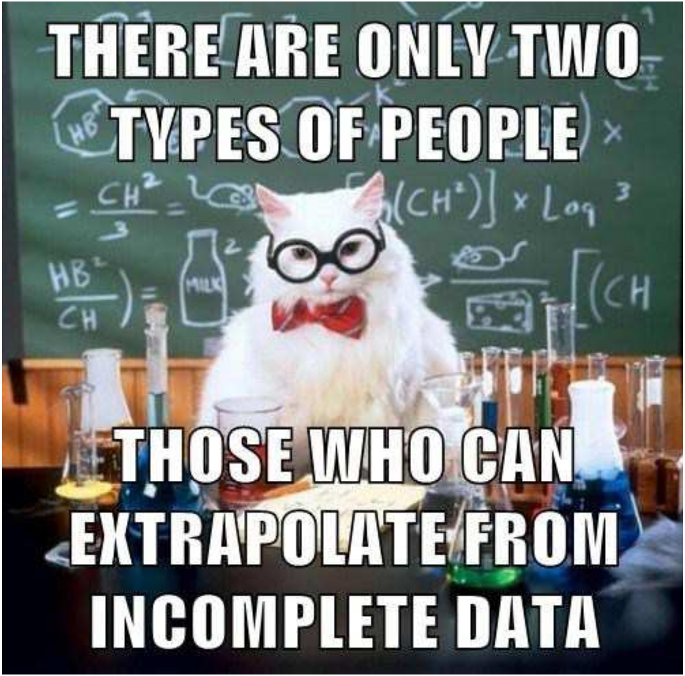

THERE ARE ONLY TWO TYPES OF PEOPLE
THOSE WHO CAN EXTRAPOLATE FROM INCOMPLETE DATA

Incomplete Data:
What Went Wrong, and How to Fix It

Leonid Libkin (University of Edinburgh)

Incomplete information

- It is everywhere.
- The more data we accumulate, the more incomplete data we accumulate.
- Sources:
  - Traditional (missing data, wrong entries, etc)
  - The Web
  - Integration/translation/exchange of data, etc
- The importance of it was recognized early
  - Codd, *"Understanding relations (installment #7)"*, 1975.
- And yet the state is very poor:
  - Both practice and theory

SQL: example 1

Orders

order_id	title
ord1	*'SQL Standard'*
ord2	*'Database Systems'*
ord3	*'Logic'*

Payments

pay_id	order_id	amount
p1	ord1	–
p2	–	\$50

SQL: example 1

Orders

order_id	title
ord1	*'SQL Standard'*
ord2	*'Database Systems'*
ord3	*'Logic'*

Payments

pay_id	order_id	amount
p1	ord1	–
p2	–	\$50

Query: all payment ids. Written as:

SELECT pay_id FROM Payments
WHERE amount $\geq$ 50 OR amount $<$ 50

SQL: example 1

Orders

order_id	title
ord1	*'SQL Standard'*
ord2	*'Database Systems'*
ord3	*'Logic'*

Payments

pay_id	order_id	amount
p1	ord1	–
p2	–	$50

Query: all payment ids. Written as:

```
SELECT pay_id FROM Payments
WHERE amount ≥ 50 OR amount < 50
```

Answer: only p2!

SQL: it gets worse

Query: unpaid orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer:

SQL: it gets worse

Query: unpaid orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer: EMPTY!

SQL: it gets worse

Query: unpaid orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer: EMPTY!

- This goes against our intuition: 3 orders, 2 payments.
- At least one must be unpaid!

SQL: it gets worse

Query: unpaid orders:

```
SELECT order_id FROM Orders
WHERE order_id NOT IN (SELECT order_id FROM Payments)
```

Answer: EMPTY!

- This goes against our intuition: 3 orders, 2 payments.
- At least one must be unpaid!

- SQL tells us that $|X| > |Y|$ and $X - Y = \emptyset$ are compatible.
- This is cast in stone (SQL standard).

SQL: quotes

"... this topic cannot be described in a manner that is simultaneously both comprehensive and comprehensible"
"Those SQL features are ... fundamentally at odds with the way the world behaves"

C. Date & H. Darwen, 'A Guide to SQL Standard'

"If you have any nulls in your database, you're getting wrong answers to some of your queries. What's more, you have no way of knowing, in general, just which queries you're getting wrong answers to; all results become suspect. You can never trust the answers you get from a database with nulls"

C. Date, 'Database in Depth'

The world, as theoreticians see it

In theory:

- We produce beautiful theoretical results
- Practitioners read our papers
- and build their systems as our results suggest.

The world, as theoreticians see it

In theory:

- We produce beautiful theoretical results
- Practitioners read our papers
- and build their systems as our results suggest.

It all looks rosy for us:

The world, as theoreticians see it

In practice:

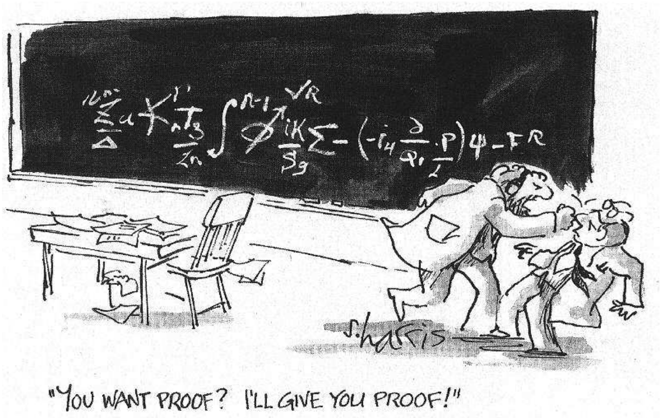
- We produce beautiful theoretical results
- Practitioners don't read our papers (usually)
- and build their systems as they please.

The world, as theoreticians see it

In practice:

- We produce beautiful theoretical results
- Practitioners don't read our papers (usually)
- and build their systems as they please.

We feel like:



Incomplete information: these scenarios don't apply

Because: we don't have the right theory yet.

We certainly don't yet have a theory that can be applied in practical settings

Incomplete information: these scenarios don't apply

Because: we don't have the right theory yet.

We certainly don't yet have a theory that can be applied in practical settings

Plan:

- A quick review of the theory of incompleteness
  - with lots of criticism
- An alternative approach
  - some basic ideas and early results
- List of things to do

Models of incompleteness

There are a few elements present in all models of incompleteness:

- A set of database objects $\mathcal{D}$
  - e.g., all databases (with or without nulls) of the same schema
- A set of complete objects $\mathcal{C} \subseteq \mathcal{D}$
  - databases of the same schema without nulls
- Semantics of incompleteness:

$$[\![\,]\!] : \mathcal{D} \;\to\; 2^{\mathcal{C}}$$

- The semantics of an incomplete object D is the set of complete objects it can possibly represent:

$$[\![D]\!] \;\subseteq\; \mathcal{C}$$

Naïve nulls

Also called marked nulls. Often arise in exchanging/integrating data:

Order(order_id,title) $\longrightarrow$ Customer(x), Prefers(x, title)

Naïve nulls

Also called marked nulls. Often arise in exchanging/integrating data:

$$\text{Order(order\_id,title)} \longrightarrow \text{Customer(x), Prefers(x, title)}$$

From Orders, we generate:

customer
$\perp_1$
$\perp_2$
$\perp_3$

customer	product
$\perp_1$	'SQL Standard'
$\perp_2$	'Database Systems'
$\perp_3$	'Logic'

Naïve nulls

Also called marked nulls. Often arise in exchanging/integrating data:

$$\text{Order(order\_id,title)} \quad \longrightarrow \quad \text{Customer(x)}, \;\; \text{Prefers(x, title)}$$

From Orders, we generate:

customer
$\bot_1$
$\bot_2$
$\bot_3$

customer	product
$\bot_1$	'SQL Standard'
$\bot_2$	'Database Systems'
$\bot_3$	'Logic'

Some nulls can repeat and denote the same value.

Naïve nulls

Also called marked nulls. Often arise in exchanging/integrating data:

$$\text{Order(order\_id,title)} \quad \longrightarrow \quad \text{Customer(x)}, \;\; \text{Prefers(x, title)}$$

From Orders, we generate:

customer
$\perp_1$
$\perp_2$
$\perp_3$

customer	product
$\perp_1$	'SQL Standard'
$\perp_2$	'Database Systems'
$\perp_3$	'Logic'

Some nulls can repeat and denote the same value.

Easily implementable (in fact used already: Clio, ++Spicy).

Naïve nulls

Also called marked nulls. Often arise in exchanging/integrating data:

$$\text{Order(order\_id,title)} \quad \longrightarrow \quad \text{Customer}(x), \;\; \text{Prefers}(x, \text{title})$$

From Orders, we generate:

customer
$\perp_1$
$\perp_2$
$\perp_3$

customer	product
$\perp_1$	'SQL Standard'
$\perp_2$	'Database Systems'
$\perp_3$	'Logic'

Some nulls can repeat and denote the same value.

Easily implementable (in fact used already: Clio, ++Spicy).

SQL model: an easy subcase – nulls don't repeat.

Two common semantics via valuations of nulls

pay_id	order_id	amount
p1	ord1	$\bot_1$
p2	$\bot_2$	\$50
p3	$\bot_3$	$\bot_1$

Two common semantics via valuations of nulls

pay_id	order_id	amount
p1	ord1	$\bot_1$
p2	$\bot_2$	\$50
p3	$\bot_3$	$\bot_1$

$$v(\bot_1) = \$100$$
$$v(\bot_2) = \text{ord2}$$
$$v(\bot_3) = \text{ord3}$$
$$\Longrightarrow$$

Two common semantics via valuations of nulls

pay_id	order_id	amount
p1	ord1	$\bot_1$
p2	$\bot_2$	\$50
p3	$\bot_3$	$\bot_1$

$v(\bot_1) = \$100$
$v(\bot_2) = \mathrm{ord2}$
$v(\bot_3) = \mathrm{ord3}$
$\Longrightarrow$

pay_id	order_id	amount
p1	ord1	\$100
p2	ord2	\$50
p3	ord3	\$100

Two common semantics via valuations of nulls

pay_id	order_id	amount
p1	ord1	$\bot_1$
p2	$\bot_2$	\$50
p3	$\bot_3$	$\bot_1$

$v(\bot_1) = \$100$
$v(\bot_2) = \text{ord2}$
$v(\bot_3) = \text{ord3}$
$\Longrightarrow$

pay_id	order_id	amount
p1	ord1	\$100
p2	ord2	\$50
p3	ord3	\$100

Closed-World-Assumption semantics (CWA semantics):

$$[\![D]\!]_{\text{cwa}} \quad = \quad \left\{ v(D) \;\middle|\; v \text{ is a valuation} \right\}$$

Two common semantics via valuations of nulls

pay_id	order_id	amount
p1	ord1	$\bot_1$
p2	$\bot_2$	\$50
p3	$\bot_3$	$\bot_1$

$v(\bot_1) = \$100$
$v(\bot_2) = \text{ord2}$
$v(\bot_3) = \text{ord3}$
$\Longrightarrow$

pay_id	order_id	amount
p1	ord1	\$100
p2	ord2	\$50
p3	ord3	\$100
p4	ord4	\$70
p5	ord5	\$65

Two common semantics via valuations of nulls

pay_id	order_id	amount
p1	ord1	$\perp_1$
p2	$\perp_2$	\$50
p3	$\perp_3$	$\perp_1$

$v(\perp_1) = \$100$
$v(\perp_2) = \text{ord2}$
$v(\perp_3) = \text{ord3}$
$\Longrightarrow$

pay_id	order_id	amount
p1	ord1	\$100
p2	ord2	\$50
p3	ord3	\$100
p4	ord4	\$70
p5	ord5	\$65

Open-World-Assumption semantics (OWA semantics):

$$[\![D]\!]_{\text{owa}} = \left\{ \text{complete } D' \;\middle|\; v(D) \subseteq D' \text{ for some valuation } v \right\}$$

Query answering

We want to answer queries Q over incomplete databases D, but only know how to answer them over complete databases D'

Query answering

We want to answer queries Q over incomplete databases D, but only know how to answer them over complete databases D'

Answers to Q in all possible worlds of D:

$$Q(\llbracket D \rrbracket) \quad = \quad \{Q(D') \mid D' \in \llbracket D \rrbracket\}$$

Query answering

We want to answer queries Q over incomplete databases D, but only know how to answer them over complete databases D'

Answers to Q in all possible worlds of D:

$$Q([\![D]\!]) \;=\; \{Q(D') \mid D' \in [\![D]\!]\}$$

First approach — strong representation systems:

$$[\![A]\!] \;=\; Q([\![D]\!])$$

The answer to Q on D is an object A that represents $Q([\![D]\!])$.

Strong representation systems are quite strong

Database

B
1
$\perp$

CWA semantics

Query $\sigma_{B=2}$

Strong representation systems are quite strong

Database

B
1
$\perp$

CWA semantics

Query $\sigma_{B=2}$

A possible world:

B
1
2

$$\sigma_{B=2}\left(\begin{array}{|c|}\hline B \\ \hline 1 \\ 2 \\ \hline\end{array}\right) = \begin{array}{|c|}\hline B \\ \hline 2 \\ \hline\end{array}.$$

Strong representation systems are quite strong

Database $\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline \bot \\ \hline \end{array}$ CWA semantics Query $\sigma_{B=2}$

A possible world: $\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 2 \\ \hline \end{array}$ $\sigma_{B=2}\left(\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 2 \\ \hline \end{array}\right) = \begin{array}{|c|}\hline B \\ \hline 2 \\ \hline \end{array}.$

Another possible world: $\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 3 \\ \hline \end{array}$ $\sigma_{B=2}\left(\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 3 \\ \hline \end{array}\right) = \emptyset.$

Strong representation systems are quite strong

Database $\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline \bot \\ \hline \end{array}$ CWA semantics Query $\sigma_{B=2}$

A possible world: $\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 2 \\ \hline \end{array}$ $\sigma_{B=2}\left(\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 2 \\ \hline \end{array}\right) = \begin{array}{|c|}\hline B \\ \hline 2 \\ \hline \end{array}.$

Another possible world: $\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 3 \\ \hline \end{array}$ $\sigma_{B=2}\left(\begin{array}{|c|}\hline B \\ \hline 1 \\ \hline 3 \\ \hline \end{array}\right) = \emptyset.$

No A so that both $\emptyset$ and $\begin{array}{|c|}\hline B \\ \hline 2 \\ \hline \end{array}$ in $[\![A]\!]_{\text{cwa}}$:

- only empty tables have $\emptyset$ in their semantics.

When strong is too strong, we need something weak

Certain answers: we are certain a tuple t is in the answer if it is in the answer in all possible worlds.

$$\boxed{\text{certain}(Q, D) \quad = \quad \bigcap_{D' \in [\![D]\!]} Q(D')}$$

When strong is too strong, we need something weak

Certain answers: we are certain a tuple t is in the answer if it is in the answer in all possible worlds.

$$\boxed{\text{certain}(Q, D) \;=\; \bigcap_{D' \in [\![D]\!]} Q(D')}$$

- Came out of weak representation systems:

$$[\![A]\!] \;=\; Q([\![D]\!]) \;\; \text{is replaced by} \;\; [\![A]\!] \;\sim\; Q([\![D]\!])$$

  - $\sim$ is an equivalence relation, weaker than equality
  - Idea: certain information in $[\![A]\!]$ and $Q([\![D]\!])$ is the same

Certain answers

Look again at

SELECT pay_id FROM Payments
WHERE amount $\geq$ 50 OR amount $<$ 50

on

pay_id	order_id	amount
p1	ord1	$\perp$
p2	$\perp'$	\$50

SQL answer =

pay_id
p2

Certain answer =

pay_id
p1
p2

Certain answers evaluation

Treat nulls as values: $\bot_1 = \bot_1$ but $\bot_1 \neq \bot_2$ and $\bot_1 \neq 1$, etc.
Often called naïve evaluation.

R:

A	B
1	$\bot_1$
2	$\bot_2$

S:

B	C
$\bot_1$	3
$\bot_2$	4

Certain answers evaluation

Treat nulls as values: $\bot_1 = \bot_1$ but $\bot_1 \neq \bot_2$ and $\bot_1 \neq 1$, etc.
Often called naïve evaluation.

R:

A	B
1	$\bot_1$
2	$\bot_2$

S:

B	C
$\bot_1$	3
$\bot_2$	4

$\pi_{A,C}(R \bowtie_B S) \Longrightarrow$

A	C
1	3
2	4

= certain answer

Certain answers evaluation

Treat nulls as values: $\bot_1 = \bot_1$ but $\bot_1 \neq \bot_2$ and $\bot_1 \neq 1$, etc.
Often called naïve evaluation.

R:

A	B
1	$\bot_1$
2	$\bot_2$

S:

B	C
$\bot_1$	3
$\bot_2$	4

$\pi_{A,C}(R \ \bowtie_B \ S) \quad \Longrightarrow$

A	C
1	3
2	4

$=$ certain answer

$\pi_A(R) \ - \rho_{A \leftarrow B}(\pi_B(S)) \quad \Longrightarrow$

A
1
2

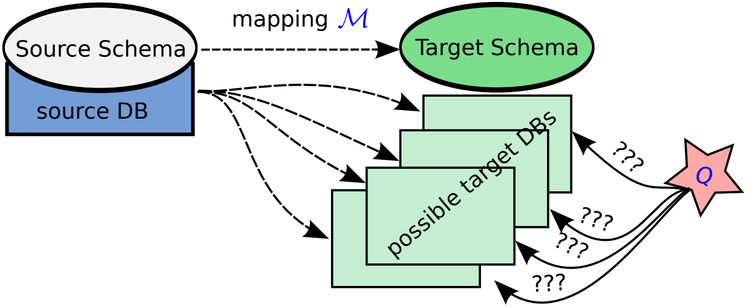
$\neq$ certain answer

Applications and certain answers

Certain answers is the standard method of query answering in applications of incompleteness:

- Data exchange
- Data integration
- Consistent query answering

Data Exchange



- A mapping $\mathcal{M}$ relates source and target schemas.
- A query Q is over the target schema.
- Potentially many target databases satisfying $\mathcal{M}$:
  - only one is materialized
- How to answer Q?

Data Exchange – certain answers

- Given: a source S, a mapping $\mathcal{M}$, a target query Q.
- Possible targets:

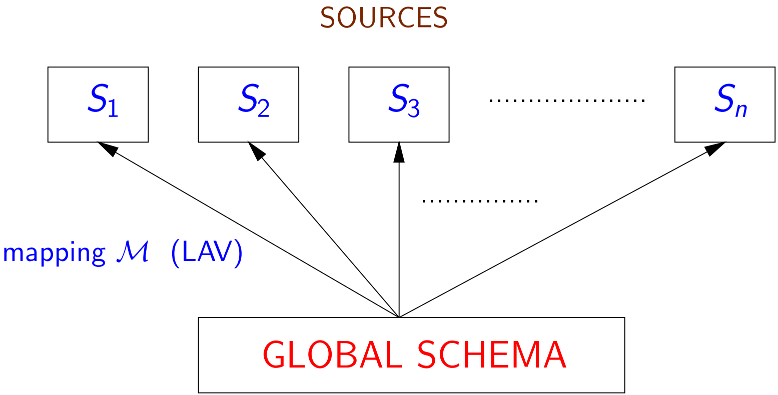
$$[\![S]\!]_{\mathcal{M}} \quad = \quad \{T \mid S \text{ and } T \text{ satisfy } \mathcal{M}\}$$

- Query answering:

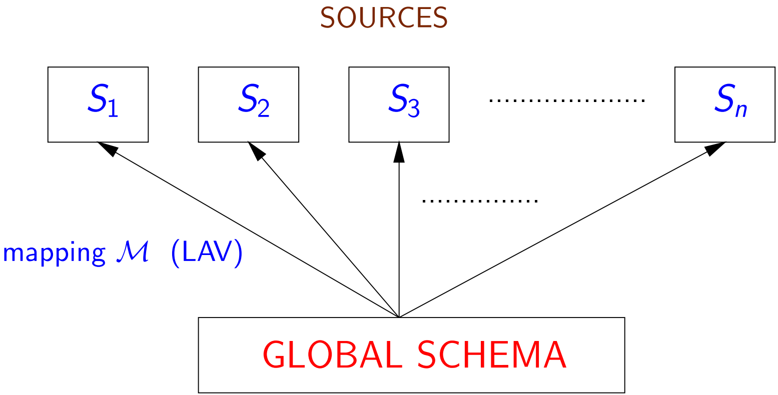
$$\text{certain}_{\mathcal{M}}(Q, S) \quad = \quad \bigcap_{T \in [\![S]\!]_{\mathcal{M}}} Q(T)$$

- We want tuples that are in the answer for all possible targets.

Virtual data integration



Virtual data integration



- Global schema database is virtual.
- Possibly multiple instances of the global schema satisfying $\mathcal{M}$.

Data Integration – certain answers

- A query Q is posed against a virtual global schema database.
- We only have access to the sources $\boldsymbol{S} = (S_1, \ldots, S_n)$
- Possible virtual databases:

$$[\![\boldsymbol{S}]\!]_{\mathcal{M}} = \{D \text{ of global schema } \mid D \text{ and } \boldsymbol{S} \text{ satisfy } \mathcal{M}\}$$

Data Integration – certain answers

- A query Q is posed against a virtual global schema database.
- We only have access to the sources $\boldsymbol{S} = (S_1, \ldots, S_n)$
- Possible virtual databases:

$$[\![\boldsymbol{S}]\!]_{\mathcal{M}} = \{D \text{ of global schema } \mid D \text{ and } \boldsymbol{S} \text{ satisfy } \mathcal{M}\}$$

- Query answering:

$$\text{certain}_{\mathcal{M}}(Q, \boldsymbol{S}) = \bigcap_{D \in [\![\boldsymbol{S}]\!]_{\mathcal{M}}} Q(D)$$

- We want tuples that are in the answer regardless of a specific instance of a global schema.

Inconsistent databases

- Often arise in data integration.
- Functional dependency name $\rightarrow$ salary but conflicting tuples (John, 50K) and (John, 60K) in two sources.
- What if we cannot clean the data and must keep inconsistent records?

Main issue: correct query answering.

Inconsistent databases – certain answers

- a database D, a query Q, a set of integrity constraints Σ.
- D violates Σ.
- Repairs: minimal changes that restore integrity
  - for functional dependencies, tuple removals

$$[\![D]\!]_\Sigma \quad = \quad \{D' \mid D' \text{ is a repair of } D \text{ wrt } \Sigma\}$$

Inconsistent databases – certain answers

- a database D, a query Q, a set of integrity constraints Σ.
- D violates Σ.
- Repairs: minimal changes that restore integrity
  - for functional dependencies, tuple removals

$$[\![D]\!]_\Sigma \quad = \quad \{D' \mid D' \text{ is a repair of } D \text{ wrt } \Sigma\}$$

- Query answering:

$$\text{certain}_\Sigma(Q, D) \quad = \quad \bigcap_{D' \in [\![D]\!]_\Sigma} Q(D')$$

We want tuples that are in the answer for all possible repairs.

Typical theoretical results

- Foundational papers:
  - Imielinski/Lipski 1984
  - Abiteboul/Kanellakis/Grahne 1991
- Null-free tuples in the result of naïve evaluation is certain answers for positive relational algebra ($\sigma, \pi, \bowtie, \cup$).
  - under both CWA and OWA
  - low complexity
- The result is optimal for OWA
- For full relational calculus, the complexity is:
  - coNP-complete under CWA
  - undecidable under OWA
  - even for data complexity

Summary

- Practice: sacrifice correctness for efficiency
  - cast in stone: SQL standard
- Theory: standard notions of correctness
  - cast in stone: representation systems, certain answers

- Theoretical notions of correctness quickly lead to high complexity but they aren't really questioned.

Summary

- Practice: sacrifice correctness for efficiency
  - cast in stone: SQL standard
- Theory: standard notions of correctness
  - cast in stone: representation systems, certain answers

- Theoretical notions of correctness quickly lead to high complexity but they aren't really questioned.

- The two sides got it wrong, and they are not talking...

Summary

- Practice: sacrifice correctness for efficiency
  - cast in stone: SQL standard
- Theory: standard notions of correctness
  - cast in stone: representation systems, certain answers

- Theoretical notions of correctness quickly lead to high complexity but they aren't really questioned.

- The two sides got it wrong, and they are not talking...
- Can we get efficiency and correctness guarantees at the same time?
- Efficiency = can use existing DBMSs for query evaluation (perhaps with just slight modifications)

Theory is not immune from criticism

- SQL's handling of nulls has been criticized a lot, but theoretical approaches have been mainly spared.
- But even the most basic notions are questionable: strong/weak representation systems, certain answers.
- We now illustrate a few problematic points.

Semantics of query answering

- Strong representation systems: $[\![A]\!] \ = \ Q([\![D]\!])$
  - A represents answers in all possible worlds

Semantics of query answering

- Strong representation systems: $[\![A]\!] \;=\; Q([\![D]\!])$
  - A represents answers in all possible worlds
- But why should the answers have the same semantics $[\![\;]\!]$?

Semantics of query answering

- Strong representation systems: $[\![A]\!] = Q([\![D]\!])$
  - A represents answers in all possible worlds
- But why should the answers have the same semantics $[\![\ ]\!]$?

- There is really no need for this.
  - XML-to-relational or relational-to-XML queries
  - why should results be open/closed if inputs are?

Semantics of query answering

- Strong representation systems: $[\![A]\!] = Q([\![D]\!])$
  - A represents answers in all possible worlds
- But why should the answers have the same semantics $[\![\;]\!]$?

- There is really no need for this.
  - XML-to-relational or relational-to-XML queries
  - why should results be open/closed if inputs are?

- One should be more flexible: $(\!|A|\!) = Q([\![D]\!])$
- But then what is the semantics of query answers $(\!|\;|\!)$?
  - How does it depend on $[\![\;]\!]$ and Q?

Why intersection?

- We always use intersection to define certain answers:

$$\text{certain}(Q, D) \;=\; \bigcap \{ Q(D') \;\mid\; D' \in [\![D]\!] \}$$

- But is this the only way? Is this the right way?
  - doesn't make sense beyond the relations: e.g., for XML queries

Why intersection?

- We always use intersection to define certain answers:

$$\text{certain}(Q, D) \;=\; \bigcap \{ Q(D') \;\mid\; D' \in [\![D]\!] \}$$

- But is this the only way? Is this the right way?
  - doesn't make sense beyond the relations: e.g., for XML queries
- More importantly, do we really get certain information?
  - Intersection takes information away from potential answers
  - But we are removing data, not information!
- Removing data can actually add information.

Why intersection? cont'd

- A single relation D:

1	2
3	$\bot$

- Query Q: return D itself
- Semantics: CWA (interpret nulls, don't add tuples)
- $\text{certain}(Q, D) \quad = \quad \boxed{1 \mid 2}$

Why intersection? cont'd

- A single relation D:

1	2
3	$\bot$

- Query Q: return D itself
- Semantics: CWA (interpret nulls, don't add tuples)

- $\text{certain}(Q, D) \;=\; $

1	2

- Removing the tuple $(3, \bot)$ adds information under CWA:

  "there is no tuple whose first component is 3."

- "Certain" answers are far from being certain!

High complexity bounds

Too much emphasis on high complexity bounds.

A typical picture:

- Certain answers computable efficiently for
  - conjunctive queries
  - sometimes their unions
  - sometimes just a subclass
  - maybe small extensions (inequality, Boolean combinations)
- Beyond that, high lower bounds
  - coNP and up, even undecidable
- One concedes defeat and moves over to the next problem.

But we still have to evaluate those queries somehow!

What to do?

- Goal: bridge correctness and efficiency.
- Sad news: not much to rely on.

What to do?

- Goal: bridge correctness and efficiency.
- Sad news: not much to rely on.

- But we can be optimistic and positive.
  This is an opportunity to rethink the whole subject.

What to do?

- Goal: bridge correctness and efficiency.
- Sad news: not much to rely on.

- But we can be optimistic and positive.
  This is an opportunity to rethink the whole subject.

Next – we propose an alternative approach:

- work in progress, no claim this is the last word
- but one must start somewhere!

and lots of issues still to be dealt with.

Basic idea

Combine three previously used approaches to incomplete information:

1. Certain answers, strong/weak representation systems;
2. Information orderings (1990s)
   - $D \preceq D'$ means that D has less information than D'
3. Databases as logical theories (1980s)
   - a database is a set of facts given by formulas

Orderings

- Popular in the early 1990s (Buneman, Ohori, PL people)
- Developed mostly for SQL's view of nulls (non-repeating nulls)

Orderings

- Popular in the early 1990s (Buneman, Ohori, PL people)
- Developed mostly for SQL's view of nulls (non-repeating nulls)
- Idea: lift simple orderings to more complex data structures
  - A null has less information that a value, e.g. $\bot \preceq 1$
  - Extend to tuples, e.g. $(1, \bot, \bot') \preceq (1, \bot, 2)$
  - Extend to sets, e.g. $X \preceq Y \quad \Leftrightarrow \quad \forall x \in X\ \exists y \in Y : \quad x \preceq y$

Orderings

- Popular in the early 1990s (Buneman, Ohori, PL people)
- Developed mostly for SQL's view of nulls (non-repeating nulls)

- Idea: lift simple orderings to more complex data structures
  - A null has less information that a value, e.g. $\perp \preceq 1$
  - Extend to tuples, e.g. $(1, \perp, \perp') \preceq (1, \perp, 2)$
  - Extend to sets, e.g. $X \preceq Y \quad \Leftrightarrow \quad \forall x \in X\ \exists y \in Y : \quad x \preceq y$

- Results:
  - orderings for different semantics
  - connections with programming semantics
  - influence on language design

Databases as logical theories

- An older approach, from the 1980s, advocated by Reiter
- A database D is viewed as a formula φ_D, or even a theory

D =

1	2
3	$\perp$

under OWA is seen as

$$\varphi_D \quad = \quad \exists x \; D(1,2) \;\wedge\; D(3,x)$$

Databases as logical theories

- An older approach, from the 1980s, advocated by Reiter
- A database D is viewed as a formula φ_D, or even a theory

$D = $

1	2
3	$\perp$

under OWA is seen as

$$\varphi_D \;=\; \exists x\; D(1,2) \;\wedge\; D(3,x)$$

- Query answering becomes logical implication.
  To see if $Q(\bar{t})$ is true with certainty, check whether

$$\varphi_D \;\models\; Q(\bar{t})$$

Databases as logical theories

- An older approach, from the 1980s, advocated by Reiter
- A database D is viewed as a formula φ_D, or even a theory

D =

1	2
3	$\bot$

under OWA is seen as

$$\varphi_D \;=\; \exists x\; D(1,2) \;\wedge\; D(3,x)$$

- Query answering becomes logical implication.
  To see if $Q(\bar{t})$ is true with certainty, check whether

$$\varphi_D \;\models\; Q(\bar{t})$$

- Didn't really take off back then:
  - complexity issues
  - finite vs infinite implication

Old approaches

They didn't deliver back then, and were dismissed.

- Reason 1: didn't concentrate on the right questions;
- Reason 2: too deeply rooted in the relational world;
  - a concrete model obscures the view!
- Reason 3: they were pursued in isolation.

Old approaches

They didn't deliver back then, and were dismissed.

- Reason 1: didn't concentrate on the right questions;
- Reason 2: too deeply rooted in the relational world;
  - a concrete model obscures the view!
- Reason 3: they were pursued in isolation.

Idea: combine the approaches

- but take just what's needed from them, no more,
- and don't be tied to just one data model.

Reminder: the basic model

- A set of database objects $\mathcal{D}$
  - sets of all databases (with or without nulls) of the same schema
- A set of complete objects $\mathcal{C} \subseteq \mathcal{D}$
  - databases of the same schema without nulls
- Semantics of incompleteness: $[\![\,]\!] : \mathcal{D} \rightarrow 2^{\mathcal{C}}$
  - the set of all complete objects represented by an incomplete object

$$[\![D]\!] \subseteq \mathcal{C}$$

Adding order

When is D less informative than D'?

Adding order

When is D less informative than D'?

- If we know nothing about D, every database is possible.

Adding order

When is D less informative than D'?

- If we know nothing about D, every database is possible.
- The more we learn about D, the fewer possible worlds there are.

Adding order

When is D less informative than D'?

- If we know nothing about D, every database is possible.
- The more we learn about D, the fewer possible worlds there are.

Information ordering:

$$D \preceq D' \quad \Leftrightarrow \quad [\![D']\!] \subseteq [\![D]\!]$$

The more informative an object is, the fewer objects it denotes.

Adding knowledge

A set $\mathcal{F}$ of formulae φ that may hold in database objects.

A minimal requirement: $[\![D]\!]$ can be described by a formula.

Adding knowledge

A set $\mathcal{F}$ of formulae φ that may hold in database objects.

A minimal requirement: $[\![D]\!]$ can be described by a formula.

- Example: $D \; = \; \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & \bot \\ \hline \end{array}$

Adding knowledge

A set $\mathcal{F}$ of formulae φ that may hold in database objects.

A minimal requirement: $[\![D]\!]$ can be described by a formula.

- Example: $D \;=\; \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & \bot \\ \hline\end{array}$
- under OWA: $\exists x \; D(1,2) \;\wedge\; D(3,x)$

Adding knowledge

A set $\mathcal{F}$ of formulae φ that may hold in database objects.

A minimal requirement: $[\![D]\!]$ can be described by a formula.

- Example: $D \; = \;$

1	2
3	$\bot$

- under OWA: $\exists x \; D(1,2) \; \wedge \; D(3,x)$
- under CWA:

$$\exists x \left(\begin{array}{ll} & D(1,2) \;\wedge\; D(3,x) \\ \wedge & \forall y,z \; D(y,z) \rightarrow \left(\begin{array}{ll} & (y,z) = (1,2) \\ \vee & (y,z) = (3,x) \end{array} \right) \end{array} \right)$$

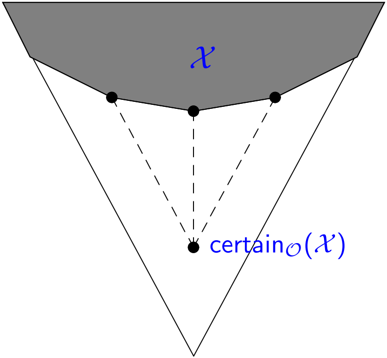
Defining certainty

- To understand certain answers, we need to define certainty in the set $Q([\![D]\!])$
- So the first basic task:

define certainty in a set of objects $\mathcal{X} \subseteq \mathcal{D}$

- It can be represented in two ways:
  - as object
  - as knowledge

Certainty as object: $\mathrm{certain}_{\mathcal{O}}(\mathcal{X})$



- Could not exceed the information content of objects in $\mathcal{X}$:
  - $\mathrm{certain}_{\mathcal{O}}(\mathcal{X}) \preceq D$ for all $D \in \mathcal{X}$;
- Must be the most informative among such objects;
- $\mathrm{certain}_{\mathcal{O}}(\mathcal{X}) = \bigwedge \mathcal{X}$ — the greatest lower bound of $\mathcal{X}$.

Certainty as knowledge: $\text{certain}_{\mathcal{K}}(\mathcal{X})$

- Formulae from $\mathcal{F}$ say what we know about objects.
- What we know with certainty about $\mathcal{X}$:

$$\text{Theory}(\mathcal{X}) \quad = \quad \{\varphi \in \mathcal{F} \mid \varphi \text{ is true in every } D \in \mathcal{X}\}$$

Certainty as knowledge: $\text{certain}_{\mathcal{K}}(\mathcal{X})$

- Formulae from $\mathcal{F}$ say what we know about objects.
- What we know with certainty about $\mathcal{X}$:

$$\text{Theory}(\mathcal{X}) \;=\; \{\varphi \in \mathcal{F} \mid \varphi \text{ is true in every } D \in \mathcal{X}\}$$

- Idea of weak representation systems: $\text{certain}_{\mathcal{K}}(\mathcal{X})$ is such that

$$\text{certain}_{\mathcal{K}}(\mathcal{X}) \;\sim\; \text{Theory}(\mathcal{X})$$

Certainty as knowledge: $\text{certain}_{\mathcal{K}}(\mathcal{X})$

- Formulae from $\mathcal{F}$ say what we know about objects.
- What we know with certainty about $\mathcal{X}$:

$$\text{Theory}(\mathcal{X}) \quad = \quad \{\varphi \in \mathcal{F} \mid \varphi \text{ is true in every } D \in \mathcal{X}\}$$

- Idea of weak representation systems: $\text{certain}_{\mathcal{K}}(\mathcal{X})$ is such that

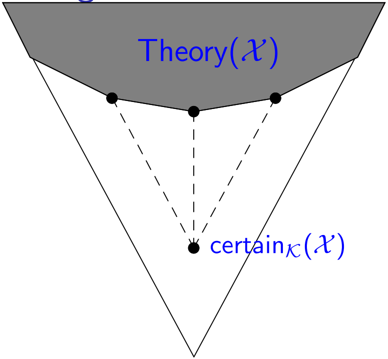
$$\text{certain}_{\mathcal{K}}(\mathcal{X}) \sim \text{Theory}(\mathcal{X})$$

- What is the equivalence $\sim$ between formulas?

  they are satisfied in exactly the same objects.

  - much more disciplined that the equivalence of WRSs

Certainty as knowledge: another look



- Ordering: implication (or containment) $\varphi \to \varphi'$
- certain$_{\mathcal{K}}(\mathcal{X})$ is the greatest lower bound of Theory($\mathcal{X}$) in this order.
- Essentially, certain$_{\mathcal{K}}(\mathcal{X}) \; = \; \bigwedge$ Theory($\mathcal{X}$).

Certain answers to queries

- Certain information in $Q(\llbracket D \rrbracket) \;=\; \{Q(D') \;\mid\; D' \in \llbracket D \rrbracket\}$
  Two ways of representing it:

$$\begin{aligned} \text{as objects}: \quad & \text{certain}_{\mathcal{O}}(Q, D) \;=\; \text{certain}_{\mathcal{O}}(Q(\llbracket D \rrbracket)) \\ \text{as knowledge}: \quad & \text{certain}_{\mathcal{K}}(Q, D) \;=\; \text{certain}_{\mathcal{K}}(Q(\llbracket D \rrbracket)) \end{aligned}$$

Certain answers to queries

- Certain information in $Q(\llbracket D \rrbracket) = \{Q(D') \mid D' \in \llbracket D \rrbracket\}$
  Two ways of representing it:

$$\text{as objects}: \quad \text{certain}_{\mathcal{O}}(Q, D) \quad = \quad \text{certain}_{\mathcal{O}}(Q(\llbracket D \rrbracket))$$

$$\text{as knowledge}: \quad \text{certain}_{\mathcal{K}}(Q, D) \quad = \quad \text{certain}_{\mathcal{K}}(Q(\llbracket D \rrbracket))$$

- Queries are mappings $Q : \mathcal{D} \to \mathcal{D}'$ between two sets of objects
  - e.g., sets of databases of different schemas
- $\mathcal{D}$ and $\mathcal{D}'$ need not have the same semantics!

Queries and semantics

- The basic principle:

we know more about the input to Q

$$\Downarrow$$

we know more about the output of Q

Queries and semantics

- The basic principle:

$$\text{we know more about the input to } Q$$
$$\Downarrow$$
$$\text{we know more about the output of } Q$$

- Looks natural? Ignored by most of the work on incompleteness.

Queries and semantics

- The basic principle:

$$\text{we know more about the input to } Q \\ \Downarrow \\ \text{we know more about the output of } Q$$

- Looks natural? Ignored by most of the work on incompleteness.
- Query $Q : \mathcal{D} \to \mathcal{D}'$
  - $\mathcal{D}$ and $\mathcal{D}'$ have semantics $[\![\,]\!]$ and $(\!|\,|\!)$
  - and information orderings $\preceq$ and $\preccurlyeq$
- We want:

$$D_1 \preceq D_2 \quad \Leftrightarrow \quad Q(D_1) \preccurlyeq Q(D_2)$$

Queries and semantics

- The basic principle:

$$D_1 \preceq D_2 \quad \Leftrightarrow \quad Q(D_1) \preccurlyeq Q(D_2)$$

  Queries preserve informativeness.

- Why was it ignored?
- Because one assumed the same semantics for inputs and outputs!
  - even though a priori there is no good reason for it.

One more condition and we are ready

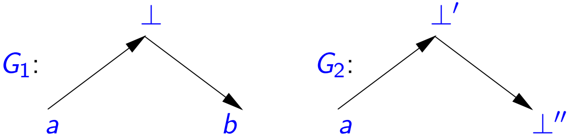
- Queries are typically written in logical languages
  - first-order logic, datalog, etc

  and they cannot distinguish isomorphic structures
- Known as genericity.

One more condition and we are ready

- Queries are typically written in logical languages
  - first-order logic, datalog, etc

  and they cannot distinguish isomorphic structures
- Known as genericity.

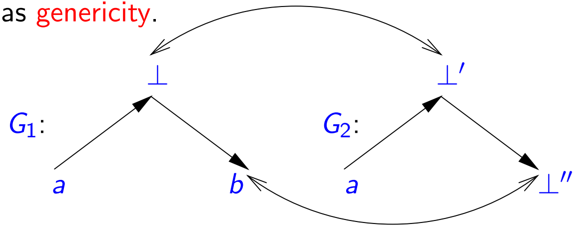


- Query Q: there is a path of length two starting in node a.
- Q cannot distinguish G_1 and G_2

One more condition and we are ready

- Queries are typically written in logical languages
  - first-order logic, datalog, etc

  and they cannot distinguish isomorphic structures
- Known as genericity.



- Query Q: there is a path of length two starting in node a.
- Q cannot distinguish G_1 and G_2 :
  there is an isomorphism preserving a: $\perp \leftrightarrow \perp'$ and $b \leftrightarrow \perp''$

Efficiency and correctness at once

Let Q

- preserve informativeness, and
- be generic.

Then

$$\text{certain}_{\mathcal{O}}(Q, D) \; = \; Q(D)$$

Efficiency and correctness at once

Let Q

- preserve informativeness, and
- be generic.

Then

$$\text{certain}_{\mathcal{O}}(Q, D) \;=\; Q(D)$$

And $\text{certain}_{\mathcal{K}}(Q, D)$ is the formula defining the semantics of $Q(D)$.

Efficiency and correctness at once

Let Q

- preserve informativeness, and
- be generic.

Then

$$\text{certain}_{\mathcal{O}}(Q, D) \;=\; Q(D)$$

And $\text{certain}_{\mathcal{K}}(Q, D)$ is the formula defining the semantics of $Q(D)$.

Magic: correct answers for free.

Efficiency and correctness at once

Let Q

- preserve informativeness, and
- be generic.

Then

$$\text{certain}_{\mathcal{O}}(Q, D) \;=\; Q(D)$$

And $\text{certain}_{\mathcal{K}}(Q, D)$ is the formula defining the semantics of $Q(D)$.

Magic: correct answers for free.

Efficiency guaranteed too: can use existing query evaluation algorithms.

The price of magic

- The right semantics of query answers:
  - it must ensure that Q preserves informativeness

The price of magic

- The right semantics of query answers:
  - it must ensure that Q preserves informativeness
- A set of formulae $\mathcal{F}$ capable of at least defining semantics of objects
  - without it, the result doesn't hold
  - but often it's easy to find one

Good bye intersection

No need to use it to get certain answers.

Recall our example: a single relation $D = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & \perp \\ \hline \end{array}$

Query Q: return D itself.

Good bye intersection

No need to use it to get certain answers.

Recall our example: a single relation $D = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & \bot \\ \hline\end{array}$

Query Q: return D itself.

- Old way: $\text{certain}(Q, D) = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline\end{array}$

Good bye intersection

No need to use it to get certain answers.

Recall our example: a single relation D $=$

1	2
3	$\perp$

Query Q: return D itself.

- Old way: $\text{certain}(Q, D)$ $=$

1	2

- New way: $\text{certain}_{\mathcal{O}}(Q, D)$ $=$

1	2
3	$\perp$

Good bye intersection

No need to use it to get certain answers.

Recall our example: a single relation D =

1	2
3	$\perp$

Query Q: return D itself.

- Old way: $\text{certain}(Q, D)$ =

1	2

- New way: $\text{certain}_{\mathcal{O}}(Q, D)$ =

1	2
3	$\perp$

We keep information about the tuple with first component 3.

When the input/output semantics coincide

- This is the setting considered most often
- Queries preserve informativeness $\Rightarrow$ efficient evaluation
  - need to understand what it means under OWA and CWA
- Good news: orderings have nice descriptions

When the input/output semantics coincide

- This is the setting considered most often
- Queries preserve informativeness $\Rightarrow$ efficient evaluation
  - need to understand what it means under OWA and CWA
- Good news: orderings have nice descriptions

- Information ordering under OWA

$$D \preceq_{\mathsf{owa}} D' \quad \Leftrightarrow \quad \exists \text{ homomorphism } D \mapsto D'$$

When the input/output semantics coincide

- This is the setting considered most often
- Queries preserve informativeness $\Rightarrow$ efficient evaluation
  - need to understand what it means under OWA and CWA
- Good news: orderings have nice descriptions

- Information ordering under OWA

$$D \preceq_{\mathsf{owa}} D' \quad \Leftrightarrow \quad \exists \text{ homomorphism } D \mapsto D'$$

- Information ordering under CWA

$$D \preceq_{\mathsf{cwa}} D' \quad \Leftrightarrow \quad \exists \text{ restricted homomorphism } D \mapsto D'$$

  - restricted = strong onto

Queries preserving informativeness

Preserving informativeness $=$ preservation under (restricted) homomorphisms

- well known and studied concept in logic
  - applications in database theory and AI (constraint satisfaction)
- It is easier to preserve informativeness under CWA

Queries preserving informativeness

Preserving informativeness
=
preservation under (restricted) homomorphisms

- well known and studied concept in logic
  - applications in database theory and AI (constraint satisfaction)
- It is easier to preserve informativeness under CWA

If Q is a positive relational algebra query $(\sigma, \pi, \bowtie, \cup)$ then

$$\text{certain}_{\mathcal{O}}(Q, D) \;=\; Q(D)$$

when both inputs and outputs have OWA semantics.

Certain answers under CWA

- Easier to preserve informativeness under CWA $\Rightarrow$ can extend positive relational algebra and still get correct answers efficiently

Certain answers under CWA

- Easier to preserve informativeness under CWA $\Rightarrow$ can extend positive relational algebra and still get correct answers efficiently
- Reminder – relational algebra division

A	B
a	1
a	2
b	1

$\div$

B
1
2

$=$

A
a

Certain answers under CWA

- Easier to preserve informativeness under CWA $\Rightarrow$ can extend positive relational algebra and still get correct answers efficiently
- Reminder – relational algebra division

A	B
a	1
a	2
b	1

$\div$

B
1
2

$=$

A
a

For a query Q expressed with

- $\sigma, \pi, \bowtie, \cup$, and
- $R \div S$, where S is a relation in the database,

$$\text{certain}_{\mathcal{O}}(Q, D) \;=\; Q(D)$$

when both inputs and outputs have CWA semantics.

Summary

We can achieve correctness and efficiency at the same time.

What we needed to do:

Summary

We can achieve correctness and efficiency at the same time.

What we needed to do:

- Drop the old intersection-based approach
  - there is life both within and beyond the relational model
- Combine previously used approaches:
  - rely on orderings to compare informativeness
  - relate orderings and semantics
  - introduce knowledge bases for query answers
    - may not be visible to the user, but needed by us to provide correctness guarantees
- Insist on the right semantics of query answers

What to do #1

- Extending query classes
  - How to handle negation? full relational algebra?
  - What is the appropriate semantics of query answers?
  - How to deal with aggregation? Intervals, distributions?
  - Recursion: datalog and fragments.
- Evaluation techniques
  - Is computing $Q(D)$ enough?
  - If not, what extra information needs to be computed?
  - How easy is it?

What to do #2

- Handling constraints
  - Many constraints – keys, foreign keys, inclusion constraints – come from classes which are hard to evaluate with certainty
  - Heavy use of universal quantification and negation
  - Led to ad hoc definitions in the past
  - How to reconcile integrity constraints and incompleteness?
- XML data
  - Incompleteness at the level of both structure and data
  - Only restrictive classes admit efficient evaluation under the old definition
  - How to apply the new theory?
  - How to incorporate constraints?
  - Emphasis on XML-to-XML queries (unlike most earlier work).

What to do #3

- Graph data and RDF
  - Models of incompleteness for graphs? Topology vs data.
  - Semantics, orderings, certainty.
  - Applications to RDF data.
- Applications
  - Revisit applications relying on certain answers.
  - Find proper semantics?
  - Can we extend good classes for data integration/exchange?
  - Can database repairs fit into our approach?

Thanks to:

- Marcelo Arenas
- Pablo Barceló
- Claire David
- Diego Figueira
- Amélie Gheerbrant
- Filip Murlak
- Juan Reutter
- Cristina Sirangelo
- Domagoj Vrgoč
- Limsoon Wong

Thanks to:

- Marcelo Arenas
- Pablo Barceló
- Claire David
- Diego Figueira
- Amélie Gheerbrant
- Filip Murlak
- Juan Reutter
- Cristina Sirangelo
- Domagoj Vrgoč
- Limsoon Wong

Questions?