Query Answering with Incomplete Information:
Conjunctive Queries over Naive Tables

Formal Methods

Giuseppe De Giacomo

Incomplete information and query answering

- Incomplete information in data: missing / unknown / partially specified data
- Query answering
  - Over usual databases (complete information):
    QA by evaluation (or "model checking")
    $$D \models Q$$
    i.e., D is seen as an interpretation (for simplicity we assume the query to be boolean, no free variables)
  - Over incomplete databases (incomplete information):
    QA by logical implication (or "entailment")
    $$\forall \mathcal{I}. \mathcal{I} \models D \text{ implies } \mathcal{I} \models Q$$

Incomplete databases

A common form of incomplete databases are the so-called "naive tables", which include values and "labelled nulls" (standing for unknown values) [IL84].

Example

Employee

name
Smith
$null_1$
Brown

Manager

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

- *Const*: we have infinite constants, corresponding to domain objects as usual;
- *Nulls*: we have a countably infinite set of nulls, corresponding to variables ranging over *Cons*;
- Tables are incomplete, i.e., more tuples may belong to them, corresponding to the so called "open-world-assumption" or OWA. (For example $null_2$ belongs to *Employee* though not reported in the table.)

Incomplete databases: semantics

Semantics of incomplete databases:

- A valuation function for nulls is a assignment function $\sigma : Nulls \rightarrow Const$ (essentially nulls are considered as individual variables in logic).
- We denote by $\mathcal{I}, \sigma \models D$ the fact that for every tuple $(t_1, \ldots, t_n) \in P$ for each table P we have $\mathcal{I}, \sigma \models P(t_1, \ldots, t_n)$.
- We define in logic the set of databases completing D as

$$Models(D) \;=\; \{\mathcal{I} \mid \text{there exists a } \sigma \text{ such that } \mathcal{I}, \sigma \models D\}$$

Example

Employee

name
Smith
$null_1$
Brown

Manager

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

Employee

name
Smith
White
Brown
Black

Manager

mgr	*mgd*
Smith	White
White	Brown
Brown	Black

Employee

name
Smith
Brown
Black
...

Manager

mgr	*mgd*
Smith	Brown
Brown	Brown
Brown	Brown

...

Certain answers to a query

An incomplete database acts like a logical theory: it selects models.

Query answering in complete databases

The answer to a query $q(\vec{x})$ over a complete database D, denoted q^D, is the set of tuples $\vec{c}$ of constants of *Const* such that the $\vec{c} \in q^D$ is to true in D.

Query answering in incomplete databases

The certain answer to a query $q(\vec{x})$ over an incomplete database D, denoted $cert(q, D)$, is the set of tuples $\vec{c}$ of constants of *Const* such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of D.

Note:

- It q is boolean, and D is incomplete: we write $D \models q$ iff q evaluates to true in every model $\mathcal{I}$ of D, (otherwise we write $D \not\models q$.
- We use the same notation as for query answering based on evaluation: the difference is in the incompleteness of the database.

Query languages for incomplete databases

Which query language to use?

1. Full SQL (or equivalently, first-order logic)
   - NO: in the presence of incomplete information, query answering becomes undecidable (FOL validity).
     (Notice this holds already for an empty incomplete database!)
2. Conjunctive queries (or better union of conjunctive queries)
   - Conjunctive queries are well behaved wrt containment. Can they be used for query answering in presence of incomplete information.
     YES! See what follows.

Conjunctive queries and incomplete databases

A conjunctive query (CQ) is a first-order query of the form

$$q(\vec{x}) \leftarrow \exists \vec{y}. R_1(\vec{x}, \vec{y}) \wedge \cdots \wedge R_k(\vec{x}, \vec{y})$$

where each $R_i(\vec{x}, \vec{y})$ is an atom using (some of) the free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

We will also use the simpler Datalog notation:

$$q(\vec{x}) \leftarrow R_1(\vec{x}, \vec{y}), \ldots, R_k(\vec{x}, \vec{y})$$

Note:

- CQs contain no disjunction, no negation, no universal quantification.
- Correspond to SQL/relational algebra select-project-join (SPJ) queries – the most frequently asked queries.
- A Boolean CQ is a CQ without free variables $\Rightarrow$ $q() \leftarrow \exists \vec{y}. R_1(\vec{y}) \wedge \cdots \wedge R_k(\vec{y})$.

Conjunctive queries and incomplete databases

Containment of conjunctive queries $q_1 \subseteq q_2$ is decidable: and LOGSPACE in q_1 and NP-complete in q_2 [CM77].

Given an incomplete database D as above we can construct in linear time a (boolean) conjunctive query q_D that fully captures it.

- For each tuple in a table of D becomes an atom in the conjunctive query q_D.
- For each labelled nulls occurring in D becomes an existentially quantified variable in q_D.

Example

E(mployee)

name
Smith
$null_1$
Brown

M(anager)

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

$$\exists x_1, x_2 . E(Smith) \wedge E(x_1) \wedge E(Brown) \wedge M(Smith, x_1) \wedge M(x_1, Brown) \wedge M(Brown, x_2)$$

E

SMITH
X
BROWN

M

SMITH	X
X	BROWN
BROWN	Y

Conjunctive queries and incomplete databases

Theorem ([IL84])

Let D be a database with incomplete information as above (naive tables), q_D the corresponding conjunctive query constructed as above, and q a boolean (union) of conjunctive query. Then:

$$D \models q \text{ iff } q_D \subseteq q$$

Proof.

1. Observe that the models of D by construction coincide with that of the formula q_D: that is $\forall \mathcal{I}. \mathcal{I} \models D$ iff $\mathcal{I} \models q_D$.
2. Moreover, $q_D \subseteq q$ in the case of boolean queries stands for $\forall \mathcal{I}. \mathcal{I} \models q_D$ implies $\mathcal{I} \models q$, or simply $q_D \models q$.
3. Hence, by (1) $D \models q$ iff $q_D \models q$. □

Conjunctive queries and incomplete databases

Using Chandra & Merlin Theorem [CM77], we get:

Theorem ([IL84])

Let D be a database with incomplete information as above (naive tables), q_D the corresponding conjunctive query constructed as above, $\mathcal{I}_{q_D}$ its canonical database, and q a boolean (union) of conjunctive query. Then:

$$D \models q \text{ iff } \mathcal{I}_{q_D} \models q$$

Note: $\mathcal{I}_{q_D}$ is exactly D with nulls interpreted as additional constants!
Hence:

Compute certain answers of non boolean CQs over incomplete databases

Given a non boolean (U)CQ q and an incomplete database D:

1. Evaluate q over D as it was a complete database
2. filter out all answers where null appears (certain answers are constituted by tuples of constants in *Const*)

Conjunctive queries and incomplete databases

As a consequence of the above theorem we have:

Computing certain answers for (union) of conjunctive queries over databases with incomplete information (naive tables) is:

- LOGSPACE in data complexity
- NP-complete in query complexity and combined complexity

Note1: Exactly as for the case of complete information!

Note2: Use of CQs is crucial, since for full FOL we get undecidability!

Examples of CQs over an incomplete database

Example

$E(mployee)$

name
Smith
$null_1$
Brown

$M(anager)$

mgr	*mgd*
Smith	$null_1$
$null_1$	Brown
Brown	$null_2$

- Queries:
  $q_1(x, y) \leftarrow M(x, y)$
  $q_2(x) \leftarrow \exists y.M(x, y)$
  $q_3(x) \leftarrow \exists y_1, y_2, y_3.M(x, y_1) \wedge M(y_1, y_2) \wedge M(y_2, y_3)$
  $q_4(x, y_3) \leftarrow \exists y_1, y_2.M(x, y_1) \wedge M(y_1, y_2) \wedge M(y_2, y_3)$
- Answers:
  q_1: { }
  q_2: { `Smith`, `Brown` }
  q_3: { `Smith` }
  q_4: { }

Conclusion

Incomplete information

Several other forms of incomplete information have been studied in the literature of Databases and especially in the literature of Knowledge Representation and Reasoning in Artificial Intelligence.

These include:

- Knowledge Bases
- Ontologies, Description Logics, Semantic Technologies
- Reasoning about Actions (incomplete information also on the dynamics)
- ...

Note

Only in very few cases dealing with incomplete information can be done through query evaluation techniques.

If interested, take the course on Knowledge Representation and Semantic Technologies.

References

[CM77] A. K. Chandra and P. M. Merlin.
Optimal implementation of conjunctive queries in relational data bases.
In *Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77)*, pages 77–90, 1977.

[IL84] T. Imielinski and W. J. Lipski.
Incomplete information in relational databases.
J. of the ACM, 31(4):761–791, 1984.