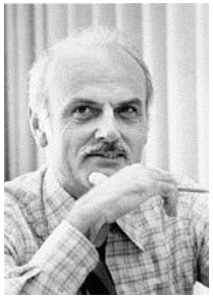
Foundations of Relational Query Languages

Advanced Topics in Foundations of Databases, University of Edinburgh, 2016/17

Relational Model

- Many ad hoc models before 1970
  - Hard to work with
  - Hard to reason about



- **1970: Relational Model by Edgar Frank Codd**
  - Data are stored in relations (or tables)
  - Queried using a declarative language
  - DBMS converts declarative queries into procedural queries that are optimized and executed
- Key Advantages
  - Simple and clean mathematical model (based on logic)
  - Separation of declarative and procedural

Relational Databases

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Constants

VIE, LHR, …

BA, U2, …

Vienna, London, …

Relational Databases

Relations

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Constants

VIE, LHR, …

BA, U2, …

Vienna, London, …

Relational Databases

Relations

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Tuples

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Constants

VIE, LHR, …

BA, U2, …

Vienna, London, …

Relational atoms

Flight(LHR,EDI,BA)

Airport(LGW,London)

Querying: Relational Algebra

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{BA, U2, OS}

π_{airline} Flight

Querying: Relational Algebra

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{LHR, LGW}

$$\pi_{code}\ (\sigma_{city=\text{'London'}}\ \text{Airport})$$

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$$\pi_{\text{airline}} \left((\text{Flight} \bowtie_{\text{origin=code}} (\sigma_{\text{city='London'}} \text{Airport})) \bowtie_{\text{destination=code}} (\sigma_{\text{city='Glasgow'}} \text{Airport}) \right)$$

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Aux	origin	destination	airline	code	city	code	city
	LGW	GLA	U2	LGW	London	GLA	Glasgow

⇩

{U2}

$$\pi_{airline}\ ((Flight\ \bowtie_{origin=code}\ (\sigma_{city=\text{'London'}}\ Airport))\ \bowtie_{destination=code}\ (\sigma_{city=\text{'Glasgow'}}\ Airport))$$

defines the auxiliary relation Aux

Relational Algebra

- **Selection: σ**
- **Projection: π**
- **Cross product:** $\times$
- Natural join: $\bowtie$
- **Rename: ρ**
- **Difference:** $-$
- **Union:** $\cup$
- Intersection: $\cap$

in bold are the primitive operators

Formal definitions can be found in any database textbook

Querying: Domain Relational Calculus

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{BA, U2, OS}

$\{z \mid \exists x \exists y \; Flight(x,y,z)\}$

Querying: Domain Relational Calculus

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{LHR, LGW}

$\{x \mid \exists y\ Airport(x,y) \wedge y = London\}$

Querying: Domain Relational Calculus

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{U2}

$$\{z \mid \exists x \exists y \exists u \exists v \; \text{Airport}(x,u) \land u = \text{London} \land \text{Airport}(y,v) \land v = \text{Glasgow} \land \text{Flight}(x,y,z)\}$$

Domain Relational Calculus

$$\{x_1,\ldots,x_k \mid \varphi\}$$

first-order formula with free variables $\{x_1,\ldots,x_k\}$

But, we can express "problematic" queries, i.e., depend on the domain

$\{x \mid \forall y\, R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

…thus, we adopt the active domain semantics – quantified variables range over the active domain, i.e., the constants occurring in the input database

Algebra = Calculus

A fundamental theorem (assuming the active domain semantics):

Theorem: The following query langauges are equally expressive

- Relational Algebra (**RA**)
- Domain Relational Calculus (**DRC**)
- Tuple Relational Calculus (**TRC**)

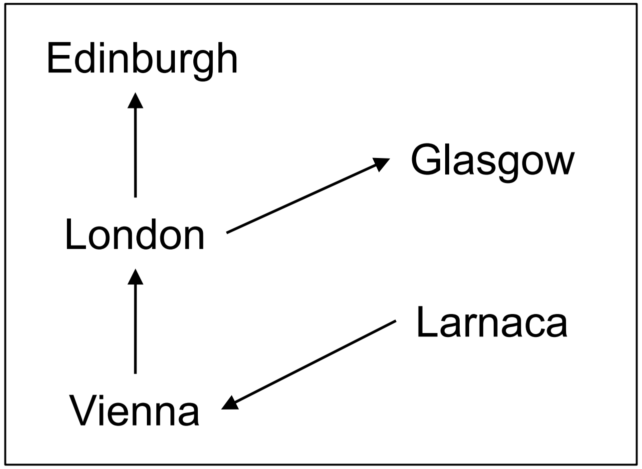
Note: Tuple relational calculus is the declarative language introduce by Codd. Domain relational calculus has been introduced later as a formalism closer to first-order logic

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Recursive query – not expressible in **RA/DRC/TRC**

(unless we bound the number of intermediate stops)

Complexity of Query Languages

- The goal is to understand the complexity of evaluating a query over a database
- Our main technical tool is complexity theory
- What to measure? Queries may have a large output, and it would be unfair to count the output as “complexity”
- We therefore consider the following decision problems:
  - Query Output Tuple (QOT)
  - Boolean Query Evaluation (BQE)

A Crash Course on Complexity Theory

we are going to recall some fundamental notions from complexity theory that will be heavily used in the context of this course – details can be found in the standard textbooks

Deterministic Turing Machine (DTM)

$$M = (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$$

- S is the set of states
- Λ is the input alphabet, not containing the blank symbol $\sqcup$
- Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Lambda \subseteq \Gamma$
- $\delta : S \times \Gamma \rightarrow S \times \Gamma \times \{L,R\}$
- s_0 is the initial state
- s_{accept} is the accept state
- s_{reject} is the reject state, where $s_{accept} \neq s_{reject}$

Deterministic Turing Machine (DTM)

$$M \;=\; (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$$

$\delta(s_1, \alpha) = (s_2, \beta, R)$

IF at some time instant τ the machine is in sate s_1, the cursor
points to cell κ, and this cell contains α

THEN at instant $\tau+1$ the machine is in state s_2, cell κ contains β,
and the cursor points to cell $\kappa+1$

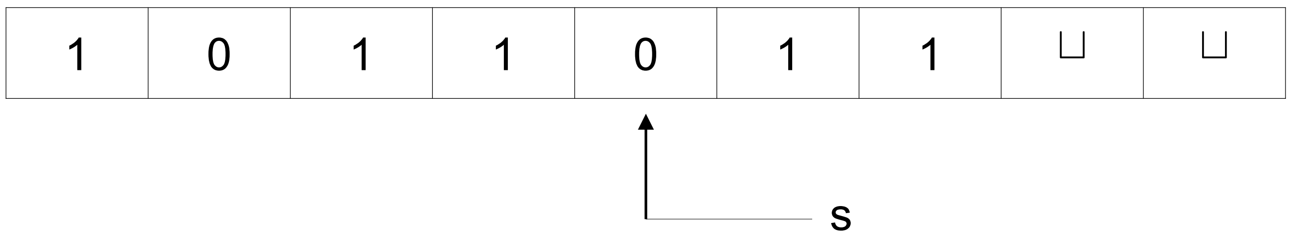
Nondeterministic Turing Machine (NTM)

$$M = (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$$

- S is the set of states
- Λ is the input alphabet, not containing the blank symbol $\sqcup$
- Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Lambda \subseteq \Gamma$
- **$\delta : S \times \Gamma \rightarrow 2^{S \times \Gamma \times \{L,R\}}$**
- s_0 is the initial state
- s_{accept} is the accept state
- s_{reject} is the reject state, where $s_{accept} \neq s_{reject}$

Turing Machine Configuration

A perfect description of the machine at a certain point in the computation

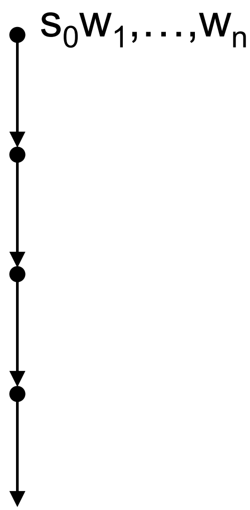


is represented as a string: **1011s011**

- Initial configuration on input $w_1,\ldots,w_n$ – $s_0 w_1,\ldots,w_n$
- Accepting configuration – $u_1,\ldots,u_k s_{accept} u_{k+1},\ldots,u_{k+m}$
- Rejecting configuration – $u_1,\ldots,u_k s_{reject} u_{k+1},\ldots,u_{k+m}$

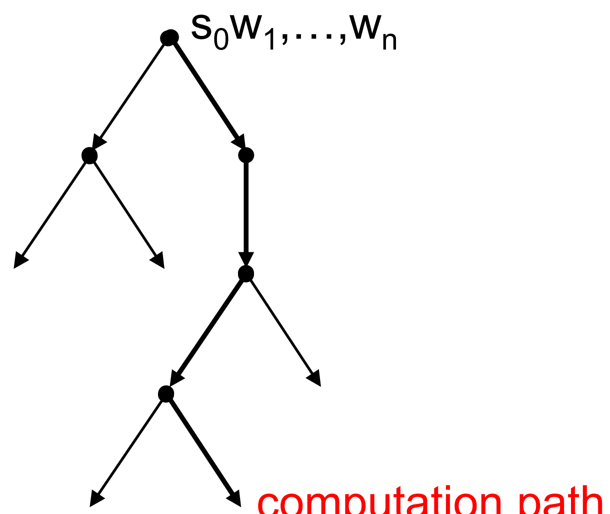
Turing Machine Computation

Deterministic



the next configuration is unique

Nondeterministic



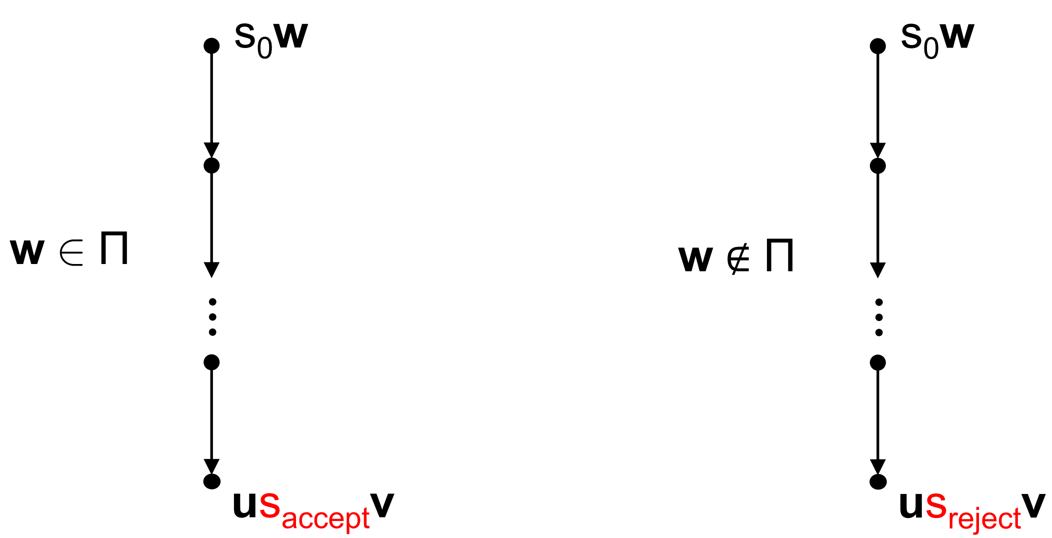
computation tree

Deciding a Problem

(recall that an instance of a decision problem Π is encoded as a word over a certain alphabet Λ – thus, Π is a set of words over Λ, i.e., $\Pi \subseteq \Lambda^*$)

A DTM $M = (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$ decides a problem Π if, for every $\mathbf{w} \in \Lambda^*$:

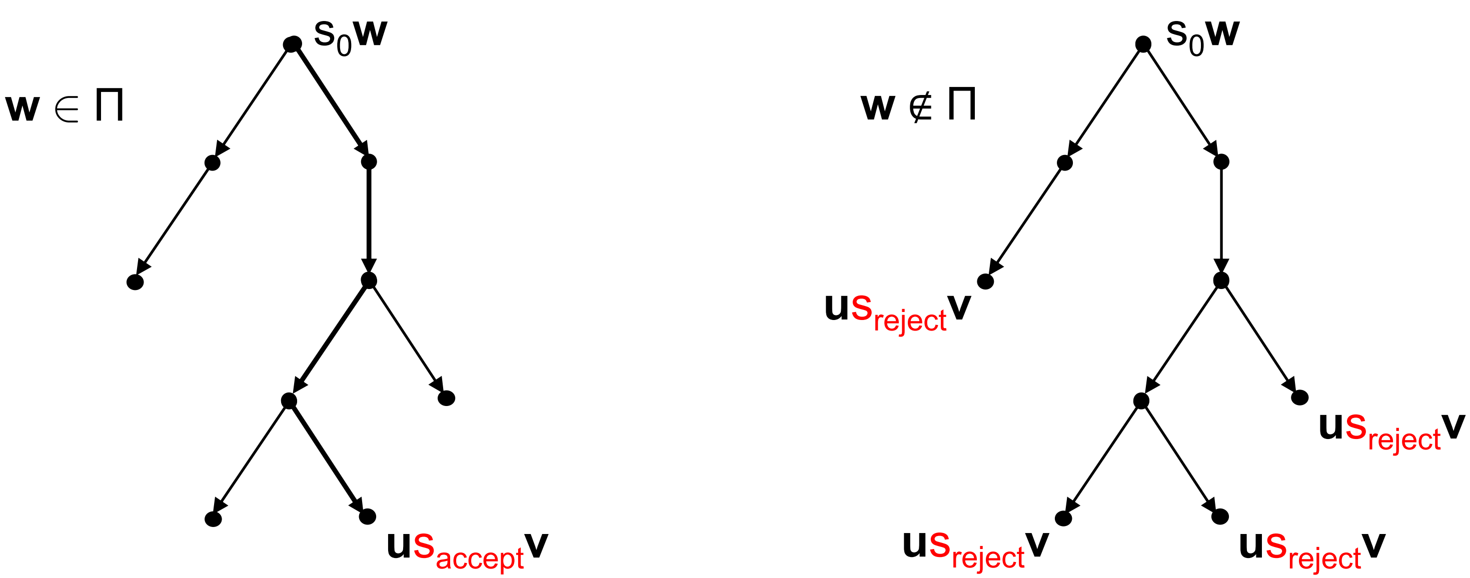
- M on input **w** halts in s_{accept} if $\mathbf{w} \in \Pi$
- M on input **w** halts in s_{reject} if $\mathbf{w} \notin \Pi$



Deciding a Problem

A NTM $M = (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$ decides a problem Π if, for every $\mathbf{w} \in \Lambda^*$:

- The computation tree of M on input $\mathbf{w}$ is finite
- There exists at least one accepting computation path if $\mathbf{w} \in \Pi$
- There is no accepting computation path if $\mathbf{w} \notin \Pi$



Complexity Classes

Consider a function $f : N \to N$

$$\mathrm{TIME}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some DTM in time } O(f(n))\}$$

$$\mathrm{NTIME}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some NTM in time } O(f(n))\}$$

$$\mathrm{SPACE}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some DTM using space } O(f(n))\}$$

$$\mathrm{NSPACE}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some NTM using space } O(f(n))\}$$

Complexity Classes

- We can now recall the standard time and space complexity classes:

PTIME	=	$\cup_{k>0}$ TIME(n^k)
NP	=	$\cup_{k>0}$ NTIME(n^k)
EXPTIME	=	$\cup_{k>0}$ TIME(2^{n^k})
NEXPTIME	=	$\cup_{k>0}$ NTIME(2^{n^k})
LOGSPACE	=	SPACE(log n)
NLOGSPACE	=	NSPACE(log n)
PSPACE	=	$\cup_{k>0}$ SPACE(n^k)
EXPSPACE	=	$\cup_{k>0}$ SPACE(2^{n^k})

(LOGSPACE and NLOGSPACE:) these definitions are relying on two-tape Turing machines with a read-only and a read/write tape

- For every complexity class C we can define its complementary class

$$\text{coC} = \{\Lambda^* \setminus \Pi \mid \Pi \in C\}$$

An Alternative Definition for NP

Theorem: Consider a problem $\Pi \subseteq \Lambda^*$. The following are equivalent:

- $\Pi \in$ NP
- There is a relation $R \subseteq \Lambda^* \times \Lambda^*$ that is polynomially decidable such that

  $\Pi = \{\mathbf{u} \mid \text{there exists } \mathbf{w} \text{ such that } |\mathbf{w}| \leq |\mathbf{u}|^k \text{ and } (\mathbf{u},\mathbf{w}) \in R\}$

witness or certificate

$\{\mathbf{xy} \in \Lambda^* \mid (\mathbf{x},\mathbf{y}) \in R\} \in$ PTIME

Example:

3SAT = $\{\varphi \mid \varphi$ is a 3CNF formula that is satisfiable$\}$

= $\{\varphi \mid \varphi$ is a 3CNF for which $\exists$ assignment α such that $|\alpha| \leq |\varphi|$ and $(\varphi,\alpha) \in R\}$

where $R = \{(\varphi,\alpha) \mid \alpha$ is a satisfying assignment for $\varphi\} \in$ PTIME

Relationship Among Complexity Classes

LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ PTIME $\subseteq$ NP, coNP $\subseteq$

PSPACE $\subseteq$ EXPTIME $\subseteq$ NEXPTIME, coNEXPTIME $\subseteq$ …

Some useful notes:

- For a deterministic complexity class C, coC = C
- coNLOGSPACE = NLOGSPACE
- It is generally believed that PTIME ≠ NP, but we don't know
- PTIME $\subset$ EXPTIME $\Rightarrow$ at least one containment between them is strict
- PSPACE = NPSPACE, EXPSPACE = NEXPSPACE, etc.
- But, we don't know whether LOGSPACE = NLOGSPACE

Complete Problems

- These are the hardest problems in a complexity class
- A problem that is complete for a class C, it is unlikely to belong in a lower class

- A problem Π is complete for a complexity class C, or simply C-complete, if:
  1. $\Pi \in C$
  2. Π is C-hard, i.e., every problem $\Pi' \in C$ can be efficiently reduced to Π

> there exists a polynomial time algorithm (resp., logspace algorithm) that computes a function f such that $\mathbf{w} \in \Pi' \Leftrightarrow f(\mathbf{w}) \in \Pi$ – in this case we write $\Pi' \leq_P \Pi$ (resp., $\Pi' \leq_L \Pi$)

- To show that Π is C-hard it suffices to reduce some C-hard problem Π' to it

Some Complete Problems

- NP-complete
  - SAT (satisfiability of propositional formulas)
  - Many graph-theoretic problems (e.g., 3-colorability)
  - Traveling salesman
  - etc.
- PSPACE-complete
  - Quantified SAT (or simply QSAT)
  - Equivalence of two regular expressions
  - Many games (e.g., Geography)
  - etc.

Back to Query Languages

Complexity of Query Languages

- The goal is to understand the complexity of evaluating a query over a database
- Our main technical tool is complexity theory
- What to measure? Queries may have a large output, and it would be unfair to count the output as "complexity"
- We therefore consider the following decision problems:
  - Query Output Tuple (QOT)
  - Boolean Query Evaluation (BQE)

Complexity of Query Languages

Some useful notation:

- Given a database *D*, and a query *Q*, *Q*(*D*) is the answer to *Q* over *D*
- **adom**(*D*) is the active domain of *D*, i.e., the constants occurring in *D*
- We write *Q*/k for the fact that the arity of *Q* is $k \geq 0$

L is some query language; for example, **RA**, **DRC**, etc. – we will see several query languages in the context of this course

QOT(**L**)

Input: a database *D*, a query $Q/k \in$ **L**, a tuple of constants $\mathbf{t} \in \mathbf{adom}(D)^k$

Question: $\mathbf{t} \in Q(D)$?

Complexity of Query Languages

Some useful notation:

- Given a database D, and a query Q, $Q(D)$ is the answer to Q over D
- **adom**(D) is the active domain of D, i.e., the constants occurring in D
- We write Q/k for the fact that the arity of Q is $k \geq 0$

L is some query language; for example, **RA**, **DRC**, etc. – we will see several query languages in the context of this course

> BQE(**L**)
>
> **Input:** a database D, a Boolean query $Q/0 \in$ **L**
>
> **Question:** $Q(D) \neq \emptyset$? (i.e., does D satisfies Q?)

Complexity of Query Languages

QOT(**L**)

Input: a database *D*, a query *Q*/k $\in$ **L**, a tuple of constants **t** $\in$ **adom**(*D*)^k

Question: **t** $\in$ *Q*(*D*)?

BQE(**L**)

Input: a database *D*, a Boolean query *Q*/0 $\in$ **L**

Question: *Q*(*D*) $\neq \emptyset$? (i.e., does *D* satisfies *Q*?)

Theorem: QOT(**L**) $\equiv_L$ BQE(**L**), where **L** $\in$ {**RA**, **DRC**, **TRC**}

($\equiv_L$ means logspace-equivalent)

Complexity of Query Languages

(let us show this for domain relational calculus)

Theorem: QOT(**DRC**) $\equiv_L$ BQE(**DRC**)

Proof: ($\leq_L$) Consider a database *D*, a k-ary query $Q = \{x_1,\ldots,x_k \mid \varphi\}$, and a tuple $(t_1,\ldots,t_k)$

Let $Q_{bool} = \{ \mid \varphi \wedge x_1 = t_1 \wedge x_2 = t_2 \wedge \ldots \wedge x_k = t_k\}$

Clearly, $(t_1,\ldots,t_k) \in Q(D)$ iff $Q_{bool}(D) \neq \emptyset$

($\geq_L$) Trivial – a Boolean domain RC query is a domain RC query

…henceforth, we focus on the Boolean Query Evaluation problem

Complexity Measures

- Combined complexity – both D and Q are part of the input

- Query complexity – fixed D, input Q

BQE[D]($\mathbf{L}$)

Input: a Boolean query $Q \in \mathbf{L}$

Question: $Q(D) \neq \emptyset$?

- Data complexity – input D, fixed Q

BQE[Q]($\mathbf{L}$)

Input: a database D

Question: $Q(D) \neq \emptyset$?

Complexity of **RA**, **DRC**, **TRC**

Theorem: For **L** $\in$ {**RA**, **DRC**, **TRC**} the following hold:

- BQE(**L**) is PSPACE-complete (combined complexity)
- BQE[D](**L**) is PSPACE-complete, for a fixed database D (query complexity)
- BQE[Q](**L**) is in LOGSPACE, for a fixed query $Q \in$ **L** (data complexity)

Proof hints:

- Recursive algorithm that uses polynomial space in Q and logarithmic space in D
- Reduction from QSAT (a standard PSPACE-hard problem)

Evaluating (Boolean) DRC Queries

Eval(D,φ) – for brevity we write φ instead of $\{ \mid \varphi\}$

- **If** $\varphi = R(t_1,\ldots,t_k)$, **then** YES iff $R(t_1,\ldots,t_k) \in D$
- **If** $\varphi = \psi_1 \wedge \psi_2$, **then** YES iff Eval($D$,$\psi_1$) = YES and Eval($D$,$\psi_2$) = YES
- **If** $\varphi = \neg\psi$, **then** NO iff Eval(D,ψ) = YES
- **If** $\varphi = \exists x\ \psi(x)$, **then** YES iff for some $t \in$ **adom**(D), Eval(D,$\psi(t)$) = YES

Lemma: It holds that

- Eval(D,φ) always terminates – in fact, this is trivial
- Eval(D,φ) = YES iff $Q(D) \neq \emptyset$, where $Q = \{ \mid \varphi\}$
- Eval(D,φ) uses $O(|\varphi| \cdot \log |\varphi| + |\varphi|^2 \cdot \log |D|)$ space

Complexity of **RA**, **DRC**, **TRC**

Theorem: For each $\mathbf{L} \in \{\mathbf{RA}, \mathbf{DRC}, \mathbf{TRC}\}$ the following holds:

- BQE(**L**) is PSPACE-complete (combined complexity)
- BQE[D](**L**) is PSPACE-complete, for a fixed database D (query complexity)
- BQE[Q](**L**) is in LOGSPACE, for a fixed query $Q \in \mathbf{L}$ (data complexity)

Proof hints:

- Recursive algorithm that uses polynomial space in Q and logarithmic space in D
- Reduction from QSAT (a standard PSPACE-hard problem)
- Actually, BQE[Q](**L**) is in $AC_0 \subset$ LOGSPACE (a highly parallelizable complexity class defined using Boolean circuits)

Other Important Algorithmic Problems

SAT(**L**)

Input: a query $Q \in \mathbf{L}$

Question: is there a (finite) database D such that $Q(D) \neq \emptyset$?

EQUIV(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database D?)

CONT(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?)

Other Important Algorithmic Problems

SAT(**L**)

Input: a query $Q \in \mathbf{L}$

Question: is there a (finite) database D such that $Q(D) \neq \emptyset$?

EQUIV(**L**)

Input: two (

Question: (

tabase D?)

these problems are important

for optimization purposes

CONT(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?)

Other Important Algorithmic Problems

SAT(**L**)

Input: a query $Q \in \mathbf{L}$

Question: is there a (finite) database D such that $Q(D) \neq \emptyset$?

- If the answer is no, then the input query Q makes no sense
- Query evaluation becomes trivial – the answer is always NO!

Other Important Algorithmic Problems

EQUIV(**L**)

Input: two queries $Q_1 \in$ **L** and $Q_2 \in$ **L**

Question: $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database D?)

- Replace a query Q_1 with a query Q_2 that is easier to evaluate
- But, we have to be sure that $Q_1(D) = Q_2(D)$ for every database D

Other Important Algorithmic Problems

CONT(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?)

- Approximate a query Q_1 with a query Q_2 that is easier to evaluate
- But, we have to be sure that $Q_2(D) \subseteq Q_1(D)$ for every database D

SAT is Undecidable

Theorem: For **L** $\in$ {**RA**, **DRC**, **TRC**}, SAT(**L**) is undecidable

Proof hint: By reduction from the halting problem.

Given a Turing machine *M*, we can construct a query $Q_M \in$ **L** such that:

M halts on the empty string $\Leftrightarrow$ there exists a database *D* such that $Q(D) \neq \emptyset$

Note: Actually, this result goes back to the 1950 when Boris A. Trakhtenbrot proved that the problem of deciding whether a first-order sentence has a finite model is undecidable

EQUIV and CONT are Undecidable

An easy consequence of the fact that SAT is undecidable is that:

Theorem: For **L** $\in$ {**RA**, **DRC**, **TRC**}, EQUIV(**L**) and CONT(**L**) are undecidable

Proof: By reduction from the complement of SAT(**L**)

- Consider a query $Q \in$ **L** – i.e., an instance of SAT(**L**)
- Let $Q_\perp$ be a query that is trivially unsatisfiable, i.e., $Q_\perp(D) = \emptyset$ for every D
- For example, when **L** = **DRC**, $Q_\perp$ can be the query $\{ \mid \exists x\ R(x) \wedge \neg R(x)\}$
- Clearly, Q is unsatisfiable $\Leftrightarrow$ $Q \equiv Q_\perp$ (or even $Q \subseteq Q_\perp$)

Recap

- The main languages for querying relational databases are:
  - Relational Algebra (**RA**)
  - Domain Relational Calcuclus (**DRC**)
  - Tuple Relational Calculus (**TRC**)

RA = DRC = TRC

(under the active domain semantics)

- Evaluation is decidable, and highly tractable in data complexity
  - **Foundations of the database industry**
  - The core of SQL is equally expressive to **RA**/**DRC**/**TRC**

- Satisfiability, equivalence and containment are undecidable
  - **Perfect query optimization is impossible**

A Crucial Question

Are there interesting sublanguages of **RA/DRC/TRC** for which

satisfiability, equivalence and containment are decidable?

Conjunctive Queries

= $\{\sigma, \pi, \bowtie\}$-fragment of relational algebra

= relational calculus without $\neg$, $\forall$, $\vee$

= simple SELECT-FROM-WHERE SQL queries
(only AND and equality in the WHERE clause)

Syntax of Conjunctive Queries (**CQ**)

$$Q(\mathbf{x}) \ := \ \exists \mathbf{y}\ (R_1(\mathbf{v_1}) \wedge \ldots \wedge R_m(\mathbf{v_m}))$$

- R_i ($1 \leq i \leq m$) are relations
- $\mathbf{x}$, $\mathbf{y}$, $\mathbf{v_1}$, …, $\mathbf{v_m}$ are tuples of variables
- each variable mentioned in $\mathbf{v_i}$ ($1 \leq i \leq m$) appears either in $\mathbf{x}$ or $\mathbf{y}$
- the variables in $\mathbf{x}$ are free called distinguished variables

It is very convenient to see conjunctive queries as rule-based queries of the form

$$Q(\mathbf{x}) \ :- \ \underbrace{R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})}$$

this is called the body of Q that can be seen as a set of atoms

Conjunctive Queries: Example 1

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{BA, U2, OS}

$\pi_{airline}$ Flight

Q(z) :- Flight(x,y,z)

{z | ∃x∃y Flight(x,y,z)}

Conjunctive Queries: Example 2

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{LHR, LGW}

$\pi_{code}\ (\sigma_{city='London'}\ Airport)$

$\{x \mid \exists y\ Airport(x,London) \wedge y = London\}$

Q(x) :- Airport(x,y), y = London

Conjunctive Queries: Example 2

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{LHR, LGW}

π_{code} $(\sigma_{city=\text{'London'}}$ Airport)

$\{x \mid \exists y \text{ Airport}(x,\text{London}) \wedge y = \text{London}\}$

Q(x) :- Airport(x,London)

Conjunctive Queries: Example 3

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

⇓

{U2}

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

π_{airline} ((Flight $\bowtie_{\text{origin=code}}$ ($\sigma_{\text{city='London'}}$ Airport)) $\bowtie_{\text{destination=code}}$ ($\sigma_{\text{city='Glasgow'}}$ Airport))

$\{z \mid \exists x \exists y \exists u \exists v\ \text{Airport}(x,u) \wedge u = \text{London} \wedge \text{Airport}(y,v) \wedge v = \text{Glasgow} \wedge \text{Flight}(x,y,z)\}$

Conjunctive Queries: Example 3

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{U2}

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

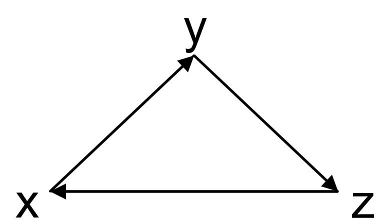
Homomorphism

- Semantics of conjunctive queries via the key notion of homomorphism
- A substitution from a set of symbols **S** to a set of symbols **T** is a function h : **S** $\rightarrow$ **T**
  i.e., h is a set of mappings of the form s $\rightarrow$ t, where s $\in$ **S** and t $\in$ **T**
- A homomorphism from a set of atoms **A** to a set of atoms **B** is a substitution
  h : terms(**A**) $\rightarrow$ terms(**B**) such that:
  1. t is a constant $\Rightarrow$ h(t) = t
  2. $R(t_1,\ldots,t_k) \in \mathbf{A} \Rightarrow h(R(t_1,\ldots,t_k)) = R(h(t_1),\ldots,h(t_k)) \in \mathbf{B}$

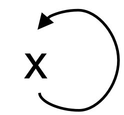
(terms(**A**) = {t | t is a variable or constant that occurs in **A**})

Exercise: Find the Homomorphisms

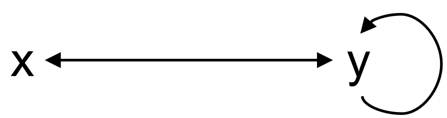
$\mathbf{S_1}$ = {P(x,y), P(y,z), P(z,x)}



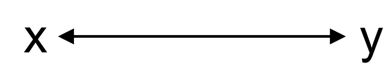
$\mathbf{S_2}$ = {P(x,x)}



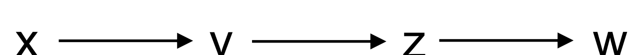
$\mathbf{S_3}$ = {P(x,y), P(y,x), P(y,y)}



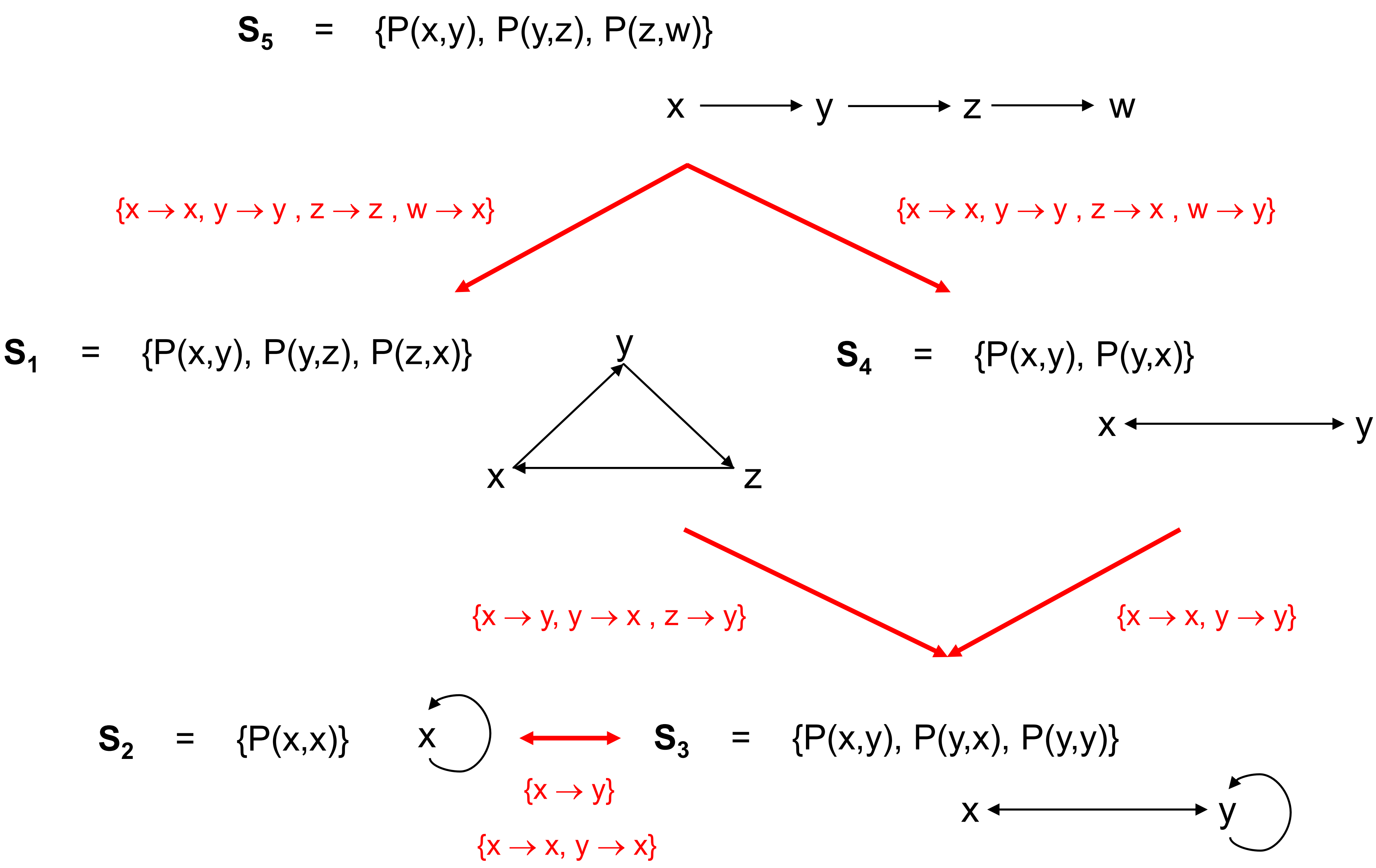
$\mathbf{S_4}$ = {P(x,y), P(y,x)}



$\mathbf{S_5}$ = {P(x,y), P(y,z), P(z,w)}



Exercise: Find the Homomorphisms



Semantics of Conjunctive Queries

- A match of a conjunctive query $Q(x_1,\ldots,x_k)$:- body in a database D is a homomorphism h such that h(body) $\subseteq D$

- The answer to $Q(x_1,\ldots,x_k)$:- body over D is the set of k-tuples

$$Q(D) := \{(h(x_1),\ldots, h(x_k)) \mid h \text{ is a match of } Q \text{ in } D\}$$

- The answer consists of the witnesses for the distinguished variables of Q

Conjunctive Queries: Example

List the airlines that fly directly from London to Glasgow

Flight	**origin**	**destination**	**airline**
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	**code**	**city**
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$\{x \rightarrow \text{LGW}, y \rightarrow \text{GLA}, z \rightarrow \text{U2}\}$

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[D](**CQ**) is NP-complete, for a fixed database D (query complexity)
- BQE[Q](**CQ**) is in LOGSPACE, for a fixed query $Q \in$ **CQ** (data complexity)

Proof:

(NP-membership) Consider a database D, and a Boolean CQ Q :- body

Guess a substitution h : terms(body) $\rightarrow$ terms(D)

Verify that h is a match of Q in D, i.e., h(body) $\subseteq D$

(NP-hardness) Reduction from 3-colorability

(LOGSPACE-membership) Inherited from BQE[Q](**DRC**) – in fact, in AC_0

NP-hardness

(NP-hardness) Reduction from 3-colorability

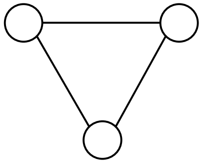
3COL

Input: an undirected graph **G** = (V,E)

Question: is there a function c : {Red,Green,Blue} $\rightarrow$ V such that

$(v,u) \in E \Rightarrow c(v) \neq c(u)$?

Lemma: G is 3-colorable $\Leftrightarrow$ **G** can be mapped to $\mathbf{K_3}$, i.e., **G** $\xrightarrow{\text{hom}}$



therefore, **G** is 3-colorable $\Leftrightarrow$ there is a match of Q_G in D = {E(x,y),E(y,z),E(z,x)}

$\Leftrightarrow$ $Q_G(D) \neq \emptyset$

the Boolean CQ that represents **G**

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[*D*](**CQ**) is NP-complete, for a fixed database *D* (query complexity)
- BQE[*Q*](**CQ**) is in LOGSPACE, for a fixed query $Q \in$ **CQ** (data complexity)

Proof:

(NP-membership) Consider a database *D*, and a Boolean CQ *Q* :- body

Guess a substitution h : terms(body) $\rightarrow$ terms(*D*)

Verify that h is a match of *Q* in *D*, i.e., h(body) $\subseteq$ *D*

(NP-hardness) Reduction from 3-colorability

(LOGSPACE-membership) Inherited from BQE[*Q*](**DRC**) – in fact, in AC_0

What About Optimization of CQs?

SAT(**CQ**)

Input: a query $Q \in$ **CQ**

Question: is there a (finite) database D such that $Q(D) \neq \emptyset$?

EQUIV(**CQ**)

Input: two queries $Q_1 \in$ **CQ** and $Q_2 \in$ **CQ**

Question: $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database D?)

CONT(**CQ**)

Input: two queries $Q_1 \in$ **CQ** and $Q_2 \in$ **CQ**

Question: $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?)

Canonical Database

- Convert a conjunctive query *Q* into a database *D*[*Q*] – the canonical database of *Q*

- Given a conjunctive query of the form *Q*(**x**) :- body, *D*[*Q*] is obtained from body by replacing each variable x with a new constant c(x) = $\underline{x}$

- E.g., given *Q*(x,y) :- R(x,y), P(y,z,w), R(z,x), then *D*[*Q*] = {R($\underline{x}$,$\underline{y}$), P($\underline{y}$,$\underline{z}$,$\underline{w}$), R($\underline{z}$,$\underline{x}$)}

- **Note:** The mapping c : {variables in body} $\rightarrow$ {new constants} is a bijection, where c(body) = *D*[*Q*] and c^{-1}(*D*[*Q*]) = body

Satisfiability of CQs

SAT(**CQ**)

Input: a query $Q \in$ **CQ**

Question: is there a (finite) database D such that $Q(D) \neq \emptyset$?

Theorem: A query $Q \in$ **CQ** is always satisfiable; thus, SAT(**CQ**) $\in O(1)$-time

Proof: Due to its canonical database – $Q(D[Q]) \neq \emptyset$

Equivalence and Containment of CQs

EQUIV(**CQ**)

Input: two queries $Q_1 \in$ **CQ** and $Q_2 \in$ **CQ**

Question: $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database D?)

CONT(**CQ**)

Input: two queries $Q_1 \in$ **CQ** and $Q_2 \in$ **CQ**

Question: $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?)

$$Q_1 \equiv Q_2 \iff Q_1 \subseteq Q_2 \text{ and } Q_2 \subseteq Q_1$$

$$Q_1 \subseteq Q_2 \iff Q_1 \equiv (Q_1 \wedge Q_2)$$

…thus, we can safely focus on CONT(**CQ**)

Homomorphism Theorem

A query homomorphism from $Q_1(x_1,\ldots,x_k)$:- $body_1$ to $Q_2(y_1,\ldots,y_k)$:- $body_2$ is a substitution $h : terms(body_1) \rightarrow terms(body_2)$ such that:

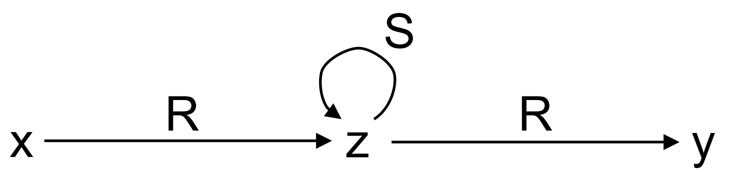
1. h is a homomorphism from $body_1$ to $body_2$
2. $(h(x_1),\ldots,h(x_k)) = (y_1,\ldots,y_k)$

Homomorphism Theorem: Let Q_1 and Q_2 be conjunctive queries. It holds that:

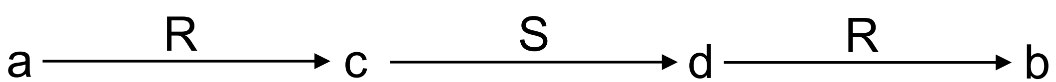
$Q_1 \subseteq Q_2 \Leftrightarrow$ there exists a query homomorphism from Q_2 to Q_1

Homomorphism Theorem: Example

$Q_1(x,y)$:- $R(x,z), S(z,z), R(z,y)$



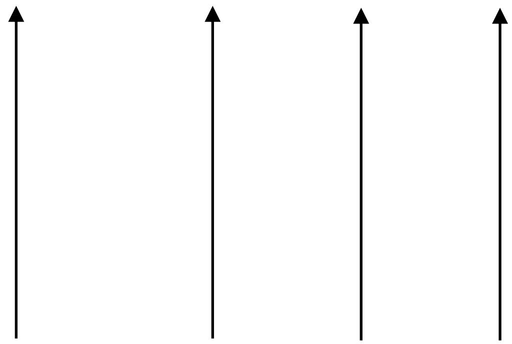
$Q_2(a,b)$:- $R(a,c), S(c,d), R(d,b)$



We expect that $Q_1 \subseteq Q_2$. Why?

Homomorphism Theorem: Example

$Q_1(x,y)$:- $R(x,z), S(z,z), R(z,y)$



$Q_2(a,b)$:- $R(a,c), S(c,d), R(d,b)$

$h = \{a \rightarrow x,\ b \rightarrow y,\ c \rightarrow z,\ d \rightarrow z\}$

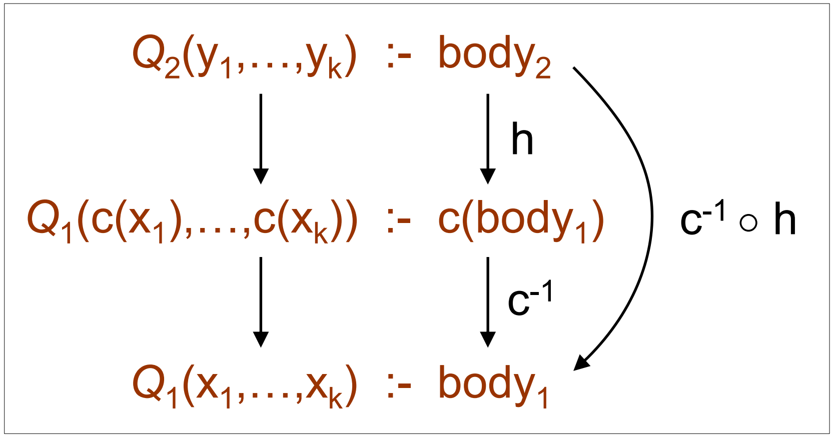
- h is a query homomorphism from Q_2 to Q_1 $\Rightarrow$ $Q_1 \subseteq Q_2$
- But, there is no homomorphism from Q_1 to Q_2 $\Rightarrow$ $Q_1 \subset Q_2$

Homomorphism Theorem: Proof

Assume that $Q_1(x_1,\ldots,x_k)$:- $body_1$ and $Q_2(y_1,\ldots,y_k)$:- $body_2$

($\Rightarrow$) $Q_1 \subseteq Q_2 \;\Rightarrow\;$ there exists a query homomorphism from Q_2 to Q_1

- Clearly, $(c(x_1),\ldots,c(x_k)) \in Q_1(D[Q_1])$ – recall that $D[Q_1] = c(body_1)$
- Since $Q_1 \subseteq Q_2$, we conclude that $(c(x_1),\ldots,c(x_k)) \in Q_2(D[Q_1])$
- Therefore, there exists a homomorphism h such that $h(body_2) \subseteq D[Q_1] = c(body_1)$ and $h((y_1,\ldots,y_k)) = (c(x_1),\ldots,c(x_k))$
- By construction, $c^{-1}(c(body_1)) = body_1$ and $c^{-1}((c(x_1),\ldots,c(x_k))) = (x_1,\ldots,x_k)$
- Therefore, $c^{-1} \circ h$ is a query homomorphism from Q_2 to Q_1

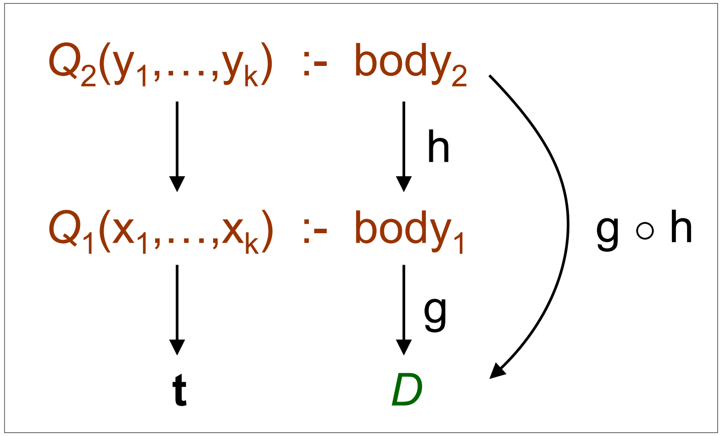


Homomorphism Theorem: Proof

Assume that $Q_1(x_1,\ldots,x_k)$:- body_1 and $Q_2(y_1,\ldots,y_k)$:- body_2

($\Leftarrow$) $Q_1 \subseteq Q_2 \;\Leftarrow\;$ there exists a query homomorphism from Q_2 to Q_1

- Consider a database D, and a tuple **t** such that $\mathbf{t} \in Q_1(D)$
- We need to show that $\mathbf{t} \in Q_2(D)$
- Clearly, there exists a homomorphism g such that $g(\text{body}_1) \subseteq D$ and $g((x_1,\ldots,x_k)) = \mathbf{t}$
- By hypothesis, there exists a query homomorphism h from Q_2 to Q_1
- Therefore, $g(h(\text{body}_2)) \subseteq D$ and $g(h((y_1,\ldots,y_k))) = \mathbf{t}$, which implies that $\mathbf{t} \in Q_2(D)$



Existence of a Query Homomorphism

Theorem: Let Q_1 and Q_2 be conjunctive queries. The problem of deciding whether there exists a query homomorphism from Q_2 to Q_1 is NP-complete

Proof:

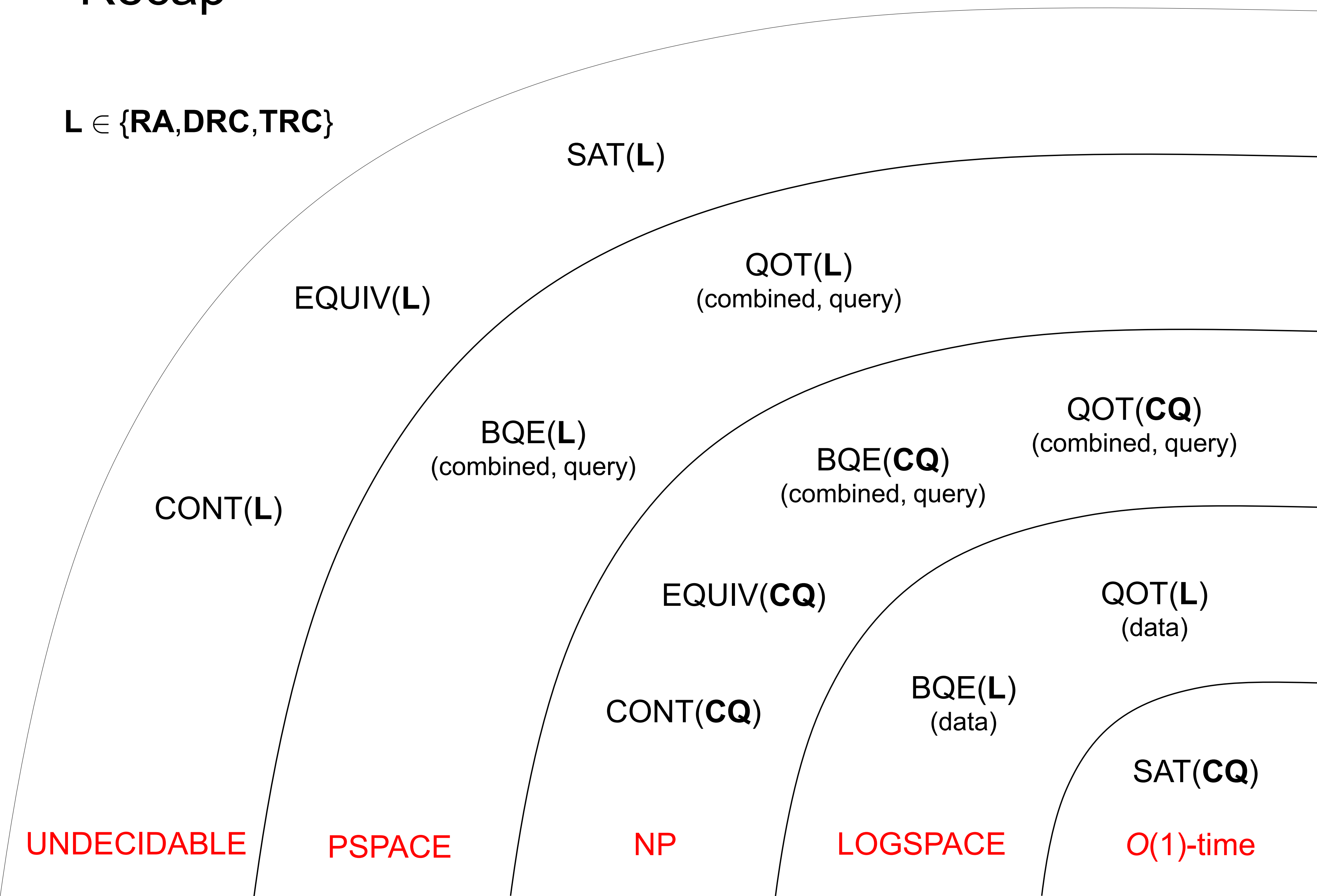
(NP-membership) Guess a substitution, and verify that is a query homomorphism

(NP-hardness) Straightforward reduction from BQE(**CQ**)

By applying the homomorphism theorem we get that:

Corollary: EQUIV(**CQ**) and CONT(**CQ**) are NP-complete

Recap



L ∈ {RA,DRC,TRC}
SAT(L)
EQUIV(L)
QOT(L)
(combined, query)
CONT(L)
BQE(L)
(combined, query)
QOT(CQ)
(combined, query)
BQE(CQ)
(combined, query)
EQUIV(CQ)
QOT(L)
(data)
CONT(CQ)
BQE(L)
(data)
SAT(CQ)
UNDECIDABLE
PSPACE
NP
LOGSPACE
O(1)-time

Minimizing Conjunctive Queries

- **Goal:** minimize the number of joins in a query

- A conjunctive query Q_1 is minimal if there is no conjunctive query Q_2 such that:
  1. $Q_1 \equiv Q_2$
  2. Q_2 has fewer atoms than Q_1

- The task of CQ minimization is, given a conjunctive query Q, to compute a minimal one that is equivalent to Q

Minimization by Deletion

By exploiting the homomorphism theorem we can show the following:

Theorem: Consider a conjunctive query $Q_1(x_1,\ldots,x_k)$:- $body_1$.

If Q_1 is equivalent to a conjunctive query $Q_2(y_1,\ldots,y_k)$:- $body_2$, where $|body_2| < |body_1|$,

then Q_1 is equivalent to a query $Q_1(x_1,\ldots,x_k)$:- $body_3$ such that $body_3 \subseteq body_1$

The above theorem says that to minimize a conjunctive query $Q_1(\mathbf{x})$:- body we simply need to remove some atoms from body

Minimization Procedure

Minimization($Q(\mathbf{x})$:- body)

Repeat until no change

choose an atom $\alpha \in$ body

if there is a query homomorphism from $Q(\mathbf{x})$:- body to $Q(\mathbf{x})$:- body $\setminus \{\alpha\}$

then body := body $\setminus \{\alpha\}$

Return $Q(\mathbf{x})$:- body

Note: if there is a query homomorphism from $Q(\mathbf{x})$:- body to $Q(\mathbf{x})$:- body $\setminus \{\alpha\}$, then the two queries are equivalent since there is trivially a query homomorphism from the latter to the former query

Minimization Procedure: Example

(a,b,c,d are constants)

Q(x) :- R(x,y), R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

$\{y \rightarrow b\}$

Q(x) :- R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

$\{v \rightarrow c\}$

Q(x) :- R(x,b), R(a,b), R(u,c), S(a,c,d)

minimal query

Note: the mapping $x \rightarrow a$ is not valid since x is a distinguished variable

Uniqueness of Minimal Queries

Natural question: does the order in which we remove atoms from the body of the input conjunctive query matter?

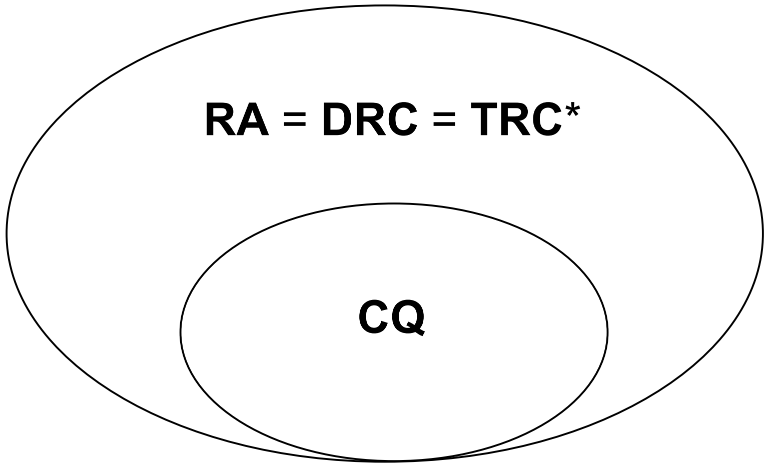
Theorem: Consider a conjunctive query Q. Let Q_1 and Q_2 be minimal conjunctive queries such that $Q_1 \equiv Q$ and $Q_2 \equiv Q$. Then, Q_1 and Q_2 are isomorphic (i.e., they are the same up to variable renaming)

Therefore, given a conjunctive query Q, the result of Minimization(Q) is unique (up to variable renaming) and is called the core of Q

Wrap-Up

- The main relational query languages – **RA/DRC/TRC**
  - Evaluation is decidable – foundations of the database industry
  - Perfect query optimization is impossible

- Conjunctive queries – an important query language
  - All the relevant algorithmic problems are decidable
  - Query minimization



*under the active domain semantics

Associated Papers

- Ashok K. Chandra, Philip M. Merlin: Optimal Implementation of Conjunctive Queries in Relational Data Bases. STOC 1977: 77-90

  Criterion for CQ containment/equivalence

- Martin Grohe: From polynomial time queries to graph structure theory. Commun. ACM 54(6): 104-112 (2011)

  A general account of connections between structural properties of databases and languages that capture efficient queries over them

- Martin Grohe: Fixed-point definability and polynomial time on graphs with excluded minors. Journal of the ACM 59(5): 27 (2012)

  We can capture PTIME on databases that satisfy certain structural (graph-theoretic) restrictions

Associated Papers

- Neil Immerman: Languages that Capture Complexity Classes. SIAM J. Comput. 16(4): 760-778 (1987)

  Query languages that correspond to complexity classes

- Phokion G. Kolaitis, Moshe Y. Vardi: Conjunctive-Query Containment and Constraint Satisfaction. J. Comput. Syst. Sci. 61(2): 302-332 (2000)

  A connection between CQs and a central AI problem of constraint satisfaction

- Leonid Libkin: The finite model theory toolbox of a database theoretician. PODS 2009: 65-76

  A toolbox for reasoning about expressivity and complexity of query languages

Associated Papers

- Leonid Libkin: Expressive power of SQL. Theor. Comput. Sci. 296(3): 379-404 (2003)

  A specific application of the above toolbox for SQL

- Moshe Y. Vardi: The Complexity of Relational Query Languages (Extended Abstract). STOC 1982: 137-146

  Different types of complexity of database queries

- Christos H. Papadimitriou, Mihalis Yannakakis: On the Complexity of Database Queries. J. Comput. Syst. Sci. 58(3): 407-427 (1999)

  A finer way of measuring complexity, between data and combined