Logic, Data, and Incomplete Information

Phokion G. Kolaitis

UC Santa Cruz & IBM Research

Lecture 3

A Fundamental Algorithmic Problem

- The Query Evaluation Problem:
  Given a query q and a database instance D, find q(D).

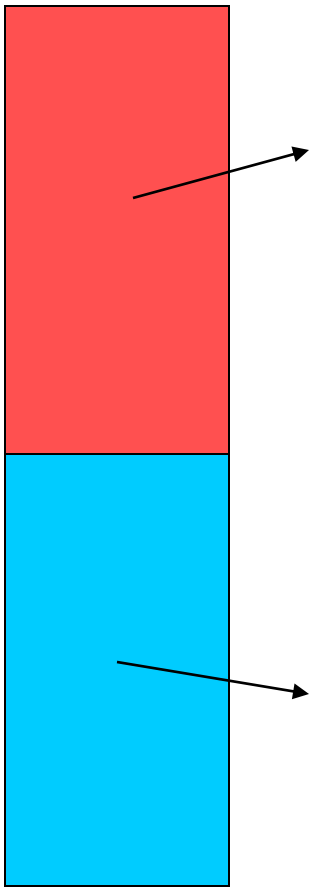
  Note:
  If q is a k-ary query for some $k \geq 1$, then q(D) is a relation of arity k.

- The Boolean Query Evaluation Problem:
  Given a Boolean query q and a database D, find q(D).

  Note:
  Here, q(D) =1 if q is true on D and q(D) = 0 if q is false on D.

Question: How "easy" or "difficult" the query evaluation problem is?

Decidable Problems & Computational Complexity



- Computational Complexity is the quantitative study of decidable problems.

The Five Basic Computational Complexity Classes

- LOGSPACE (or, L): All decision problems solvable by a TM using extra memory bounded by a logarithmic amount in the input size.
- NLOGSPACE (or, NL): All decision problems solvable by a NTM using extra memory bounded by a logarithmic amount in the input size.
- P (or, PTIME): All decision problems solvable by a TM in time bounded by some polynomial in the input size.
- NP: All decision problems solvable by a NTM in time bounded by some polynomial in the input size.
- PSPACE: All decision problems solvable by a TM using memory bounded by a polynomial in the input size.

The Five Basic Computational Complexity Classes

Theorem:

- The following inclusions hold:
  $\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE}$.

- Moreover, it is known that $\text{LOGSPACE} \subset \text{PSPACE}$.

- No other proper inclusion between these classes is known at present. In particular, it is not known whether P = NP.

Note:

- The question: “is P = NP?” is the central open problem in computational complexity.
- It is one of the Millennium Prize Problems put forward by the Clay Mathematics Institute.

Complete Problems for Complexity Classes

- Definition: A decision problem Q is PSPACE-complete if
  - Q is in PSPACE.
  - If Q' is in PSPACE, then there is a polynomial-time computable function f such that for every input x, we have that

$$x \in Q' \iff f(x) \in Q.$$

- Definition: A decision problem Q is NP-complete if
  - Q is in NP.
  - If Q' is in NP, then there is a polynomial-time computable function f such that for every input x, we have that

$$x \in Q' \iff f(x) \in Q.$$

- Such an f is a polynomial-time reduction of Q' to Q ($Q' \preccurlyeq_p Q$)

NP-Completeness

Definition:

- CNF-formula: a propositional formula in conjunctive normal form.
- SAT (Satisfiability): Given a CNF-formula, is it satisfiable?

Theorem (S. Cook – 1971)

SAT is NP-complete

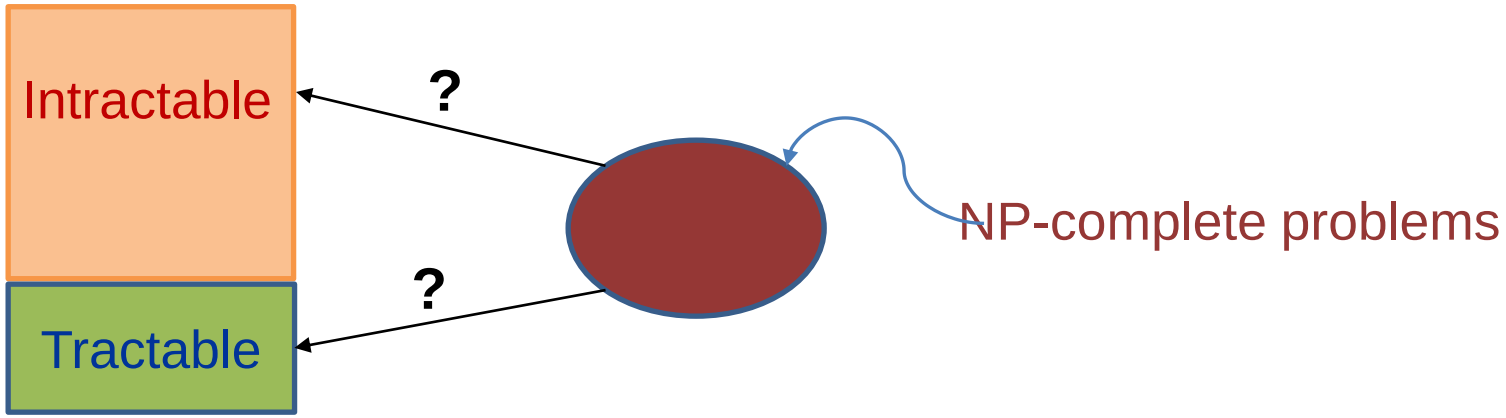
Hint of Proof:

- Membership in NP is easy (uses the fact that the model checking problem for propositional formulas is in P).
- NP-hardness: Encoding of non-deterministic Turing machines running in polynomial-time.

More on NP-Complete Problems

Note: The “P = NP?” question is the same as the question:
Are NP-complete problems tractable or intractable?

Fact: Either all NP-complete problems are tractable or all are intractable.



Note: Showing that a problem is NP-complete is regarded as evidence that the problem is intractable.

Complete Problems for PSPACE

Theorem (Stockmeyer – 1976):
Quantified Boolean Formulas (QBF) is PSPACE-complete:
Given a quantified Boolean formula $\forall x_1 \exists x_2 \ldots. \forall x_k \varphi$, is it true?

Example:

- $\forall x_1 \exists x_2 ((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$
  - Is true.
  - If $x_1 = 1$, then put $x_2 = 0$.
  - If $x_1 = 0$, then put $x_2 = 1$.

- $\forall x_1 \exists x_2 ((x_1 \vee x_2) \wedge (x_1 \vee \neg x_2))$
  - Is false.
  - If $x_1 = 1$, then we put $x_2 = 0$ ($x_2 = 1$ also works).
  - If $x_1 = 0$, then the formula evaluates to false, no matter what x_2 is.

Complexity of the Query Evaluation Problem

The Query Evaluation Problem for Relational Calculus:
Given a relational calculus formula φ and a database instance D, find $\varphi^{adom}(D)$.

Theorem: The Query Evaluation Problem for Relational Calculus is PSPACE-complete.

Corollary: The Query Evaluation Problem for Relational Algebra is PSPACE-complete.

Theorem: The query evaluation problem for conjunctive queries $\exists z_1 \cdots \exists z_n \chi(x_1, \ldots, x_n)$ is NP-complete.

Combined Complexity vs. Data Complexity

Theorem: Let φ be a fixed relational calculus formula. Then the following problem is solvable in polynomial time: given a database instance D, find φ(D). In fact, this problem is in LOGSPACE.

Language	Combined Complexity	Data Complexity
Relational Calculus	PSPACE-complete	LOGSPACE (hence, in P)
Conjunctive Queries	NP-complete	LOGSPACE (hence, in P)
Unions of Conj. Queries	NP-complete	LOGSPACE (hence, in P)

For most query languages L, the data complexity of L is of lower complexity than the combined complexity of L.

Query Evaluation on a Collection of Databases

Note:

- Thus far, the query evaluation problem involves a query and a single database.
- There are many natural scenarios, however, in which the query evaluation problem involves a query and a collection of databases.
- Typically, this collection of databases are the "completions" of some "incomplete" database.
- These "completions" can be thought of as the possible worlds arising from an "incomplete" database.

Question: What is the semantics of a query posed on a collection of databases?

Certain Answers

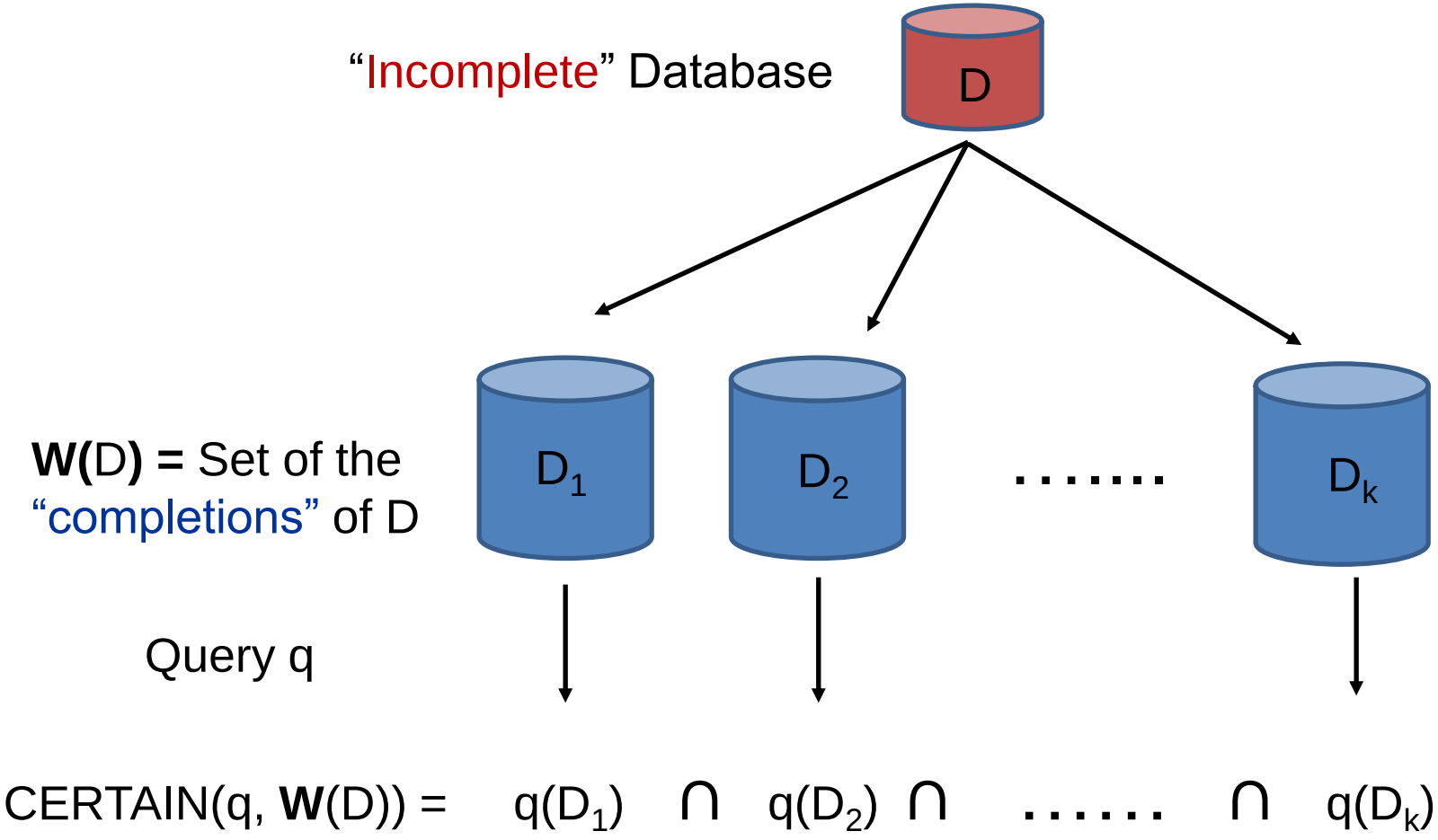
Definition:

- Let q be a k-ary query and ***W*** a collection of databases.
  The certain answers of q on ***W*** is the set

  CERTAIN(q,***W***) = $\bigcap_{D \text{ in } \boldsymbol{W}} q(D) = \{ \mathbf{a}: \mathbf{a} \in q(D), \text{ for every D in } \boldsymbol{W} \}$

- Let q be a Boolean query and ***W*** a collection of databases.
  The certain answers of q on ***W*** is
  CERTAIN(q, ***W***) = 1, if q(D) = 1 for every D in ***W***;
  CERTAIN(q, ***W***) = 0, if q(D) = 0 for some D in ***W***.

Visualizing the Certain Answers



Possible Answers

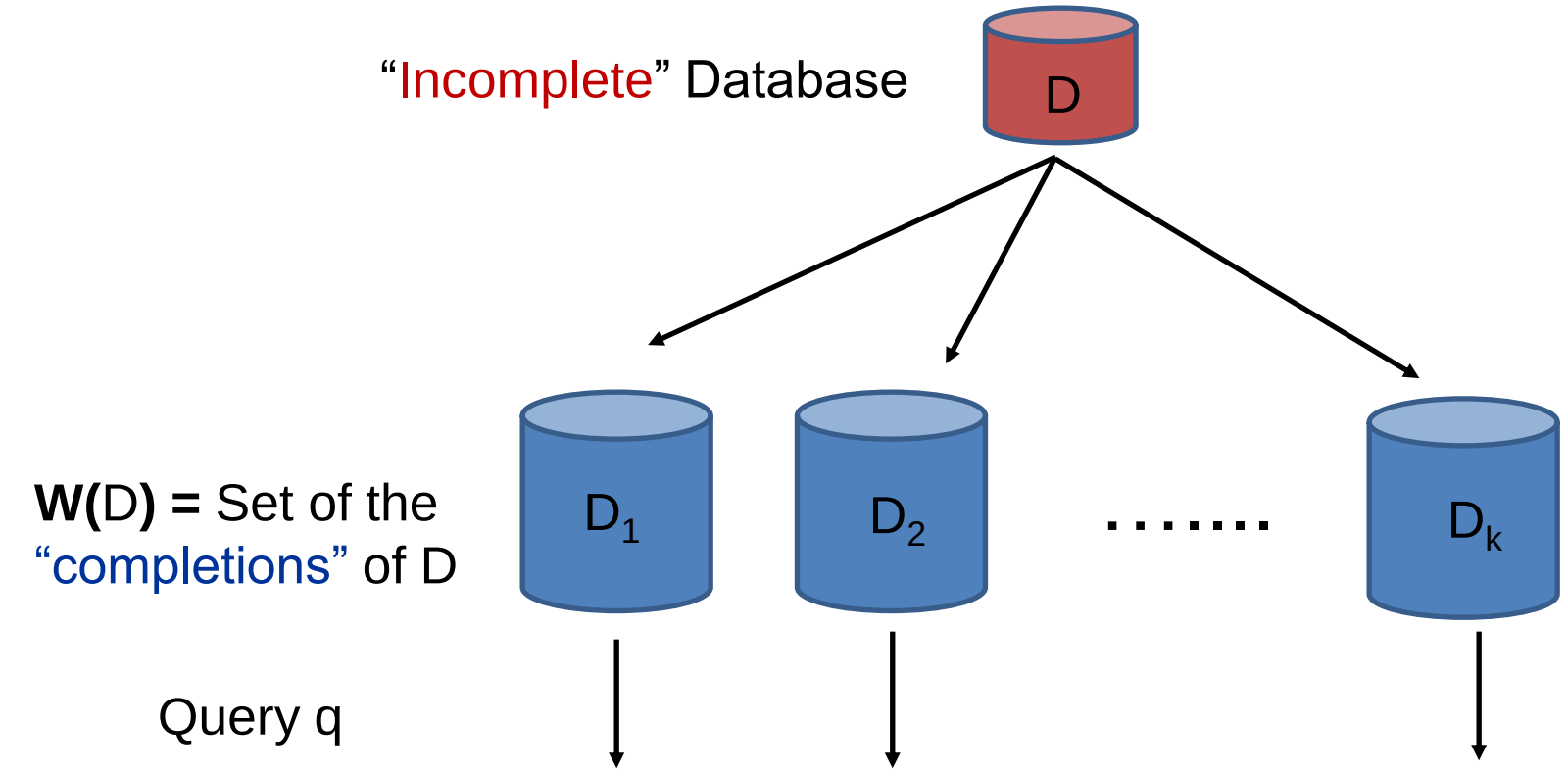
Definition:

- Let q be a k-ary query and ***W*** a collection of databases.
  The possible answers of q on ***W*** is the set

  POSSIBLE(q,***W***) $= \bigcup_{D \text{ in } \boldsymbol{W}} q(D) = \{ \mathbf{a}: \mathbf{a} \in q(D), \text{ for some D in } \boldsymbol{W} \}$

- Let q be a Boolean query and ***W*** a collection of databases.
  The possible answers of q on ***W*** is
  POSSIBLE(q, ***W***) = 1, if q(D) = 1 for some D in ***W***;
  POSSIBLE(q, ***W***) = 0, if q(D) = 0 for every D in ***W***.

Visualizing the Possible Answers



Certain Answers vs. Possible Answers

Note:

- A certain answer a provides a strong guarantee:
  It is an answer to the query on every possible world.

- A possible answer a provides a weaker guarantee:
  It is an answer to the query in at least one possible world.

In what follows, we will focus on the certain answers to queries.

Certain Answers in Different Contexts

- Inconsistent Databases
  - Possible Worlds: The repairs of an inconsistent database.
- Data Exchange
  - Possible Worlds: The solutions of a source instance.
- Probabilistic Databases
  - Possible Worlds: The databases represented by a probabilistic database.
- Computational Social Choice:
  - Possible Worlds: The completions of a set of partial orders representing partial preferences of voters.

Database Dependencies

- Database dependencies are semantic restrictions on databases.

- Codd introduced functional dependencies in 1972.
- Soon after this, several different classes of integrity constraints were introduced and studied, including inclusion dependencies, multivalued dependencies, and join dependencies.

- Eventually, it was realized that:
  - Most of the integrity constraints considered are special cases of embedded implicational dependencies.

  - Embedded implicational dependencies can be expressed by formulas of relational calculus.

Functional Dependencies

Definition: Let **R** be a relational schema and R an instance of **R**.

- If $A_1,\ldots,A_m$, B are attributes of **R**, then we say that
  R satisfies the functional dependency
  $$A_1,\ldots,A_m \rightarrow B$$
  if whenever two tuples in R agree on the values of $A_1,\ldots,A_m$, then they also agree on the value of B.
  (in other words, there are no two tuples in R that have the same value on the attributes of $A_1,\ldots,A_m$, but differ on the value of B).

- If $A_1,\ldots,A_m$, $B_1,\ldots,B_k$ are attributes of **R**, then we say that
  R satisfies the functional dependency
  $$A_1,\ldots,A_m \rightarrow B_1,..,B_k$$
  if R satisfies the functional dependencies
  $A_1,\ldots,A_m \rightarrow B_1$, …, $A_1,\ldots,A_m \rightarrow B_k$.

Functional Dependencies

Example: **R**(A,B,C,D,E)

- R satisfies:
  - A,B $\rightarrow$ D
  - C $\rightarrow$ D
  - B $\rightarrow$ A
  - A,B,C $\rightarrow$ D,E
  - …
- R does not satisfy:
  - A $\rightarrow$ B
  - C $\rightarrow$ B
  - D $\rightarrow$ E
  - …

Instance R

A	B	C	D	E
1	2	3	4	5
2	1	3	4	6
1	2	4	4	3
2	3	5	1	2

Functional Dependencies

- Definition: A relational schema **R** satisfies the functional dependency

  $$A_1,\ldots,A_m \rightarrow B_1,..,B_k$$

  if every instance R of **R** satisfies $A_1,\ldots,A_m \rightarrow B_1,..,B_k$.

- Fact: In effect, the above definition imposes a semantic restriction on the instances of **R**, namely, we disallow all instances that violate the functional dependency $A_1,\ldots,A_m \rightarrow B_1,..,B_k$.

- Notation: If X and Y are sets of attributes of **R**, then we write

  $$X \rightarrow Y$$

  to denote the functional dependency with the members of X in the left-hand side and the members of Y in the right-hand side.

Functional Dependencies

- From a given instance R, we can only derive information about FDs that do **not** hold on the schema **R** (and **not** about FDs that **hold** on the schema).

- CUSTOMER
  (name,ssn,acc-no,balance)
  Consider the instance R to the right.
  - The following FD **fails:**
    balance $\rightarrow$ acc-no
  - The following FDs **hold:**
    - name $\rightarrow$ ssn
    - ssn $\rightarrow$ acc-no

  However, one should **not** expect these to be true of every instance of the schema.

Instance R of CUSTOMER

name	ssn	acc-no	balance
Smith	123456	3394	1500
Jones	213457	3457	1200
Glenn	345678	4567	1500
Wirt	346789	8543	3450

Functional Dependencies

Example: COMPANY(employee, dpt, manager)

- Some plausible FDs are:
  - employee $\rightarrow$ dpt
  - dpt $\rightarrow$ manager
  - manager $\rightarrow$ dpt
  - employee $\rightarrow$ manager

- Some implausible FDs are:
  - manager $\rightarrow$ employee
  - dpt $\rightarrow$ employee

Superkeys and Candidate Keys

Definition: **R** a relational schema, X a set of attributes of **R**

- X is a superkey of **R** if $X \rightarrow A$, for every attribute A of **R**.

- X is a candidate key of **R** if X is a minimal superkey of **R**, that is,
  - X is a superkey of **R**.
  - There is **no** superkey Y of **R** such that $Y \subset X$ (proper subset).

Note: From the candidate keys of **R**, one is selected and designated as the primary key or, simply, the key of **R**.

Note: SQL supports the declaration of keys.

Superkeys and Candidate Keys

Example: COMPANY(employee, dpt, manager)

Assume that the FD dpt $\rightarrow$ manager holds.

- {employee, dpt, manager} is a superkey (trivially)
- {dpt, employee} is a candidate key (in fact, it is the only one).

Example: COMPANY(employee, dpt, manager)

Assume that the FDs dpt $\rightarrow$ manager and manager $\rightarrow$ dpt hold.

- {employee, dpt, manager} is a superkey (trivially).
- {dpt, employee} is a candidate key.
- {manager, employee} is a candidate key.

Functional Dependencies and Relational Calculus

Fact: Every functional dependency $A_1,\ldots,A_m \rightarrow B$ can be expressed in relational calculus. More formally, there is a relational calculus formula ψ such that for every instance R, the following are equivalent:

- R satisfies $A_1,\ldots,A_m \rightarrow B$
- $R \models \psi$.

Proof (by example): Assume that **R** has attributes A,B,C,D.
Then the following are equivalent for the FD $A,B \rightarrow C$.

- R satisfies $A,B \rightarrow C$.
- $R \models \forall x,y,z,w,z',w'(R(x,y,z,w) \wedge R(x,y,z',w') \rightarrow z = z')$.

Note: The formula $\forall x,y,z,w,z',w'(R(x,y,z,w) \wedge R(x,y,z',w') \rightarrow z = z')$ is an example of an equality-generating dependency (egd).

Equality-Generating Dependencies

Definition: An equality-generating dependency (egd) is a formula of relational calculus of the form:

$$\forall x_1,\ldots,x_n(\varphi(x_1,\ldots,x_n) \rightarrow x_i = x_j),$$

where $\varphi(x_1,\ldots,x_n)$ is a conjunction of atomic formulas (i.e., φ is a conjunctive query)

Examples:

- $\forall x_1,x_2,x_3(R(x_1,x_2) \wedge P(x_2,x_3) \wedge T(x_2) \rightarrow x_2 = x_3)$
  - This is an egd, but not a FD.

- $\forall x_1,x_2,x_3(R(x_1,x_2) \wedge R(x_1,x_3) \rightarrow x_2 = x_3)$
  - This is both an egd and a FD, namely $A_1 \rightarrow A_2$.

Inclusion Dependencies

Example: ENROLLS(student-id, name, course),
PERFORM(student-id, course, grade)

Consider the integrity constraint:

– "every student enrolled in a course is assigned a grade"

This is an example of an inclusion dependency;
it is denoted by:

ENROLLS[student-id,course] $\subseteq$ PERFORM[student-id,course,].

Inclusion Dependencies

Definition: An inclusion dependency (ID) is an expression of the form

$S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n]$, where

- $A_1,\ldots,A_n$ are distinct attributes from S
- $B_1,\ldots,B_n$ are distinct attributes from T.

- An instance satisfies $S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n]$ if
  for every tuple $s \in S$ with values $c_1,\ldots,c_n$ for the attributes $A_1,\ldots,A_n$,
  there is a tuple $t \in T$ with values $c_1,\ldots,c_n$ for the attributes $B_1,\ldots,B_n$.

- A database schema satisfies $S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n]$
  if every instance of the schema satisfies this ID.

Inclusion Dependencies and Relational Calculus

Fact: Every inclusion dependency $S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n]$ can be expressed in relational calculus.

Proof (by example): ENROLLS(student-id, name, course),
PERFORM(student-id, course, grade)

Consider the ID

ENROLLS[student-id,course] $\subseteq$ PERFORM[student-id,course],

"every student enrolled in a course is assigned a grade".

This ID is equivalent to the relational calculus formula

$\forall s,n,c\ (\text{ENROLLS}(s,n,c) \rightarrow \exists g\ \text{PERFORM}(s,c,g))$.

Note: The formula $\forall s,n,c\ (\text{ENROLLS}(s,n,c) \rightarrow \exists g\ \text{PERFORM}(s,c,g))$ is an example of a tuple-generating dependency (tgd).

Tuple-Generating Dependencies

Definition: A tuple-generating dependency (tgd) is a formula of relational calculus of the form:

$$\forall x_1,\ldots,x_n(\varphi(x_1,\ldots,x_n) \rightarrow \exists y_1,..,y_m\, \psi(x'_1,\ldots,x'_k,y_1,\ldots,y_m)),$$

where

- $\varphi(x_1,\ldots,x_n)$ and $\psi(x'_1,\ldots,x'_k,y_1,\ldots,y_m$ are conjunctions of atomic formulas
- The variables $x'_1,\ldots,x'_k$ are among the variables $x_1,\ldots,x_n$.

Note: In effect, a tuple-generating dependency asserts that the conjunctive query defined by $\varphi(x_1,\ldots,x_n)$ is contained in the conjunctive query defined by $\exists y_1,..,y_m\psi(x'_1,\ldots,x'_k,y_1,\ldots,y_m)$.

Tuple-Generating Dependencies

Examples:

- Every inclusion dependency is a tuple-generating dependency.

- $\forall$ x,y,z (E(x,y)$\wedge$ E(y,z) $\rightarrow$ E(x,z))
  - This is a tgd, but not an ID. It asserts that E is transitive.

- $\forall$ x,y(E(x,y) $\rightarrow \exists$ z(F(x,z) $\wedge$ F(z,y)).
  - This says that for every edge in E, there is a path of length 2 in F.

- $\forall$ x,y,z (P(x,y,z) $\rightarrow$ R(x,y)$\wedge$ T(y,z))
  - This says that P is decomposed to R and T.

Embedded Implicational Dependencies

Definition: A database integrity constraint is an embedded implicational dependency if it is either a tuple-generating dependency or an equality-generating dependency.

Fact: Embedded implicational dependencies contain as special cases the various classes of integrity constraints studied in the 1970s and the early 1980s, such as:

- Functional dependencies
- Join dependencies
- Inclusion dependencies.
- Multivalued dependencies.

(see the survey paper on database dependencies by Fagin and Vardi)

Inconsistent Databases and Certain Answers

- We will now focus on inconsistent databases and on the certain answers of queries in that context.
- Recall that there are two main uses of logic in databases:
  - Logic is used as a database query language to express questions asked against databases.
  - Logic is used as a specification language to express integrity constraints in databases, that is, semantic restrictions that the data at hand ought to obey.

Equality-Generating Dependencies

Definition

- Functional Dependency $R : X \rightarrow Y$
  If two tuples in R agree on X, then they agree on Y.
- Superkey Constraint $R : X \rightarrow Y$, where Y is the set of attributes of R that are not in X.

Example $R(A, B, C, D)$

- Functional Dependency $R : A, B \rightarrow D$ as an egd:

  $\forall a, b, c, c', d, d'(R(a, b, c, d) \wedge R(a, b, c', d') \rightarrow d = d')$

- Superkey Constraint $R : A, B \rightarrow C, D$ as two egds:

  $\forall a, b, c, c', d, d'(R(a, b, c, d) \wedge R(a, b, c', d') \rightarrow c = c')$
  $\forall a, b, c, c', d, d'(R(a, b, c, d) \wedge R(a, b, c', d') \rightarrow d = d')$

Inconsistent Databases

- In designing databases, one specifies a schema **S** and a set Σ of integrity constraints on **S**.
- An inconsistent database is a database *I* that does not satisfy Σ.
- Inconsistent databases arise in a variety of contexts and for different reasons:
  - For lack of support of particular integrity constraints.
  - In data integration of heterogeneous data obeying different integrity constraints.
  - In data warehousing and in Extract-Transform-Load (ETL) applications, where data resides in different sources and has to be “cleaned” before it can be processed.

Coping with Inconsistent Databases

Two different approaches:

- Data Cleaning: Based on heuristics or specific domain knowledge, the inconsistent database is transformed to a consistent one by modifying (adding, deleting, updating) tuples in relations.
  - This is the main approach in industry (e.g., IBM InfoSphere Quality Stage, Microsoft DQS).
  - More engineering than science as quite often arbitrary choices have to be made.

Coping with Inconsistent Databases

Two different approaches:

- Data Cleaning: Based on heuristics or specific domain knowledge, the inconsistent database is transformed to a consistent one by modifying (adding, deleting, updating) tuples in relations.
  - This is the main approach in industry (e.g., IBM InfoSphere Quality Stage, Microsoft DQS).
  - More engineering than science as quite often arbitrary choices have to be made.

- Database Repairs: A framework for coping with inconsistent databases in a principled way and without "cleaning" dirty data first.

Database Repairs

Definition (Arenas, Bertossi, Chomicki – 1999)

Σ a set of integrity constraints and I an inconsistent database. A database J is a *repair* of I w.r.t. Σ if

- J is a consistent database (i.e., $J \models \Sigma$);
- J differs from I in a minimal way.

Database Repairs

Definition (Arenas, Bertossi, Chomicki – 1999)

Σ a set of integrity constraints and I an inconsistent database.
A database J is a *repair* of I w.r.t. Σ if

- J is a consistent database (i.e., $J \models \Sigma$);
- J differs from I in a minimal way.

Fact

Several different types of repairs have been considered:

- Set-based repairs (subset, superset, $\oplus$-repairs).
- Cardinality-based repairs
- Attribute-based repairs
- Preferred repairs

Subset Repairs

Definition

Σ a set of integrity constraints and I an inconsistent database.
J is a *subset-repair* of I w.r.t. Σ if

- $J \subset I$
- $J \models \Sigma$ (i.e., J is consistent)
- there is no J' such that $J' \models \Sigma$ and $J \subset J' \subset I$.

Note

Until further notice, we will use the term repair, instead of the term subset repair.

Subset Repairs

Example

Key constraint

$\Sigma = \{\forall x \forall y \forall z (R(x, y) \wedge R(x, z) \rightarrow y = z)\}$

Database

$I = \{R(a_1, b_1), R(a_1, b_2), R(a_2, b_1), R(a_2, b_2)\}$

I has four (subset) repairs w.r.t. Σ:

- $J_1 = \{R(a_1, b_1), R(a_2, b_1)\}$
- $J_2 = \{R(a_1, b_1), R(a_2, b_2)\}$
- $J_3 = \{R(a_1, b_2), R(a_2, b_1)\}$
- $J_4 = \{R(a_1, b_2), R(a_2, b_2)\}$.

Exponentially many repairs exist, in general.

Consistent Query Answering (CQA)

Definition (Arenas, Bertossi, Chomicki)

Σ a set of integrity constraints, q a query, and I a database. The *consistent answers of q on I w.r.t. Σ* is the set

$$\text{CONS}(q, I, \Sigma) = \bigcap \{q(J) : J \text{ is a repair of } I \text{ w.r.t. } \Sigma\}.$$

Consistent Query Answering (CQA)

Definition (Arenas, Bertossi, Chomicki)

Σ a set of integrity constraints, q a query, and I a database. The *consistent answers of q on I w.r.t. Σ* is the set

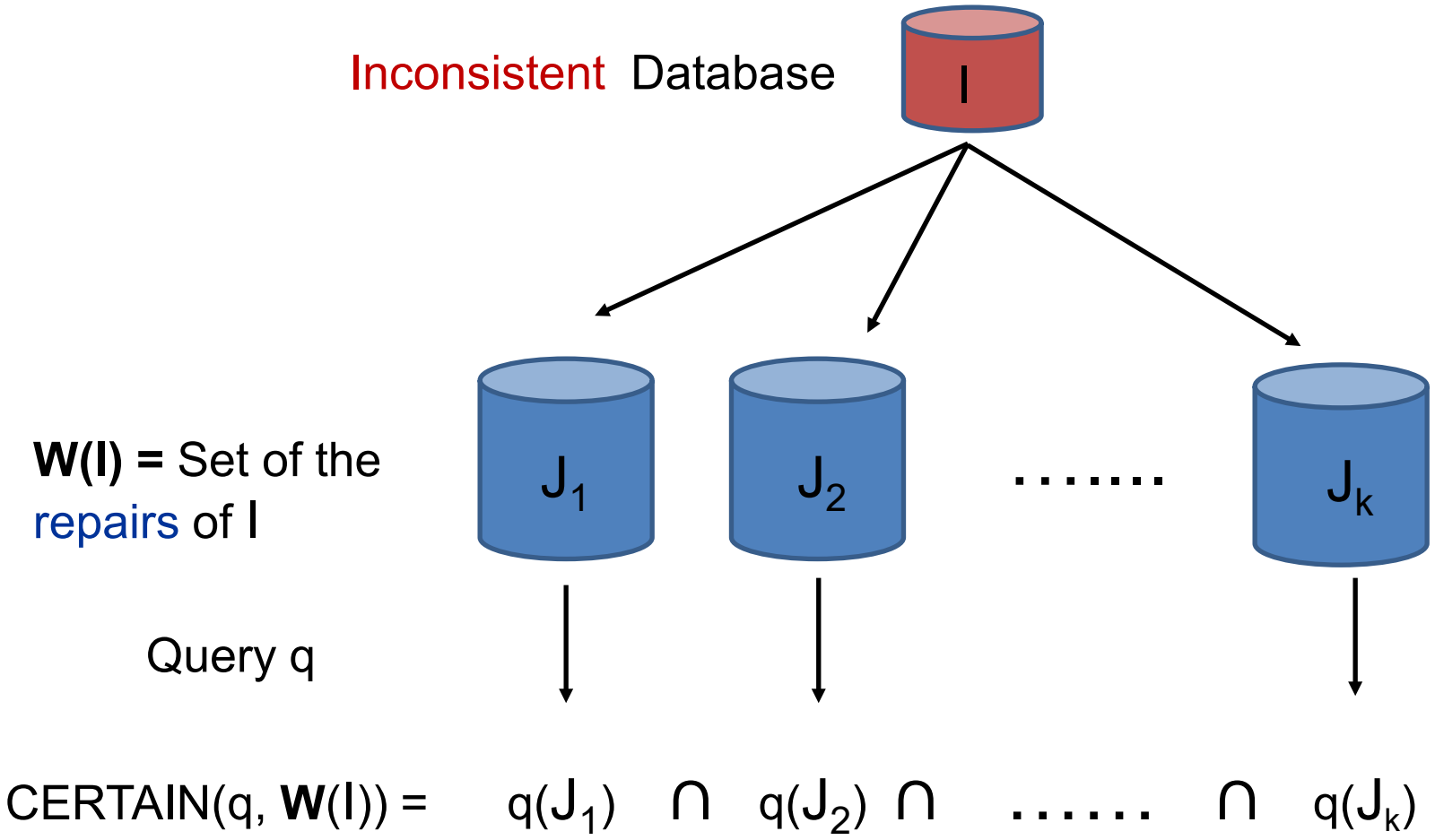
$$\text{CONS}(q, I, \Sigma) = \bigcap \{q(J) : J \text{ is a repair of } I \text{ w.r.t. } \Sigma\}.$$

Fact: Consistent answers *are* certain answers, where the possible worlds are the repairs.

- If Σ is a set of integrity constraints and I is a database, we write $\mathbf{W}(I)$ for the set of all repairs of I w.r.t. Σ. Then

$$\text{CONS}(q, I, \Sigma) = \text{CERTAIN}(q, \mathbf{W}(I)).$$

Visualizing the Consistent Answers



Consistent Query Answering (CQA)

Example (Revisited)

$\Sigma = \{\forall x \forall y \forall z((R(x,y) \wedge R(x,z) \rightarrow y = z)\}$
$I = \{R(a_1, b_1), R(a_1, b_2), R(a_2, b_1), R(a_2, b_2)\}$

Recall that I has four repairs w.r.t. Σ:

- $J_1 = \{R(a_1, b_1), R(a_2, b_1)\}$, $J_2 = \{R(a_1, b_1), R(a_2, b_2)\}$
- $J_3 = \{R(a_1, b_2), R(a_2, b_1)\}$, $J_4 = \{R(a_1, b_2), R(a_2, b_2)\}$.

- If $q(x)$ is the query $\exists y R(x, y)$, then

$$\text{CONS}(q, I, \Sigma) = \{a_1, a_2\}.$$

- If $q(x)$ is the query $\exists z R(z, x)$, then

$$\text{CONS}(q, I, \Sigma) = \emptyset.$$

Overview of Research on Database Repairs

Main themes explored in the study of repairs and consistent query answering:

- Complexity of CQA for conjunctive queries
- Complexity of Repair Checking: Given I and J, is J a repair of I w.r.t. Σ?
- Prototype CQA Systems for selected classes of constraints (e.g., key constraints) and selected classes of queries (e.g., conjunctive queries).

Note

For a survey, see the monograph Database Repairing and Consistent Query Answering by L. Bertossi.

The Complexity Class coNP

Definition

- If Q is a decision problem, then the complement of Q is the decision problem $\overline{Q}$ such that

$$\overline{Q}(x) = \begin{cases} 1 & \text{if } Q(x) = 0, \\ 0 & \text{if } Q(x) = 1. \end{cases}$$

- coNP is the class of all decision problems Q whose complement $\overline{Q}$ is in NP.

Fact

- $\overline{\mathrm{SAT}}$ = UNSAT: Given a Boolean formula φ, is φ unsatisfiable?
- UNSAT is coNP-complete.

Complexity of CQA: A "Simple" Case Study

Fact Let Σ be a set of key constraints with one key per relation. Then the repair-checking problem w.r.t. Σ is in P (in fact, in L).

Proof: Exercise.

Complexity of CQA: A "Simple" Case Study

Fact Let Σ be a set of key constraints with one key per relation. Then the repair-checking problem w.r.t. Σ is in P (in fact, in L).

Proof: Exercise.

Definition Let Σ be a set of key constraints with one key per relation and q a Boolean conjunctive query (no free variables). $\mathrm{CERTAINTY}(q, \Sigma)$ is the following decision problem: Given a database I, is $\mathrm{CONS}(q, I, \Sigma)$ true? (is q true on every repair of I?)

Complexity of CQA: A "Simple" Case Study

Fact Let Σ be a set of key constraints with one key per relation. Then the repair-checking problem w.r.t. Σ is in P (in fact, in L).

Proof: Exercise.

Definition Let Σ be a set of key constraints with one key per relation and q a Boolean conjunctive query (no free variables). $\textsc{certainty}(q, \Sigma)$ is the following decision problem: Given a database I, is $\textsc{cons}(q, I, \Sigma)$ true? (is q true on every repair of I?)

Fact $\textsc{certainty}(q, \Sigma)$ is in coNP.

Proof: Exercise

Hint: Use the previous fact and the definitions.

Note: $\mathrm{LOGSPACE} \subseteq \mathrm{P} \subseteq \mathrm{coNP} \subseteq \mathrm{PSPACE}$.

Complexity of CQA: An Illustration

Binary relations R and S having the first attribute as key, i.e.,

$$\Sigma = \{R(u, v) \wedge R(u, w) \rightarrow v = w, \quad S(u, v) \wedge S(u, w) \rightarrow v = w\}.$$

Consider the following three Boolean queries:

- Let q_1 be the "path" query $\exists x, y, z(R(x, y) \wedge S(y, z))$.
- Let q_2 be the "cycle" query $\exists x, y(R(x, y) \wedge S(y, x))$.
- Let q_3 be the "sink" query $\exists x, y, z(R(x, y) \wedge S(z, y))$.

Question:
What can we say about the complexity of CERTAINTY(q_i, Σ), where $i = 1, 2, 3$?

Complexity of CQA: An Illustration

Binary relations R and S having the first attribute as key, i.e.,

$$\Sigma = \{R(u,v) \wedge R(u,w) \rightarrow v = w, \quad S(u,v) \wedge S(u,w) \rightarrow v = w\}.$$

Fact: Let q_1 be the "path" query $\exists x, y, z(R(x,y) \wedge S(y,z))$.
Then CERTAINTY(q_1, Σ) is in L, hence it is also in P.
Actually, CERTAINTY(q_1, Σ) is FO-rewritable as

$$q^* \equiv \exists x, y, z(R(x,y) \wedge S(y,z) \wedge \forall y'(R(x,y') \rightarrow \exists z' S(y',z'))).$$

This means that for every database I, we have that

$$\text{CONS}(q_1, I, \Sigma) = 1 \text{ if and only if } q^*(I) = 1.$$

Complexity of CQA: An Illustration

Binary relations R and S having the first attribute as key, i.e.,

$$\Sigma = \{R(u,v) \wedge R(u,w) \rightarrow v = w, \quad S(u,v) \wedge S(u,w) \rightarrow v = w\}.$$

Fact: Let q_1 be the "path" query $\exists x, y, z(R(x,y) \wedge S(y,z))$.
Then CERTAINTY(q_1, Σ) is in L, hence it is also in P.
Actually, CERTAINTY(q_1, Σ) is FO-rewritable as

$$q^* \equiv \exists x, y, z(R(x,y) \wedge S(y,z) \wedge \forall y'(R(x,y') \rightarrow \exists z' S(y',z'))).$$

This means that for every database I, we have that

$$\text{CONS}(q_1, I, \Sigma) = 1 \text{ if and only if } q^*(I) = 1.$$

Note: FO-rewritability is the most desirable outcome from a complexity viewpoint: we can compute the consistent answers by evaluation a FO-query on the inconsistent database.

Complexity of CQA: An Illustration

Binary relations R and S having the first attribute as key, i.e.,

$$\Sigma = \{R(u,v) \wedge R(u,w) \rightarrow v = w, \quad S(u,v) \wedge S(u,w) \rightarrow v = w\}.$$

- Let q_1 be the "path" query $\exists x, y, z(R(x,y) \wedge S(y,z))$.
  CERTAINTY(q_1, Σ) is in P; in fact, it is FO-rewritable.
- Let q_2 be the "cycle" query $\exists x, y(R(x,y) \wedge S(y,x))$.
  CERTAINTY(q_2, Σ) is in P, but it is not FO-rewritable.
- Let q_3 be the "sink" query $\exists x, y, z(R(x,y) \wedge S(z,y))$.
  CERTAINTY(q_3, Σ) is coNP-complete.

Classifying the Complexity of CQA

Question: Can we classify the complexity of CERTAINTY(q, Σ)?

Classifying the Complexity of CQA

Question: Can we classify the complexity of CERTAINTY(q, Σ)?

Conjecture (Dichotomy Conjecture for CERTAINTY(q, Σ))

If Σ is a set of key constraints with one key per relation and q is a Boolean conjunctive query, then one of the following holds:

- CERTAINTY(q, Σ) is in P.
- CERTAINTY(q, Σ) is coNP-complete.

Moreover, the dichotomy is effective: we can decide in PTIME whether CERTAINTY(q, Σ) is in P or it is coNP-complete.

Ladner's Theorem and Dichotomies in Complexity

Theorem (Ladner - 1975)

If $\mathrm{P} \neq \mathrm{NP}$, then there is a decision problem Q such that

- Q is in NP, but not in P.
- Q is not NP-complete.

The Fine Structure of NP

NP-complete
not NP-complete, not in P
P

Ladner's Theorem and Dichotomies in Complexity

Theorem (Ladner - 1975)

If $\mathrm{P} \neq \mathrm{NP}$, then there is a decision problem Q such that

- Q is in NP, but not in P.
- Q is not NP-complete.

The Fine Structure of NP

NP-complete
not NP-complete, not in P
P

Dichotomy Conjecture for CERTAINTY(q, Σ)

	coNP-complete
CERTAINTY(q, Σ)	not coNP-complete, not in P
	P

Progress towards the Dichotomy for CERTAINTY(q, Σ)

Theorem (Koutris and Wijsen - 2017)

If Σ is a set of key constraints with one key per relation and q is a Boolean self-join free conjunctive query, then one of the following holds:

- CERTAINTY(q, Σ) is in P.
- CERTAINTY(q, Σ) is coNP-complete.

Moreover, this dichotomy is decidable in quadratic time.

Progress towards the Dichotomy for CERTAINTY(q, Σ)

Theorem (Koutris and Wijsen - 2017)

If Σ is a set of key constraints with one key per relation and q is a Boolean self-join free conjunctive query, then one of the following holds:

- CERTAINTY(q, Σ) is in P.
- CERTAINTY(q, Σ) is coNP-complete.

Moreover, this dichotomy is decidable in quadratic time.

Key Notion: The attack graph associated with Σ and q.

- The nodes of the attack graph are the atoms of q.
- The edges of the attack graph are determined by the functional dependencies on the variables of an atom that are implied by the keys of the other atoms.

The Attack Graph

Σ a set of key constraints with one key per relation.
q a Boolean self-join free conjunctive query

- $K(q) = \{\text{key}(F) \to \text{Var}(F) : F \text{ is an atom of } q\}$.
- $F^{+,q} = \{x \in \text{Var}(q) : K(q \setminus F) \models \text{key}(F) \to x\}$.
- F attacks G, denoted $F \rightsquigarrow G$, if there is a sequence $F_1, \ldots F_n$ such that
  - $F_1 = F$ and $F_n = G$.
  - $\text{Var}(F_i) \cap \text{Var}(F_{i+1}) \not\subseteq F^{+,q}$, for every $i \leq n-1$,
- An attack $F \rightsquigarrow G$ is weak if $K(q) \models \text{key}(F) \to \text{key}(G)$; otherwise, the attack is strong.
- A cycle in the attack graph is strong if it contains at least one strong attack.

Progress towards the Dichotomy for CERTAINTY(q, Σ)

Theorem (Koutris and Wijsen - 2017)

Let Σ be a set of key constraints with one key per relation and let q is a Boolean self-join free conjunctive query.

- If the attack graph is acyclic, then CERTAINTY(q, Σ) is in P and, in fact, it FO-rewritable; otherwise, CERTAINTY(q, Σ) is L-hard, hence it is not FO-rewritable.
- If the attack graph contains no strong cycle, then CERTAINTY(q, Σ) is in P.
- If the attack graph contains a strong cycle, then CERTAINTY(q, Σ) is coNP-complete.

Moreover, these conditions can be checked in quadratic time.

A Precursor to the Koutris-Wisjen Dichotomy

Theorem (K . . . and Pema - 2012)

Assume Σ consists of a key for R and a key for S, and let q be a Boolean query with two atoms, one R-atom and one S-atom. If CERTAINTY(q, Σ) is not FO-rewritable, then the following hold:

- If $\mathrm{key}(R) \cup \mathrm{key}(S) \subseteq \mathrm{Var}(R) \cap \mathrm{Var}(S)$, then CERTAINTY$(q, \Sigma)$ is in P.
- If $\mathrm{key}(R) \cup \mathrm{key}(S) \not\subseteq \mathrm{Var}(R) \cap \mathrm{Var}(S)$, then CERTAINTY$(q, \Sigma)$ is coNP-complete.

A Precursor to the Koutris-Wisjen Dichotomy

Theorem (K . . . and Pema - 2012)

Assume Σ consists of a key for R and a key for S, and let q be a Boolean query with two atoms, one R-atom and one S-atom. If CERTAINTY(q, Σ) is not FO-rewritable, then the following hold:

- If $\text{key}(R) \cup \text{key}(S) \subseteq \text{Var}(R) \cap \text{Var}(S)$, then CERTAINTY$(q, \Sigma)$ is in P.
- If $\text{key}(R) \cup \text{key}(S) \not\subseteq \text{Var}(R) \cap \text{Var}(S)$, then CERTAINTY$(q, \Sigma)$ is coNP-complete.

Examples:

- Let q_2 be the "cycle" query $\exists x, y(R(x, y) \wedge S(y, x))$. CERTAINTY$(q_2, \Sigma)$ is in P, because $\text{key}(R) \cup \text{key}(S) = \{x, y\}$, $\text{Var}(R) \cap \text{Var}(S) = \{x, y\}$.
- Let q_3 be the "sink" query $\exists x, y, z(R(x, y) \wedge S(z, y))$. CERTAINTY$(q_3, \Sigma)$ is coNP-complete, because $\text{key}(R) \cup \text{key}(S) = \{x, z\}$, $\text{Var}(R) \cap \text{Var}(S) = \{y\}$.

The Current Frontier

Open Problems

- Prove the Dichotomy Conjecture for CERTAINTY(q, Σ), where Σ is a set of keys, one for each relation, and q is an arbitrary Boolean conjunctive query.
  - Partial progress (path queries) has been made by Koutris, Ouyang, Wijsen - 2021.
- Prove a Dichotomy Theorem for CERTAINTY(q, Σ), where Σ is a set of functional dependencies and q is a union of Boolean conjunctive queries.
  - Partial progress (multiple keys per relation) has been made by Koutris and Wijsen - 2020.