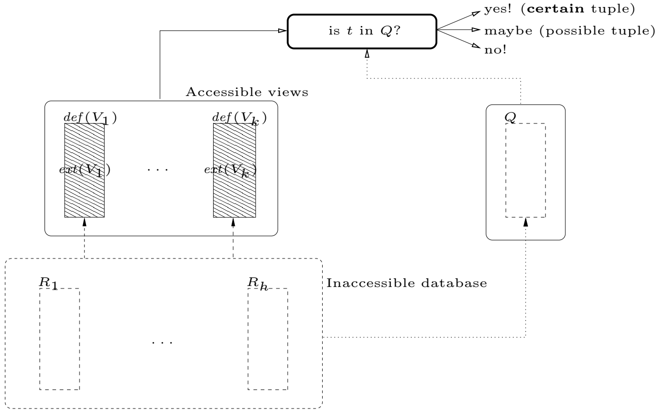
View-based query processing: Computing the answer to a query based on a set of views, rather than on the raw data in the database.

- View-based query rewriting (indirect)
- View-based query answering (direct)
- Relationship between query answering and query rewriting

We are studying view-based query processing within the **relational data model**.

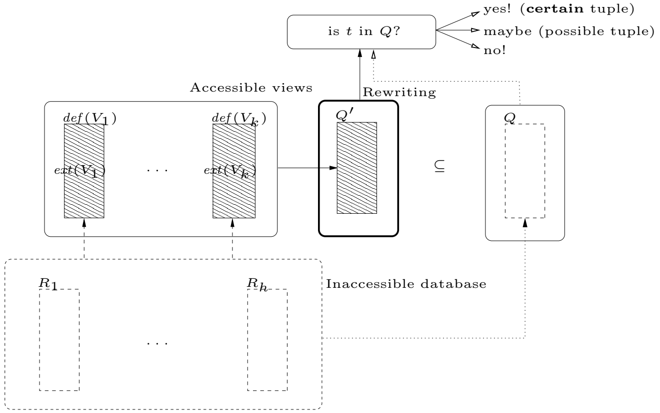
View-based query answering

Basic Idea: Given a query Q, a set of views $\mathcal{V}$ with definitions $def(\mathcal{V})$ and extensions $ext(\mathcal{V})$, compute the tuples t which are in the answer to Q in all databases consistent with the views (certain tuples).



View-based query rewriting

Basic Idea: Given a query Q and a set of views $\mathcal{V}$ with definitions $def(\mathcal{V})$, reformulate Q into a new query Q' expressed in some language on the alphabet of the view names $\mathcal{V}$.



View-based query answering

There are several results also on view-based query answering.

- Complexity for several query and view languages
- CQs and UCQs under Description Logics constraints
- Regular path queries

View-based query answering

As said above query answering (QA) is defined as follows:

$t \in cert(Q, \mathcal{V})$ **iff** $t \in ans(Q, DB)$**, for each** DB **such that:**

- DB **satisfies** $\mathcal{G}$
- DB **is coherent with** $V_1, \ldots, V_m$

For sound views, LAV mapping and no constraint on the global view, the definition of certain answers becomes as follows:

$$cert(Q, \mathcal{V}) = \{t \mid \forall DB.\ ext(\mathcal{V}) \subseteq ans(def(\mathcal{V}), DB) \Rightarrow t \in ans(Q, DB)\}$$

View-based query answering and query containment

There is a strong relationship between QA and QC when the views are **sound**.

For query languages at least as powerful as CQs (CQs, UCQs, datalog, FOL, etc.).

QA and QC are mutually reducible one into the other in polynomial time!

Reduction to query containment

Consider the query Q' on the alphabet of the views defined as:

$$Q'(x) \text{ :- } x = t \wedge V_1(t_{11}) \wedge \cdots \wedge V_1(t_{1k_1}) \wedge \cdots \wedge V_n(t_{n1}) \wedge \cdots \wedge V_n(t_{nk_n})$$

where $\mathcal{V} = \{V_1, \ldots V_n\}$, and $ext(V_i) = \{t_{i,1} \ldots, t_{ik_i}\}$ for each $i = 1, \ldots, n$.

Theorem 6 $t \in cert(Q, \mathcal{V})$ *iff* $Q'^{expd} \subseteq Q$.

Proof.

$\Rightarrow$ If $t \in cert(Q, \mathcal{V})$ then

- for all DB conforming with the views $ans(Q'^{expd}, DB) = \{t\} \subseteq ans(Q, DB)$
- while for those DB' not conforming with the views $ans(Q'^{expd}, DB') = \emptyset \subseteq ans(Q, DB')$.

hence $Q'^{expd} \subseteq Q$.

$\Leftarrow$ Assume $Q'^{expd} \subseteq Q$. For all DB conforming with the views $ans(Q'^{expd}, DB) = \{t\}$ and since $Q'^{expd} \subseteq Q$, we have $t \in ans(Q, DB)$. □

Reduction from query containment

Let Q_1 and Q_2 be two queries (x is a tuple of variables):

$$Q_1(x) \text{ :- } \Phi_1(x)$$
$$Q_2(x) \text{ :- } \Phi_2(x)$$

Consider a single view V:

- $def(V)$: $V(x) \text{ :- } x = c \ \wedge \ \exists y.(\Phi(y) \wedge p(y))$, where p is a new predicate
- $ext(V) = \{c\}$
- $as(V) = sound$

and the query Q defined as

$$Q(x) \text{ :- } x = c \wedge \exists y.(\Phi(y) \wedge p(y))$$

Theorem 7 $Q_1 \subseteq Q_2$ *iff* $c \in cert(Q, \{V\})$.

Proof.

$\Rightarrow$

If $c \notin cert(Q, \mathcal{V})$, then there exists a database DB conforming with the views such that $c \notin ans(Q, DB)$. This implies that there exists a tuple in $t \in ans(Q_1, DB)$ but $t \notin ans(Q_2, DB)$, i.e., $Q_1 \not\subseteq Q_2$.

$\Leftarrow$ Assume $Q_1 \not\subseteq Q_2$. Then there exists a database DB and a tuple t such that $t \in ans(Q_1, DB)$ but $t \notin ans(Q_2, DB)$. We can extend DB by assigning to new predicate p the interpretation $ans(p, DB) = \{t\}$. But then we have that DB conforms to the view V while $ans(Q, DB) = \emptyset$. Hence $c \notin cert(Q, \mathcal{V})$. □

QA and QC: observations

- We can transfer **upper-bounds** from QC to QA, using the reduction form QA to QC.
- We can transfer **lower-bounds** from QC to QA, using the reduction from QC to QA.
- We can use algorithms for query containment to get **algorithms** for query answering.
- *What kind of complexity are we characterizing for QA?* **Combined complexity.**

Combined complexity of QA for sound views

$\text{QC}[Q_1, Q_2]$	CQs	PQs*	datalog	FOL
CQs	NP	NP	EXPTIME	undec.
PQ	Π_2^p	Π_2^p	EXPTIME	undec.
datalog	2EXPTIME	2EXPTIME	undec.	undec.
FOL	undec.	undec.	undec.	undec.
$\text{QA}[\mathcal{V}, Q]$	CQs	PQs	datalog	FOL
CQs	**NP**	**NP**	**EXPTIME**	**undec.**
PQs	Π_2^p	Π_2^p	**EXPTIME**	**undec.**
datalog	**2EXPTIME**	**2EXPTIME**	**undec.**	**undec.**
FOL	**undec.**	**undec.**	**undec.**	**undec.**

* PQs, i.e., are UCQs that allow to nest disjunctions in conjunctions.

What about data complexity?

We want to refine the complexity analysis to take into account that data in the database are orders of magnitude bigger that the size of the queries.

In other words we would like to characterize the **data complexity** of QA!

Complexity of view based query answering

Can be measured in three different ways:

data complexity: as a function of the size of the view extensions $ext(V_1) \cup \cdots \cup ext(V_k)$

expression complexity: as a function of the size of the query Q and of the view definitions $def(V_1), \ldots, def(V_k)$

combined complexity: as a function of the size of both $ext(V_1) \cup \cdots \cup ext(V_k)$ and the expressions Q, $def(V_1), \ldots, def(V_k)$

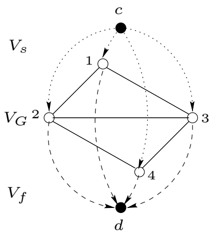
QA coNP-hard for UCQs views

Theorem 8 *Let $\mathcal{L}_Q$ and $\mathcal{L}_\mathcal{V}$ be UCQs, and let view be sound. Then QA is coNP-hard in data complexity.*

Proof. Reduction from graph 3-colorability.

$$\begin{array}{lllll} \text{Views:} & V_s(x,y) & \text{-:} & R_s(x,y) \\ & V_G(x,y) & \text{-:} & R_{rg}(x,y) \vee R_{gr}(x,y) \vee R_{rb}(x,y) \vee R_{br}(x,y) \vee R_{gb}(x,y) \vee R_{bg}(x,y) \\ & V_f(x,y) & \text{-:} & R_f(x,y) \\ \text{Query:} & Q(x,y) & \text{:-} & \bigvee_{\beta \neq \gamma} \exists z, v, w.(R_s(x,z) \wedge R_{\alpha\beta}(z,v) \wedge R_{\gamma\delta}(v,w) \wedge R_f(w,y)) \end{array}$$

Only the view extensions depend on graph $G = (N, E)$



$$\begin{array}{llll} \text{Extensions:} & ext(V_s) & = & \{(c,a) \mid a \in N\} \\ & ext(V_G) & = & \{(a,b),(b,a) \mid (a,b) \in E\} \\ & ext(V_f) & = & \{(a,d) \mid a \in N\} \end{array}$$

G is 3-colorable iff (c, d) is not a certain answer of Q.

Note: queries and view definitions used in this proof are UPQs! See Part 3.

QA for PQs views: algorithm

Let n be the number of tuples in $ext(\mathcal{V})$ and k the number of atoms in the longest (wrt atoms) view definition.

Theorem 9 *Let $\mathcal{L}_\mathcal{V}$ be PQs and $\mathcal{L}_Q$ be datalog, then $t \notin cert(Q, \mathcal{V})$ iff there exists a database DB' of* **size** *nk conforming with the views such that $t \notin ans(Q, DB)$. (For both sound and exact views).*

Proof. Since $t \notin cert(Q, \mathcal{V})$, there exists a database DB conforming with the views such that $t \notin ans(Q, DB)$.

Consider the database $DB' \subseteq DB$ having only the nk tuples required by the views. DB' still conforms with the views, moreover still have $t \notin ans(Q, DB')$ (it is sufficient only that Q is monotone!). □

QA for PQs views: algorithm (cont.)

Theorem 10 *Let $\mathcal{L}_\mathcal{V}$ be PQs and $\mathcal{L}_Q$ be datalog, then QA is in* **coNP**. *(For both sound and exact views.)*

Algorithm:

```
bool certain (tuple $t$, query $Q$, views $\mathcal{V}$)
{
   guess(a database $DB'$ of size $nk$) {   (nondet.)
      verify whether $DB'$ conforms to the views   (poly.)
      verify whether $t \notin ans(Q, DB')$   (poly.)
      if (both test positive ) return false
   }
   return true;
}
```

Data complexity for sound views

Sound Views	CQ	PQ	datalog	FOL
CQ	**PTIME***	**PTIME***	**PTIME***	**undec.**
PQ	**coNP**	**coNP**	**coNP**	**undec.**
datalog	**coNP**	**coNP**	**undec.**	**undec.**
FOL	**undec.**	**undec.**	**undec.**	**undec.**

* This is shown, by proving that the maximal rewriting wrt a PTIME query language (namely CQs, PQs, datalog) computes exactly the certain answers. See Part 1.

What about exact views?

Till now we have focused on sound views. What happen when we consider **exact views***?*

QA becomes coNP-hard even for views defined by CQs! See below.

QA coNP-hard for exact CQs views

Theorem 11 *Let $\mathcal{L}^{\mathcal{V}}$ and $\mathcal{L}^{Q}$ be CQs, and let $as(\mathcal{V}) = exact$. Then verifying whether $c \in cert(Q, \mathcal{V})$ is* **coNP-hard** *in data complexity.*

Proof. Reduction from graph 3-colorability. Let $G = (N, E)$ be an arbitrary graph. Consider three exact views $\mathcal{V} = \{V_1, V_2, V_3\}$:

definitions	extensions
$V_1(x)$:- $color(x, y)$	$ext(V_1) = N$
$V_2(y)$:- $color(x, y)$	$ext(V_2) = \{red, green, blue\}$
$V_3(y)$:- $edge(x, y)$	$ext(V_3) = E$

and the query Q:

$$Q(c) \text{ :- } edge(x, y), color(x, z), color(y, z)$$

Then $c \in cert(Q, \mathcal{V})$ iff the graph G is not 3-colorable. □

Data complexity for exact views

Complete Views	CQ	PQ	datalog	FOL
CQ	**coNP**	**coNP**	**coNP**	**undec.**
PQ	**coNP**	**coNP**	**coNP**	**undec.**
datalog	**undec.**	**undec.**	**undec.**	**undec.**
FOL	**undec.**	**undec.**	**undec.**	**undec.**

Part 2: conclusions

We have gathered a lot of results on query rewriting (Part 1) and on query answering (Part 2).

What can we say now about the relationships between these two ways of performing view-based query processing? See Part 3.

Part 3

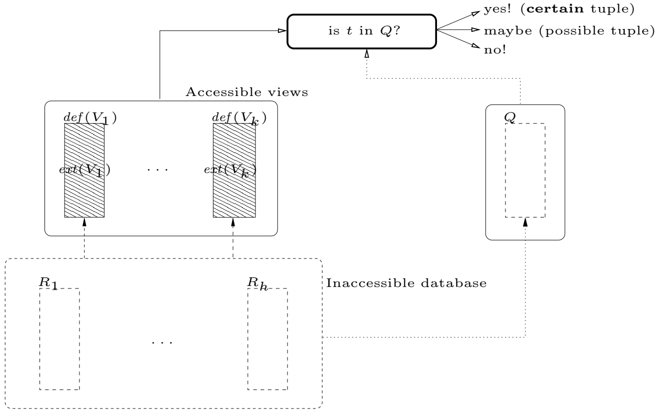
View-based query processing: Computing the answer to a query based on a set of views, rather than on the raw data in the database.

- View-based query rewriting (indirect)
- View-based query answering (direct)
- Relationship between query answering and query rewriting

We are studying view-based query processing within the **relational data model**.

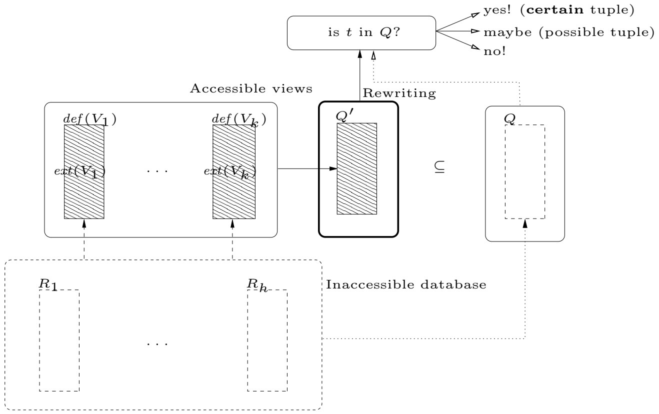
View-based query answering

Basic Idea: Given a query Q, a set of views $\mathcal{V}$ with definitions $def(\mathcal{V})$ and extensions $ext(\mathcal{V})$, compute the tuples t which are in the answer to Q in all databases consistent with the views (certain tuples).



View-based query rewriting

Basic Idea: Given a query Q and a set of views $\mathcal{V}$ with definitions $def(\mathcal{V})$, reformulate Q into a new query Q' expressed in some language on the alphabet of the view names $\mathcal{V}$.



Previous results

There are many previous results on view-based query processing.

View-based query rewriting:

- Conjunctive queries
- Queries with aggregates
- Under constraints (inclusion and functional dependencies)
- Recursive queries
- Description Logics queries
- Queries for semi-structured data

View-based query answering:

- Complexity for several query and view languages
- Under Description Logics constraints
- Regular path queries

But: *For a long time, no clear understanding of the relationships between query answering and query rewriting!!!*

Query rewriting vs. query answering

Query answering by rewriting:

- Rewrite the query in the alphabet of the view names;
- Evaluate the rewriting on the view extension.

Typically people are interested in **rewritings** that are:

- Contained in the original query
- Expressed in a given **query language**
- **Maximal** for the given class of queries
- Exact(??), i.e., rewritings that are logically equivalent to the query, if possible (observe that such rewritings may not exists).

But:

- **Q1:** *When is the rewriting also* **complete** *– i.e., computes all certain tuples?*
- **Q2:** *What do we gain or lose by focusing on a given class of queries?*

Perfect rewriting

Query answering: Let $cert(Q, def(\mathcal{V}), ext(\mathcal{V}))$ be the function that computes the certain tuples for Q wrt $def(\mathcal{V})$ and $ext(\mathcal{V})$.

Perfect rewriting:

Define $cert_{[Q,def(\mathcal{V})]}$ to be the **Currying** of $cert$ wrt Q and $def(\mathcal{V})$.

⇒ – $cert_{[Q,def(\mathcal{V})]}$ is a query on the alphabet of the view names that given $ext(\mathcal{V})$ returns the certain tuples for the query Q wrt $def(\mathcal{V})$ and $ext(\mathcal{V})$;

– $cert_{[Q,def(\mathcal{V})]}$ is a (*sound*) **rewriting** of Q wrt $def(\mathcal{V})$;

– $cert_{[Q,def(\mathcal{V})]}$ is **complete** (no better rewritings may exist);

– $cert_{[Q,def(\mathcal{V})]}$ is called the **perfect rewriting** of Q wrt $def(\mathcal{V})$.

Comparing with the perfect rewriting

- **Q1:** *Can we express the perfect rewriting in a certain query language?*
- **Q2:** *How does maximal rewriting for a given class of queries compare with the perfect rewriting?*
  - *From a semantical point of view?*
  - *From a computational point of view?*
- *Which is the computational complexity of the perfect rewriting?*

The case of conjunctive queries

Rewriting: [Levy,Mendelzon,Sagiv,Srivastava-PODS95], [DuschkaGenesereth-PODS97], [AbiteboulDuschka-PODS98])

Let Q and $def(\mathcal{V})$ be CQs, and let Q' be the **union of all maximal rewritings for the class of CQs**. Then:

- Q' is the maximal rewriting for the class of unions of conjunctive queries (UCQs);
- **Query answering:** generate Q', evaluate Q' on $ext(\mathcal{V})$;
- **Q' is the perfect rewriting**;
- Q' is a PTIME query (in fact, LOGSPACE).
- Q' is an exact rewriting, if an exact rewriting exists.

Q: *Does this "ideal situation" carry on to cases where Q and $def(\mathcal{V})$ allow for union?*

Unions of path queries (UPQs)

Very simple query language defined as follows:

$$\begin{array}{lcl} Q & \longrightarrow & P \mid Q_1 \cup Q_2 \\ P & \longrightarrow & R \mid P_1 \circ P_2 \end{array}$$

where R denotes a **binary database relation**, P denotes a **path query**, which is a chaining of database relations, and Q denotes a **union of path queries**.

Observe: UPQs are a simple form of:

- Unions of conjunctive queries;
- Regular path queries.

View-based query processing for UPQs

Thm: View-based query answering for UPQs is coNP-complete in data complexity [Calvanese,DeGiacomo,Lenzerini,Vardi-ICDE'00].

In other words: $cert(Q, def(\mathcal{V}), ext(\mathcal{V}))$ with Q and $def(\mathcal{V})$ fixed is coNP-complete.

$\Rightarrow$ **The perfect rewriting $cert_{[Q,def(\mathcal{V})]}$ is a coNP-complete query.**

Bad news: *For query languages that include UPQs the perfect rewriting is coNP-hard!*

PTIME perfect rewritings

Typically we are interested in **PTIME queries**, (or even better LOGSPACE queries).

Program: *Isolate those UPQs Q and $def(\mathcal{V})$ for which the perfect rewriting is PTIME (assuming P≠NP).*

Unfortunately, this reduces to one of the most difficult open questions in computer science: the non-uniform CSP PTIME dicotomy (here phrased directly on homomorphisms): *Characterize the structures B such that for each strcture A over the same alphabet, findingchecking the existence of an homorphism from A to B is PTIME*[Calvanese,DeGiacomo,Lenzerini,Vardi-LICS'00].

Rewritings in PTIME query languages

Fall back Program: *Fix the language of the rewriting, choosing a PTIME query language.*

Observe: This is exactly what is done in most papers on rewriting!!!

Important problem:

- How can we test the rewriting obtained for perfectness?
  In general is an hard problem, see also [Calvanese,DeGiacomo,Lenzerini,Vardi-ICDT'05].