Modern Database Dependency Theory

Lecture 1: Overview

Michael Benedikt & Balder ten Cate
SQL Execution Error.

Executed SQL statement: INSERT INTO Reservation SELECT 'Meeting102', '11:00-12:00', 'SQL Development Team'
INSERT INTO Reservation SELECT 'Meeting102', '11:00-12:00', 'NET Programmers'
Error Source: .Net SqlClient Data Provider
Error Message: Violation of UNIQUE KEY constraint 'sqlUniqueConstraint'. Cannot insert duplicate key in object 'dbo.Reservation'.
The statement has been terminated.
Integrity constraint violation

Reservation:

<table>
<thead>
<tr>
<th>location</th>
<th>time</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>+ Meeting102</td>
<td>11:00-12:00</td>
<td>SQL Development team</td>
</tr>
<tr>
<td>+ Meeting102</td>
<td>11:00-12:00</td>
<td>.NET programmers</td>
</tr>
</tbody>
</table>

**Integrity constraint:**
- No two distinct rows should have the same location and time.
- Reservation: location, time \( \rightarrow \) title
- In first-order logic:
  \[ \forall x y u v \text{ Reservation}(x, y, u) \land \text{ Reservation}(x, y, v) \rightarrow u = v \]
Integrity constraints

- **Integrity constraints**: structural properties that are required to hold in a database after every transaction.

- **Database dependencies**: a fragment of first-order logic for expressing (among other things) integrity constraints.
Another Example

Staff:

<table>
<thead>
<tr>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“John”</td>
<td>“<a href="mailto:john@gmail.com">john@gmail.com</a>”</td>
<td>23</td>
</tr>
</tbody>
</table>

Department:

<table>
<thead>
<tr>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>“Comp.Sci”</td>
<td>“Parks Rd.”</td>
</tr>
</tbody>
</table>

Functional dependencies:
- Staff: staffid → name, email, deptid
- Department: deptid → name, address

Inclusion dependency:
- Staff[deptid] ⊆ Department[deptid]
How are Integrity Constraints Used?

- **Maintain integrity** of the database
- Allow the database management system to use more efficient data structures and algorithms.
- ...
Schema Design

<table>
<thead>
<tr>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptName</th>
<th>deptAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“John”</td>
<td>“<a href="mailto:john@gmail.com">john@gmail.com</a>”</td>
<td>“Comp.Sci”</td>
<td>“Parks Rd.”</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Functional dependencies
- **Staff**: staffid → *
- **Staff**: deptName → deptAddress

Redundancy:
- The address of the Comp.Sci. department is stored many times.
- If the address changes, need to change many entries in the DB.
- Decomposing the relation into two relations is more sensible.

Schema design
- A traditional topic in databases where constraints play a key role.
Query Containment under Constraints

<table>
<thead>
<tr>
<th>Staff:</th>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department:</th>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Inclusion dependency:**
- $\text{Staff}[\text{deptid}] \subseteq \text{Department}[\text{deptid}]$

**Query 1:** $q(y) = \exists xzu \text{ Staff}(x,y,z,u)$

**Query 2:** $q(y) = \exists xzuwv (\text{Staff}(x,y,z,u) \land \text{Department}(u,v,w))$
Open-World Query Answering

<table>
<thead>
<tr>
<th>Course:</th>
<th>Employee:</th>
</tr>
</thead>
<tbody>
<tr>
<td>course_id</td>
<td>teacher</td>
</tr>
<tr>
<td>301</td>
<td>john</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department:</th>
<th>ReportsTo:</th>
</tr>
</thead>
<tbody>
<tr>
<td>dept</td>
<td>manager</td>
</tr>
<tr>
<td>CS</td>
<td>kurt</td>
</tr>
<tr>
<td>Phil</td>
<td>saul</td>
</tr>
<tr>
<td>Math</td>
<td>alfred</td>
</tr>
<tr>
<td>emp</td>
<td>manager</td>
</tr>
<tr>
<td>john</td>
<td>kurt</td>
</tr>
</tbody>
</table>

Query $q(x,y) = \text{Employee}(x,y)$

Constraints:
- $\text{Course}[\text{teacher}] \subseteq \text{Employee}[\text{emp}]$
- $\text{Employee}[\text{dept}] \subseteq \text{Department}[\text{dept}]$
- $\text{Department}: \text{dept} \rightarrow \text{manager}$
- $\text{Department}: \text{manager} \rightarrow \text{dept}$
- $\text{ReportsTo}: \text{emp} \rightarrow \text{manager}$
- $\forall xyz (\text{Employee}(x,y) \land \text{Department}(y,z) \rightarrow \text{ReportsTo}(x,z))$
What this course is about

• Modern applications of database dependencies
• Fundamental tasks involving database dependencies:
  – Implication problem
  – Query containment under constraints
  – Open-world query answering
  – Answering queries under access restrictions
  – …
• Theory of database dependencies:
  – Decidability and algorithms
  – Well behaved classes of database dependencies
  – Relationships to knowledge representation languages.
Why you should keep coming

• This course is on the interface between data management, knowledge representation, and automated reasoning.

• At the same time, the course relates to fundamental topics in “pure” mathematical logic (e.g., proofs, interpolation, finite model theory)
Today

1. Relational databases
2. Database dependencies (tgds and egds)
3. Three important computational problems:
   – The implication problem for constraints
   – Conjunctive query containment under constraints
   – Open world query answering
4. Some classic results
Relational Databases

The relational database model (Codd, 1970):

- A schema $S = \{R_1, \ldots, R_n \}$ is a set of relation names, each with an associated “arity”. For convenience, we often give names to the different attributes (argument positions) of a relation.
- A fact over $S$ is an expression of the form $R(a_1, \ldots, a_n)$, where $R \in S$ is an $n$-ary relation and $a_1, \ldots, a_n$ is a tuple of constants (i.e. values from some infinite universe of constants).
- A database instance (or simply database) $D$ for $S$ is a finite set of facts over $S$. 
Example

- **Schema S**
  - Staff(staffid, name, email, deptid)
  - Department(deptid, name, address)

- **Database instance D**
  - Staff(1, “John”, “john@gmail.com”, 23)
  - Department(23, “Comp.Sci”, “Parks Rd.”)

<table>
<thead>
<tr>
<th>Staff:</th>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>“John”</td>
<td>“<a href="mailto:john@gmail.com">john@gmail.com</a>”</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department:</th>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>23</td>
<td>“Comp.Sci”</td>
<td>“Parks Rd.”</td>
</tr>
</tbody>
</table>
Relational Databases (ct’d)

- A database instance $D$ for schema $S = \{R_1, \ldots, R_n\}$ can be viewed as a **finite first-order structure** $(\text{dom}(D), R_1^D, \ldots, R_n^D)$
  - $\text{dom}(D)$ is the **active domain** of $D$, i.e., the set of all constants occurring in the facts of $D$.
  - $R_k^D = \{ (a_1, \ldots, a_m) \mid R_k(a_1, \ldots, a_m) \in D \}$ is the set of tuples belonging to relation $R_k$ in $D$. 

Queries

• An n-ary **query** is expressed by a formula with n free variables.
  – An n-ary query returns an n-ary relation.
    • Notation: q(D) is the relation computed by q in a database D.
  – A **boolean query** is a query without free variables (i.e., n=0).
    • q(D) = true/false
Example

- Query:
  - \( q(y,z) = \exists x u b (\text{Staff}(x,y,z,u) \land \text{Department}(u,\text{"Comp.Sci."},b)) \)
  - “Return the name and email address of all staff of the Computer Science department.”

- Answers:
  - \( q(D) = \{ (\text{"John"}, \text{"john@gmail.com"}) \} \)
Query Languages

- The four most important query languages:
  1. Conjunctive queries
  2. Unions of Conjunctive Queries (UCQ)
  3. FO queries
  4. Datalog
First-Order Logic (FO)

- Constants (a, b, c, …) and variables (x, y, z, …)
- Atomic formulas: R(t₁, …, tₙ), t₁ = t₂
- Boolean connectives and quantifiers: ∧, ∨, ¬, ∀, ∃
- **Active domain semantics**: quantifiers range over the **active domain** of the instance
- **Unique names semantics**: for every value there is a constant symbol that always denotes that value (distinct constants denote distinct values).
Conjunctive queries

- **Conjunctive Query (CQ):** \( \exists y \ ( \alpha_1(x,y) \land \ldots \land \alpha_n(x,y) ) \) where \( \alpha_1, \ldots, \alpha_n \) are atomic formulas.

- CQs are the most important queries used in practice. They form the SELECT-FROM-WHERE fragment of SQL.

- Example:
  \[ q(y,z) = \exists x \exists u \ ( \text{Staff}(x,y,z,u) \land \text{Department}(u,"\text{Comp.Sci."},b) ) \]

- A **match** (satisfying assignment / homomorphism) of a CQ in a database \( D \) is a map from the variables in the query to \( \text{adom}(I) \), such that all atoms of the query are satisfied.

- For a CQ \( q(x) = \exists y \ \varphi(x,y) \) and a database \( D \),
  \[ q(D) = \{ (h(x)) \mid h:\{x,y\} \rightarrow \text{adom}(D) \text{ is a match of } q \text{ in } D \} \]
Unions of Conjunctive Queries

• A n-ary union of conjunctive queries (UCQ) is a disjunction of finitely many n-ary CQs.

• Example:
  \[ q(x, y) = \text{Edge}(x, y) \lor [\exists z \ (\text{Edge}(x, z) \land \text{Edge}(z, y))] \]

• Every UCQ is a positive existential FO formula,
  – and every positive existential FO formula is equiv. to a UCQ.
Datalog

• Datalog extends UCQ with recursion.

• Example of a datalog query:
  - \textit{Reach}(x, y) \leftarrow \textit{Edge}(x, y)
  - \textit{Reach}(x, y) \leftarrow \textit{Edge}(x, z) \land \textit{Reach}(z, y)

• Fixpoint semantics: “Reach is the smallest relation s.t. …”

• \textit{Reach}(x, y) is \textbf{not FO definable}.

• Datalog will come back later on in the course.
Query Languages

- FO
- UCQ
- CQ
- Datalog
An Aside: Queries vs. Plans

• Important distinction in the relational database model:
  – **Query**: declarative description (think: logical formula)
  – **Query plan**: concrete procedure (sequence of operations)
    by which the answer to a query can be obtained.

• Example:
  – Query: \( q(x) = \exists yz \ (R(x,y) \land S(y,z) \land T(x,z)) \)
  – Query plan: \( \pi_1(\pi_{1,3}(R \bowtie_{R.2=S.1} S) \cap T) \)

• **Codd’s Completeness Theorem**: every FO query has a
  query plan in the relational algebra (and vice versa).
Summary

• **Database schema** ~~ a finite relational signature. E.g.,
  - \{ Staff(staffid, name, email, deptid) ,
    Department(deptid, name, address) \}

• **Database instance** (of a given schema) ~~ a set of facts.
  - Think: a finite first-order structure (active domain semantics)

• **Database queries** ~~ logical formulas \( \phi(x) \). For example:
  - \( \phi(y,z) = \exists u \forall x \text{ Staff}(x,y,z,u) \& \text{Department}(u,"Comp.Sci.",b) \)
  - Four important query languages: CQ, UCQ, FO, Datalog

• Next up: **Database constraints** ~~ logical sentences (in a fragment of FO).
Database Dependencies

- A **database constraint** is a condition that the database instance is expected to satisfy. A database instance D is **consistent** with a set of constraint if D satisfies all the constraints.

- In the 1970s many different classes of database dependencies (constraints) were introduced and studied.
  - Functional dependencies (FDs),
  - Inclusion dependencies (ICs),
  - Multivalued dependencies (MVDs), ...

- In the early 1980s, **tuple-generating dependencies** (TGDs) and **equality-generating dependencies** (EGDs) arose as a unifying general constraint language.
Functional Dependencies (FDs)

• Syntax: \( R: A_1, \ldots, A_n \rightarrow B \)
  – \( R \) is a relation name and \( A_1, \ldots, A_n, B \) are attributes of \( R \).
• Semantics:
  – “Every two \( R \)-facts that agree on \( A_1, \ldots, A_n \) agree on \( B \).”

• Example (for Reservation[location, time, title]):
  – Reservation: location, time \( \rightarrow \) title

• Expressed in first-order logic:
  • \( \forall xyuv(\text{Reservation}(x, y, u) \land \text{Reservation}(x, y, v) \rightarrow u=v) \)
• Special case of an equality-generating dependency (EGD):
  • \( \forall \mathbf{x} (\phi(\mathbf{x}) \rightarrow x_i=x_j) \) where \( \phi(\mathbf{x}) \) is a conjunction of atoms.
Inclusion Dependencies (IDs)

- **Syntax**: \( R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n] \)
  - \( R \) is a relation name and \( A_1, \ldots, A_n \) are attributes of \( R \).
  - \( S \) is a relation name and \( B_1, \ldots, B_n \) are attributes of \( S \).

- **Example (for Staff[staffid, name, email, deptid] and Department[deptid, name, address])**:
  - \( \text{Staff[deptid]} \subseteq \text{Department[deptid]} \)

- **Expressed in first-order logic**:
  - \( \forall x y u \text{ Staff}(x,y,u) \rightarrow \exists v w \text{ Department}(x,v,w) \)

- **Special case of a tuple-generating dependency (TGD)**:
  - \( \forall x (\phi(x) \rightarrow \exists y \psi(x,y)) \) where \( \phi(x) \) and \( \psi(x,y) \) are conjunctions of atoms.
### Named vs Unnamed Perspective

<table>
<thead>
<tr>
<th>Staff:</th>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department:</th>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- In database languages, attributes of a relation are often referred to **by name**.
  - Staff[deptid] ⊆ Department[deptid]
  - Staff: staffid → name, email, deptid

- In logic, they are usually referred to **by position** ("unnamed perspective"): 

- From now on, we will adhere to the **unnamed perspective**.
  - Staff[4] ⊆ Department[1]
  - Staff: 1 → 2,3,4
Key constraints

Staff:

<table>
<thead>
<tr>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Department:

<table>
<thead>
<tr>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **FDs:**
  - (a) Staff: 1 → 2,3,4
  - (b) Department: 1 → 2,3

- **ID:**
  - (c) Staff[4] ⊆ Department[1]

- The first attribute of Staff (i.e., staffid) is called a **key**:
  - it functionally determines all other attributes of the relation.

- (a) and (b) are also called **key constraints**.
- The combination of (c) and (b) is called a **foreign key constraint**.
Keys and foreign keys in SQL

CREATE TABLE Staff
(
    staffid int,
    name char(50),
    email char(50),
    deptid int,
    PRIMARY KEY (staffid),
    FOREIGN KEY (deptid) REFERENCES Department(deptid)
)
TGDs and EGDs

- Many other classes of dependencies have been introduced and studied. They all turned out to be expressible in one of the following two fragments of FO:
  - **Tuple-generating dependencies** (TGD):
    \[ \forall x ( \phi (x) \rightarrow \exists y \psi (x, y) ) \]
  - **Equality-generating dependencies** (EGDs)
    \[ \forall x ( \phi (x) \rightarrow x_1 = x_j ) \]

- TGDs and EGDs will be the main focus of this course.

- **Notational convention**: universal quantifiers omitted for readability.
• **Special cases of TGDs:**
  – Inclusion dependencies
  – Full TGDs (TGDS without existential quantifiers)
  – View definitions
    • \( \text{TwoHop}(x, y) \leftrightarrow \exists z \ (\text{Edge}(x, z) \land \text{Edge}(z, y)) \)
  – Schema mapping constraints
    • Describe relationships across different schemas.
    • \( \text{Dept}(x, y, z, u) \rightarrow \exists w \ \text{Department}(x, u, z, w) \)

• **Special cases of EGDs:**
  – Functional dependencies
  – Key constraints
Geography of Dependency Classes

EGDs
  FDs
    Keys

TGDs
  IDs
  Full TGDs
Implication

- Dependencies $C_1, \ldots, C_n$ imply $C_{n+1}$ if every database satisfying $C_1, \ldots, C_n$ satisfies $C_{n+1}$.

- Looks like a special case of FO entailment (i.e., $C_1, \ldots, C_n \models C_{n+1}$), but it is not quite the same. Databases are normally assumed to be finite.

- Example (binary relation $R$):

- True in all databases but there are infinite structures that are counterexamples
# Complexity of Testing Implication

<table>
<thead>
<tr>
<th>Dependency class</th>
<th>Complexity of testing implication</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDs</td>
<td>PTIME</td>
<td>Beeri-Bernstein</td>
</tr>
<tr>
<td>EGDs</td>
<td>NP-complete</td>
<td>Beeri-Vardi</td>
</tr>
<tr>
<td>IDs</td>
<td>PSPACE-complete</td>
<td>Casanova-Fagin-Papadimitriou</td>
</tr>
<tr>
<td>Full TGDS (i.e., TGDs without $\exists$ quantifiers)</td>
<td>EXPTIME-complete</td>
<td>Chandra-Lewis-Makowsky</td>
</tr>
<tr>
<td>TGDs</td>
<td>Undecidable</td>
<td>Mitchell / Chandra-Vardi</td>
</tr>
<tr>
<td>FDs + IDs</td>
<td>Undecidable</td>
<td>Gurevich-Lewis / Vardi</td>
</tr>
</tbody>
</table>
Sound and Complete Axiomatizations

• The following axioms form a sound and complete proof system for FD implication (Armstrong 1974):
  – Reflexivity: \( R:X \rightarrow x \) for \( x \in X \)
  – Transitivity: If \( R:X \rightarrow Y \) and \( R:X \cup \{y\} \rightarrow z \) then \( R:X \rightarrow z \)
  – Augmentation: If \( R:X \rightarrow z \) then \( R:X \cup Y \rightarrow z \)

• A similar complete axiomatization exists for other constraints, such as IDs (Casanova-Fagin-Papadimitriou 1982).
Query Containment

• If $q_1(x)$ and $q_2(x)$ are queries and $C$ is a set of dependencies, then $q_1(x)$ is contained in $q_2(x)$ w.r.t. $C$ if, for all databases $D$ that are consistent w.r.t. $C$, $q_1(D) \subseteq q_2(D)$.

• Example:
  - Query $q_1(x) = \exists yz \ R(x, y) \land S(y, z)$
  - Query $q_2(x) = \exists y \ R(x, y)$
Open World Query Answering

- **Input**: a database instance \( D \), a set of constraints \( C \), a query \( q \).
  - The database may not be consistent w.r.t. the constraints
  - \( \text{worlds}(D,C) = \{ D' | D \subseteq D' \text{ and } D' \text{ is consistent w.r.t. } C \} \)
  - \( \text{certain}(q,D,C) = \bigcap \{ q(D') | D' \text{ in } \text{worlds}(D,C) \} \)
  - For boolean \( q \), \( \text{certain}(q,D,C) = \text{true} \) iff \( q(D') = \text{true} \) for all \( D' \in \text{worlds}(D,C) \).

- **Motivation**:
  - \( D \) contains true facts but may be incomplete.
  - Each \( D' \in \text{worlds}(D,C) \) is a “possible world”.

- **The open world query answering problem**:
  - given \( q, D, C \), and a tuple \( t \), decide if \( t \in \text{certain}(q,D,C) \).
  - If \( q \) is boolean, decide if \( \text{certain}(q,D,C) = \text{true} \)
Open-World Query Answering

Course:

<table>
<thead>
<tr>
<th>course_id</th>
<th>teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>301</td>
<td>john</td>
</tr>
</tbody>
</table>

Employee:

<table>
<thead>
<tr>
<th>emp</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>bill</td>
<td>Phil</td>
</tr>
</tbody>
</table>

Department:

<table>
<thead>
<tr>
<th>dept</th>
<th>manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>kurt</td>
</tr>
<tr>
<td>Phil</td>
<td>saul</td>
</tr>
<tr>
<td>Math</td>
<td>alfred</td>
</tr>
</tbody>
</table>

ReportsTo:

<table>
<thead>
<tr>
<th>emp</th>
<th>manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>kurt</td>
</tr>
</tbody>
</table>

Query \( q(x,y) = \text{Employee}(x,y) \)

Constraints:

- Course[teacher] ⊆ Employee[emp]
- Employee[dept] ⊆ Department[dept]
- Department: dept → manager
- Department: manager → dept
- ReportsTo: emp → manager
- \( \forall x y z . \text{Employee}(x,y) \land \text{Department}(y,z) \rightarrow \text{ReportsTo}(x,z) \)
More on Open World Querying

• Situations where open-world querying naturally arises:
  – **Data integration and data exchange** (multiple incomplete sources, constraints relating source to each other)
  – **Ontology-mediated data access** (some relations are not stored in the database, but constraints, in the form of an ontology, are provided that allow us to still infer things involving these relations)
  – **Querying under access restrictions** (we cannot freely access all relations in the database. Using the integrity constraints, we can still infer things involving inaccessible relations)

• These topics will be covered extensively on Thursday and Friday.
**Theorem:** the following problems are effectively equivalent, for every class of constraints $C$:

1. The CQ containment problem w.r.t. constraints from $C$
2. Open-world query answering for CQs w.r.t. constraints from $C$.

For $C$=TGDs, both problems are undecidable.

**Proof of equivalence:** involves some ingredients that we discuss next.
1st ingredient: homomorphisms

- **Homomorphisms** are a fundamental tool in the study of databases, conjunctive queries, and dependencies.

- Let D and D’ be two database instances over the same schema. A **homomorphism from D to D’** is a function 
  \[ h : \text{dom}(D) \rightarrow \text{dom}(D') \]
  such that for every fact \( R(a_1\ldots a_n) \in D \), the corresponding fact \( R(h(a_1),\ldots,h(a_n)) \) belongs to D’.
Example

<table>
<thead>
<tr>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“James”</td>
<td>“<a href="mailto:james@gmail.com">james@gmail.com</a>”</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>“Philosophy”</td>
<td>“Woodstock Rd.”</td>
</tr>
<tr>
<td>25</td>
<td>“Linguistics”</td>
<td>“Walton Str.”</td>
</tr>
</tbody>
</table>

maps homomorphically to

<table>
<thead>
<tr>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“John”</td>
<td>“<a href="mailto:john@gmail.com">john@gmail.com</a>”</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>“Comp.Sci”</td>
<td>“Parks Rd.”</td>
</tr>
</tbody>
</table>
Theorem: let \( h : D \rightarrow D' \) be a homomorphism and \( q \) a UCQ not containing any constants

- If \((a_1, \ldots, a_n)\) in \( q(D) \) then \((h(a_1), \ldots, h(a_n))\) in \( q(D')\)
- If \( q \) is Boolean and \( q \) is \( q(D) = true \) then \( q(D') = true \).

In other words: “UCQs are preserved by homomorphisms”.
(In fact, all Datalog queries are preserved by homomorphisms.)

The same holds for queries with constants, provided that \( h(a) = a \) for all constants \( a \) appearing in the query.
Example revisited

Staff:

<table>
<thead>
<tr>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“James”</td>
<td>“<a href="mailto:james@gmail.com">james@gmail.com</a>”</td>
<td>24</td>
</tr>
</tbody>
</table>

Department:

<table>
<thead>
<tr>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>“Philosophy”</td>
<td>“Woodstock Rd.”</td>
</tr>
<tr>
<td>25</td>
<td>“Linguistics”</td>
<td>“Walton Str.”</td>
</tr>
</tbody>
</table>

Staff:

<table>
<thead>
<tr>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“John”</td>
<td>“<a href="mailto:john@gmail.com">john@gmail.com</a>”</td>
<td>23</td>
</tr>
</tbody>
</table>

Department:

<table>
<thead>
<tr>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>“Comp.Sci”</td>
<td>“Parks Rd.”</td>
</tr>
</tbody>
</table>

For every UCQ q (without constants), if “Linguistics” ∈ q(D) then “Comp.Sci” ∈ q(D’).
2nd ingredient: canonical queries and canonical database

Correspondence between **boolean conjunctive queries** and **databases**:

- Consider the boolean CQ q: \( \exists xyz \ R(x,y) \land S(y,z) \)
  - Canonical database \( D_q \): \( \{ R(a,b), \ S(b,c) \} \)
  - We replaced each existential variable by a constant

- Consider the database \( D \): \( \{ T(a,b,c), \ U(c,b,d) \} \)
  - Canonical query \( q_D \): \( \exists xyzu \ T(x,y,z) \land U(z,y,u) \)
  - We replaced each constant by an existential variable

- We can freely move around between (boolean) CQs and databases.
Back to Business

**Theorem:** the following problems are effectively equivalent for every class of constraints $C$:

1. The CQ containment problem w.r.t. constraints from $C$
2. Open world query answering for CQs w.r.t. constraints from $C$.

For $C=$TGDs, both problems are undecidable.

**Proof of equivalence:** blackboard

- (using homomorphisms, canonical queries and canonical databases).
**Theorem:** the following problems are effectively equivalent for every set of tgds and egds C:

1. The CQ containment problem w.r.t. the constraints in C.
2. Open world query answering for CQs w.r.t. the constraints in C.

**Proof** For simplicity we prove the case for boolean conjunctive queries only. The argument extends to arbitrary conjunctive queries.

\[ q_1 \subseteq_C q_2 \]

iff

for every database \( D \models C \), if \( q_1(D) = \text{true} \) then \( q_2(D) = \text{true} \)

iff

for every database \( D \models C \), if \( D_q_1 \rightarrow^{\text{hom}} D \) then \( q_2(D) = \text{true} \)

iff

for every database \( D \models C \), if \( D_q_1 \rightarrow^{\text{hom}} D \) then \( \text{certain}(q_2, D, C) = \text{true} \)

iff

\( \text{certain}(q_2, D_q_1, C) = \text{true} \)

(\textbf{Lemma}: whenever \( D_1 \rightarrow^{\text{hom}} D_2 \) then for all boolean CQs \( q \), \( \text{certain}(q, D_1, C) = \text{true} \) implies \( \text{certain}(q, D_2, C) = \text{true} \)
References


• Phokion Kolaitis (2009). Relational databases, logic, and and complexity. Course slides for 2009 GII Doctoral School on Advances in Databases

• Riccardo Rosati (2011). On the finite controllability of conjunctive query answering in databases under open-world assumption. JCSS 77(3): 572-594
Overview of the course

✓ Monday. Introduction

• Tuesday. Basic techniques
  – forward chaining ("chase"); backward chaining ("query rewriting")

• Wednesday. Advanced techniques
  – beyond the terminating chase; guarded tgds; datalog-rewritings

• Thursday. Application: querying reformulation under constraints

• Friday. More applications: data exchange; ontology-based data access
Modern Database Dependency Theory

Lecture 2: Basic Techniques

Michael Benedikt & Balder ten Cate
Day 2 Overview

Essential computational question: **open world query answering.**

**Given:** "partial database" \( D \), query \( Q \), and dependencies \( C \)

- **Tuple-generating dependencies (TGD):**
  \[ \forall x \phi(x) \rightarrow \exists y \psi(x, y) \]

- **Equality-generating dependencies (EGDs)**
  \[ \forall x \phi(x) \rightarrow x_i = x_j \]

**Determine:** which answers to \( Q \) can be inferred from \( C \) and \( D \).

**Equivalent problem:** deciding containment of queries under constraints. Determine if \( q_1 \subseteq_C q_2 \)

Overview two basic techniques:
- Forward chaining (the chase)
- Backward chaining (query-rewriting)
Forward and Backward Chaining (or: why you should *still* stay in this course)
The Chase by Example

Enroll[enrollid, studid, coursename]
Classification[coursename, topic]

∀ studid Enroll₁(studid, 323) → ∃ enrollid Enroll(enrollid, studid, "Intro to Physics")

Records stored with courseid 323 in table Enroll₁ correspond to records stored with coursename "Intro to Physics"

∀ studid lname phone Enroll₂(studid, lname, phone, 221) → ∃ enrollid Enroll(enrollid, studid, "Intro to Botany")

Records stored with courseid 221 in table Enroll₂ correspond to records stored with coursename "Intro to Botany"
The Chase by Example

Enroll[enrollid, studid, coursename]

Enroll_1[studid, courseid]

Enroll_2[studid, lname, phone, courseid]

Classification[coursename, topic]

Facts:

Enroll_1(11234,323)...
Classification("Intro to physics", "science")
Enroll_2(11456,21)...
Classification("Intro to botany", "science")
The Chase by Example

Constraints: \( \forall \text{studid} \ Enroll_1(\text{studid}, \ 323) \rightarrow \exists \text{enrollid} \ Enroll(\text{enrollid}, \ \text{studid}, \ "Intro to physics") \)

\( \forall \text{studid} \ \text{lname} \ \text{phone} \ Enroll_2(\text{studid}, \ \text{lname}, \ \text{phone}, \ 221) \rightarrow \exists \text{enrollid} \ Enroll(\text{enrollid}, \ \text{studid}, \ "Intro to botany") \)

Facts: \( Enroll_1(11234, \ 323), \ Enroll_2(11456, \ 21), \ldots \)
\( \text{Classification("Intro to physics", \ "science")} \)
\( \text{Classification("Intro to botany", \ "science")} \)

Goal: find certain answers to \( Q = \{ \text{studid}: \exists \text{eid} \ \text{coursename} \ \text{studid} \ Enroll(\text{eid}, \ \text{studid}, \ \text{coursename}) \ \wedge \ \text{Classification(\text{coursename}, \ "science")} \} \)

*studids of students enrolled in a course classified as Science*
Chase Example

Constraints:
\[ \forall \text{studid} \, \text{Enroll}_1(\text{studid}, 323) \rightarrow \]
\[ \exists \text{enrollid} \, \text{Enroll}(\text{enrollid}, \text{studid}, "Intro to physics") \]

\[ \forall \text{studid lname phone} \, \text{Enroll}_2(\text{studid}, \text{lname}, \text{phone}, 221) \] 
\[ \rightarrow \exists \text{enrollid} \, \text{Enroll}(\text{enrollid}, \text{studid}, "Intro to botany") \]

Facts: \text{Enroll}_1(11234, 323), \ldots 
\text{Enroll}_2(11376, "Jones", 3232333, 221), \ldots 
\text{Classification}("Intro to physics", "science")
\text{Classification}("Intro to botany", "science")
Chase Example

Constraints:
\[
\forall \text{studid } \text{Enroll}_1(\text{studid}, 323) \rightarrow \\
\exists \text{enrollid } \text{Enroll}(\text{enrollid}, \text{studid}, "Intro to physics")
\]

\[
\forall \text{studid lname phone } \text{Enroll}_2(\text{studid}, \text{lname}, \text{phone}, 221) \rightarrow \\
\exists \text{enrollid } \text{Enroll}(\text{enrollid}, \text{studid}, "Intro to botany")
\]

Facts: \text{Enroll}_1(11234, 323), \ldots
\text{Enroll}_2(11376, "Jones", 3232333, 221), \ldots
\text{Classification("Intro to physics", "science")}
\text{Classification("Intro to botany", "science")}

Forward chain: add new fact
\text{Enroll}(e_0, 11234, "Intro to physics")
Chase Example

Constraints:
\[ \forall \text{studid} \; \text{Enroll}_1(\text{studid}, 323) \rightarrow \exists \text{enrollid} \; \text{Enroll}(\text{enrollid}, \text{studid}, "Intro to physics") \]

\[ \forall \text{studid lname phone} \; \text{Enroll}_2(\text{studid, lname, phone, 221}) \rightarrow \exists \text{enrollid} \; \text{Enroll}(\text{enrollid}, \text{studid, "Intro to botany")} \]

Facts: \text{Enroll}_1(11234, 323), \ldots \\
\text{Enroll}_2(11376, "Jones", 3232333, 221), \ldots \\
\text{Classification("Intro to physics", "science")} \\
\text{Classification("Intro to botany", "science")} \\

Forward chain: add new fact \\
\text{Enroll}(e_0, 11234, "Intro to physics")

New value, distinct from every value in the database. Sometimes called a "null" or "chase constant".
Chase Example

Constraints:
\( \forall \text{studid}\ Enroll_1(\text{studid}, 323) \rightarrow \exists \text{enrollid}\ Enroll(\text{enrollid}, \text{studid}, "Intro to physics") \)

\( \forall \text{studid lname phone}\ Enroll_2(\text{studid}, \text{lngame}, \text{phone}, 221) \rightarrow \exists \text{enrollid}\ Enroll(\text{enrollid}, \text{studid}, "Intro to botany") \)

Facts: \( Enroll_1(11234, 323), \ldots \)
\( Enroll_2(11376, "Jones", 3232333, 221), \ldots \)
Classification("Intro to physics", "science")
Classification("Intro to botany", "science")

Forward chain: \( Enroll(e_0, 11234, "Intro to physics"), Enroll(e_1, 11376, "Intro to botany") \)
Chase Example

Constraints:
\[ \forall \text{studid} \: \text{Enroll}_1(\text{studid}, 323) \rightarrow \exists \text{enrollid} \: \text{Enroll}(\text{enrollid}, \text{studid}, "Intro to physics") \]

\[ \forall \text{studid lname phone} \: \text{Enroll}_2(\text{studid}, \text{lname}, \text{phone}, 221) \rightarrow \exists \text{enrollid} \: \text{Enroll}(\text{enrollid}, \text{studid}, "Intro to botany") \]

Facts: \text{Enroll}_1(11234, 323), ...
Classification("Intro to physics", "science")
Classification("Intro to botany", "science")

Forward chain: \text{Enroll}(e_0, 11234, "Intro to physics"), 
\text{Enroll}(e_1, 11376, "Intro to botany")

Forward chaining is now complete.
Chase Example

Facts after forward chaining: \( \text{Enroll}_1(11234, 323) \)...
Classification("Intro to physics", "science")
Classification("Intro to botany", "science")
Enroll\((e_0, 11234, "Intro to physics")\)
Enroll\((e_1, 11376,"Intro to botany")\)

Goal: find certain answers to \( Q = \{\text{studid} : \exists \text{eid coursename Enroll(eid,studid,coursename)} \land \text{Classification(coursename, "science")}\} \)

studids of students enrolled in a course classified as Science

To compute the certain answers of conjunctive query \( Q \), just evaluate it on the database after forward chaining.
Chase Example

Facts after forward chaining: \( \text{Enroll}_1(11234, 323) \) ...
- Classification("Intro to physics", "science")
- Classification("Intro to botany", "science")
- \( \text{Enroll}(e_0, 11234, \text{"Intro to physics"}) \)
- \( \text{Enroll}(e_1, 11376, \text{"Intro to botany"}) \)

Goal: find certain answers to \( Q = \{\text{studid: } \exists \text{eid coursename } \text{Enroll(eid, studid, coursename)} \land \text{Classification(coursename, "science")}\} \)

**studids of students enrolled in a course classified as Science**

To compute the certain answers of conjunctive query \( Q \), just evaluate it on the database after forward chaining:

Certain answers to \( Q = \{11234 \ldots \} \)
Chase Example

Facts after forward chaining: \( \text{Enroll}_1(11234,323) \)...
Classification("Intro to physics", "science")
Classification("Intro to botany", "science")
Enroll\( (e_0,11234, "Intro to physics") \)
Enroll\( (e_1,11376,"Intro to botany") \)

Goal: find certain answers to \( Q = \{ \text{studid: } \exists \text{ eid coursename Enroll(eid, studid, coursename)} \land \text{Classification(coursename, "science")} \} \)

\textit{studids of students enrolled in a course classified as Science}

To compute the certain answers of conjunctive query \( Q \), just evaluate it on the database after forward chaining:

Certain answers to \( Q = \{11234,11376\} \)
Example 2

Constraints:

\[ \forall \text{did dname mgrid} \text{Department(did, dname, mgrid)} \rightarrow \exists \text{N Employee(mgrid, N, did)} \]

/* Every Department Manager is an employee */

Queries:

\[ Q_1 = \{ \text{did} \mid \exists \text{ dname mgrid} \text{Department(did, dname, mgrid)} \} \]

\[ Q_2 = \{ \text{did} \mid \exists \text{ mgrid ename} \text{Employee(mgrid, ename, did)} \} \]

To check if \( Q_1 \) is contained in \( Q_2 \) w.r.t. constraints we make \( Q_1 \) into a database, forward chain, and then see if \( Q_2 \) holds.
Example 2

Constraints:

\[ \forall \text{did dname mgrid} \text{Department} (\text{did, dname, mgrid}) \rightarrow \exists \text{N Employee} (\text{mgrid, N, did}) \]

/* Every Department Manager is an employee

Queries:

\[ Q_1 = \{ \text{did} | \exists \text{ dname mgrid Department} (\text{did, dname, mgrid}) \} \]

\[ Q_2 = \{ \text{did} | \exists \text{ mgrid ename Employee} (\text{mgrid, ename, did}) \} \]

Department (\text{did}_0, \text{ dname}_0, \text{ mgrid}_0)
Example 2

Constraints:

\[ \forall \text{did dname mgrid} \quad \text{Department}(\text{did, dname, mgrid}) \rightarrow \exists \text{N} \quad \text{Employee}(\text{mgrid, N, did}) \]

/* Every Department Manager is an employee */

Queries:

\[ Q_1 = \{\text{did} | \exists \text{dname mgrid Department}(\text{did, dname, mgrid})\} \]

\[ Q_2 = \{\text{did} | \exists \text{mgrid ename Employee}(\text{mgrid, ename, did}) \} \]

\[ \text{Department}(\text{did}_0, \text{dname}_0, \text{mgrid}_0) \]

Forward chain: \[ \text{Employee}(\text{mgrid}_0, \text{N}_0, \text{did}_0) \]
Example 2

Constraints:
∀ i ∀ n ∀ m Department (did, dname, mgrid) →
∃ N Employee (mgrid, N, did)
/* Every Department Manager is an employee

Queries:
Q₁ = \{ did | ∃ dname mgrid Department (did, dname, mgrid) \}

Q₂ = \{ did | ∃ mgrid ename Employee (mgrid, ename, did) \}

Forward chained database:
Department (did₀, dname₀, mgrid₀), Employee (mgrid₀, N₀, did₀)

To see if Q₁ is contained in Q₂ just check if there is a match of Q₂ in the forward chained database mapping the free variable did to its image in the canonical database did₀.
Example 2

Queries:
\( Q_1 = \{ \text{did} | \exists \text{ dname mgrid} \text{ Department} (\text{did, dname, mgrid}) \} \)

\( Q_2 = \{ \text{did} | \exists \text{ mgrid ename} \text{ Employee} (\text{mgrid, ename, did}) \} \)

Forward chained database:
Department (\( \text{did}_0, \text{ dname}_0, \text{ mgrid}_0 \)), Employee (\( \text{mgrid}_0, \text{ N}_0, \text{ did}_0 \))

There is a match of \( Q_2 \) in the forward chained database, mapping the shared free variable \( \text{did} \) to its image in the canonical database \( \text{did}_0 \).

From this we can conclude that \( Q_1 \) is contained in \( Q_2 \).
TGD Chase Step

To see if database $\mathcal{D}$ satisfies $\mathcal{Q}$ under TGDs $\mathcal{C}$ start reasoning forward to throw in consequences of $\mathcal{D}$ under $\mathcal{C}$.

- For a TGD
  \[ \forall x_1 \ldots x_n A_1(x_1\ldots) \land A_2(x_1\ldots) \ldots \rightarrow B_1(x_1\ldots) \]
  If we have a match of the left side, just add the facts on the right.

- For TGD $\forall x_1 \ldots x_n A_1(x_1\ldots) \land A_2(x_1\ldots) \ldots \rightarrow$ 
  $\exists y_1 \ldots y_k B_1(x_1\ldots y_1\ldots y_k)$

If we have a match of the left, **create new constants** $d_1 \ldots d_k$, generate new facts to give a match for the right.
Chase Step

**Match** of a conjunctive query in a database: homomorphism of the query into the database. **Unfulfilled Match** for TGD $\tau$ in database $\mathcal{I}$: homomorphism of the left side of $\tau$ that does not extend to a match of the right side of $\tau$.

**Chase Step**: take match $\mathbf{m}$ of the left of a TGD $\tau$ in $\mathcal{I}$, throw in facts, using fresh witnesses for the existential quantifiers on the right, to make the right side true.
Chase Step: take match $m$ of the left of a TGD $\tau$ in $I$, throw in fresh witnesses and facts to make the right true.

Chase step is relative to a set of "Base Values" $B$, which should include all values mentioned in databases, queries and constraints. We choose values that are not only not in $D$ but also not in $B$. We omit the dependence on $B$ from now on.
Chase

Chase Step for match $\mathbf{M}$, TGD $\tau$, database $\mathbf{I}$: throw in witnesses and facts to make the right true.

**Parallel Chase Step**($\mathbf{I}$): perform chase step on $\mathbf{I}$ for all unfulfilled matches of constraints.

**Standard Chase** for $\mathbf{D}$ and $\mathbf{C}$ (abbrev. Chase($\mathbf{D}, \mathbf{C}$))

Start with $\mathbf{I} := \mathbf{D}$, then iterate the following as long as there is a TGD $\tau$ with an unfulfilled match $\mathbf{M}$ for left of $\tau$ in $\mathbf{I}$

\[
\mathbf{I} := \text{ParallelChaseStep}(\mathbf{I})
\]

Return $\mathbf{I}$
Chase

ParallelChaseStep(I): perform chase step on I for all unmatched TGDs.

**Standard Chase** for D and C (abbrev. Chase(D, C)) formed by:

Start with $I := D$, then iterate the following as long as there is TGD $\tau$ with an unfulfilled match $M$ for left of $\tau$ in $I$

$I := \text{ParallelChaseStep}(I)$

When it terminates, the Standard Chase ensures that every unfulfilled match is eventually fulfilled.
Properties of the Chase

**Theorem** For a boolean conjunctive query $Q$ (mentioning only constants in $B$) and set of TGDs $C$ such that the chase terminates:

$$\text{certain}(Q,D,C) = \text{true} \text{ iff } Q \text{ holds in Chase}(D,C)$$

Reduce checking if $Q$ follows from $D, C$ to checking in a single database.
Properties of the Chase

**Theorem** For a non-boolean conjunctive query $Q$ (mentioning only constants in $B$) and set of TGDs $C$ such that the chase terminates:

A tuple $t$ using values from $D$ or $B$ is a certain answer to $Q$ with respect to constraints $C$ iff $t$ is in the result of $Q$ evaluated on $\text{Chase}(D, C)$.

Reduce calculation of certain answers for $Q$ with respect to $D, C$ to evaluating $Q$ on a single database $\text{Chase}(D, C)$. 
Properties of the Chase

**Theorem** For boolean conjunctive queries $Q_1$, $Q_2$ and set of TGDs $C$ such that the chase terminates on $\text{Canon}(Q_1)$ where $\text{Canon}(Q_1)$ is the canonical database of $Q_1$:

$Q_1$ is contained in $Q_2$ with respect to constraints $C$ iff $Q_2$ evaluates to true on $\text{Chase}(\text{Canon}(Q_1), C)$

Similarly for non-boolean queries.
Universality of the Chase

All of these results follow from a more general theorem

Universality Theorem

If $U = \text{Chase}(D, C)$ exists, it is a universal model:

- $U$ is a model: it satisfies the constraints and contains $D$
- for any model $I$ for $(D, C)$, there is a homomorphism $h$ from $U$ to $I$ that is constant on $B$. 
Universality Theorem

Arbitrary Solution

Chase \((D, C)\)
Properties of the Chase

**Theorem** For a boolean conjunctive query \( Q \) (mentioning only constants in \( \mathcal{B} \)), set of TGDs \( C \), and database \( D \) for which \( \text{Chase}(D, C) \) exists, we have:

\[
\text{certain}(Q, D, C) = \text{true} \iff Q \text{ holds in } \text{Chase}(D, C)
\]

**Proof:**

- If \( \text{certain}(Q, D, C) = \text{true} \) then \( Q \) holds in every model, hence in \( \mathcal{U} \).
- If \( \text{certain}(Q, D, C) = \text{false} \), then for some model \( I \) \( Q \) is not satisfied in \( I \). But then \( Q \) could not be satisfied in \( \mathcal{U} \) (since \( Q \) would be preserved under the homomorphism \( h \)).
More General Chase

Chase Step for match $\mathbf{M}$, TGD $\tau$, database $\mathbf{I}$: throw in witnesses and facts to make the right true.

**Parallel Chase Step**($\mathbf{I}$): perform chase step on $\mathbf{I}$ for all unfulfilled matches of constraints.

**Standard Chase** for $\mathbf{D}$ and $\mathbf{C}$ (abbrev. $\text{Chase(D,C)}$)

Start with $\mathbf{I} := \mathbf{D}$, then iterate the following as long as there is a TGD $\tau$ with an unfulfilled match $\mathbf{M}$ for left of $\tau$ in $\mathbf{I}$

$I := \text{ParallelChaseStep}(\mathbf{I})$

If process terminates Return $\mathbf{I}$
Else Return "limit of $\mathbf{I}$"
Infinite Chase Model

\[ D \]

\[ I_1 = \text{Perform Chase Steps on } D \]

\[ I_2 = \text{Parallel Chase Steps on } I_1 \]

\[ I_3 = \text{Parallel Chase Steps on } I_2 \]

\[ I_4 = \text{Parallel Chase Steps on } I_3 \]

\[ \ldots \]

\[ I_\infty = \text{union of } I_i \]
Universality of the Chase

Without assuming termination of the chase, we can state a universality result.

Theorem \( U = \text{Chase}(D, C) \), even when infinite, is a universal model:

- \( U \) is a model: it satisfies the constraints and contains \( D \)
- for any model \( I \) for \( (D, C) \), there is a homomorphism \( h \) from \( U \) to \( I \) that is constant on \( B \).

From this we can see that evaluating \( Q \) on \( \text{Chase}(D, C) \) gets the infinitely certain answers of \( Q \): tuples that are returned by \( Q \) in all models, finite or infinite.
Complications

Start with $I := D$,

While (there is a TGD $\tau$ with an unfulfilled match $M$ for left of $\tau$ in $I$)
    $I := \text{Parallel ChaseStep}(I)$

The chase can be infinite.

Binary relation $R$ with inclusion dependency:
$R[2] \subseteq R[1]$
$\forall x \ y \ R(x, y) \rightarrow \exists z \ R(y, z)$
Infinite Chase

Binary relation \( R \) with inclusion dependency:
\[
\forall \; x \; y \; R(x, y) \rightarrow \exists \; z \; R(y, z)
\]
Simple Termination

For some classes of TGDs, we can argue that the chase terminates: after a finite number of steps, there will be no more chase steps to fire.

Consider **source-to-target TGDs**: relations on the right are distinct from relations on the left.

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enroll₁[studid, courseid]</td>
<td>Enroll[enrollid, studid, coursename]</td>
</tr>
<tr>
<td>Enroll₂[studid, lname, phone, courseid]</td>
<td>Classification[coursename, topic]</td>
</tr>
</tbody>
</table>

Then the chase terminates.
Simple Termination

For some classes of TGDs, we can argue that the chase terminates: after a finite number of steps, there will be no more chase steps to fire.

Consider full TGDs: no restrictions on the relations, but no existential quantifiers allowed on the right.

Then the chase terminates.
Geography of Dependency Classes

- **EGDs**
  - **FDs**
  - **Keys**

- **TGDs**
  - **IDs**
  - **Full TGDs**
Termination via Acyclicity

For some classes of TGDs, we can argue that the chase terminates: after a finite number of steps, there will be no more rules to fire.

Consider **acyclic inclusion dependencies**.

Inclusion dependencies where the following graph is acyclic:

- nodes are all relations in the IDs,
- an edge from $R_i$ to $R_j$ if there is a dependency $R_i[p_1 \ldots p_k] \subseteq R_j[q_1 \ldots q_k]$

Then the chase terminates.
Termination via Acyclicity

Given a set of TGDs, form the dependency graph:

Nodes = positions within relation

Regular edge:
**Constraint:** $\forall x_1 \ldots R(x_1, \ldots) \rightarrow S(y_1, x_1, \ldots)$
**Edge:** $R[1] \rightarrow S[2]$

Existential edge:
**Constraint:** $\forall x_1 \ldots R(x_1, \ldots) \rightarrow \exists z S(y_1, z \ldots x_1, \ldots)$
**Edge:** $R[1] \rightarrow S[2]$
Termination via Acyclicity

Regular edge:
**Constraint:**  $\forall x_1 \ldots R(x_1, \ldots) \rightarrow S(y_1, x_1, \ldots)$
**Edge:** $R[1] \rightarrow S[2]$

Existential edge:
**Constraint:**  $\forall x_1 \ldots R(x_1, \ldots) \rightarrow \exists z S(y_1, z \ldots x_1, \ldots)$
**Edge:** $R[1] \rightarrow S[2]$

**Weakly acyclic** set of TGDs: no cycle through an existential edge
Weakly Acyclic

Example:
\[ \forall \text{did} \ \forall \text{dname} \ \forall \text{mgrid} \ \text{Department}(\text{did}, \ \text{dname}, \ \text{mgrid}) \rightarrow \]
\[ \exists \text{N} \ \text{Employee}(\text{mgrid}, \ \text{N}, \ \text{did}) \]
/* Every Department Manager is an employee

\[ \forall \text{eid} \ \forall \text{ename} \ \forall \text{did} \ \text{Employee}(\text{eid}, \ \text{ename}, \ \text{did}) \rightarrow \]
\[ \exists \text{D} \ \exists \text{M} \ \text{Department}(\text{did}, \ \text{D}, \ \text{M}) \]
/* Every Employee works in a department
Termination via Acyclicity

Weakly acyclic: no existential cycle in Dependecny Graph.
Nodes= positions within relation
Regular edge:
Constraint: \( \forall x_1 \ldots R(x_1, \ldots) \rightarrow S(y_1, x_1, \ldots) \)
Existential edge
Constraint: \( \forall x_1 \ldots R(x_1, \ldots) \rightarrow \exists z \ S(y_1, z \ldots, x_1, \ldots) \)

**Theorem** For any weakly acyclic set of TGDs, the chase terminates.
Termination via "boring cycles"

For some classes of TGDs, we can argue that the chase terminates: after a finite number of steps, there will be no more rules to fire.

Consider constraints corresponding to **conjunctive view definitions**:

$$\begin{align*}
R_1(x_1 \ldots x_n) & \leftrightarrow Q_1(x_1 \ldots x_n) \\
\ldots \\
R_n(...) & \leftrightarrow Q_n(...) 
\end{align*}$$

Assume $Q_i$ conjunctive queries using only base relations, not other $R_i$'s.
Conjunctive View Definitions

Base signature has $R(x, y)$
"Two-hop view" $V_1(x) = \{ x | \exists y z \ R(x, y) \land R(y, z) \}$

$V_1(x) \rightarrow \exists y z \ R(x, y) \land R(y, z)$ \hspace{1cm} (1)
$R(x, y) \land R(y, z) \rightarrow V_1(x)$ \hspace{1cm} (2)

In chasing a database $D$, we can apply the rules (1) first then the rules (2).
We will not need to re-apply rules from (1) afterwards.
Termination via boring cycles

**Theorem** For constraints coming from conjunctive view definitions, the chase terminates.
Bad news: Undecidability

**Theorem**
The following problem is undecidable: 😖

**Input:** TGDs $c$,
**Output:** yes if for every database $\mathcal{D}$ and base values $\mathcal{B}$ there is a terminating chase sequence.
Theorem
The following problem is undecidable: 😞
Input: database \(D\), values \(B\), and TGDs \(C\)
Output: whether or not there is a terminating chase sequence for \(D, B, C\)

Related undecidability result: existence of finite universal models
Forward Chaining
Summary

• **For now:** there are many classes for which we can guarantee chase termination (and get a bound) and hence perform forward chaining.
• **Later:** there are classes for which the chase does not terminate, but still we can compute query answers, containment, implication, via forward chaining.
Further Work

Weakly acyclic and conjunctive view definitions represent two of the first known classes with terminating chase. See the references for information on more complex conditions ensuring termination.
Backward Chaining

**Forward chaining:** Pre-pre-process the data $\mathcal{D}$. Calculate consequences of $\mathcal{D}$ under constraints $\mathcal{C}$. Once we have calculated "maximal consequences" we can just do query evaluation (independent of the query $\mathcal{Q}$).

**Backward chaining:** pre-process the query $\mathcal{Q}$. Calculate sufficient conditions for $\mathcal{Q}$ to be implied by the constraints $\mathcal{C}$. Once we have pre-processed the query to calculate "all sufficient conditions", we have reduced to query evaluation. Pre-processing is independent of the database $\mathcal{D}$. 
Rewritability
Example

Suppose $c$ consists of:

$$\forall x \ y \ A(x, y) \rightarrow \exists z \ B(x, z)$$

Consider query $Q = \exists x \ z \ B(x, z) \land C(x)$

Rewriting $Q^{-1}_c =$

$$[\exists x \ z \ B(x, z) \land C(x)] \lor$$

$$[\exists x \ y \ A(x, y) \land C(x)]$$
Rewritability
By Example

\[ C = \{ \forall x \ y \ A(x, y) \rightarrow \exists z \ B(x, z), \ \forall u \ v \ D(u, v) \rightarrow A(v, u) \} \]

\[ Q = \exists x \ z \ B(x, z) \land C(x) \]

Rewriting \( R = Q^{-1}_C = \)
\[ [ \exists x \ z \ B(x, z) \land C(x) ] \lor [ \exists x \ y \ A(x, y) \land C(x) ] \]

Iterate Rewriting \( S = R^{-1}_C \)
\[ [ \exists x \ z \ B(x, z) \land C(x) ] \lor [ \exists x \ y \ A(x, y) \land C(x) ] \lor [ \exists x \ y \ D(y, x) \land C(x) ] \]

No further way to "backward chain".
So to get certain answers to \( Q \) on any \( D \), just evaluate \( S \) on \( D \).
Backward Chaining Step

Given boolean query without constants and no repeated variables in an atom \( Q = \exists x_1 \ldots x_m A_1 \land \ldots A_n \)

and TGD constraint \( \rho \)
\[
\forall x_1 \ldots x_u R_1 \land \ldots \land R_j \rightarrow \exists y_1 \ldots y_v S
\]

\( \rho \) is **applicable** if there is a variable mapping \( \sigma \) that maps \( S \) into some \( A_i \) and no \( \sigma(y_i) \) in \( \sigma(S) \) is shared with another \( A_j \)

If we have such a mapping, we let \( \rho^{-1}(Q) \) be the result of replacing \( A_i \) by \( \sigma(R_1) \land \ldots \land \sigma(R_j) \) and adding existential quantifiers with fresh variables as needed.
Shared Variable Restriction

\( C = \{ \tau = \forall x \ y \ A(x, y) \rightarrow \exists z \ B(x, z) \} \)

\( Q = \exists u \ v \ B(u, v) \land C(u, u) \)

We can map right hand side of \( \tau \) to atom \( B(u, v) \)

The "inverse" query \( \exists u \ v \ A(u, v) \land C(u, u) \) does imply \( Q \) under \( C \)
Shared Variable Restriction

\[ C = \{ \tau = \forall x \ y \ A(x, y) \rightarrow \exists z \ B(x, z) \} \]

\[ Q = \exists u \ v \ B(u, v) \land C(u, v) \]

We can map right hand side of \( \tau \) to atom \( B(u, v) \)

The "inverse" query \( \exists u \ v \ A(u, v) \land C(u, v) \) does not imply \( Q \) under \( C \).
Iterative Backward Chaining

Start with $Q'' := Q$, then iterate until $Q''$ does not change:

Choose a conjunctive query $Q_i$ in $Q''$ an applicable rule $\rho$ and mapping $\sigma$ of the right of $\rho$ into an atom of $Q_i$

$Q'_i := \text{BackwardChaseStep}(Q_i, \rho, \sigma)$

If $Q'_i$ not a renaming of something in $Q''$

Then Add $Q'_i$ to $Q''$

Return $Q''$
Backward Chaining Step

**Theorem** if iterative backward chasing on \( \mathcal{C}, \mathcal{Q} \), terminates returning query \( \mathcal{Q}' \) then for any database \( \mathcal{D} \), \( \mathcal{Q}' \) evaluated on \( \mathcal{D} \) returns the Infinitely-certain answers of \( \mathcal{Q} \) for \( \mathcal{D}, \mathcal{C} \).

**Recall:** Infinitely-certain answers of \( \mathcal{Q} \) are those that are returned by \( \mathcal{Q} \) in all models, finite or infinite.
Backward Chaining Step

**Theorem** If iterative backward chasing on $C$, $Q$, terminates returning query $Q'$ then for any database $D$, $Q'$ evaluated on $D$ returns the infinitely-certain answers of $Q$ for $D$, $C$.

**Easier direction:** if tuple $t$ is returned by $Q'$ on $D$, show that $t$ is an infinitely-certain answer.

Show that if tuple $t$ is returned by $\rho^{-1}(Q)$ for some TGD $\rho$ then $t$ is an infinitely-certain answer of $Q$.

Then use induction.
Backward Chaining
Step

**Theorem** If iterative backward chasing on \( C, Q \), terminates returning query \( Q' \) then for any database \( D \), \( Q' \) evaluated on \( D \) returns the infinitely-certain answers of \( Q \) for \( D, C \).

**Slightly harder direction:** if tuple \( t \) is an infinitely-certain answer, show that \( t \) is returned by \( Q' \) on \( D \).

Suppose backward chaining terminates at stage \( m \).
If \( t \) is an infinitely-certain answer, then it must appear in the chase (by previous result using universality of the chase). It thus must appear at some stage of the chase, say stage \( n \).
Show that \( t \) is returned by \( n \) stage backward rewriting of \( Q \).
If \( n \leq m \), clear that \( t \) is returned also at stage \( m \) (since larger stages return more tuples).
If \( n > m \) then \( n \) stage returns the same as \( m \) stage, since fixpoint reached at \( m \).
FO Rewritability

A query $Q$ is **first-order rewritable** according to constraints $C$ if there is some first-order query $Q'$ such that:

for any database $D$, $Q'$ evaluated on $D$ returns the certain answers of $Q$ for $D$, $C$

A query $Q$ is **weakly first-order rewritable** according to constraints $C$ if there is some first-order query $Q'$ such that:

for any database $D$, $Q'$ evaluated on $D$ returns the infinitely-certain answers of $Q$ for $D$, $C$
Significance of Backward Chaining

A query $Q$ is **first-order rewritable** according to constraints $C$ if there is some first-order query $Q'$ such that:

for any database $D$, $Q'$ evaluated on $D$ returns the certain answers of $Q$ for $D$, $C$

First-order rewritability implies that the certain answers can be computed "scalably in the data".
- with low complexity in the size of the data
- using only off-the-shelf database technology
  (thus taking advantage of DB techniques for speeding up evaluation –e.g. indexing, query optimization)
FO Rewritability

A query $\mathcal{Q}$ is **weakly first-order rewritable over arbitrary instances** according to constraints $\mathcal{C}$ if there is some first-order query $\mathcal{Q}'$ such that:

for any database $\mathcal{D}$, $\mathcal{Q}'$ evaluated on $\mathcal{D}$
returns the certain answers of $\mathcal{Q}$ for $\mathcal{D}$, $\mathcal{C}$

We have seen:
when backwards chaining terminates, the result is a weak first-order rewriting of the query $\mathcal{Q}$
Rewriting

**Theorem** If \( q \) is a conjunctive query and \( c \) is a collection of **Linear TGDs**, then the backward chaining approach terminates.

Linear TGD: single atom on the left

\[
\forall x_1 \ldots x_m \\
A(x_1 \ldots x_m) \rightarrow \exists y_1 \ldots y_j \ B_1 \land \ldots \land B_n
\]
**Linear TGDs**

**Theorem** If $q$ is a conjunctive query and $c$ is a collection of Linear TGDs, then the backward chaining approach terminates with a weak first-order rewriting.

When we perform a backward chase step with a linear TGD, we may add on a new rewritten query, but only one of the same size as the original conjunctive query.

There are only a finite number of conjunctive queries of a given size, thus eventually must not add on anything new.
Linear TGDs

In fact, we have a stronger result:

**Theorem** If \( q \) is a conjunctive query and \( C \) is a collection of Linear TGDs, then the backward chaining approach terminates with a first-order rewriting.

**Proof:** finite model construction due to Rosati. Closely-related to techniques for construction finite models in logic see BaranyGottlobOtto in the references.
Limits of Rewriting

Consider constraint

\[ \forall x \ y \ R(x, y) \land R(y, z) \rightarrow R(x, z) \]

Query \[ Q = \exists x \ y \ U(x) \land R(x, y) \land V(y) \text{ is not first-order re-writable.} \]
Limits of Rewriting

Consider constraint

\[ \forall x \ y \ R(x, y) \land R(y, z) \to R(x, z) \]

Query \( Q = \exists x \ y \ U(x) \land R(x, y) \land V(y) \) is implied by a database \( D \) and the constraint above exactly when there is a path from a \( U \) node to a \( V \) node. One can show, using this, that \( Q \) is not first-order rewritable.
Undecidability

**Theorem**
The following problem is undecidable:

**Input:** TGDs $\mathbf{c}$ and query $\mathbf{q}$,
**Output:** yes if $\mathbf{q}$ is first-order rewritable with respect to $\mathbf{c}$
Backward Chaining
Summary

• **For now:** there are some classes for which backward chaining terminates (and we get a bound). For these we get first-order rewritability.
  – Many consequences for implementation.

• **Later:** there are classes for which we cannot get a first-order rewriting, but we can compute query answers via backward chaining.
References

Forward chaining (the chase).
Much of the material in this tutorial comes from
the following survey article, which contains much more detail on
the chase procedure:

Backward reasoning (Query rewriting)
An early source is:
Andrea Calì, Domenico Lembo, Riccardo Rosati: *Query rewriting and answering under constraints in data integration systems*. IJCAI 2003: 16-21

Ininitely-certain answers and certain answers for IDs.
See:

Vince Bárány, Georg Gottlob, Martin Otto: *Querying the Guarded Fragment* Logical Methods in Computer Science 10(2) (2014))
Modern Database Dependency Theory

Lecture 3: Advanced Techniques

Michael Benedikt & Balder ten Cate
Day 2 Revisited

Essential computational question: **open world query answering.**

**Given:** "partial database" $\mathcal{D}$, query $\mathcal{Q}$, and dependencies $\mathcal{C}$

- **Tuple-generating dependencies (TGD):**
  $\forall x ( \phi(x) \rightarrow \exists y \psi(x, y))$

- **Equality-generating dependencies (EGDs)**
  $\forall x (\phi(x) \rightarrow x_i = x_j)$

**Determine:** which answers to $\mathcal{Q}$ can be inferred from $\mathcal{C}$ and $\mathcal{D}$.

**Equivalent problem:** deciding containment of queries under constraints: Determine if $\mathcal{Q}_1 \subseteq_c \mathcal{Q}_2$

Overviewed two basic techniques:
- Forward chaining (the chase)
- Backward chaining (query-rewriting)
Day 3 Overview

- We will discuss how these extend to Equality Generating Dependencies (EGDs).
- We will show how to extend forward chaining and backward chaining to wider sets of TGDs.
Recall: EGDs

- **Tuple-generating dependencies (TGD):**
  \[ \forall x \left( \phi (x) \rightarrow \exists y \; \psi (x, y) \right) \]

- **Equality-generating dependencies (EGDs)**
  \[ \forall x \left( \phi (x) \rightarrow x_i = x_j \right) \]

<table>
<thead>
<tr>
<th>Staff:</th>
<th>staffid</th>
<th>name</th>
<th>email</th>
<th>deptid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department:</th>
<th>deptid</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

EGDs include **Functional Dependencies** (Staff: \( 1 \rightarrow 2,3,4 \)) and **Keys** (Department: \( 1 \rightarrow 2,3 \))
Forward Reasoning

Recall the TGD chase:
• TGD $\forall x (\phi (x) \rightarrow \exists y \psi (x, y))$ has an **unfulfilled match** if there is homomorphism onto the left that does not extend to a homomorphism of the right.
• TGD chase step: add fresh elements and facts to make the right true.
  • **Fresh** relative to a set $B$ of values containing those in $D$

Modifying for EGDs $\forall x (\phi (x) \rightarrow x_i = x_j)$:

EGD has an unfulfilled match if there is a homomorphism on the left and the homomorphic images of variables on the right are not equal

**EGD chase step**: identify elements on the right to make the EGD true.
Chasing EGDs

Modifying for EGDs:

EGD has an unfulfilled match if there is a homomorphism on the left that does not have equalities for the right.

**EGD chase step:** identify elements on the right to make the EGD true.
In one chase step, just "mark" two elements as equivalent.
Chasing EGDs

Recall the TGD chase:

• TGD $\forall x (\phi(x) \rightarrow \exists y \psi(x, y))$ has an **unfulfilled match** if there is homomorphism onto the left that does not extend to a homomorphism of the right.

• TGD chase step: add fresh elements and facts to make the right true.
  • **Fresh** relative to a set $B$ of values containing those in $D$

One **round** of the TGD chase: do chase steps for all unfulfilled matches.
Chasing EGDs

Modifying for EGDs:
EGD has an unfulfilled match if there is a homomorphism on the left side that does not extend to a homomorphism of the right.

**EGD chase step:** mark elements on the right as equivalent to make the EGD true.

**EGDParallelChaseStep**(I,B):
create equivalence classes for all applicable EGD chase steps, close under transitivity, and keep one representative per class
• if no elements of B in a class, choose representative arbitrarily
• if exactly one element of B in the class, choose it as a representative
• if more than one element of B in a class, fail
EGD Chase
Step

\[ D \]

\[ R(p, q) \]

\[ R(p, s) \]

\[ \forall x \ y \ R(x, y) \rightarrow \exists u \ v \ S(x, u, v) \]

\[ \forall x \ y \ v \ w \ z \ S(x, y, w) \land S(x, v, z) \rightarrow w = z \]
EGD Chase
Step

\[ D \]

\[ R(p, q) \quad S(p, c_1, d_1) \]

\[ R(p, s) \]

\[ \forall x \ y \ R(x, y) \rightarrow \exists u \ v \ S(x, u, v) \]

\[ \forall x \ y \ v \ w \ z \ S(x, y, w) \land S(x, v, z) \rightarrow w = z \]
EGD Chase Step

\[
\begin{align*}
D \\
R(p, q) & \quad S(p, c_1, d_1) \\
R(p, s) & \quad S(p, c_2, d_2) \\
\forall x \ y \ R(x, y) & \rightarrow \exists u \ v \ S(x, u, v) \\
\forall x \ y \ v \ w \ z \ S(x, y, w) \land S(x, v, z) & \rightarrow w = z
\end{align*}
\]
EGD Chase
Step

D
R(p,q) S(p,c₁,d₁)
R(p,s) S(p,c₂,d₂)

∀ x y R(x,y) → ∃ u v S(x,u,v)

∀ x y v w z S(x,y,w) ∧ S(x,v,z) → w=z
EGD Chase
Step

\[ D \]

\[ R(p, q) \quad S(p, c_1, d_1) \]

\[ R(p, s) \quad S(p, c_2, d_1) \]

\[ \forall x \; y \; R(x, y) \rightarrow \exists u \; v \; S(x, u, v) \]

\[ \forall x \; y \; v \; w \; z \; S(x, y, w) \land S(x, v, z) \rightarrow w = z \]
EGD Chase
Step Example 2

\[ D \]
\[ R(p, q) \]
\[ R(p, s) \]

\[ \forall x \ y \ R(x, w) \land R(y, z) \rightarrow w=z \]

Chase Step fails

Chase step can only identify "nulls"
EGD/TGD Chase

**Recall:** TGD chase:

Start with \( I := D \), then iterate the following as long as there is a TGD \( \tau \) with an unfulfilled match \( M \) for left side of \( \tau \) in \( I \)

\[
I := \text{TGDParallelChaseStep}(I,B)
\]

EGD and TGD chase:

Start with \( I := D \), then iterate the following as long as there is a TGD or EGD with an unfulfilled match \( M \) for left side in \( I \)

If \( \text{EGDParallelChaseStep}(I,B) \) succeeds

\[
I := \text{EGDParallelChase}(I,B)
\]

\[
I := \text{TGDParallelChase}(I,B)
\]

Else Fail
Universality

**Theorem** If the EGD-TGD chase terminates successfully then it produces a universal model.

**Corollary:** For a non-boolean conjunctive query $Q$ (mentioning only values in $B$) and set of TGDs $C$ such that the EGD-TGD chase terminates successfully:

a tuple $t$ using values from $D$ or $B$ is a certain answer to $Q$ with respect to constraints $C$ iff $t$ is in the result of $Q$ evaluated on $\text{Chase}(D,C)$.

Similarly, we can compute the certain answers of a non-boolean query using the chase.
Quiz: Failure and Success

**Corollary:** For a non-boolean conjunctive query $Q$ (mentioning only values in $B$) and set of TGDs $C$ such that the EGD-TGD chase terminates successfully:

a tuple $t$ using values from $D$ or $B$ is a certain answer to $Q$ with respect to constraints $C$ iff $t$ is in the result of $Q$ evaluated on $\text{Chase}(D, C)$.

Similarly, we can compute the certain answers of a non-boolean query using the chase.

What can we say about our basic computational problems if the chase fails?
More General EGD-TGD chase

If the EGD-TGD chase does not terminate, one can still define the chase as a limit.
Recall: Infinite TGD Chase Model

\[ I_0 = D \]

\[ I_1 = \text{Perform Chase Steps on } D \]

\[ I_2 = \text{Parallel Chase Steps on } I_1 \]

\[ I_3 = \text{Parallel Chase Steps on } I_2 \]

\[ I_4 = \text{Parallel Chase Steps on } I_3 \]

... 

\[ I_\infty = \text{union of } I_i \]
Infinite EGD-TGD Chase Model

\[ I_0 = D \quad I_1 = \text{Perform Chase Steps on } D \]
Infinite EGD-TGD Chase Model

$I_0 = D$  $I_1 = \text{Perform Chase Steps on } D$  $I_2 = \text{Parallel Chase Steps on } I_1$
Infinite EGD-TGD Chase Model

\[ I_0 = D \quad I_1 = \text{Perform Chase Steps on } D \quad I_2 = \text{Parallel Chase Steps on } I_1 \]

\[ I_3 = \text{Parallel Chase Steps on } I_2 \ldots \]
Infinite EGD-TGD Chase Model

\[ I_0 = D \quad I_1 = \text{Perform Chase Steps on } D \quad I_2 = \text{Parallel Chase Steps on } I_1 \]

\[ I_3 = \text{Parallel Chase Steps on } I_2 \ldots \]

\[ I_\infty = \text{direct limit of } I_1 \]
More General Universality

**Theorem** If the EGD-TGD chase does not fail then it produces a universal model.

**Corollary:** For a non-boolean conjunctive query $Q$ (mentioning only values in $B$) and set of TGDs $C$ such that the EGD-TGD chase does not fail:

a tuple $t$ using values from $D$ or $B$ is an infinitely-certain answer to $Q$ with respect to constraints $C$ iff $t$ is in the result of $Q$ evaluated on $\text{Chase}(D, C)$.
Since the chase may not terminate with just TGDs, clearly it can fail to terminate with TGDs and EGDs. There are a few obvious termination conditions.

**Theorem** For constraints consisting of source-to-target TGDs as well as EGDs, the chase will terminate (with success or failure).

**Theorem** For constraints consisting of full TGDs as well as EGDs, the chase will terminate.
Tame EGD/TGD cases

Intuitive reasons why a key constraint would not impact the chase. The keys "do not conflict" with any chase step using a TGD $\tau$.

$$\text{TGD } \tau = \forall x \ y \ S(x, y) \rightarrow \exists z \ R(x, z)$$

A key constraint on any relation other than $R$ does not interfere with a chase step on $\tau$.

A key constraint on $R[1]$ does not interfere with a chase step on $\tau$. 
Wild EGD/TGD cases

Intuitive reasons why a key constraint might impact the chase.

\[ \text{TGD } \tau = \forall \ x_1 \ x_2 \ y \ S(x_1, x_2, y) \rightarrow \exists \ z \ R(x_1, x_2, z) \]

A key constraint on \( R[1] \) does interfere with a chase step on \( \tau \).
Wild EGD/TGD cases

Intuitive reasons why a key constraint might impact the chase.

\[ \text{TGD } \tau = \forall x \ S(x) \rightarrow \exists z \ \ R(x, z, z) \]

Database \( D \) has \( R(a, b, c) \) and \( S(a) \)

A key constraint on \( R[1] \) does interfere with a chase step on \( \tau \).
EGD/TGD chase termination

Let $K$ be a key constraint $S : i_1 \ldots i_n \rightarrow \ast$ and $\tau$ a TGD of the form

$$\forall x_1 \ldots x_j y_1 \ldots y_k \phi(x_1 \ldots , y_1 \ldots ) \rightarrow$$

$$\exists z_1 \ldots z_1 R(x_1 \ldots , z_1 \ldots )$$

Then, $K$ is non-conflicting with $\tau$ iff either
1. the relation $S$ is different from $R$ or
2. some $i_1 \ldots i_n$ does not carry an $x_i$ on the right or
3. every variable $z_i$ appears only once on the right of $\tau$ and
   $i_1 \ldots i_n$ are exactly the positions in $R$ carrying an $x_i$ on the right of $\tau$

A set of keys $K$ is non-conflicting with a set $T$ of TGDs if each key in $K$ is non-conflicting with every TGD in $T$. 
Theorem If \( K \) is non-conflicting with TGDs \( T \) and the initial database satisfies \( K \) then the chase on \( K \cup T \) is the same as chasing with \( T \).

In particular, if \( T \) is in one of the terminating classes of TGDs, then the chase on \( K \cup T \) terminates as well.

**Moral:** Don't bother doing the EGD-TGD chase: just do the easier TGD chase.
Undecidability

Recall: basic undecidability result about IDs and FDs:

Theorem The implication problem for a collection $c$ of IDs and Functional dependencies is undecidable

Theorem The following problem is undecidable:
Input: a boolean conjunctive query $Q$, database $D$, constraints $C$ consisting of Inclusion dependencies and keys
Output: yes if $Q$ follows from $D$ under the constraints

Theorem The following problem is undecidable:
Input: database $D$, constraints $C$ consisting of Inclusion dependencies and keys
Output: yes if the chase terminates on $D$
EGD/TGD Summary

- The chase can be extended to EGDs (thus in particular to keys and foreign keys).
- Universality still holds: when the chase converges, it can be used to answer queries.
- Termination conditions in the presences of EGDs are much more limited.
Recall: Infinite Chase

Binary relation $R$ with inclusion dependency:

$R[2] \subseteq R[1]
\forall x \ y \ R(x, y) \rightarrow \exists z \ R(y, z)$
Beyond The Terminating Chase

**Recall:** the only technique for forward chained reasoning involved showing that the chase terminates.

But terminating classes are very restrictive.
Tame TGDs

New approach to get decidability of open world query answering of $\mathcal{Q}$ w.r.t. a set of TGDs $\mathcal{C}$

Restrict TGDs in $\mathcal{C}$ so that, although chase may be infinite, we can stop chasing at some point and know that "nothing new in terms of $\mathcal{Q}$ can happen later on".
Chase

Recall:

**Standard Chase** for D and C (abbrev. Chase \((D, C)\))

Start with \(I := D\), then iterate the following as long as there is a TGD \(\tau\) with an unfulfilled match \(M\) for left of \(\tau\) in \(I\)

\[ I := \text{ParallelChaseStep}(I) \]

Return \(I\)

**Approximations to the chase:**

\(\text{Chase}_n (D, C) = \) do the above for at most \(n\) steps
Guarded TGDs

TGD:
constraint of form
\[ \forall x_1 \ldots x_j \ A_1(x_1 \ldots) \land \ldots \land A_m(x_j \ldots) \to \exists y_1 \ldots y_k \ B_1(x_1 \ldots x_j, y_1 \ldots y_k) \land \ldots \]

Guarded TGD:
constraint of form
\[ \forall x_1 \ldots x_j \ A(x_1 \ldots x_j) \land (\text{other atoms using only } x_1 \ldots x_j) \to \exists y_1 \ldots y_k \ B_1(x_1 \ldots x_m, y_1 \ldots y_k) \land \ldots \]
Examples of Guarded TGDs

Guarded TGD:
constraint of form
\[ \forall x_1 \ldots x_j \ A(x_1 \ldots x_j) \]
\[ \land \ (\text{other atoms using only } x_1 \ldots x_j) \]
\[ \rightarrow \]
\[ \exists y_1 \ldots y_k \ B_1(x_1 \ldots x_j, y_1 \ldots y_k) \land \ldots \]

Guarded:
\[ \forall x_1 \ x_2 \ R(x_1, x_2) \land S(x_1) \rightarrow G(x_1, x_2) \]

Not Guarded:
\[ \forall x_1 \ x_2 \ x_3 \ R(x_1, x_2) \land R(x_2, x_3) \rightarrow R(x_1, x_3) \]
Examples of Guarded TGDs

Guarded TGD:
constraint of form
\[ \forall x_1 \ldots x_j \ A(x_1 \ldots x_j) \wedge (\text{other atoms using only } x_1 \ldots x_j) \rightarrow \exists y_1 \ldots y_k \ B_1(x_1 \ldots x_j, y_1 \ldots y_k) \wedge \ldots \]

Includes IDs and other classes that arise in data integration and data exchange.
Guar ded
TGDs

**Theorem** For every set $C$ of Guarded TGDs, database $D$, and boolean conjunctive query $Q$, if $D$ implies $Q$ under $C$, then $\text{Chase}_n(D, C)$ satisfies $Q$

where $n$ is bounded by a doubly exponential function in $D, C, Q$

For Guarded TGDs we can search for a proof and know when to cut off the search – effective proof procedure.
Example: chasing with Guarded TGDs

\[ \forall x \ y \ R(x, y) \land Q(x) \rightarrow \exists z \ R(y, z) \land S(z) \]

\[ \forall x \ y \ R(x, y) \land S(y) \rightarrow \exists z \ R(y, z) \land Q(y) \]

\[ D = \{ R(c, d), Q(c) \} \]
Example: chasing with Guarded TGDs

\[ \forall x \ y \ R(x, y) \land Q(x) \rightarrow \exists z \ R(y, z) \land S(z) \]

\[ \forall x \ y \ R(x, y) \land S(y) \rightarrow \exists z \ R(y, z) \land Q(y) \]

\[ R(c, d), Q(c) \]

\[ R(d, e), S(e) \]
Example: chasing with Guarded TGDs

\[ \forall x \ y \ R(x, y) \land Q(x) \rightarrow \exists z \ R(y, z) \land S(z) \]

\[ \forall x \ y \ R(x, y) \land S(y) \rightarrow \exists z \ R(y, z) \land Q(y) \]

\[ R(c, d), Q(c) \]

\[ R(d, e), S(e) \]
Example: chasing with Guarded TGDs

\[ \forall x \ y \ R(x, y) \land Q(x) \rightarrow \exists z \ R(y, z) \land S(z) \]

\[ \forall x \ y \ R(x, y) \land S(y) \rightarrow \exists z \ R(y, z) \land Q(y) \]

\[ R(c, d), Q(c) \]

\[ R(d, e), S(e) \]
Example: chasing with Guarded TGDs

\[ \forall x \ y \ R(x, y) \land Q(x) \rightarrow \exists z \ R(y, z) \land S(z) \]

\[ \forall x \ y \ R(x, y) \land S(y) \rightarrow \exists z \ R(y, z) \land Q(y) \]

\[ R(c, d), Q(c) \]
\[ R(d, e), S(e) \]
\[ R(e, f), Q(e) \]
Example: chasing with Guarded TGDs

\[ \forall x \ y \ R(x, y) \land Q(x) \rightarrow \exists z \ R(y, z) \land S(z) \]

\[ \forall x \ y \ R(x, y) \land S(y) \rightarrow \exists z \ R(y, z) \land Q(y) \]

\[ R(c, d), Q(c) \]

\[ R(d, e), S(e) \]

\[ R(e, f), Q(e) \]

... 

Abstract pattern (i.e. local isomorphism type) repeats.
Tree-like Chase

As chase progresses, place facts into bags, with each bag guarded by an atom. Bags form a tree.
Cutting off the Chase

We can cut off the chase whenever bags "repeat".

We can cut off the chase whenever bags "repeat".
Cutting off the Chase

B_1 \quad B_2

R(c,d) \quad H(c,c)

B_3 \quad B_4
Cutting off the Chase: Mysteries

- Why can the chase be grouped as a tree?
- Why does local information suffice to characterize when two bags act "the same"?

If $\mathcal{Q}$ is very simple, then we can cut off a branch ending in bag $b$ if $b$ has an isomorphic ancestor $b'$.  
- For general CQs, have to maintain additional "state" information about a bag, and only cut-off the chase if there is a mapping from $b$ to ancestor $b'$ that preserves this information.
Adding EGDs to the infinite chase

For non-conflicting keys, our prior results already tell us that the technique extends:

**Theorem** For every set \( C \) of Guarded TGDs, database \( D \), and boolean conjunctive query \( Q \), we can compute an \( n \) such that if \( D \) implies \( Q \) under \( C \), then \( \text{Chase}_n(D,C) \) satisfies \( Q \)

**Theorem** If \( K \) is non-conflicting with TGDs \( T \) and the initial database satisfies \( K \) then the chase on \( K \cup T \) is the same as chasing with \( T \)

**Corollary** For Guarded TGDs and non-conflicting keys, if \( D \) satisfies the keys, then we just need to chase with the Guarded TGDs up to \( n \)
Further Work: Infinite Chasing

Guarded TGDs are only one class for which one can determine when to cut off the chase. This is an extremely active area of research with close ties to finite model theory and automata theory.
Further Work: Infinite Chasing

Recall:

**certain answers** for query \( Q \), database \( D \), constraints \( C \)

= results returned by \( Q \) on all (finite) databases

**infinitely certain answers** for \( Q, D, C \) = results returned by \( Q \) on all models for \( D \) and \( C \)

It is easy to show that for terminating chase classes, there is no difference between these notions.

It is not at all clear that this is true for the "tame infinite classes", like Guarded TGDs.

**Deeper results**: finite implication and general implication also co-incide for these classes.

See [Rosati11](#) and [BaranyGottlobOtto14](#) in the references.
Beyond First-order rewritability

**Recall:** A query $Q$ is **first-order rewritable** according to constraints $C$ if there is some first-order query $Q'$ such that:

for any database $D$, $Q'$ evaluated on $D$ returns the certain answers of $Q$ for $D, C$

We have seen that Linear TGDs are FO rewritable.

What about Guarded TGDs?
Recall: Limits of Rewriting

Constraint

$$\forall x \ y \ z \ R(x, y) \land R(y, z) \rightarrow R(x, z)$$

Query  $$Q = \exists x \ y \ U(x) \land R(x, y) \land V(y)$$ is not first-order rewritable.

Is it a Guarded TGD?
Non-Rewritability of Guarded TGDs

Consider **Guarded** TGD constraint:

\[ \forall x \ y \ R(x, y) \land V(y) \rightarrow V(x) \]

Query \( Q = \exists x \ y \ U(x) \land R(x, y) \land V(y) \) is **not** first-order rewritable.

\( Q \) is certain iff there is a \( U \) node that transitively reaches \( V \)

\[ U \rightarrow V \rightarrow V \rightarrow V \]
Beyond
First-order rewritability

**Recall:** A query $Q$ is **first-order rewritable** according to constraints $C$ if there is some first-order query $Q'$ such that:

for any database $D$, $Q'$ evaluated on $D$ returns the certain answers of $Q$ for $D$, $C$

We have seen that Linear TGDs are FO rewritable. Guarded TGDs are not.

Thus for Guarded TGDs, we need to look to a more general query language for rewritings.
Recall: Query Languages

- FO
- UCQ
- CQ
- Datalog
A Datalog query consists of:

- Collection of intensional predicates with one distinguished goal predicate.

- Rules of form

\[ U(x_1 \ldots x_m) \leftarrow A_1(x_1 \ldots y_1 \ldots) \land \ldots \land A_n(x_1 \ldots y_1 \ldots) \]

\( U \) must be an intensional predicate. 
\( A_i \) are either intensional predicates or extensional (schema) predicates.
Datalog Example

Reach(x, y) ← R(x, y)
Reach(x, y) ← ∃ z Reach(x, z) ∧ Reach(z, y)
Datalog Semantics

To evaluate $\varphi$, calculate iteratively for each $\mathbf{u}$, the $k$th approximation to each intensional predicate $\mathbf{u}^k$.

- $\mathbf{u}^0$ is empty

- We form each $\mathbf{u}^{k+1}$ by adding to $\mathbf{u}^k$ tuples formed from evaluating rules using the $k$ approximations

For each rule

$\mathbf{U}(x_1 \ldots x_m) \leftarrow A_1(x_1 \ldots y_1 \ldots) \land \ldots A_n(x_1 \ldots y_1 \ldots)$

$\mathbf{U}_{k+1} = \mathbf{U}_k \cup \{x_1 \ldots x_m \mid \exists y_1 \ldots A_1^k(x_1 \ldots y_1 \ldots) \land \ldots A_n^k(x_1 \ldots y_1 \ldots)\}$
Datalog Semantics

To evaluate $Q$, calculate iteratively for each $U$, the $k^{th}$ approximation to each intensional predicate $U^k$.

- $U^0$ is empty
- We form each $U^{k+1}$ by adding to $U^k$ tuples formed from evaluating rules using the $k$ approximations

Eventually predicates reach a fixpoint. Return value of goal predicate.
Back to Example

\[
\begin{align*}
\text{Reach}(x, y) & \leftarrow \text{R}(x, y) \\
\text{Reach}(x, y) & \leftarrow \exists \ z \ \text{Reach}(x, z) \land \text{Reach}(z, y)
\end{align*}
\]

\[
\begin{align*}
\text{Reach}_0 & = \emptyset \\
\text{Reach}_1 & = (A, B), (A, C), (C, D) \\
\text{Reach}_2 & = \text{Reach}_1 \cup \{ (A, D) \} \\
\text{Reach}_3 & = \text{Reach}_2: \text{fixpoint} \\
\text{Return} \ \text{Reach}_3
\end{align*}
\]
Some important complexity classes

- **PTIME (a.k.a. P)**
  - Problems that can be solved by an algorithm that runs in **polynomial time** (e.g., linear, quadratic, cubic, etc.)

- **NP**
- **coNP**

- **PSPACE**
  - Problems that can be solved by an algorithm that uses a **polynomial amount of memory**

- **EXPTIME**
  - Problems that can be solved by an algorithm that runs in **exponential time** (e.g., $2^n$)
NP

• NP: problems that can be solved by a non-deterministic algorithm that runs in polynomial time.
  – “problems whose solutions are easy to check but not necessarily easy to find.”

• The million dollar question: P =? NP

• Certain problems in NP are known to be NP-complete:
  – Every problem in NP has a PTIME-reduction to this problem.
  – If this problem is in PTIME then every NP-problem is in PTIME.
  – In other words, if this problem is in PTIME then P=NP.
Complexity of CQ evaluation

• **CQ evaluation:**
  – **Input**: a database $D$, conjunctive query $q$, and a tuple $t$.
  – **Problem**: Is it the case that $t \in q(D)$?

• Algorithm for conjunctive query evaluation:
  – When there is an existential quantifier, try all possibilities.
  – Worst-case time of this algorithm: at most $|D|^{|q|}$.

• Does that count as polynomial time or exponential time?
  – It’s exponential in $|q|$.
  – It’s polynomial in $|D|$, but the degree depends on $|q|$.
Data Complexity vs. Combined Complexity

• **Data complexity** of query evaluation:
  – The complexity of evaluating a fixed query (where the input of the problem is only the database and tuple).

• **Combined complexity** of query evaluation:
  – The complexity of query evaluation, where the query is part of the input

• Data complexity tends to be a more meaningful measure:
  – Queries tend to be small. Databases tend to be big.
  – We care most about *scalability to large databases* and not so much about scalability to large queries.
## Complexity of Query Evaluation

<table>
<thead>
<tr>
<th>Query language</th>
<th>Data complexity</th>
<th>Combined complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ and UCQ</td>
<td>PTIME</td>
<td>NP-complete</td>
</tr>
<tr>
<td>FO</td>
<td>PTIME</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>Datalog</td>
<td>PTIME</td>
<td>EXPTIME-complete</td>
</tr>
</tbody>
</table>

We only considered **worst-case data/combined complexity** but there are many other, more refined notions of complexity: (e.g., average-case complexity)
Quiz: graph 3-colorability

• Famous NP-complete problem: **graph 3-colorability**

• **Graph**: database D for schema \{E\} where E is a symmetric and irreflexive binary relation.

• **Problem**: Can I color the nodes of the graph using Red, Green and Blue so that adjacent nodes have different colors?

![3-colorable graph](image1)

![non-3-colorable graph](image2)
Quiz: graph 3-colorability

1. Is there a FO query q such that q(G)=true iff G is 3-colorable?
2. Is there a Datalog query q such that q(G)=true iff G is 3-colorable?

• Answer for FO:
  – FO queries have PTIME data complexity. That means: every FO query has a PTIME algorithm.
  – If there is a FO query that defines 3-colorability, then 3-colorability is in PTIME, and therefore P=NP.
  – Conclusion: assuming P≠NP, 3-colorability is not FO-definable.
  – Indeed, we can prove (without assuming P≠NP) that 3-colorability is not FO-definable.

• Answer for Datalog: same.
Datalog Complexity

To evaluate $\mathcal{Q}$, calculate iteratively for each $\mathcal{U}$, the $k^{th}$ approximation to each intensional predicate $\mathcal{U}^k$

- $\mathcal{U}^0$ is empty
- We form each $\mathcal{U}^{k+1}$ by adding to $\mathcal{U}^k$ tuples formed from evaluating rules using the $k$ approximations

**Theorem:** A Datalog query $\mathcal{Q}$ can be evaluated on a database $\mathcal{D}$ in time polynomial in $\mathcal{D}$
Non-recursive Datalog

Can restrict Datalog so that intensional predicates are divided into integer levels, where predicates in level \( n \) are defined only using predicates of level \( < n \)

\[
\begin{align*}
\text{TwoHop}(x, y) & \leftarrow \exists z \ R(x, z) \land R(z, y) \\
\text{FourHop}(x, y) & \leftarrow \exists z \ \text{TwoHop}(x, z) \land \text{TwoHop}(z, y)
\end{align*}
\]

Proposition
Non-recursive Datalog has the same expressiveness as UCQs.
Datalog vs. Dependencies

Datalog has a similar syntax as dependencies. But Datalog queries compute an output relation from input relations using a fixpoint.

\[ R(x, y) \rightarrow \text{Reach}(x, y) \]
\[ \text{Reach}(x, z) \land \text{Reach}(z, y) \rightarrow \text{Reach}(x, y) \]

Constraints which give a property that an input database with \text{Reach} and \text{R} should satisfy.

\[ \text{Reach}(x, y) \leftarrow R(x, y) \]
\[ \text{Reach}(x, y) \leftarrow \exists z \text{ Reach}(x, z) \land \text{Reach}(z, y) \]

Datalog query taking as input a database with only \text{R} and computes \text{Reach} using a fixpoint computation.
Datalog Rewritability

A query \( Q \) is **Datalog rewritable** according to constraints \( C \) if there is some Datalog query \( Q' \) such that:

for any database \( D \), \( Q' \) evaluated on \( D \) returns the certain answers of \( Q \) for \( D, C \)

Datalog rewritability also implies that the certain answers can be computed "scalably in the data".
Theorem If the constraints consist of guarded TGDs only, then every conjunctive query is Datalog-rewritable.

Proof: not so easy. See the references.
Inefficient Rewritability

Consider constraints

\[ A_1(x) \rightarrow A_0(x) \]
\[ A_2(x) \rightarrow A_1(x) \]
\[ \ldots \]
\[ A_n(x) \rightarrow A_{n-1}(x) \]

And query
\[ \exists x \ y \ D(x) \land A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]
\[ A_2(x) \rightarrow A_1(x) \]
\[ \ldots \]
\[ A_n(x) \rightarrow A_{n-1}(x) \]

One step:

\[ [A_0(x) \land A_0(y) \land D(x) \land E(y)] \]

And query

\[ \exists x \ y \ D(x) \land A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]
\[ A_2(x) \rightarrow A_1(x) \]
\[ \ldots \]
\[ A_n(x) \rightarrow A_{n-1}(x) \]

One step:
\[ [A_0(x) \land A_0(y) \land D(x) \land E(y)] \lor \]
\[ [A_0(x) \land A_1(y) \land D(x) \land E(y)] \]

And query
\[ \exists x y \ D(x) \land \ A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]

\[ A_2(x) \rightarrow A_1(x) \]

\[ \ldots \]

\[ A_n(x) \rightarrow A_{n-1}(x) \]

Two step:

\[ [A_0(x) \land A_0(y) \land D(x) \land E(y)] \lor \]

\[ [A_0(x) \land A_1(y) \land D(x) \land E(y)] \lor \]

\[ [A_1(x) \land A_1(y) \land D(x) \land E(y)] \]

And query

\[ \exists x \, y \, D(x) \land A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]

\[ A_2(x) \rightarrow A_1(x) \]

\[ \ldots \]

\[ A_n(x) \rightarrow A_{n-1}(x) \]

Continue rewriting:
\[ [A_0(x) \land A_0(y) \land D(x) \land E(y)] \lor \]
\[ [A_0(x) \land A_1(y) \land D(x) \land E(y)] \lor \]
\[ [A_1(x) \land A_1(y) \land D(x) \land E(y)] \]

And query
\[ \exists x \ y \ D(x) \land \]
\[ A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]

\[ A_2(x) \rightarrow A_1(x) \]

\[ \ldots \]

\[ A_n(x) \rightarrow A_{n-1}(x) \]

Continue rewriting:

\[ [A_0(x) \land A_0(y) \land D(x) \land E(y)] \lor \]

\[ [A_0(x) \land A_1(y) \land D(x) \land E(y)] \lor \]

\[ [A_1(x) \land A_1(y) \land D(x) \land E(y)] \]

\[ [A_1(x) \land A_1(y) \land D(x) \land E(y)] \lor \]

\[ [A_1(x) \land A_2(y) \land D(x) \land E(y)] \]

And query

\[ \exists x \ y \ D(x) \land A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]

\[ A_2(x) \rightarrow A_1(x) \]

\[ \ldots \]

\[ A_n(x) \rightarrow A_{n-1}(x) \]

Continue rewriting:

\[ [A_0(x) \land A_0(y) \land D(x) \land E(y)] \lor \]

\[ [A_0(x) \land A_1(y) \land D(x) \land E(y)] \lor \]

\[ [A_1(x) \land A_1(y) \land D(x) \land E(y)] \]

And query

\[ \exists x \ y \ D(x) \land A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]
\[ A_2(x) \rightarrow A_1(x) \]
\[ \ldots \]
\[ A_n(x) \rightarrow A_{n-1}(x) \]

Continue rewriting:
\[ [A_0(x) \land A_0(y) \land D(x) \land E(y)] \lor \]
\[ [A_0(x) \land A_1(y) \land D(x) \land E(y)] \lor \]
\[ [A_1(x) \land A_1(y) \land D(x) \land E(y)] \]
\[ [A_1(x) \land A_1(y) \land D(x) \land E(y)] \lor \]
\[ [A_1(x) \land A_2(y) \land D(x) \land E(y)] \lor \]
\[ [A_2(x) \land A_2(y) \land D(x) \land E(y)] \ldots \]

And query
\[ \exists x \; y \; D(x) \land A_0(x) \land A_0(y) \land E(y) \]
Inefficient Rewritability

One can show that in the previous family of examples, any UCQ rewriting must be at least exponential.

**Solution:** get more succinct rewriting by going to a more powerful query language.
Efficient Rewritability

\[ A_1(x) \rightarrow A_0(x) \]

\[ A_2(x) \rightarrow A_1(x) \]

\[ \ldots \]

\[ A_n(x) \rightarrow A_{n-1}(x) \]

And query

\[ \exists x \ y \ D(x) \land A_0(x) \land A_0(y) \land E(y) \]

Non-recursive Datalog rewriting:

\[ \text{Goal}() \leftarrow U_0(x) \land U_0(y) \land D(x) \land E(y) \]

\[ U_0(x) \leftarrow A_0(x) \]

\[ U_0(x) \leftarrow A_1(x) \]

\[ \ldots \]
Efficient Rewritability

**Theorem** If the constraints $c$ consisting of linear TGDs only, then every conjunctive query $q$ is efficiently non-recursive Datalog-rewritable

That is, there is a rewriting that is polynomial in the size of $q$ and $c$. 
Efficient Datalog Rewritability

**Theorem** If the constraints consist of Full TGDs only, then every conjunctive query is non-recursive-Datalog-rewritable

Proof: same idea as in the example.
EGDs and Efficient Rewritability

**Theorem** If the constraints \( c \) consist of linear TGDs and non-conflicting keys, then every conjunctive query \( q \) is efficiently Datalog-rewritable.

That is, there is a rewriting that is polynomial in the size of \( q \) and \( c \).
New summary

- We can extend forward chaining to larger classes of TGDs.

- We can extend backward-chaining to larger classes, by having rewritings in Datalog. We could also use Datalog to get smaller rewritings even when First Order rewritings exist.

- The framework can be extended to EGDs, but under very restricted conditions.
References

EGDs and the chase:
Much of the material here is also covered in the survey paper mentioned for day 2:
Adrian Onet The Chase Procedure and Its Applications in Data Exchange In Data, Exchange, Information, and Streams (2013)

Non-conflicting keys and TGDs are discussed in:
Andrea Calì, Domenico Lembo, Riccardo Rosati
Query rewriting and answering under constraints in data integration systems
References

Guarded TGDs and non-terminating chase:


Vince Bárány, Georg Gottlob, Martin Otto: *Querying the Guarded Fragment* Logical Methods in Computer Science 10(2) (2014)
References

Datalog rewritability:
Vince Bárány, Michael Benedikt, Balder ten Cate: *Rewriting Guarded Negation Queries*. MFCS 2013: 98-110

Efficient Datalog rewritability:
Modern Database Dependency Theory

Lecture 4: Applications to Query Reformulation

Michael Benedikt & Balder ten Cate
Days 2-3 Revisited

Essential computational question: **open world query answering.**

**Given:** "partial database" $\mathcal{D}$, query $\mathcal{Q}$, and dependencies $\mathcal{C}$

- **Tuple-generating dependencies (TGD):**
  $$\forall x ( \phi (x) \rightarrow \exists y \; \psi (x, y) )$$
- **Equality-generating dependencies (EGDs)**
  $$\forall x ( \phi (x) \rightarrow x_i = x_j)$$

**Determine:** which answers to $\mathcal{Q}$ can be inferred from $\mathcal{C}$ and $\mathcal{D}$.

**Equivalent problem:** deciding containment of queries under constraints: Determine if $\mathcal{Q}_1 \subseteq_c \mathcal{Q}_2$

Overviewed two techniques:
- Forward chaining (the chase)
- Backward chaining (query-rewriting)
Applications overview
Or: why you should *still* stay in this course.
Query Reformulation under Access Restrictions

**Given**: a source query $Q$ set of dependencies describing the schema, and some restriction $T$ on a target query

**Goal**: generate a target query $Q_T$ (Conjunctive Query, First Order) that obeys the restriction of $T$ and gives the same answers as $Q$ (on all databases satisfying the constraints)

**Source Query**: conjunctive query over full schema

**Restriction on target query**: a subset $T$ of the relations that are allowed in the target
Example

\[\forall \text{did} \ \text{dname} \ \text{mgrid} \ \text{Department}(\text{did}, \ \text{dname}, \ \text{mgrid}) \rightarrow \exists \ \text{N Employee}(\text{mgrid}, \ \text{N}, \ \text{did})\]

// Every Department Manager is an employee

\[\forall \text{eid} \ \text{ename} \ \text{did} \ \text{Employee}(\text{eid}, \ \text{ename}, \ \text{did}) \rightarrow \exists \ \text{D M Department}(\text{did}, \ \text{D}, \ \text{M})\]

// Every Employee works in a department

Goal: a CQ reformulation of \(Q = \{\text{did} | \exists \ \text{eid} \ \text{ename Employee}(\text{eid}, \ \text{ename}, \ \text{did})\}\) using restricted vocabulary \text{Department}

Desired Output:
\(Q_T = \{\text{did}: \exists \ \text{dname} \ \text{mgrid Department}(\text{did}, \ \text{dname}, \ \text{mgrid})\}\)
Reformulations vs. Rewritings

In reformulations:

• Target vocabulary specified
• Target vocabulary data is complete
• Answers must be complete
  – Reformulations may not exist.
Subsumes Many DB Problems

Querying using materialized views

Querying in data integration

Optimizing queries using constraints
### Querying using Materialized Views

<table>
<thead>
<tr>
<th>User Query</th>
<th>Exported Views</th>
<th>Backend DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q ) defined over ( R_1 \ldots R_n )</td>
<td>( v_1 ) defined by ( Q_1 )</td>
<td>( R_1 \ldots R_n )</td>
</tr>
<tr>
<td>( v_2 ) defined by ( Q_2 )</td>
<td>( \ldots )</td>
<td>( v_n ) defined by ( Q_n )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

**Goal:** see if \( Q \) can be answered using only information available in the views.

Translating to reformulation problem:
We have a schema that includes the \( v_i \) and also the base relations, with **view-based integrity constraints**

\[
\forall x_1 \ldots x_k \quad v_i(x_1 \ldots x_k) \leftrightarrow Q_i(R_1, \ldots, x_1 \ldots x_k)
\]

The relations \( v_i \) are in the restricted subset \( T \), while the backend relations \( R_1 \ldots R_n \) are not.
## Data Integration

<table>
<thead>
<tr>
<th>Integrated Schema</th>
<th>Mappings</th>
<th>Backend Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall \text{address},... )</td>
<td>( \text{Hotels(&quot;Hilton&quot;, address,...) } \leftrightarrow \text{HiltonHotels(address,...)} )</td>
<td>( \text{Hilton Hotels(address,...)} )</td>
</tr>
<tr>
<td>( \text{Hotels(chain, address,...)} )</td>
<td>( \text{Marriott Hotels(...)} )</td>
<td>( \text{UnitedFlights(flightnum,...)} )</td>
</tr>
<tr>
<td>( \text{Flights(flightnum,...)} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Users write \( Q \) over integrated schema

Virtual "Global Relations" on which queries are posed are not in the target \( T \), remote sources related to the global schema by constraints are in \( T \).
Simplifying Queries using Constraints

Start with $Q$ over $S$, and find $Q_T$ (also over $T=S$) that is simpler, using the constraints.

**Simpler:** $Q_T$ uses fewer relations, $Q_T$ is smaller, $Q_T$ is existential while $Q$ has quantifier alternation.
And now for something completely different ....
Proofs and Validity

\( \rho_1 \models \rho_2 \) aka “\( \rho_1 \) entails \( \rho_2 \)” in any instance where \( \rho_1 \) holds, \( \rho_2 \) holds. Same as: \( \rho_1 \to \rho_2 \) holds on all instances.

When \( \rho_1 (x_1 \ldots x_n) \) and \( \rho_2 (x_1 \ldots x_n) \) are formulas with free variables, \( \rho_1 \models \rho_2 \) means that \( \forall x_1 \ldots x_n \rho_1 (x_1 \ldots x_n) \to \rho_2 (x_1 \ldots x_n) \) is valid.

It is a semantic notion of implication.

We can develop a proof system that allows us to show entailments. We will always look for proof systems that are complete:
\( \rho_1 \) proves \( \rho_2 \) (written \( \rho_1 \vdash \rho_2 \)) exactly when \( \rho_1 \models \rho_2 \)
Proofs and Validity

\( \rho_1 \models \rho_2 \) aka “\( \rho_1 \) entails \( \rho_2 \)” : in any instance where \( \rho_1 \) holds, \( \rho_2 \) holds.
Same as: \( \rho_1 \rightarrow \rho_2 \) holds on all instances.

We can develop a **proof system** that allows us to show entailments. We will always look for proof systems that are **complete**:

\( \rho_1 \) proves \( \rho_2 \) (written \( \rho_1 \vdash \rho_2 \)) exactly when \( \rho_1 \models \rho_2 \)

**Note:** entailment considers **all** databases even infinite ones.
Most proof systems capture this "strong entailment", not entailment over finite structures.
For the next few slides, by default we will allow infinite structures.
The Chase and Proof Systems

The "universality theorem" implies that the chase is a complete proof system for query containment under constraints entailments:

\[ Q_1 \wedge c \models Q_2 \quad Q_1 \subseteq_c Q_2 \]

Where \( Q \) and \( Q' \) are conjunctive queries and \( c \) is a conjunction of TGDs
Tableau Proof System

We look at a more general proof system for first-order logic entailments.

Procedure for proving entailments $\rho_1 \vdash \rho_2$ where $\rho_1, \rho_2$ are in **Negation Normal Form (NNF)**: built up from atoms and negated atoms using $\forall, \exists, \land, \lor.$

**Proposition** There is a procedure $\text{NNF}(\phi)$ converting any FO formula to NNF.

Our tableau proceed by attempting to prove inconsistency of $\rho_1 \land \text{NNF}(\neg \rho_2)$
Tableau Proof System

Given FO sentence in NNF, try to generate a contradiction.

Tableau = tree containing collections of sets of formulas
Tree can grow downwards by adding children to a leaf node $s$:

Given $s$ containing $\forall x \phi$, spawn a child where we add $\phi(a)$ for some existing $a$ (show we can get a contradiction for some $x$)

Given $s$ containing $\exists x \phi$ can generate a child adding to $s \phi(a)$ where $a$ is not mentioned in $s$ (show we can get a contradiction for an $x$, while making no special assumptions on $x$)
Tableau Proof System

Given first-order logic sentence, try to generate a contradiction.
Tableau = tree containing collections of sets of formulas

Tree can grow downwards by adding children to a leaf node $s$:

Given $s$ containing $A \land B$, spawn a child which adds $A$ and $B$

Given $s$ containing $A \lor B$, spawn two children $s \cup \{A\}$ and $s \cup \{B\}$ (*show that either $A$ or $B$ leads to a contradiction*)
Tableau Proof System

Given first-order logic sentence, try to generate a contradiction.
Tableau = tree containing collections of sets of formulas

A node is inconsistent if it contains both ρ and ¬ρ for some atomic formula ρ

A tableau is inconsistent if every path leads to a contradiction.
Tableau Proof System

Given first-order logic sentence, try to generate a contradiction.

\[
\exists x \ A(x) \land \neg B(x) \land C(x) \land \forall y \ [ (\neg A(y) \land E(y)) \lor B(y) ]
\]

\[
A(c) \land \neg B(c) \land C(c) \land \forall y \ [ (\neg A(y) \land E(y)) \lor B(y) ]
\]

\[
A(c), \neg B(c), C(c), \forall y \ [ (\neg A(y) \land E(y)) \lor B(y) ]
\]

\[
A(c), \neg B(c), C(c), \ [ (\neg A(c) \land E(c)) \lor B(c) ]
\]
Tableau Proof System

Given first-order logic sentence, try to generate a contradiction.

$$\exists x \ A(x) \land \neg B(x) \land C(x) \land \forall y [ (\neg A(y) \land E(y)) \lor B(y) ]$$

$$A(c) \land \neg B(c) \land C(c) \land \forall y [ (\neg A(y) \land E(y)) \lor B(y) ]$$

$$A(c), \neg B(c), C(c), \forall y [ (\neg A(y) \land E(y)) \lor B(y) ]$$

$$A(c), \neg B(c), C(c), [ (\neg A(c) \land E(c)) \lor B(c) ]$$

$$A(c), \neg B(c), C(c), \neg A(c) \land E(c) \quad A(c), C(c), \neg B(c), B(c)$$

$$A(c), \neg B(c), C(c), \neg A(c), E(c) \quad x$$
Completeness of Tableaux

**Theorem** Tableaux are a complete proof system for first order logic.

For $\rho_1$ and $\rho_2$ in first order logic we have: $\rho_1$ proves $\rho_2$ (written $\rho_1 \vdash \rho_2$) exactly when $\rho_1 \models \rho_2$. 
The Chase and Tableaux

The chase can be seen as a specialized kind of tableau.

To prove \( Q_1 \subseteq_C Q_2 \) (for boolean CQs \( Q_1, Q_2 \)) in the chase, we start with the canonical database of \( Q_1 \) and forward chain until we get a database satisfying \( Q_2 \).

In the corresponding tableau, we would start with root node \( Q_1 \wedge \text{NNF}(C) \wedge \text{NNF}(\neg Q_2) \) and derive a contradiction, where \( \text{NNF}(C) \) denotes the conjunction of the TGDs in \( C \) converted to \( \text{NNF} \).

**First step in simulation:**
use the "And"-rule to generate \( Q_1, \text{NNF}(C), \text{NNF}(\neg Q_2) \) and then instantiate the existential quantifiers of \( Q_1 \) with fresh constants ("exists"-rule of a tableau).
The Chase and Tableaux

The chase can be seen as a specialized kind of tableau.

**Simulating a chase step:**

For TGD \( \forall x_1 \ldots x_n A_1(x_1 \ldots) \land A_2(x_1 \ldots) \ldots \rightarrow \exists y_1 \ldots y_k B(x_1 \ldots y_1 \ldots y_k) \)

If we have a match of the left to \( A_1(c_1 \ldots) \land \ldots \) create new constants \( f_1 \ldots f_k \) and generate new facts to give \( B(c_1 \ldots f_1 \ldots f_k) \).

In our tableau, we would have a node with the fact \( A_1(c_1 \ldots) \land \ldots \) and also a version of the TGD written in NNF:

\[
\forall x_1 \ldots x_n [\neg A_1(x_1 \ldots) \lor \neg A_2(x_1 \ldots) \ldots \lor \exists y_1 \ldots y_k B(x_1 \ldots y_1 \ldots y_k)]
\]

Instantiate the left with existing constants \( c_1 \ldots c_n \) using the "For all" rule of Tableau gives:

\[
[\neg A_1(c_1 \ldots) \lor \neg A_2(c_1 \ldots) \ldots \lor \exists y_1 \ldots y_k B(c_1 \ldots y_1 \ldots y_k)]
\]

Using the "Or" rule on this, the "And" rule on \( A_1(c_1 \ldots) \land \ldots \) we get a bunch of terminating branches, with a remaining branch containing \( \exists y_1 \ldots y_k B(c_1 \ldots y_1 \ldots y_k) \)

Finally, use the "Exists rule" to generate \( B(c_1 \ldots f_1 \ldots f_k) \) for fresh constants \( f_1 \ldots f_k \).
The Chase and Tableaux

The chase can be seen as a specialized kind of tableau.

**Simulating the final conjunctive query match step of the chase:**

Our goal is to prove the target query $Q = \exists x_1 \ldots x_j \ A_1 (x_1 \ldots ) \land \ldots A_k (x_1 \ldots )$

In the chase, when we have a match of this with some facts $A_1 (c_1 \ldots ) \ldots A_k (c_1 \ldots )$

then the chase is successful.

In our tableau, in the goal we have $\text{NNF} (\neg Q) = \forall x_1 \ldots x_j \ [\neg A_1 (x_1 \ldots ) \lor \ldots \neg A_k (x_1 \ldots ) ]$

which will be carried down throw the tableau.

If we have derived $A_1 (c_1 \ldots ) \ldots A_k (c_1 \ldots )$ we can use the "Forall"-rule with $c_1 \ldots c_j$

to generate

$\neg A_1 (c_1 \ldots ) \lor \ldots \neg A_k (c_1 \ldots )$

and then the Or-rule repeatedly with $A_1 (c_1 \ldots ) \ldots A_k (c_1 \ldots )$ to close the tableau.
The Chase and Tableaux

**Proposition** There is a linear transformation taking every chase proof of $Q_1 \subseteq_C Q_2$ to a tableau proof, where every intermediate chase configuration maps to a tableau node consisting of the facts in that configuration along with $\text{NNF}(C) \land \text{NNF}(\neg Q_2)$
**Back to Reformulation**

Q is **first-order reformulatable** over T with respect to constraints C if there is first-order Q_T that only mentions relations in T such that for every database D satisfying the constraints C

\[ Q(D) = Q_T(D) \]

**Our goal:** determine if Q is first-order reformulatable over T w.r.t. C, and if so produce a Q_T
Necessary condition

Q is determined with respect to subvocabulary T and constraints C if:

Whenever databases D and D' satisfy C and agree on their restriction to T facts, then D and D' agree on Q.
Determinacy
and Reformulations

Q is **first-order reformulatable** over T with respect to constraints C if there is first-order $Q_T$ that only mentions relations in T such that for every database D satisfying the constraints C

$$Q(D) = Q_T(D)$$

Q is **determined** with respect to subvocabulary T and constraints C if:
Whenever databases D and D' satisfy C and agree on their restriction to T facts, then D and D' agree on Q.

**Observation:** if Q is first-order reformulatable over T, then Q is determined over T.
Converse

We will show the following result

**Theorem**
If conjunctive query $Q$ is determined over restricted vocabulary $T$ with respect to constraints $C$, then $Q$ is FO reformulatable over $T$ w.r.t. $C$

In the process, we will see how to:
• Test whether $Q$ is determined
• When $Q$ is determined, find a reformulation

These results are related to a whole theory in mathematical logic, Beth Definability and Preservation Theorems.
Determinacy as a Proof Goal

**Goal:** given a conjunctive query $Q$ we want to a) see if it is determined over $T$ by $C$, and b) if so produce a reformulation of $Q$ over $T$ w.r.t. $C$.

Let $Q'$ be formed by replacing each relation $R$ in $Q$ by a copy $R'$. Let $C'$ be formed by replacing $R$ by $R'$ in $C$.

Then determinacy of $Q$ over $T$ can be restated as:

$$C \land Q \land [\bigwedge_{R \in T} (\forall x_1 \ldots x_n \ R(x_1 \ldots x_n) \leftrightarrow R'(x_1 \ldots x_n)) \land C'] \models Q'$$
Determinacy can also be restated as:

\[(c \land q) \models [\bigwedge_{R \in T} (\forall x_1 ... x_n R(x_1 ... x_n) \leftrightarrow R'(x_1 ... x_n)) \land c'] \rightarrow q'\]

We can witness determinacy with a tableau showing (omitting NNF( )):

\[(c \land q) \vdash [\bigwedge_{R \in T} (\forall x_1 ... x_n R(x_1 ... x_n) \leftrightarrow R'(x_1 ... x_n)) \land c'] \rightarrow q'\]

Recall: tableau for entailment above is a tree with root

\[(c \land q) \land \neg ( [\bigwedge_{R \in T} (\forall x_1 ... x_n R(x_1 ... x_n) \leftrightarrow R'(x_1 ... x_n)) \land c'] \rightarrow q')\]

in which every leaf a clash.

We will see how to extract a reformulation from such a proof.
Interpolation

Given two formulas $\rho_1$ and $\rho_2$ (of first order logic) with different (but overlapping) vocabularies,

if $\rho_1 \vdash \rho_2$

a (Craig) interpolant for this entailment is a first-order formula $\psi$ such that

$\rho_1 \vdash \psi \vdash \rho_2$

and $\psi$ uses only the common symbols of $\rho_1$ and $\rho_2$.

The logician William Craig showed that interpolants always exist – proving the Craig Interpolation Theorem.
Interpolation \( \rightarrow \) Reformulations

Determinacy can be restated as:

\[(\mathcal{C} \land \mathcal{Q}) \models [\bigwedge_{R \in \mathcal{T}} (\forall x_1 \ldots x_n R(x_1 \ldots x_n) \leftrightarrow R'(x_1 \ldots x_n)) \land C'] \rightarrow \mathcal{Q}'\]

Common relations on the left and right are all in \( \mathcal{T} \).

Thus applying interpolation gets \( \mathcal{Q}_T \) mentioning only \( \mathcal{T} \) relations such that:

\[\mathcal{C} \land \mathcal{Q} \models \mathcal{Q}_T \models [\bigwedge_{R \in \mathcal{T}} (...) \land C'] \rightarrow \mathcal{Q}'\]

**Claim:** \( \mathcal{Q}_T \) is the reformulation we want.

That is: for any \( \mathcal{D} \) satisfying the constraints, \( \mathcal{D} \) satisfies \( \mathcal{Q} \) iff \( \mathcal{D} \) satisfies \( \mathcal{Q}_T \).
Interpolation → Reformulations

Thus applying interpolation gets $Q_T$ mentioning only $T$ relations such that:

$$c \land Q \vDash Q_T \vDash [\bigwedge_{R \in T} (...) \land c'] \rightarrow Q'$$

**Claim:** $Q_T$ is the reformulation we want. That is: for any $D$ satisfying the constraints, $D$ satisfies $Q$ iff $D$ satisfies $Q_T$.

Assume $Q$ is a sentence (no free vars) for simplicity.

**Proof:** $(\Rightarrow) c \land Q \vDash Q_T$ so if $D$ satisfies $c \land Q$, it satisfies $Q_T$.
Interpolation gets Reformulations

Applying interpolation gets $Q_T$ mentioning only $T$ relations such that:

$$C \land Q \models Q_T \models [\bigwedge_{R \in T} (...) \land C'] \rightarrow Q'$$

**Claim:** $Q_T$ is the reformulation we want. That is: for any $D$ satisfying the constraints, $D$ satisfies $Q$ iff $D$ satisfies $Q_T$.

**Proof:** ($\iff$) Suppose $D$ satisfies $C \land Q_T$. Extend $D$ to $D^*$ by defining each $R'$ in the full signature to be the same as $R$.

$$Q_T \models [\bigwedge_{R \in T} (\forall x_1 \ldots x_n \ R(x_1 \ldots x_n) \leftrightarrow R'(x_1 \ldots x_n)) \land C'] \rightarrow Q'$$

so $D^*$ satisfies $Q'$

Thus $D$ satisfies $Q$
Proving Craig Interpolation

Given two formulas $\rho_1$ and $\rho_2$ of first order logic with different (but overlapping) vocabularies, if $\rho_1 \vdash \rho_2$ then there is $\psi$ such that $\rho_1 \vdash \psi \vdash \rho_2$

and $\psi$ uses only the common symbols of $\rho_1$ and $\rho_2$ ($\psi$ is a Craig Interpolant for $\rho_1$ and $\rho_2$).

We have seen: how applying an interpolation algorithm can get us a reformulation.
Now give an idea of: how an interpolant is generated from a proof.
To see if $\rho_1 \vdash \rho_2$, develop a tableau for $\rho_1 \land \neg \rho_2$

Tableau for $\exists x \ A(x) \land \neg B(x) \land C(x) \vdash \neg \forall y \ [(\neg A(y) \land E(y)) \lor B(y)]$

$\exists x \ A(x) \land \neg B(x) \land C(x) \land \forall y \ [(\neg A(y) \land E(y)) \lor B(y)]$

$A(c) \land \neg B(c) \land C(x) \land \forall y \ [(\neg A(y) \land E(y)) \lor B(y)]$

$A(c) \land \neg B(c) \land C(c) \land [ (\neg A(c) \land E(c)) \lor B(c)]$

$A(c), \neg B(c), C(c), [ (\neg A(c) \land E(c)) \lor B(c)]$

$A(c), \neg B(c), C(c), \neg A(c) \land E(c)$

$A(c), C(c), \neg B(c), B(c)$

$A(c), \neg B(c), C(c), \neg A(c), E(c)$
Interpolation and Tableaux

\[ \exists x \ A(x) \land \neg B(x) \land C(x) \vdash \neg \forall y \ [ (\neg A(y) \land E(y)) \lor B(y) ] \]

\[ \exists x \ A(x) \land \neg B(x) \land C(x) \land \forall y \ [ (\neg A(y) \land E(y)) \lor B(y) ] \]

\[ A(c) \land \neg B(c) \land C(c) \land \forall y \ [ (\neg A(y) \land E(y)) \lor B(y) ] \]

\[ A(c) \land \neg B(c) \land C(c) \land [ (\neg A(c) \land E(c)) \lor B(c) ] \]

\[ A(c), \neg B(c), C(c), [ (\neg A(c) \land E(c)) \lor B(c) ] \]

A(c), \neg B(c), C(c), \neg A(c), E(c) \hspace{2cm} A(c), \neg B(c), C(c), B(c)

Can associate every formula with a “provenance”, telling whether it came from the left or right.
Interpolation and Tableaux

\[ \exists x \; A(x) \land \neg B(x) \land C(x) \vdash \neg \forall y \; [ (\neg A(y) \land E(y)) \lor B(y) ] \]

\[ \exists x \; A(x) \land \neg B(x) \land C(x) \land \forall y \; [ (\neg A(y) \land E(y)) \lor B(y) ] \]

\[ A(c) \land \neg B(c) \land C(c) \land \forall y \; [ (\neg A(y) \land E(y)) \lor B(y) ] \]

\[ A(c) \land \neg B(c) \land C(c) \land [ (\neg A(c) \land E(c)) \lor B(c) ] \]

\[ A(c), \neg B(c), C(c), \neg A(c), E(c) \]

\[ A(c), \neg B(c), C(c), B(c) \]

Bottom-up algorithm creates a formula that interpolates between the left and the right formulas in every node.
Interpolation and Tableaux

\[ \exists x \, A(x) \land \neg B(x) \]
\[ \exists x \, A(x) \land \neg B(x) \land C(x) \vdash \neg \forall y \left[ (\neg A(y) \land E(y)) \lor B(y) \right] \]

\[ \exists x \, A(x) \land \neg B(x) \land C(x) \land \forall y \left[ (\neg A(y) \land E(y)) \lor B(y) \right] \]

\[ A(c) \land \neg B(c) \land C(c) \land \forall y \left[ (\neg A(y) \land E(y)) \lor B(y) \right] \]

\[ \cdots \]

\[ A(c) \land \neg B(c) \land C(c) \land \left[ (\neg A(c) \land E(c)) \lor B(c) \right] \]

\[ \cdots \]

\[ A(c), \neg B(c), C(c), \left[ (\neg A(c) \land E(c)) \lor B(c) \right] \]

\[ A(c) \land \neg B(c), C(c), \neg A(c), E(c) \]

\[ A(c), \neg B(c), C(c), B(c) \]
Previous example was using “classical semantics”. To handle active-domain semantics, need some custom proof rules and interpolation rules for “relative quantifications”

\[
\exists x_1 \ldots x_n \ R(x_1 \ldots x_n) \land \phi
\]

\[
\forall x_1 \ldots x_n \ R(x_1 \ldots x_n) \rightarrow \phi
\]
Bottom line

To show that $Q$ can be reformulated in first-order logic over target schema $T$, we need to show that $Q$ is determined over $T$.

We can do this by finding a proof that:

$$(c \land Q) \vdash [\bigwedge_{R \in T} (\forall x_1 \ldots x_n R(x_1 \ldots x_n) \leftrightarrow R'(x_1 \ldots x_n)) \land C'] \rightarrow Q'$$

and then use a Craig Interpolation Procedure to generate the reformulation $Q_T$. 
Where are we?

- Covered the connection between interpolation results and first-order reformulation

- Methodology:
  - translate reformulation need into a semantic property;
  - translate the property to an entailment
  - find a proof of the entailment
  - apply an interpolation theorem to extract the reformulation

- Now extend this to conjunctive reformulation.
Conjunctive Reformulation

It is easy to see that there are constraints $c$, conjunctive queries $q$, and targets $t$ such that $q$ can be reformulated over $t$ with respect to $c$, but the reformulation is not a conjunctive query.
Monotonicity

**Q** is **monotonically determined** by relations **T** with respect to constraints **C** if:

For any instances **D**, **D'** satisfying **C** with every **R ∈ T**, 
**D(R) ⊆ D'(R) then Q(D) ⊆ Q(D')**

**Q** is **∃+ reformulatable** over **T** with respect to constraints **C** if: 
there is a positive existential FO **Q_T** such that for any instance **D** satisfying **C**, **Q(D) = Q'(D)**

**Observation:** Every query **∃+ reformulatable** w.r.t **T** and **C** is monotonically determined by **T** w.r.t. **C**
Monotonicity

Recall:
If $CQ \varphi$ is determined over $\mathcal{T}$ by $c$, then $\varphi$ is first-order reformulatable over $\mathcal{T}$ w.r.t. $c$.

This is a "database analog" of a theorem in logic, the Projective Beth Definability Theorem.
We now state the analogous result for $\exists^+$ reformulatability.

Theorem
If $CQ \varphi$ is monotonically determined by $\mathcal{T}$ w.r.t. $c$, then $\varphi$ is reformulatable as a positive existential (i.e union of conjunctive queries) formula over $\mathcal{T}$ w.r.t. $c$. 

Monotonicity and Preservation

If $CQ \varphi$ is monotonically determined by $T$ w.r.t $C$, then $\varphi$ is positively rewritable over $T$ w.r.t. $C$.

Special case: If $CQ \varphi$ is monotonically determined over the full signature w.r.t $C$, then $\varphi$ is equivalent to a positive existential formula w.r.t. $C$.

This is a variation of one of the Preservation Theorems in First-Order Logic.
Monotonicity as a proof goal

**Goal:** $\exists^+ \text{ reformulation of } Q \text{ over } T \text{ w.r.t } C$

**Necessary semantic condition:** Monotonically determined

For any databases $D, D'$ satisfying $C$ if for every $R \in T$, $D(R) \subseteq D'(R)$ then $Q(D) \subseteq Q(D')$

Find a proof that witnesses (omitting $\text{NNF}(..)$):

\[
c \land Q \land [\bigwedge_{R \in T} \forall x_1 \ldots x_n \ R(x_1 \ldots x_n) \rightarrow R'(x_1 \ldots x_n)] \land C \vdash Q'
\]

Rewriting

\[
c \land Q \land [\bigwedge_{R \in T} \forall x_1 \ldots x_n \ R(x_1 \ldots x_n) \rightarrow R'(x_1 \ldots x_n)] \vdash C' \rightarrow Q'
\]

Where $C'$ and $Q'$ use a copy of the signature

Apply interpolation algorithm to this proof and "remove primes".
Monotonicity as a proof goal

Find a proof that witnesses (omitting NNF(..) ) :

\[ C \land Q \land [\bigwedge_{R \in T} \forall x_1 \ldots x_n \ R(x_1 \ldots x_n) \rightarrow R'(x_1 \ldots x_n)] \land C' \vdash Q' \]

Rewriting

\[ C \land Q \land [\bigwedge_{R \in T} \forall x_1 \ldots x_n \ R(x_1 \ldots x_n) \rightarrow R'(x_1 \ldots x_n)] \vdash C' \rightarrow Q' \]

Where \( C' \) and \( Q' \) use a copy of the signature

Apply interpolation algorithm to this proof and remove primes. Argue as before that the resulting interpolant is a reformulation.

**Claim:** resulting interpolant will be positive existential.

**Reason:** Common relations only occur positively on the left. Therefore common relations can only occur positively in the interpolant.
Summary

We have reduced reformulation of query $Q$ over subvocabulary $T$ with respect to $C$ to getting a tableau proof of:

$$C \land C' \land (\text{additional axioms capturing, e.g. monotonicity}) \land Q \vdash Q'$$

If the schema constraints $C$ are TGDs, then we are seeing if $Q$ is contained in $Q'$ with respect to a set of TGDs $\text{Reform(C)}$.

Then we can apply interpolation to this tableau proof.

*Chase is a special case of the Tableau proof system.*

Thus we can:
- use the chase to decide $Q \subseteq_{\text{Reform(C)}} Q'$
- consider the resulting proof as a tableau proof and apply interpolation to it
- get a reformulation from the interpolant
How does this help?

Of course, it does not help for general TGDs:

**Theorem** The following problem is undecidable:
**Input:** a boolean conjunctive query $Q$, TGD constraints $C$
**Output:** yes if $Q$ has a first-order reformulation under the constraints

But for the specialized classes studied previously, it is useful.
Recall: Tame TGDs

One approach to get decidability of implication of $\varphi'$ by $\varphi$ w.r.t. $C$

Restrict TGDs in $C$ so that the chase terminates.

E.g. referential integrity constraints that are acyclic
Tame TGDs

One approach to get decidability of \( Q_1 \subseteq_c Q_2 \)

Restrict TGDs in \( c \) so that the chase terminates. E.g. weakly acyclic TGDs.

Given a set of TGDs, form the dependency graph
Nodes= positions within relation

Regular edge:
\[ R(x_1, \ldots) \rightarrow S(y_1, x_1, \ldots) \quad R[1] \rightarrow S[2] \]

Existential edge
\[ R(x_1 \ldots) \rightarrow \exists z S(y_1, z \ldots x_1 \ldots) \quad R[1] \rightarrow S[2] \]

**Weakly acyclic:** no cycle through an existential edge
Weakly Acyclic

Example:
∀ did dname mgrid Department (did, dname, mgrid) →
∃ N Employee (mgrid, N, did)
// Every Department Manager is an employee

∀ eid ename did Employee (eid, ename, did) →
∃ D M Department (did, D, M)
// Every Employee works in a department
Effectivity

Recall the methodology:
• use the chase to decide $Q \subseteq_{\text{Reform}(C)} Q'$
• consider the resulting proof as a tableau proof and apply interpolation to it
• get a reformulation from the interpolant

Still need to check: If constraints $C$ are in restricted classes, TGDs $\text{Reform}(C)$ is also in a restricted class where the chase behaves well.

If constraints $C$ are weakly acyclic TGDs, then proof goal is of form $Q \land \text{Reform}(C) \vdash Q'$ where $\text{Reform}(C)$ is again a weakly acyclic set of TGDs.

Terminating Chase $\rightarrow$ Effective procedure 😊
If $C$ is weakly acyclic, derived TGDs $\text{Reform}(C)$ will also be weakly acyclic.
Reformulation with Weakly Acyclic TGDs

Back to First Reformulation Example:

\[ \forall \text{did} \ \text{dname} \ \text{mgrid} \ \text{Department} (\text{did, dname, mgrid}) \rightarrow \exists \text{N Employee} (\text{mgrid, N, did}) \]

// Every Department Manager is an employee

\[ \forall \text{eid} \ \text{ename} \ \text{did} \ \text{Employee} (\text{eid, ename, did}) \rightarrow \exists \text{D M Department} (\text{did, D, M}) \]

// Every Employee works in a department

Suppose only Department is a table in the interface set \( T \)

**Goal:** a CQ reformulation of:

\[ Q = \{ \text{did} | \exists \text{eid ename Employee(eid, ename, did)} \} \] using Department
Reformulation Example

Suppose only Department is in the set of interface tables \( T \).

**Goal:** a CQ reformulation of: \( Q(z) = \{ z \mid \exists x y \text{Employee}(x, y, z) \} \) using Department

Apply the methodology: Look at constraints (1) + (2) + copies on new table names (1') + (2') + "Transfer Axioms".
\[
\forall \ x \ y \ z \ \text{Department}(x, y, z) \rightarrow \text{Department'}(x, y, z)
\]

Look for proof of goal:
\[
Q(z) \land \text{new constraints} \vdash \exists x y \ \text{Employee'}(x, y, z)
\]

Then apply interpolation to the proof.
Reformulation Example

∀i n m D(i, n, m) → ∃o E(m, o, i)  (1)
∀i n d E(i, n, d) → ∃o p D(d, o, p)  (2)

Suppose only D is a table in the interface set T.

We want a CQ reformulation of: Q={z | ∃x y E(x, y, z)}

Proof (using chase):

E(x, y, z)

D(z, o, p)  Chase step with (2)

D'(z, o, p)  ("transfer axiom")

E'(p, o', z)  Chase with (1')

Applying interpolation to this proof gets reformulation {z | ∃o p D(z, o, p)}

Chase terminates with success
Other Tame TGD classes

Methodology:
• use the chase to decide $\varphi \subseteq_{\text{Reform}(C)} \varphi'$
• consider the resulting proof as a tableau proof and apply interpolation to it
• get a reformulation from the interpolant

Still need to check: If constraints $c$ are in restricted classes, TGDs $\text{Reform}(C)$ is also in a restricted class where the chase behaves well.

If constraints $c$ come from conjunctive view definitions, and we are looking for a positive existential reformulation, then $\text{Reform}(C)$ has a terminating chase.
Terminating Chase $\rightarrow$ Effective procedure
View-based Conjunctive Reformulation

Consider case of constraints generated by conjunctive views

\[ V_1(x, y) = \exists u v w \ R(x, w) \land S(w, u) \land T(v, y) \]

\[ R(x, w) \land S(w, u) \land T(v, y) \rightarrow V_1(x, y) \]
(Integrity constraint: Definition-to-View-Relation)

\[ V_1(x, y) \rightarrow \exists u v w R(x, w) \land S(w, u) \land T(v, y) \]
(Integrity constraint: View-Relation-to-Definition)

\[ V_1(x, y) \rightarrow V'_1(x, y) \] (Monotonicity axiom)

\[ V'_1(x, y) \rightarrow \exists u v w R'(x, w) \land S'(w, u) \land T'(v, y) \]
\[ R'(x, w) \land S'(w, u) \land T'(v, y) \rightarrow V'_1(x, y) \]

Not weakly acyclic. But chase still terminates.
Bottom line on Conjunctive Reformulation with Views

Methodology:
• use the chase to decide $\mathcal{Q} \subseteq_{\text{Reform}(C)} \mathcal{Q}'$
• consider the resulting proof as a tableau proof and apply interpolation to it
• get a reformulation from the interpolant

Applied to positive existential reformulation of constraints coming from CQ views, gives new proof of:

**Theorem [Levy Mendelzon Sagiv Srivastava 95]** There is an effective procedure to decide whether a conjunctive query is rewritable using a conjunctive query over a set of views.
Recall: Guarded TGDs

TGD: constraint of form
\[ \forall x_1 \ldots x_j \]
\[ A_1(x_1 \ldots) \land \ldots \land A_m(x_j \ldots) \]
\[ \rightarrow \]
\[ \exists y_1 \ldots y_k \ B_1(x_1 \ldots x_j, y_1 \ldots y_k) \land \ldots \]

Guarded TGD: constraint of form
\[ \forall x_1 \ldots x_j \ A(x_1 \ldots x_j) \land (\text{other atoms using only } x_1 \ldots x_j) \]
\[ \rightarrow \]
\[ \exists y_1 \ldots y_k \ B_1(x_1 \ldots x_m, y_1 \ldots y_k) \land \ldots \]
Guarded TGDs

Methodology:
• use the chase to decide $\mathcal{Q} \subseteq_{\text{Reform}(C)} \mathcal{Q}'$
• consider the resulting proof as a tableau proof and apply interpolation to it
• get a reformulation from the interpolant

Need to check: If constraints $\mathcal{C}$ are in restricted classes, TGDs $\text{Reform}(\mathcal{C})$ is also in a restricted class where the chase behaves well.

If constraints $\mathcal{C}$ are Guarded TGDs, then $\text{Reform}(\mathcal{C})$ is again a set of Guarded TGDs.
Finite vs. Infinite

Interpolation-based Reformulation Methodology:
• use the chase to decide $\mathcal{Q} \subseteq_{\text{Reform}(\mathcal{C})} \mathcal{Q}'$
• consider the resulting proof as a tableau proof and apply interpolation to it
• get a reformulation from the interpolant

Limitation/Concern: methodology is only complete for finding reformulations that work over all databases.

Theorem If constraints $\mathcal{C}$ are Guarded TGDs, acyclic classes, conjunctive view constraints (for positive existential reformulation) then the interpolation-based method is also complete for finding reformulations over finite databases.

Follows from the fact that for each of these implications, they hold on all finite databases iff they hold on all databases.
Summary

- One can combine "tame chasing" with interpolation to get methods for reformulating queries over restricted interfaces.
- Connects proof methods for dependencies with techniques used for first-order theorem proving.
Constructive interpolation is due to

The first application of interpolation to "reformulation" (extracting explicit definitions
from proofs) is found in:
The connection between database query reformulation problems and interpolation is first noted in:
Luc Segoufin, Victor Vianu: Views and queries: determinacy and rewriting. PODS 2005: 49-60

and explored further in:

The connection between interpolation and preservation theorems in model theory, as well as an interpolation theorem for "restricted quantifier formulas" can be found in:

Constructive implications of Segoufin and Vianu’s results for databases were noted in: David Toman, Grant Weddell: Fundamentals of Physical Design and Query Compilation. Synthesis Lectures on Data Management, Morgan & Claypool Publishers 2011

Reformulation problems for databases and their connection with databases are also explored in:

References

The theoretical results as formulated here are due to:

Michael Benedikt, Balder ten Cate, Efthymia Tsamoura: *Generating low-cost plans from proofs*. PODS 2014: 200-211

There is a related demonstration system shown at:

Modern Database Dependency Theory

Lecture 5: Further applications -- Schema Mappings

Michael Benedikt & Balder ten Cate
Query Reformulation

- **Base schema**: $\text{Road}(x, y)$

- **View definitions**:
  - $P_2(x, y) = \exists u \; \text{Road}(x, u) \land \text{Road}(u, y)$
  - $P_3(x, y) = \exists uv \; \text{Road}(x, u) \land \text{Road}(u, v) \land \text{Road}(v, y)$
  - ...

- **Observation**: $P_4$ can be reformulated in terms of $P_2$.

- **Puzzle [Afrati’07]**: reformulate $P_5$ in terms of $P_3$ and $P_4$
Answer

\[ P_5(x,y) = \exists u \left( P_4(x,u) \land \forall v \left( P_3(v,u) \rightarrow P_4(v,y) \right) \right) \]
Yesterday’s Results

Q a conjunctive query, C a set of TGDs, and T a set of relation symbols.

**Thm 1.** The following are equivalent*:
1. Q is FO reformulatable over T w.r.t. C
2. Q is determined over T by C
   A reformulation can be computed from a proof of determinacy.

**Thm 2.** The following are equivalent*:
1. Q is positive-existential FO reformulatable over T w.r.t. C
2. Q is monotonically determined over T by C,
   A reformulation can be computed from a proof of monotonic determinacy.

**Puzzle (continued):** in the previous example, show that P₅ is not monotonically determined by P₃ and P₄.

* over all (finite and infinite) databases
Some important complexity classes

- **PTIME** (a.k.a. P)
  - Problems that can be solved by an algorithm that runs in **polynomial time** (e.g., linear, quadratic, cubic, etc.)

- **NP**
  - Problems that can be solved by an algorithm that uses a polynomial amount of memory

- **coNP**

- **PSPACE**
  - Problems that can be solved by an algorithm that runs in exponential time (e.g., $2^n$)

- **EXPTIME**
## Complexity of Query Evaluation

<table>
<thead>
<tr>
<th>Query language</th>
<th>Data complexity</th>
<th>Combined complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ and UCQ</td>
<td>PTIME</td>
<td>NP-complete</td>
</tr>
<tr>
<td>FO</td>
<td>PTIME</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>Datalog</td>
<td>PTIME</td>
<td>EXPTIME-complete</td>
</tr>
</tbody>
</table>

For every Datalog query \( q \) there is a polynomial time algorithm \( A_q \) that, given a database....

There is an algorithm \( A \) that, given a Datalog query and database, ...
Homomorphisms

• Let D and D’ be two database instances over the same schema. A **homomorphism from D to D’** is a function

  \[ h : \text{adom}(D) \rightarrow \text{adom}(D’) \]

  such that for every fact \( R(a_1...a_n) \in D \), the corresponding fact \( R(h(a_1),...,h(a_n)) \) belongs to D’.

**Notation:**

\( D \rightarrow D’ \) means that there is a homomorphism from D to D’.
Example
The Homomorphism Order

• Fix a schema $S$ and consider $(\text{FinStr}[S], \rightarrow)$

• What can we say about the binary relation $\rightarrow$?
  – Reflexive and transitive (pre-order)
  – Lattice (every two structures have a least upper bound and a greatest lower bound)
• Upset generated by a structure corresponds to
  – a **CQ**.

• Upset generated by a finite set of structures corresponds to
  – a **UCQ**.

• Downset generated by a structure corresponds to
  – a **Constraint Satisfaction Problem (CSP)**.
$T_k$ is the linear order with $k$ elements, $P_k$ is the path with $k+1$ elements.

**Gallai-Hasse-Roy-Vitaver Theorem (~1965)**

$(P_{k+1}, T_k)$ is a duality pair, i.e., for every $H$,

$H \rightarrow T_k$ if and only if $P_{k+1} \nrightarrow H$. 
Information integration

• The Information Integration Problem: combining data from different sources, structured under different schemas.

• Aspects of information integration:
  - Data exchange (transforming data structured under a source schema into data structured under a target schema)
  - Data integration (querying data from different sources using a single unified global schema)
Data exchange

Translating data structured under a source schema into data structured under a target schema.
Data integration

Querying heterogeneous data in different sources via a virtual global schema
Schema mappings

• **Schema mappings** are an essential piece in the formalization of data exchange and data integration.

• A schema mapping is a high-level declarative specification of the relationships between two schemas, in the form of a set of constraints $\Sigma$. 

```plaintext
Source S

\Sigma

Target T
```
ST-TGDs (a.k.a. GLAV constraints)

A source-to-target tuple generating dependency (ST-TGD) is a TGD
\[ \forall x \ (\varphi(x) \rightarrow \exists y \ \psi(x, y)) \]
where
- \( \varphi(x) \) is a conjunction of atoms over the source schema;
- \( \psi(x, y) \) is a conjunction of atoms over the target schema.

Example:
- \( \text{Student}(s) \land \text{Enrolls}(s,c) \rightarrow \exists t,g \ (\text{Teaches}(t,c) \land \text{Grade}(s,c,g)) \)
GAV and LAV constraints

• ST-TGDs are also known as GLAV constraints.

• Special case: GAV constraints
  \( \forall x (\varphi(x) \rightarrow R(x_{i_1}, \ldots, x_{i_n})) \), where R is a target relation.

• Examples:
  - Copy (Nicknaming): \( \forall x_1, \ldots, x_n (P(x_1, \ldots, x_n) \rightarrow R(x_1, \ldots, x_n)) \)
  - Projection: \( \forall x, y, z (P(x, y, z) \rightarrow R(x, y)) \)
  - Join: \( \forall x, y, z (E(x, z) \land F(z, y) \rightarrow R(x, z, y)) \)

• Another special case: LAV constraints
  \( \forall x (R(x) \rightarrow \exists y \psi(x, y)) \), where R is a source relation.

• GAV and LAV constraints are the most widely supported schema mapping constraints.
**Definition**: Schema mapping $M = (S, T, \Sigma)$. If $I$ is a source database, then a **solution** for $I$ is a target database $J$ such that $(I, J)$ satisfy $\Sigma$.

The **data exchange problem** associated with $M$: Given a source database $I$, construct a database $J$ for $I$ (provided a solution exists).

For a given source instance, there are, in general, multiple target databases satisfying the specifications of the schema mapping.

- When more than one solution exist, which solutions are “better” than others?
- How do we compute a “good” solution?
Universal Solutions

**Definition:** $M = (S, T, \Sigma)$ schema mapping, I source database. A target database $J$ is a universal solution for $I$ w.r.t. $M$ if

- $J$ is a solution for $I$ w.r.t. $M$.
- If $J'$ is a solution for $I$ w.r.t. $M$, then there is a homomorphism $h: J \rightarrow J'$ that is constant on $\text{adom}(I)$.

**Note:** Intuitively, a universal solution for $I$ is a most general (= least specific) solution for $I$. 

Universal Solutions in Data Exchange

\[ \Sigma \]

Schema \( S \)  
\[ \text{Universal Solution} \]

I  
\[ J \]

\( h_1 \)  
\[ J_1 \]

\( h_2 \)  
\[ J_2 \]

\( h_3 \)  
\[ J_3 \]

Solutions
Universal Solutions and Examples

- Consider the schema mapping $M = (\{E\}, \{F\}, \Sigma)$, where
  \[ \Sigma = \{ E(x, y) \rightarrow \exists z \ (F(x, z) \land F(z, y)) \} \]

- Source database $I = \{ E(1, 2) \}

- Solutions for $I$:
  - $J_1 = \{ F(1, 2), F(2, 2) \}$
    - $J_1$ is a solution but not universal
  - $J_2 = \{ F(1, X), F(X, 2) \}$
    - $J_2$ is a universal solution
  - $J_3 = \{ F(1, X), F(X, 2), F(1, Y), F(Y, 2) \}$
    - $J_3$ is a universal solution
  - $J_4 = \{ F(1, X), F(X, 2), F(3, 3) \}$
    - $J_4$ is a solution but not universal
Universal Solutions and Schema Mappings

**Note:** A key property of GLAV schema mappings is the **existence of universal solutions.**

**Theorem (FKMP 2003)** \( M = (S, T, \Sigma) \) a GLAV schema mapping.

- Every source database \( I \) has a universal solution \( J \) w.r.t. \( M \),
- The chase can be used to construct, given a source instance \( I \), a canonical universal solution \( \text{chase}_M(I) \).

Universal solutions have become the preferred semantics in data exchange (the preferred solutions to materialize).

**Note:** The chase method applies (and terminates) even if \( \Sigma \) includes a weakly acyclic set of target tgds and egds.
Schema Mapping Design

How do we **obtain a schema mapping** in the first place?

- Traditionally: graphical tools (user draws correspondences between attributes of the relations in the two schemas).

- Data examples are a helpful resource.
  - Deriving schema mappings from data examples
  - Using data examples to illustrate candidate schema mappings
Data Examples

![Diagram showing source and target databases]

\[ M = (S, T, \Sigma) \] a GLAV schema mapping

- **Data Example**: A pair \((I, J)\) where \(I\) is a source database and \(J\) is a target database.

- **Positive** Data Example for \(M\):
  - A data example \((I, J)\) that satisfies \(\Sigma\), i.e., \((I, J) \models \Sigma\)
  - In other words, \(J\) is a solution for \(I\) w.r.t. \(M\).
Data Examples

Example: $M = (\{E\}, \{F\}, \Sigma)$, where $\Sigma = \{ E(x, y) \rightarrow \exists z (F(x, z) \land F(z, y)) \}$

Positive Data Examples $(I, J)$ (J a solution for I w.r.t. M)

- $I = \{ E(1, 2) \}$ $J = \{ F(1, 3), F(3, 2) \}$
- $I = \{ E(1, 2) \}$ $J = \{ F(1, X), F(X, 2) \}$
- $I = \{ E(1, 2) \}$ $J = \{ F(1, 3), F(3, 2), F(3, 4) \}$
- $I = \{ E(1, 2), E(3, 4) \}$ $J = \{ F(1, 3), F(3, 2), F(3, Y), F(Y, 4) \}$

- Negative Data Examples $(I, J)$ (J not a solution for I w.r.t. M)
  - $I = \{ E(1, 2) \}$ $J = \{ F(1, 3) \}$
  - $I = \{ E(1, 2) \}$ $J = \{ F(1, 3), F(4, 2) \}$
Schema Mappings and Data Examples

- \( M = (S, T, \Sigma) \) GLAV schema mapping
- \( \text{Sem}(M) = \{ (I,J): (I,J) \text{ is a positive data example for } M \} \)

**Fact:** \( \text{Sem}(M) \) is an infinite set

**Reason:**
If \((I,J)\) is a positive data example for \( M \) and if \( J \subseteq J' \), then \((I,J')\) is a positive data example for \( M \).

**Question:**
Can \( M \) be “characterized” using finitely many data examples?
Types of Data Examples

\[ M = (S, T, \Sigma) \] a GLAV schema mapping

• **Positive Data Example:** a data example \((I,J)\) such that \((I,J)\) satisfies \(\Sigma\), i.e., \(J\) is a solution for \(I\) w.r.t. \(M\).

• **Negative Data Example:** a data example \((I,J)\) such that \((I,J)\) does not satisfy \(\Sigma\), i.e., \(J\) is not a solution for \(I\) w.r.t. \(M\).

• **Universal Data Example:**
A data example \((I,J)\) such that \(J\) is a **universal** solution for \(I\) w.r.t. \(M\).
Characterizing Schema Mappings

- \( M = (S, T, \Sigma) \) GLAV schema mapping
- \( \text{Sem}(M) = \{ (I,J): (I,J) \text{ is a positive data example for } M \} \)

**Question:**
Can \( M \) be “characterized” using finitely many data examples?

Formally:
- Is there a finite set \( D \) of data examples (labeled as positive/negative/universal) such that \( M \) is the only (up to logical equivalence) schema mapping that fits the data examples?
Warm-up: The Copy Schema Mapping

Let M be the binary copy schema mapping specified by the constraint
\[ \forall x, y \ (E(x, y) \rightarrow F(x, y)) \).

**Question:** Which is the “most representative” data example for M, hence a good candidate for “characterizing” it?

**Intuitive Answer:** \((I_1, J_1)\) with \(I_1 = \{ E(a, b) \} \), \(J_1 = \{ F(a, b) \} \)

**Facts:** It will turn out that:
- \((I_1, J_1)\) “characterizes” M among all LAV schema mappings.
- \((I_1, J_1)\) does not “characterize” M among all GLAV schema mappings; in fact, not even among all GAV schema mappings.

**Reason:** \((I_1, J_1)\) is also a universal example for the GAV schema mapping specified by
\[ \forall x, y, u, v \ (E(x, y) \land E(u, v) \rightarrow F(x, v)) \).

Proposition: \( M = (S, T, \Sigma) \) a GLAV schema mapping, \( C \) a class of schema mappings. If \( M \) is uniquely characterizable within \( C \) by a finite set of positive and negative examples, then \( M \) is also uniquely characterizable within \( C \) by a finite set of universal examples.

Proof Idea: Uniquely characterizing

positive examples: \((I^+_1, J^+_1), (I^+_2, J^+_2), \ldots\) and

negative examples: \((I^-_1, J^-_1), (I^-_2, J^-_2), \ldots\)

give rise to uniquely characterizing

universal examples: \((I^+_1, \text{chase}_M(I^+_1)), (I^+_2, \text{chase}_M(I^+_2)), \ldots\)
\((I^-_1, \text{chase}_M(I^-_1)), (I^+_2, \text{chase}_M(I^+_2)), \ldots\)
• So, unique characterizability via positive and negative examples implies unique characterizability via universal examples.

• The converse is not always true.

• For this reason, we will focus on unique characterizability via universal examples.
Unique Characterizations Warm-Up

**Theorem:** Let $M$ be the binary copy schema mapping specified by the constraint $\forall x, y \ (E(x, y) \rightarrow F(x, y))$.

- The set $U = \{(I_1, J_1)\}$ with $I_1 = \{E(a, b)\}$, $J_1 = \{F(a, b)\}$ uniquely characterizes $M$ within the class of all LAV schema mappings.

- There is a finite set $U'$ consisting of three universal examples that uniquely characterizes $M$ within the class of all GAV schema mappings.

- There is no finite set of universal examples that uniquely characterizes $M$ within the class of all GLAV schema mappings.
The following universal examples uniquely characterizes the binary copy schema mapping within the class of all GAV schema mappings.
Unique Characterizations of LAV Mappings

**Theorem:** If $M = (S, T, \Sigma)$ is a LAV schema mapping, then there is a finite set $U$ of universal examples that uniquely characterizes $M$ within the class of all LAV schema mappings.

**Hint of Proof:**
- $U$ is the set of all universal examples $(I, J)$ where $I$ consists of a single fact and $J = \text{chase}_M(I)$. 
Illustration of Unique Characterizability

Let $M$ be the binary projection schema mapping specified by

$$\forall x, y \ (P(x, y) \rightarrow Q(x))$$

- The following set $U$ of universal examples uniquely characterizes $M$ within the class of all LAV schema mappings:

  $$U = \{ (I_1, J_1), (I_2, J_2) \}, \text{ where}$$

  $$I_1 = \{ P(a_1, a_2) \}, \quad J_1 = \{ Q(a_1) \}$$

  $$I_2 = \{ P(a_1, a_1) \}, \quad J_2 = \{ Q(a_1) \}.$$
Illustration of Unique Characterizability

Let $M$ be the schema mapping specified by

$$\forall x, y \ (P(x, y) \rightarrow Q(x)) \quad \text{and} \quad \forall x \ (P(x, x) \rightarrow \exists y \ R(x, y))$$

- The following set $U$ of universal examples uniquely characterizes $M$ within the class of all LAV schema mappings:

$$U = \{ (I_1, J_1), (I_2, J_2) \}, \quad \text{where}$$

$$I_1 = \{ P(a_1, a_2) \}, \quad J_1 = \{ Q(a_1) \}$$

$$I_2 = \{ P(a_1, a_1) \}, \quad J_2 = \{ Q(a_1), R(a_1, Y) \}.$$
Unique Characterizations of GAV Mappings

**Note:** Recall that for the schema mapping specified by the binary copy constraint \( \forall x, y (E(x, y) \rightarrow F(x, y)) \), there is a finite set of universal examples that uniquely characterizes it within the class of all GAV schema mappings.

In contrast,

**Theorem:** Let \( M \) be the GAV schema mapping specified by
\[ \forall x, y, u, v, w \ (E(x, y) \land E(u, v) \land E(v, w) \land E(w, u) \rightarrow F(x, y)). \]
There is no finite set of universal examples that uniquely characterizes \( M \) within the class of all GAV schema mappings.

**Question:** How can this difference be explained?
Characterizing GAV Schema Mappings

• **Question:**
  – What is the reason that some GAV schema mappings are uniquely characterizable within the class of all GAV schema mappings while some others are not?
  – Is there an algorithm for deciding whether or not a given GAV schema mapping is uniquely characterizable?

• **Answer:**
  – The answers to these questions are closely connected to database constraints and homomorphism dualities.
Homomorphisms

Notation: A, B finite structures

- \[ \rightarrow A = \{ B : B \rightarrow A \} \]
  - Example (for graphs): \[ \rightarrow K_2 \] = Class of 2-colorable graphs

- \[ A \rightarrow = \{ B : A \rightarrow B \} \]
  - Example (for graphs): \[ K_2 \rightarrow \] = Class of graphs with at least one edge.
Homomorphism Dualities

• **Definition:** Let $D$ and $F$ be two finite structures
  
  – $(F,D)$ is a **duality pair** if for every finite structure $A$ we have
    
    $$A \rightarrow D \text{ if and only if } (F \leftrightarrow A).$$

  – In symbols, $\Rightarrow_{D} = F \leftrightarrow$

  – $F$ is called an **obstruction (or, frustration)** for $D$. 
Homomorphism Dualities

• **Theorem** (*König 1936*): A graph is 2-colorable if and only if it contains no cycle of odd length.
  In symbols, \( \rightarrow K_2 = \bigcap_{i \geq 0} (C_{2i+1} \not\leftrightarrow) \).

• **Definition**: Let \( F \) and \( D \) be two sets of structures. We say that \((F, D)\) is a **duality pair** if for every structure \( A \), TFAE
  – There is a structure \( D \) in \( D \) such that \( A \rightarrow D \).
  – For every structure \( F \) in \( F \), we have \( F \not\leftrightarrow A \).

  In symbols, \( \bigcup_{D \in D} (\rightarrow D) = \bigcap_{F \in F} (\not\leftrightarrow F) \).

  In this case, we say that \( F \) is an **obstruction set** for \( D \).
Homomorphism Dualities

**Duality Pair** \((F, D)\), where

\[ F = \{F_1, F_2, \ldots\} \]

\[ D = \{D_1, D_2, \ldots\} \]
Unique Characterizations and Homomorphism Dualities

**Theorem:** Let $M = (S, T, \Sigma)$ be a GAV mapping. Then the following statements are equivalent:

- $M$ is uniquely characterizable via universal examples within the class of all GAV schema mappings.

- For every target relation symbol $R$, the set $F(M,R)$ of the **canonical structures** of the GAV constraints in $\Sigma$ with $R$ as their head is the obstruction set of some finite set $D$ of structures.
Canonical Structures of GAV Constraints

Definition:
• The **canonical structure** of a GAV constraint
  \[ \forall x \ (\varphi_1(x) \land \ldots \land \varphi_k(x) \rightarrow R(x_{i_1}, \ldots, x_{i_m})) \]
  is the structure consisting of the atomic facts \( \varphi_1(x), \ldots, \varphi_k(x) \) and having constant symbols \( c_1, \ldots, c_m \) interpreted by the variables \( x_{i_1}, \ldots, x_{i_m} \) in the atom \( R(x_{i_1}, \ldots, x_{i_m}) \).

Let \( M = (S, T, \Sigma) \) be a GAV schema mapping. For every relation symbol \( R \) in \( T \), let \( F(M, R) \) be the set of all canonical structures of GAV constraints in \( \Sigma \) with the target relation symbol \( R \) in their head.
Canonical Structures

Examples:

- GAV constraint $\sigma = \forall x, y, z \ (E(x, y) \land E(y, z) \rightarrow F(x, z))$
  - Canonical structure: $A_\sigma = \{x, y, z\}, \{(E(x, y), E(y, z))\}, x, z$
  - Constants $c_1$ and $c_2$ interpreted by the distinguished elements $x$ and $z$.

- GAV constraint $\theta = \forall x, y, z \ (E(x, y) \land E(y, z) \rightarrow F(x, x))$
  - Canonical structure: $A_\tau = \{x, y, z\}, \{E(x, y), E(y, z)\}, x, x$
  - Constants $c_1$ and $c_2$ both interpreted by the distinguished element $x$. 

Unique Characterizations and Homomorphism Dualities

**Theorem:** Let $M = (S, T, \Sigma)$ be a GAV mapping. Then the following statements are equivalent:

- $M$ is uniquely characterizable via universal examples w.r.t. the class of all GAV constraints.

- For every target relation symbol $R$, the set $F(M, R)$ of the canonical structures of the GAV constraints in $\Sigma$ with $R$ as their head is the obstruction set of some finite set $D$ of structures.
Let $M$ be the GAV schema mapping specified by
\[ \forall x \ (R(x, x) \rightarrow P(x)) . \]

- Canonical structure $F = (\{x\}, \{R(x, x)\}, x)$
- Consider $D = (\{a, b\}, \{R(a, b), R(b, a), R(b, b)\}, a)$

**Fact:** $(F, D)$ is a duality pair, because it is easy to see that for every structure $G=(V, R, d)$, we have that
\[ G \rightarrow D \text{ if and only if } F \leftrightarrow G. \]

Consequently, $M$ is uniquely characterizable via universal examples within the class of all GAV schema mappings.
Unique Characterizations and Homomorphism Dualities

**Theorem:** Let $M = (S, T, \Sigma)$ be a GAV schema mapping in suitable normal form. Then the following statements are equivalent:

- $M$ is *uniquely characterizable via universal examples* w.r.t. the class of all GAV constraints.

- For every target relation symbol $R$, the set $F(M, R)$ is the *obstruction set* of some finite set of structures.

- For every target relation symbol $R$, the set $F(M, R)$ consists entirely of *c-acyclic* structures.
c-Acyclicity

Definition: Let $A = (A, R_1, \ldots, R_m, c_1, \ldots, c_k)$ be a relational structure with constants $c_1, \ldots, c_k$.

- The **incidence graph** $\text{inc}(A)$ of $A$ is the bipartite graph with
  - nodes the elements of $A$ and the facts of $A$
  - edges between elements and facts in which they occur
- The structure $A$ is **c-acyclic** if
  - Every cycle of $\text{Inc}(A)$ contains at least one constant $c_i$, and
  - Only constants may occur more than once in the same fact.

Example:

- $A = \{1,2,3\}, \{R((1,2,3), Q(1,2)), 1\}$ is c-acyclic
  - the cycle $1, R(1,2,3), 2, Q(1,2), 1$ contains the constant 1, and it is the only cycle of $\text{inc}(A)$.
- $A = \{1,2,3\}, \{R((1,2,3), Q(1,2)), 3\}$ is not c-acyclic
  - the cycle $1, R(1,2,3), 2, Q(1,2), 1$ contains no constant.
Applications

• The GAV schema mapping \( M \) specified by
  \[
  \forall x, y, u, v, w \ (E(x, y) \land E(u, v) \land E(v, w) \land E(w, u) \rightarrow F(x, y))
  \]
is not uniquely characterizable: the canonical structure contains a cycle with no constant on it
Applications

- The GAV schema mapping specified by the constraint
  \[ \forall x, y, z \ (E(x, y) \land E(y, z) \rightarrow F(x, z)) \]
is uniquely characterizable via universal examples.

- Let \( M \) be the GAV schema mappings specified by the constraints
  \begin{align*}
  \sigma & : \forall x, y, z \ (E(x, y) \land E(y, z) \land E(z, x) \rightarrow F(x, z)) \\
  \tau & : \forall x, y \ (E(x, y) \land E(y, x) \rightarrow F(x, x))
  \end{align*}
The canonical structures of these constraints are
  \begin{align*}
  A_\sigma & = ([x, y, x] \{E(x, y), E(y, z), E(z, x)\}, x, z) \\
  A_\tau & = ([x, y] \{E(x, y), E(y, x)\}, x, x)
  \end{align*}
  - Both are c-acyclic; hence \( \{A_\sigma, A_\tau\} \) is an obstruction set of a finite set of structures.
  - Therefore, \( M \) is uniquely characterizable via universal examples.
Synopsis

- Introduced and studied the notion of unique characterization of a schema mapping by a finite set of universal examples.

- Every LAV schema mapping is uniquely characterizable via universal examples within the class of all LAV schema mappings.

- Necessary and sufficient condition, and an algorithmic criterion for a GAV schema mapping to be uniquely characterizable via universal examples within the class of all GAV schema mappings.
  - Tight connection with homomorphism dualities.
Open Problems

• When is a LAV schema mapping uniquely characterizable by a “small” number of universal examples within the class of all LAV schema mappings?
  – Same question for GAV schema mappings.

• When is a GLAV schema mapping uniquely characterizable by finitely many universal examples within to the class of all GLAV schema mappings?
  – We do not even know whether this problem is decidable.
References


• The part of the lecture on data examples is based mainly on the paper “Characterizing Schema Mappings via Data Examples” by B. Alexe, B. ten Cate, Ph. Kolaitis, W.-C. Tan in ACM TODS 2011.
Other Research on Schema Mappings and Data Exchange

- Obtaining schema mappings:
  - Deriving schema mappings from data examples
- Managing schema mappings
  - Operations on schema mappings (e.g., composition and inversion)
  - Schema mapping optimization (e.g., rewriting schema mappings to simpler ones)
- Using schema mappings in data exchange
  - E.g., obtaining minimal (“core”) universal solutions
  - Dealing with inconsistencies during data exchange
- Query answering for richer schema mapping languages and query languages.
  - E.g., aggregate queries, schema mappings with arithmetic operations
Ontology-Based Data Access

- **Ontology** - a formal representation of knowledge as concepts within a domain, and the relationships between instances of those concepts
Ontology-Mediated Queries

- **Data D**: relational database instances (finite set of ground facts) over some schema $S$; e.g.,
  
  \[
  \text{diagnose}(\text{sue, fibrillation}), \nonumber \\
  \text{heartdisease}(\text{fibrillation})
  \]

- **Ontology O**: a set of constraints **over some schema** $S'$ **extending** $S$; e.g.,
  
  \[
  \forall xy. (\text{diagnose}(x, y) \land \text{heartdisease}(y) \rightarrow \text{heartpatient}(x))
  \]

- **Query $q(x)$**: a conjunctive query **over** $S'$; e.g.,
  
  \[
  q(x) = \text{heartpatient}(x)
  \]
Description logic $\mathcal{ALC}$

Ontologies consist of finite sets of implications $C \lor D$, where $C$, $D$ are concepts:

$$C, D := A \mid \top \mid \bot \mid C \cup D \mid C \cap D \mid \neg C \mid \exists R.C \mid \forall R.C$$

The ontology

- $\forall x (Employee(x) \rightarrow \exists y (WorksIn(x, y) \land Dept(y)))$
- $\forall x (Employee(x) \rightarrow \neg Dept(x))$
- $\forall x (Manager(x) \land Manages(x, y) \rightarrow (Employee(y) \lor Manager(y)))$

is written in $\mathcal{ALC}$ as

- $Employee \sqsubseteq \exists WorksIn.Dept$
- $Employee \sqsubseteq \neg Dept$
- $Manager \sqsubseteq \forall Manages.(Employee \cup Manager)$
• **Theorem**: For some $\mathcal{ALC}$ ontology $O$ and boolean CQ $q$, certain($q,O$) is coNP-complete.

• **Proof**: we can express 3-colorability:
  - **Ontology**:
    - $\exists x \ (R \land \neg G \land \neg B) \lor (R \land G \land \neg B) \lor (R \land \neg G \land B)$
    - $R \land \exists \text{Edge}.R \subseteq \bot$
    - $G \land \exists \text{Edge}.G \subseteq \bot$
    - $B \land \exists \text{Edge}.B \subseteq \bot$
  - **Query**
    - $\exists x \ Ax$ (where $A$ is a fresh predicate)

Certain($q,G,O$)=true for a graph $G$ iff $G$ is not 3-colorable.
Guarded TGDs are only one class for which one can determine when to cut off the chase. This is an extremely active area of research with close ties to finite model theory and automata theory.
Theorem. For every $(O,q)$ in $(\mathcal{ALC},\text{UCQ})$, the certain answers can be evaluated in coNP (data complexity).

Proof idea: use tableaux (disjunctive chase). Just as in the case of guarded TGDs, we can cut off the search after some bounded number of steps.

We want to know:

- For which pairs $(O,q)$ are the certain answers PTIME computable?
- For which pairs $(O,q)$ are the certain answers FO-rewritable?
- For which pairs $(O,q)$ are the certain answers Datalog-rewritable?

Focus on the $(\mathcal{ALC}, \text{boolean atomic query})$ case.
Surprising connection with CSP

**Theorem**: for every \((O,q)\) in \((\mathcal{ALC}, \text{boolean atomic query})\), \(\text{certain}(q,O)\) is the complement of some CSP. Conversely, every CSP is the complement of \(\text{certain}(q,O)\) for some \((O,q)\) in \((\mathcal{ALC}, \text{boolean atomic query})\).

**Proof idea**:
- From CSP to \((O,q)\): straightforward generalization of the 3-colorability example.
- From \((O,q)\) to CSP: define a structure whose elements are all the “consistent 1-types”, i.e., maximal consistent subsets of subformulas(O) consistent with \(\neg q\).

**Consequence of the Theorem**: we can apply known results from CSP literature.
1. There is an algorithm for deciding FO-rewritability of \(\text{certain}(O,q)\)
2. There is an algorithm for deciding Datalog-rewritability of \(\text{certain}(O,q)\)
3. (Feder-Vardi conjecture) there is an algorithm for deciding if \(\text{certain}(O,q)\) is in PTIME.
Summary

• Ontology-mediated queries \((O,q)\) in \((\mathcal{ALC},\text{boolean atomic query})\) have the same expressive power as (the complement of) CSPs.

• This connection allows us to tap into a deep source of results and techniques.

• Results generalize to richer ontology-mediated query language (with more work).
References


• The part of the lecture on data examples is based mainly on the paper “Characterizing Schema Mappings via Data Examples” by B. Alexe, B. ten Cate, Ph. Kolaitis, W.-C. Tan in ACM TODS 2011.

• The last part, on ontology-based data access, was based on “Ontology-based data access: a study through disjunctive datalog, CSP, and MMSNP” by M. Bienvenu, B. ten Cate, C. Lutz, F. Wolter in PODS, 2013.