CS 294 Special Topics in Database Theory
Unit 1: Logic and Queries

Dan Suciu

University of Washington

Welcome!

This course is intended for graduate students interested in getting deeper into data management technologies: understanding the underlying theory.

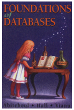
I am a professor at the University of Washington, attending the SIMONS institute Logic and Algorithms in Database Theory and AI, and the recipient of the Theory of Computing Chancellor's Professorship at UC Berkeley.

So, this course is a one-time offering.

Tentative Course Outline

Tue	Thu	Unit	Topic	Lecturer
8/29 9/5	8/31	U1	Queries and Static Analysis.	
	9/7	U2	Hypertree Decomposition.	
9/12	9/14	U3	Incremental View Maintenance	Dan Olteanu
9/19	 9/21	U4	AGM Bound WCOJ	 Hung Ngo
9/25-9/27: WS 1: Fine-grained Complexity, Logic, Query Eval				
10/3	10/5	U5	Database Constraints.	
10/10	10/11	U6	Probabilistic databases	
10/16-10/20: WS 2: Probabilistic Circuits and Logic				
10/24	10/26	U7	Semirings, K-Relations.	
10/31		U8	FAQ	Hung Ngo
 11/7	11/2 11/9	U9	Datalog, Chase.	
11/13-11/17: WS 3: Logic and Algebra for Query Evaluation				
11/21	11/28	U10	TBD	

Recommended Readings



The "Alice Book" [Abiteboul et al., 1995]

Libkin's *Finite Model Theory* [Libkin, 2004]
A much shorter tutorial in PODS [Libkin, 2009].

New upcoming book on Database Theory [Arenas et al., 2022].

Basic Definitions

Structures

A vocabulary σ is a set of relation symbols $R_1, \ldots, R_k$ and function symbols $f_1, \ldots, f_m$, each with a fixed arity.

A structure is $\boldsymbol{D} = (D, R_1^D, \ldots, R_k^D, f_1^D, \ldots, f_m^D)$, where $R_i^D \subseteq (D)^{\text{arity}(R_i)}$ and $f_j^D : (D)^{\text{arity}(f_j)} \rightarrow D$.

D = the *domain* or the *universe*; always assumed $\neq \emptyset$.
$v \in D$ is called an *element* or a *value* or a *point*.
$\boldsymbol{D}$ called a *structure* or a *model* or *database*.

Structures

A vocabulary σ is a set of relation symbols $R_1, \ldots, R_k$ and function symbols $f_1, \ldots, f_m$, each with a fixed arity.

A structure is $\boldsymbol{D} = (D, R_1^D, \ldots, R_k^D, f_1^D, \ldots, f_m^D)$,
where $R_i^D \subseteq (D)^{\text{arity}(R_i)}$ and $f_j^D : (D)^{\text{arity}(f_j)} \rightarrow D$.

$D =$ the *domain* or the *universe*; always assumed $\neq \emptyset$.
$v \in D$ is called an *element* or a *value* or a *point*.
$\boldsymbol{D}$ called a *structure* or a *model* or *database*.

Structures

A vocabulary σ is a set of relation symbols $R_1, \ldots, R_k$ and function symbols $f_1, \ldots, f_m$, each with a fixed arity.

A structure is $\boldsymbol{D} = (D, R_1^D, \ldots, R_k^D, f_1^D, \ldots, f_m^D)$,
where $R_i^D \subseteq (D)^{\text{arity}(R_i)}$ and $f_j^D : (D)^{\text{arity}(f_j)} \to D$.

D = the *domain* or the *universe*; always assumed $\neq \emptyset$.
$v \in D$ is called an *element* or a *value* or a *point*.
$\boldsymbol{D}$ called a *structure* or a *model* or *database*.

Examples

A graph is $G = (V, E)$, $E \subseteq V \times V$.

A field is $\mathbb{F} = (F, 0, 1, +, \cdot)$ where

- F is a set.
- 0 and 1 are constants (i.e. functions $F^0 \to F$).
- $+$ and $\cdot$ are functions $F^2 \to F$.

An ordered set is $\boldsymbol{S} = (S, \leq)$ where $\leq \subseteq S \times S$.

A database is $\boldsymbol{D} = (\text{Domain}, \text{Customer}, \text{Order}, \text{Product})$.

Discussion

- We don't really need functions, since $f : D^k \rightarrow D$ is represented by its graph $\subseteq D^{k+1}$, but we keep them when convenient.
- If f is a 0-ary function $D^0 \rightarrow D$, then it is a constant D, and we denote it c rather than f.
- $\boldsymbol{D}$ can be a finite or an infinite structure.

First Order Logic

Fix a vocabulary σ and a set of variables $x_1, x_2, \ldots$

Terms:

- Every constant c and every variable x is a term.
- If $t_1, \ldots, t_k$ are terms then $f(t_1, \ldots, t_k)$ is a term.

Formulas:

- $\boldsymbol{F}$ is a formula (means *false*).
- If $t_1, \ldots, t_k$ are terms, then $t_1 = t_2$ and $R(t_1, \ldots, t_k)$ are formulas.
- If φ, ψ are formulas, then so are $\varphi \to \psi$ and $\forall x(\varphi)$.

Discussion

We were very frugal! We used only $\boldsymbol{F}, \rightarrow, \forall$.

In practice we use several derived operations:

- $\neg\varphi$ is a shorthand for $\varphi \rightarrow \boldsymbol{F}$.
- $\varphi \vee \psi$ is a shorthand for $(\neg\varphi) \rightarrow \psi$.
- $\varphi \wedge \psi$ is a shorthand for $\neg(\varphi \vee \psi)$.
- $\exists x(\varphi)$ is a shorthand for $\neg(\forall x(\neg\varphi))$.

$\boldsymbol{F}$ often denoted: `false` or $\perp$ or 0.

$=$ is not always part of the language

Formulas and Sentences

We say that $\forall x(\varphi)$ *binds* x in φ. Every occurrence of x in φ is *bound*. Otherwise, it is *free*.

A sentence is a formula φ without free variables.

E.g. formula $\varphi(x,z) = \exists y(E(x,y) \wedge E(y,z))$. Free variables: x, z.

E.g. sentence $\varphi = \exists x \forall z \exists y(E(x,y) \wedge E(y,z))$.

Truth

Let φ be a sentence, and $\boldsymbol{D}$ a structure

Definition

We say that φ is true in $\boldsymbol{D}$, written $\boxed{\boldsymbol{D} \models \varphi}$, if:

- φ is $c = c'$ and c, c' are the same constant.
- φ is $R(c_1, \ldots, c_n)$ and $(c_1, \ldots, c_n) \in R^D$.
- φ is $\psi_1 \rightarrow \psi_2$ and $\boldsymbol{D} \not\models \psi_1$, or $\boldsymbol{D} \models \psi_1$ and $\boldsymbol{D} \models \psi_2$.
- φ is $\forall y(\psi)$, and, forall $b \in D$, $\boldsymbol{D} \models \psi[b/y]$.

This definition is boring but important!

Special Case: Propositional Logic

A nullary relation, $A()$, is the same as a propositional variable:

- In any structure $\boldsymbol{D}$, A^D can be either $\emptyset$ or $\{()\}$.
- If $A^D = \{()\}$ then we say that A^D is true.
- If $A^D = \emptyset$ then we say that A^D is false.

Sentences over nullary predicates are the same as propositional formulas:

$$A() \wedge (B() \vee \neg A())$$

What do these sentences say about *D*?

$$\exists x \exists y \exists z (x \neq y) \wedge (x \neq z) \wedge (y \neq z)$$

$$\exists x \exists y \forall z (z = x) \vee (z = y)$$

What do these sentences say about D?

$$\exists x \exists y \exists z (x \neq y) \wedge (x \neq z) \wedge (y \neq z)$$

"There are at least three elements", i.e. $|D| \geq 3$

$$\exists x \exists y \forall z (z = x) \vee (z = y)$$

What do these sentences say about D?

$$\exists x \exists y \exists z (x \neq y) \wedge (x \neq z) \wedge (y \neq z)$$

"There are at least three elements", i.e. $|D| \geq 3$

$$\exists x \exists y \forall z (z = x) \vee (z = y)$$

"There are at most two elements", i.e. $|D| \leq 2$

What do these sentences say about D?

$$\forall x \exists y E(x,y) \vee E(y,x)$$

$$\forall x \forall y \exists z E(x,z) \wedge E(z,y)$$

$$\begin{aligned} &\exists x \exists y \exists z (\forall u (u = x) \vee (u = y) \vee (u = z)) \\ &\wedge \neg E(x,x) \wedge E(x,y) \wedge \neg E(x,z) \\ &\wedge \neg E(y,z) \wedge \neg E(y,y) \wedge E(y,z) \\ &\wedge E(z,x) \wedge \neg E(z,y) \wedge \neg E(z,z) \end{aligned}$$

What do these sentences say about *D*?

$$\forall x \exists y E(x,y) \vee E(y,x)$$

"There are no isolated nodes"

$$\forall x \forall y \exists z E(x,z) \wedge E(z,y)$$

$$\begin{gathered}\exists x \exists y \exists z (\forall u (u = x) \vee (u = y) \vee (u = z)) \\ \wedge \neg E(x,x) \wedge E(x,y) \wedge \neg E(x,z) \\ \wedge \neg E(y,z) \wedge \neg E(y,y) \wedge E(y,z) \\ \wedge E(z,x) \wedge \neg E(z,y) \wedge \neg E(z,z)\end{gathered}$$

What do these sentences say about ***D***?

$$\forall x \exists y E(x, y) \vee E(y, x)$$

"There are no isolated nodes"

$$\forall x \forall y \exists z E(x, z) \wedge E(z, y)$$

"Every two nodes are connected by a path of length 2"

$$\begin{gathered}\exists x \exists y \exists z (\forall u (u = x) \vee (u = y) \vee (u = z)) \\ \wedge \neg E(x, x) \wedge E(x, y) \wedge \neg E(x, z) \\ \wedge \neg E(y, z) \wedge \neg E(y, y) \wedge E(y, z) \\ \wedge E(z, x) \wedge \neg E(z, y) \wedge \neg E(z, z)\end{gathered}$$

What do these sentences say about D?

$$\forall x \exists y E(x, y) \vee E(y, x)$$

"There are no isolated nodes"

$$\forall x \forall y \exists z E(x, z) \wedge E(z, y)$$

"Every two nodes are connected by a path of length 2"

$$\begin{gathered}
\exists x \exists y \exists z (\forall u (u = x) \vee (u = y) \vee (u = z)) \\
\wedge \neg E(x, x) \wedge E(x, y) \wedge \neg E(x, z) \\
\wedge \neg E(y, z) \wedge \neg E(y, y) \wedge E(y, z) \\
\wedge E(z, x) \wedge \neg E(z, y) \wedge \neg E(z, z)
\end{gathered}$$

It completely determines the graph: $D = \{a, b, c\}$ and $a \rightarrow b \rightarrow c \rightarrow a$.

Classical Model Theory

Fix a sentence φ, and a set of sentences Σ (may be infinite).

- Satisfiability: Σ is satisfiable if $\exists \boldsymbol{D}$ such that $\boldsymbol{D} \models \Sigma$. $\texttt{SAT}(\Sigma)$.
- Implication: Σ implies φ if $\forall \boldsymbol{D}$, $\boldsymbol{D} \models \Sigma$ implies $\boldsymbol{D} \models \varphi$. $\Sigma \models \varphi$.
- Validity: φ is valid if $\forall \boldsymbol{D}$, $\boldsymbol{D} \models \varphi$. We write $\models \varphi$ or $\texttt{VAL}(\varphi)$.

$$\neg\texttt{SAT}(\varphi) \text{ iff } \texttt{VAL}(\neg\varphi)$$

Completeness, Undecidability

Gödels Completeness Thm: $\Sigma \models \varphi$ iff there exists a finite proof $\Sigma \vdash \varphi$.
Church's Undecidability Thm: `VAL` is undecidable. Hence, so is `SAT`.

We will not discuss what a "proof" $\Sigma \vdash \varphi$ means.

Corollary

There exists an algorithm that enumerates all valid sentences:

$$VAL = \{\varphi_0, \varphi_1, \varphi_2, \ldots\}$$

There exists an algorithm that enumerates all unsatisfiable sentences:

$$UNSAT = \{\varphi_0, \varphi_1, \varphi_2, \ldots\}$$

We say that `VAL` is recursively enumerable, r.e., and `SAT` is co-r.e.

Completeness, Undecidability

Gödels Completeness Thm: $\Sigma \models \varphi$ iff there exists a finite proof $\Sigma \vdash \varphi$.
Church's Undecidability Thm: $\mathtt{VAL}$ is undecidable. Hence, so is $\mathtt{SAT}$.

We will not discuss what a "proof" $\Sigma \vdash \varphi$ means.

Corollary

There exists an algorithm that enumerates all valid sentences:

$$VAL = \{\varphi_0, \varphi_1, \varphi_2, \ldots\}$$

There exists an algorithm that enumerates all unsatisfiable sentences:

$$UNSAT = \{\varphi_0, \varphi_1, \varphi_2, \ldots\}$$

We say that $\mathtt{VAL}$ is recursively enumerable, r.e., and $\mathtt{SAT}$ is co-r.e.

Finite Model Theory, Databases, Verification

All previous problems, where the models are restricted to be finite:

- Finite satisfiability: $\mathtt{SAT}_{\text{fin}}(\Sigma)$.
- Finite implication: $\Sigma \models_{\text{fin}} \varphi$.
- Finite validity: $\models_{\text{fin}} \varphi$ or $\mathtt{VAL}_{\text{fin}}(\varphi)$.

New problems that make sense only in the finite:

- Model checking: Given φ, $\boldsymbol{D}$, determine whether $\boldsymbol{D} \models \varphi$.
- Query evaluation: Given $\varphi(\boldsymbol{x})$, $\boldsymbol{D}$, compute $\{\boldsymbol{a} \mid \boldsymbol{D} \models \varphi[\boldsymbol{a}/\boldsymbol{x}]\}$.

Examples

- $\boldsymbol{F} \models \varphi$ for any sentence φ why?.

Examples

- $\boldsymbol{F} \models \varphi$ for any sentence φ why?.
- $\Sigma \models \boldsymbol{F}$ iff Σ is unsatisfiable why?.

Examples

- $\boldsymbol{F} \models \varphi$ for any sentence φ why?.
- $\Sigma \models \boldsymbol{F}$ iff Σ is unsatisfiable why?.
- Is $\Phi \stackrel{\text{def}}{=} (\forall x \exists y E(x, y)) \wedge (\forall x_1 \forall x_2 \forall y (E(x_1, y) \wedge E(x_2, y) \Rightarrow x_1 = x_2))$ satisfiable?

Examples

- $\boldsymbol{F} \models \varphi$ for any sentence φ why?.
- $\Sigma \models \boldsymbol{F}$ iff Σ is unsatisfiable why?.
- Is $\Phi \stackrel{\text{def}}{=} (\forall x \exists y E(x,y)) \wedge (\forall x_1 \forall x_2 \forall y (E(x_1,y) \wedge E(x_2,y) \Rightarrow x_1 = x_2))$ satisfiable?
  Yes: E.g. $E = \{(1,2),(2,3),(3,1)\}$.

Examples

- $\boldsymbol{F} \models \varphi$ for any sentence φ why?.
- $\Sigma \models \boldsymbol{F}$ iff Σ is unsatisfiable why?.
- Is $\Phi \stackrel{\text{def}}{=} (\forall x \exists y E(x, y)) \wedge (\forall x_1 \forall x_2 \forall y (E(x_1, y) \wedge E(x_2, y) \Rightarrow x_1 = x_2))$ satisfiable?
  Yes: E.g. $E = \{(1, 2), (2, 3), (3, 1)\}$.
- Is $\Phi \wedge (\exists y \forall x \neg E(x, y))$ (where Φ is defined above) satisfiable?

Examples

- $\boldsymbol{F} \models \varphi$ for any sentence φ why?.
- $\Sigma \models \boldsymbol{F}$ iff Σ is unsatisfiable why?.
- Is $\Phi \stackrel{\text{def}}{=} (\forall x \exists y E(x, y)) \wedge (\forall x_1 \forall x_2 \forall y(E(x_1, y) \wedge E(x_2, y) \Rightarrow x_1 = x_2))$ satisfiable?
  Yes: E.g. $E = \{(1,2),(2,3),(3,1)\}$.
- Is $\Phi \wedge (\exists y \forall x \neg E(x, y))$ (where Φ is defined above) satisfiable?
  Yes: $E = \{(0,1),(1,2),(2,3),\ldots\}$

Examples

- $\boldsymbol{F} \models \varphi$ for any sentence φ why?.
- $\Sigma \models \boldsymbol{F}$ iff Σ is unsatisfiable why?.
- Is $\Phi \stackrel{\text{def}}{=} (\forall x \exists y E(x,y)) \wedge (\forall x_1 \forall x_2 \forall y (E(x_1,y) \wedge E(x_2,y) \Rightarrow x_1 = x_2))$ satisfiable?
  Yes: E.g. $E = \{(1,2),(2,3),(3,1)\}$.
- Is $\Phi \wedge (\exists y \forall x \neg E(x,y))$ (where Φ is defined above) satisfiable?
  Yes: $E = \{(0,1),(1,2),(2,3),\ldots\}$
  But Not satisfiable in the finite. "Axioms of infinity" [Börger et al., 1997]

$$\boxed{\mathtt{SAT}_{\text{fin}}(\varphi) \Rightarrow \mathtt{SAT}(\varphi)}$$

Finite v.s. Classical Model Theory

In relational databases we are interested in Finite Model Theory.

$\texttt{VAL}_{\text{fin}}$, $\texttt{SAT}_{\text{fin}}$ differ from $\texttt{VAL}$, $\texttt{SAT}$.
Could they be $\texttt{VAL}_{\text{fin}}$, $\texttt{SAT}_{\text{fin}}$ decidable

Finite v.s. Classical Model Theory

In relational databases we are interested in Finite Model Theory.

$\texttt{VAL}_{\text{fin}}$, $\texttt{SAT}_{\text{fin}}$ differ from $\texttt{VAL}$, $\texttt{SAT}$.
Could they be $\texttt{VAL}_{\text{fin}}$, $\texttt{SAT}_{\text{fin}}$ decidable

There is hope:

- In classical model theory $\texttt{VAL}$ is r.e., $\texttt{SAT}$ is co-r.e.
- In finite model theory $\texttt{SAT}_{\text{fin}}$ is r.e. why?.

Trakhtenbrot's Undecidability Theorem

Theorem (Trakhtenbrot)

If the vocabulary includes at least one relation of arity ≥ 2, then SAT_{fin} is undecidable. (We will prove it later.)

Therefore static analysis of arbitrary FO formulas is undecidable; same as for Turing-complete programming languages: this justifies studying fragments of FO, where static analysis is possible.

We will prove Trakthenbrot's theorem later.

The condition at least one relation of arity ≥ 2 is necessary. See Homework 1.

Summary

Classical Model Theory:

- Concerned with satisfiability, validity, provability.
- Major, fundamental results: Gödel's completeness; Church undecidability; the Compactness Theorem; Löwenheim–Skolem; Gödel's incompleteness.

Finite Model Theory:

- Concerned with similar questions, plus evaluation.
- Major, fundamental results: Trakhtentbrot's undecidability; Fagin's $0/1$-law; Fagin's SO=NP theorem.

Relational Model

Origins

In 1970-1971 Tedd Codd proposed that databases should be modeled as finite structures, and queries represented by formulas.

A decade of debates followed, where the relational data model had to compete against the established CODASYL model.

This story is now the founding legend, par of the folklore of our community. A great reading is *What Goes Around Comes Around* in [Bailis et al., 2015].

Relational Databases

Fix the schema (vocabulary): $R_1, R_2, \ldots$.

A relational database instance is a finite structure $\boldsymbol{D} = (D, R_1^D, R_2^D, \ldots)$

We often omit the domain and write $\boldsymbol{D} = (R_1^D, R_2^D, \ldots)$.

The active domain, $\text{ADom}(\boldsymbol{D})$, is the set of constants that occur in $R_1^D, R_2^D, \ldots$

A query, $Q(\boldsymbol{x})$, is an FO formula with free variables $\boldsymbol{x}$. We write (with some overloading) $Q(\boldsymbol{D})$ for the result of Q on a database $\boldsymbol{D}$.

The Drinkers-Beer-Bar Example

Introduced by [Ullman, 1980].

```
Frequents(Drinker,Bar)
Serves(Bar,Beer)
Likes(Drinker,Beer)
```

The Drinkers-Beer-Bar Example

Introduced by [Ullman, 1980].

```
Frequents(Drinker,Bar)
Serves(Bar,Beer)
Likes(Drinker,Beer)
```

Drinkers who frequent some bar who serve some beer that they like:

$$Q(x) = \exists y \exists z(\texttt{Frequents}(x, y) \wedge \texttt{Serves}(y, z) \wedge \texttt{Likes}(x, z))$$

The Drinkers-Beer-Bar Example

Introduced by [Ullman, 1980].

```
Frequents(Drinker,Bar)
Serves(Bar,Beer)
Likes(Drinker,Beer)
```

Drinkers who frequent <u>some</u> bar who serve <u>some</u> beer that they like:

$$Q(x) = \exists y \exists z(\texttt{Frequents}(x, y) \wedge \texttt{Serves}(y, z) \wedge \texttt{Likes}(x, z))$$

Drinkers who frequent <u>only</u> bars who serve <u>only</u> beers that they like:

$$Q(x) = \forall y(\texttt{Frequents}(x, y) \Rightarrow \forall z(\texttt{Serves}(y, z) \Rightarrow \texttt{Likes}(x, z)))$$

The Drinkers-Beer-Bar Example

Introduced by [Ullman, 1980].

```
Frequents(Drinker,Bar)
Serves(Bar,Beer)
Likes(Drinker,Beer)
```

Drinkers who frequent some bar who serve some beer that they like:

$$Q(x) = \exists y \exists z(\texttt{Frequents}(x, y) \wedge \texttt{Serves}(y, z) \wedge \texttt{Likes}(x, z))$$

Drinkers who frequent only bars who serve only beers that they like:

$$Q(x) = \forall y(\texttt{Frequents}(x, y) \Rightarrow \forall z(\texttt{Serves}(y, z) \Rightarrow \texttt{Likes}(x, z)))$$

The last query is incorrect!

Domain Independence

The last query is actually incorrect! Let's look at a simpler query:

$$\boxed{Q(x) = \forall y(R(x,y) \Rightarrow S(x,y))}$$

Domain Independence

The last query is actually incorrect! Let's look at a simpler query:

$$\boxed{Q(x) = \forall y(R(x,y) \Rightarrow S(x,y))}$$

Recall the "boring" definition: if $c \in D$, $c \notin \Pi_x(R^D)$ then $Q(c)$ is true.
Q returns values c that are not in the active domain; domain dependent.

Domain Independence

The last query is actually incorrect! Let's look at a simpler query:

$$\boxed{Q(x) = \forall y(R(x, y) \Rightarrow S(x, y))}$$

Recall the "boring" definition: if $c \in D$, $c \notin \Pi_x(R^D)$ then $Q(c)$ is true.
Q returns values c that are not in the active domain; domain dependent.

Definition

An FO formula (query) is domain independent if it does not depend on the domain D of the structure $(D, R_1^D, R_2^D, \ldots)$.

Domain Independence

The last query is actually incorrect! Let's look at a simpler query:

$$\boxed{Q(x) = \forall y(R(x, y) \Rightarrow S(x, y))}$$

Recall the "boring" definition: if $c \in D$, $c \notin \Pi_x(R^D)$ then $Q(c)$ is true.
Q returns values c that are not in the active domain; domain dependent.

Definition

An FO formula (query) is domain independent if it does not depend on the domain D of the structure $(D, R_1^D, R_2^D, \ldots)$.

Are these queries independent?
$Q(x, y) = R(x) \vee S(y)$

Domain Independence

The last query is actually incorrect! Let's look at a simpler query:

$$\boxed{Q(x) = \forall y(R(x, y) \Rightarrow S(x, y))}$$

Recall the "boring" definition: if $c \in D$, $c \notin \Pi_x(R^D)$ then $Q(c)$ is true.
Q returns values c that are not in the active domain; domain dependent.

Definition

An FO formula (query) is domain independent if it does not depend on the domain D of the structure $(D, R_1^D, R_2^D, \ldots)$.

Are these queries independent?
$Q(x, y) = R(x) \vee S(y)$ Domain dependent.

Domain Independence

The last query is actually incorrect! Let's look at a simpler query:

$$\boxed{Q(x) = \forall y(R(x,y) \Rightarrow S(x,y))}$$

Recall the "boring" definition: if $c \in D$, $c \notin \Pi_x(R^D)$ then $Q(c)$ is true.
Q returns values c that are not in the active domain; domain dependent.

Definition

An FO formula (query) is domain independent if it does not depend on the domain D of the structure $(D, R_1^D, R_2^D, \ldots)$.

Are these queries independent?

$Q(x,y) = R(x) \vee S(y)$ Domain dependent.

$Q(x) = R(x) \wedge \exists y(\neg S(x,y))$

Domain Independence

The last query is actually incorrect! Let's look at a simpler query:

$$\boxed{Q(x) = \forall y(R(x,y) \Rightarrow S(x,y))}$$

Recall the "boring" definition: if $c \in D$, $c \notin \Pi_x(R^D)$ then $Q(c)$ is true.
Q returns values c that are not in the active domain; domain dependent.

Definition

An FO formula (query) is domain independent if it does not depend on the domain D of the structure $(D, R_1^D, R_2^D, \ldots)$.

Are these queries independent?

$Q(x,y) = R(x) \vee S(y)$ Domain dependent.

$Q(x) = R(x) \wedge \exists y(\neg S(x,y))$ Domain dependent.

Domain Independence

Given a formula φ, how do we check whether it is domain independent?

Domain Independence

Given a formula φ, how do we check whether it is domain independent?

Theorem

Checking domain independence is undecidable.

Domain Independence

Given a formula φ, how do we check whether it is domain independent?

Theorem

Checking domain independence is undecidable.

Proof Assuming an algorithm for checking domain independence, we solve $\texttt{SAT}_{\text{fin}}$, which contradicts Trakhtenbrot's theorem:

- Fix some domain-dependent query, say $\varphi = \forall x R(x)$.
- Given an FO sentence Φ, construct a new sentence $\psi \stackrel{\text{def}}{=} \Phi \wedge \varphi$.
- Then ψ is domain independent iff Φ is unsatisfiable.

Domain Independence

Given a formula φ, how do we check whether it is domain independent?

Theorem

Checking domain independence is undecidable.

Proof Assuming an algorithm for checking domain independence, we solve $\mathtt{SAT}_{\mathsf{fin}}$, which contradicts Trakhtenbrot's theorem:

- Fix some domain-dependent query, say $\varphi = \forall x R(x)$.
- Given an FO sentence Φ, construct a new sentence $\psi \stackrel{\text{def}}{=} \Phi \wedge \varphi$.
- Then ψ is domain independent iff Φ is unsatisfiable.

Q is range-restricted if each var is restricted to (a subset of) ADom.

$$\boxed{Q(x) = \exists u(R(x,u) \vee S(x,u)) \bigwedge (\forall y(R(x,y) \Rightarrow S(x,y)))}$$

Relational Algebra – Quick Review

Five operators:

- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

Relational Algebra – Quick Review

Five operators:

- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

Who likes Leffe?

$Q_1(x) = \texttt{Likes}(x, \texttt{'Leffe'})$

Relational Algebra – Quick Review

Five operators:

- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

Who likes Leffe?

$Q_1(x) = \texttt{Likes}(x, \texttt{'Leffe'})$

$$\begin{array}{c} \Pi_x \\ | \\ \sigma_{y=\texttt{'Leffe'}} \\ | \\ \texttt{Likes}(x,y) \end{array}$$

Relational Algebra – Quick Review

$Q_2(x, y, z) = \texttt{Frequents}(x, y) \wedge \texttt{Serves}(y, z) \wedge \texttt{Likes}(x, z)$

Five operators:

- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

Who likes Leffe?

$Q_1(x) = \texttt{Likes}(x, \texttt{'Leffe'})$

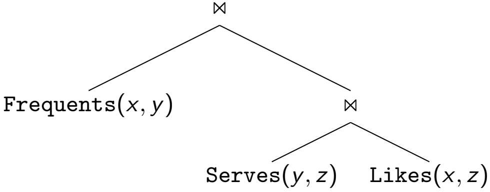
$$\begin{array}{c} \Pi_x \\ | \\ \sigma_{y=\texttt{'Leffe'}} \\ | \\ \texttt{Likes}(x,y) \end{array}$$

Relational Algebra – Quick Review

Five operators:

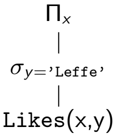
- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

$Q_2(x, y, z) = \texttt{Frequents}(x, y) \wedge \texttt{Serves}(y, z) \wedge \texttt{Likes}(x, z)$



Who likes Leffe?

$Q_1(x) = \texttt{Likes}(x, \texttt{'Leffe'})$



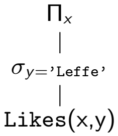
Relational Algebra – Quick Review

Five operators:

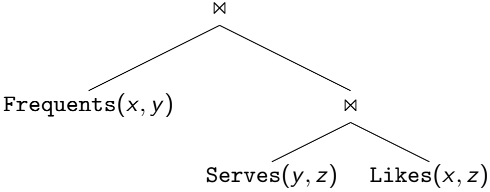
- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

Who likes Leffe?

$$Q_1(x) = \texttt{Likes}(x, \texttt{'Leffe'})$$



$$Q_2(x, y, z) = \texttt{Frequents}(x, y) \wedge \texttt{Serves}(y, z) \wedge \texttt{Likes}(x, z)$$



$$Q_3(x) = A(x) \wedge \forall y(B(y) \Rightarrow C(x, y))$$

Relational Algebra – Quick Review

Five operators:

- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

Who likes Leffe?

$$Q_1(x) = \texttt{Likes}(x, \texttt{'Leffe'})$$

$$\Pi_x$$
$$\sigma_{y=\texttt{'Leffe'}}$$
$$\texttt{Likes}(x,y)$$

$$Q_2(x,y,z) = \texttt{Frequents}(x,y) \wedge \texttt{Serves}(y,z) \wedge \texttt{Likes}(x,z)$$

$$\bowtie$$
$$\texttt{Frequents}(x,y) \qquad \bowtie$$
$$\texttt{Serves}(y,z) \quad \texttt{Likes}(x,z)$$

$$Q_3(x) = A(x) \wedge \forall y(B(y) \Rightarrow C(x,y))$$
$$Q_3(x) = A(x) \wedge \neg\exists y(B(y) \wedge \neg C(x,y))$$

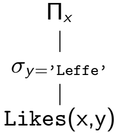
Relational Algebra – Quick Review

Five operators:

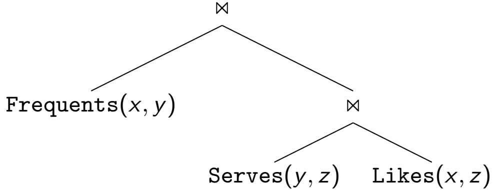
- Selection σ
- Projection Π
- Join $\bowtie$
- Union $\cup$
- Difference $-$

Who likes Leffe?

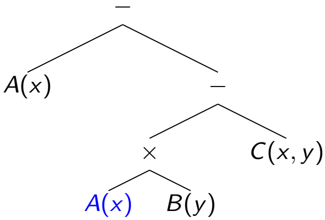
$$Q_1(x) = \texttt{Likes}(x, \texttt{'Leffe'})$$



$$Q_2(x, y, z) = \texttt{Frequents}(x, y) \wedge \texttt{Serves}(y, z) \wedge \texttt{Likes}(x, z)$$



$$Q_3(x) = A(x) \wedge \forall y(B(y) \Rightarrow C(x, y))$$
$$Q_3(x) = A(x) \wedge \neg\exists y(B(y) \wedge \neg C(x, y))$$



Easier with an anti-semijoin (look it up).

FO and RA are Equivalent

Theorem

Domain-independent FO and Relational Algebra express the same class of queries.

Proof: exercise.

FO and RA are Equivalent

Theorem

Domain-independent FO and Relational Algebra express the same class of queries.

Proof: exercise.

Physical independence principle: separation of What from How.

- Users write what they want, in a declarative language (FO).
- System decides how to compute the query most efficiently (RA plan).

Summary

- Relational data model is founded on finite model theory.
- Physical Data Independence is perhaps the deepest reason why it is still successful 50 years later: separate the What from the How.
- What is in FO. But too abstract for the real world (e.g. domain independence!), hence SQL and its history.
- Why is RA. But too limited for the real world, hence extended with aggregates, group-by, dependent joins, anti-semijoins, etc, etc.
- FO used in databases beyond query expressions: for constraints, optimization rules, verification.

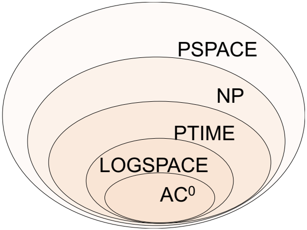
The Query Evaluation Problem

Complexity

A Turing-complete language can express any computable problem.

But FO is restricted. What is the complexity of the problems it can express?

First, are interested in the complexity class. Later we will study efficient algorithms.



The Query Evaluation Problem

Given a query Q and a database instance $\boldsymbol{D}$, compute $Q(\boldsymbol{D})$.
This is the bread-and-butter of database engines.

The Query Evaluation Problem

Given a query Q and a database instance $\boldsymbol{D}$, compute $Q(\boldsymbol{D})$.
This is the bread-and-butter of database engines.

Definition (Complexity of Query Evaluation [Vardi, 1982])

Three ways to define the complexity:

- Data Complexity. Fix the query Q, complexity is $f(|\boldsymbol{D}|)$.
- Query Complexity. Fix the database $\boldsymbol{D}$, complexity is $f(|Q|)$.
  A.k.a. expression complexity.
- Combined Complexity, $f(|Q|, |\boldsymbol{D}|)$.

The Query Evaluation Problem

Given a query Q and a database instance $\boldsymbol{D}$, compute $Q(\boldsymbol{D})$.
This is the bread-and-butter of database engines.

Definition (Complexity of Query Evaluation [Vardi, 1982])

Three ways to define the complexity:

- Data Complexity. Fix the query Q, complexity is $f(|\boldsymbol{D}|)$.
- Query Complexity. Fix the database $\boldsymbol{D}$, complexity is $f(|Q|)$.
  A.k.a. expression complexity.
- Combined Complexity, $f(|Q|, |\boldsymbol{D}|)$.

Which is most important in practice?

Data Complexity of FO is in AC^0

Theorem

The Data Complexity of FO is in AC^0

(Stronger: it is in uniform AC^0, but we will ignore this.)

Recall that AC^0 is at the bottom of the hierarchy:

$$AC^0 \subseteq \text{LOGSPACE} \subseteq \cdots \subseteq \text{PTIME}$$

Before we prove the theorem let's prove something simpler:
The Data Complexity of FO is in PTIME.

Data Complexity of FO is in PTIME: Proof

How do we evaluate this? $\boxed{Q = \exists x\,(A(x) \wedge \forall y(B(y) \Rightarrow C(x,y)))}$

Data Complexity of FO is in PTIME: Proof

How do we evaluate this? $\boxed{Q = \exists x\,(A(x) \wedge \forall y(B(y) \Rightarrow C(x,y)))}$

```
some_x = false;
for x = 1,n do:
    if A(x) then:
            all_y = true
            for y = 1,n do:
                if not (B(y) => C(x,y))
                then: all_y = false;
    if all_y then: some_x = true;
return some_x
```

Data Complexity of FO is in PTIME: Proof

How do we evaluate this? $\boxed{Q = \exists x\,(A(x) \wedge \forall y(B(y) \Rightarrow C(x,y)))}$

```
some_x = false;
for x = 1,n do:
    if A(x) then:
            all_y = true
            for y = 1,n do:
                if not (B(y) => C(x,y))
                then: all_y = false;
    if all_y then: some_x = true;
return some_x
```

- Generalizes to any sentence φ.
- Runtime $O(N^k)$, where:
  $N = |\text{ADom}|$
  $k = |\text{Vars}(\varphi)|$
- In PTIME (and in LOGSPACE), for fixed φ.

Data Complexity of FO is in PTIME: Proof

How do we evaluate this? $\boxed{Q = \exists x\,(A(x) \wedge \forall y(B(y) \Rightarrow C(x,y)))}$

```
some_x = false;
for x = 1,n do:
    if A(x) then:
            all_y = true
            for y = 1,n do:
                if not (B(y) => C(x,y))
                then: all_y = false;
    if all_y then: some_x = true;
return some_x
```

- Generalizes to any sentence φ.
- Runtime $O(N^k)$, where:
  $N = |\text{ADom}|$
  $k = |\text{Vars}(\varphi)|$
- In PTIME (and in LOGSPACE), for fixed φ.

Many texts state that the data complexity is in LOGSPACE, or in PTIME. The correct complexity is AC^0. Let's prove it

Definition of AC^0

Definition

A problem is in AC^0 if $\forall N$, there exists a circuit of polynomial size and constant depth, consisting NOT gates and unbounded fan-in AND and OR gates, that computes the problem on inputs of size N encoded using N bits.

Definition of AC^0

Definition

A problem is in AC^0 if $\forall N$, there exists a circuit of polynomial size and constant depth, consisting NOT gates and unbounded fan-in AND and OR gates, that computes the problem on inputs of size N encoded using N bits.

E.g. Given an undirected graph with n nodes, check if it has a triangle.

Definition of AC^0

Definition

A problem is in AC^0 if $\forall N$, there exists a circuit of polynomial size and constant depth, consisting NOT gates and unbounded fan-in AND and OR gates, that computes the problem on inputs of size N encoded using N bits.

E.g. Given an undirected graph with n nodes, check if it has a triangle.
When $n = 4$ there are $N = 6$ possible edges, E_{ij} for $1 \leq i < j \leq 4$

Definition of AC^0

Definition

A problem is in AC^0 if $\forall N$, there exists a circuit of polynomial size and constant depth, consisting NOT gates and unbounded fan-in AND and OR gates, that computes the problem on inputs of size N encoded using N bits.

E.g. Given an undirected graph with n nodes, check if it has a triangle. When $n = 4$ there are $N = 6$ possible edges, E_{ij} for $1 \leq i < j \leq 4$

$$E_{12} \wedge E_{23} \wedge E_{13} \vee E_{12} \wedge E_{24} \wedge E_{14} \vee E_{23} \wedge E_{34} \wedge E_{24} \vee E_{34} \wedge E_{14} \wedge E_{13}$$

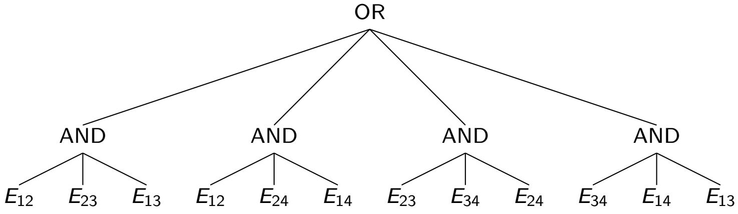
Definition of AC^0

Definition

A problem is in AC^0 if $\forall N$, there exists a circuit of polynomial size and constant depth, consisting NOT gates and unbounded fan-in AND and OR gates, that computes the problem on inputs of size N encoded using N bits.

E.g. Given an undirected graph with n nodes, check if it has a triangle. When $n = 4$ there are $N = 6$ possible edges, E_{ij} for $1 \leq i < j \leq 4$

$$E_{12} \wedge E_{23} \wedge E_{13} \vee E_{12} \wedge E_{24} \wedge E_{14} \vee E_{23} \wedge E_{34} \wedge E_{24} \vee E_{34} \wedge E_{14} \wedge E_{13}$$



Data Complexity FO is in AC^0: Proof

Fix a Boolean query Q in FO. Encode the input database $\boldsymbol{D}$ using bits:

- Let $N = |\text{ADom}(\boldsymbol{D})|$.

Data Complexity FO is in AC^0: Proof

Fix a Boolean query Q in FO. Encode the input database $\boldsymbol{D}$ using bits:

- Let $N = |\text{ADom}(\boldsymbol{D})|$.
- Encode a relation of arity k using N^k bits.
  E.g. $R(X, Y)$ encoded using Boolean matrix R_{ij}.

Data Complexity FO is in AC^0: Proof

Fix a Boolean query Q in FO. Encode the input database $\boldsymbol{D}$ using bits:

- Let $N = |\text{ADom}(\boldsymbol{D})|$.
- Encode a relation of arity k using N^k bits.
  E.g. $R(X, Y)$ encoded using Boolean matrix R_{ij}.
- Expand Q into a Boolean formula $\Phi_{Q,\boldsymbol{D}}$, called the lineage of Q on $\boldsymbol{D}$.

Data Complexity FO is in AC^0: Proof

Fix a Boolean query Q in FO. Encode the input database $\boldsymbol{D}$ using bits:

- Let $N = |\text{ADom}(\boldsymbol{D})|$.
- Encode a relation of arity k using N^k bits.
  E.g. $R(X, Y)$ encoded using Boolean matrix R_{ij}.
- Expand Q into a Boolean formula $\Phi_{Q,\boldsymbol{D}}$, called the lineage of Q on $\boldsymbol{D}$.
- Represent $\Phi_{Q,\boldsymbol{D}}$ using an AC^0 circuit.

Data Complexity FO is in AC^0: Proof

Fix a Boolean query Q in FO. Encode the input database $\boldsymbol{D}$ using bits:

- Let $N = |\text{ADom}(\boldsymbol{D})|$.
- Encode a relation of arity k using N^k bits.
  E.g. $R(X, Y)$ encoded using Boolean matrix R_{ij}.
- Expand Q into a Boolean formula $\Phi_{Q,\boldsymbol{D}}$, called the lineage of Q on $\boldsymbol{D}$.
- Represent $\Phi_{Q,\boldsymbol{D}}$ using an AC^0 circuit.

Example: $\boxed{Q = \exists x\,(A(x) \wedge \forall y(B(y) \Rightarrow C(x, y)))}$

Data Complexity FO is in AC^0: Proof

Fix a Boolean query Q in FO. Encode the input database $\boldsymbol{D}$ using bits:

- Let $N = |\text{ADom}(\boldsymbol{D})|$.
- Encode a relation of arity k using N^k bits.
  E.g. $R(X, Y)$ encoded using Boolean matrix R_{ij}.
- Expand Q into a Boolean formula $\Phi_{Q,\boldsymbol{D}}$, called the lineage of Q on $\boldsymbol{D}$.
- Represent $\Phi_{Q,\boldsymbol{D}}$ using an AC^0 circuit.

Example: $\boxed{Q = \exists x\, (A(x) \wedge \forall y(B(y) \Rightarrow C(x, y)))}$

Its lineage is: $\boxed{\Phi_{Q,\boldsymbol{D}} = \bigvee_{i=1,n} \left(A_i \wedge \bigwedge_{j=1,n} (\neg B_j \vee C_{ij})\right)}$

Data Complexity FO is in AC^0: Proof

Fix a Boolean query Q in FO. Encode the input database $\boldsymbol{D}$ using bits:

- Let $N = |\text{ADom}(\boldsymbol{D})|$.
- Encode a relation of arity k using N^k bits.
  E.g. $R(X, Y)$ encoded using Boolean matrix R_{ij}.
- Expand Q into a Boolean formula $\Phi_{Q,\boldsymbol{D}}$, called the lineage of Q on $\boldsymbol{D}$.
- Represent $\Phi_{Q,\boldsymbol{D}}$ using an AC^0 circuit.

Example: $\boxed{Q = \exists x\,(A(x) \wedge \forall y(B(y) \Rightarrow C(x, y)))}$

Its lineage is: $\boxed{\Phi_{Q,\boldsymbol{D}} = \bigvee_{i=1,n}\left(A_i \wedge \bigwedge_{j=1,n}(\neg B_j \vee C_{ij})\right)}$

In class: construct a circuit of depth 5 and size $O(n^2)$.

Summary

- Data complexity is in AC^0; this implies LOGSPACE, PTIME.
- Expression complexity, combined complexity: PSPACE complete
  We will discuss this later.
- AC^0 is the class of highly parallelizable problems.
- "SQL is embarrassingly parallel"

Restricted Query Languages

Motivation

- FO is too rich for powerful optimizations: Trakhtenbrot's theorem is a fundamental limit.
- For fragments of FO static analysis is possible, and they still capture the most important queries in practice.
- Assuming FO consists of $\exists, \forall, \wedge, \vee, \neg, =$, we will obtain fragments by restricting the connectives.

Conjunctive Queries

Definition

A Conjunctive Query (CQ) is an expression of the form:

$$\boxed{Q(\boldsymbol{x}_0) = \exists \boldsymbol{y} \bigwedge_i R_i(\boldsymbol{x}_i)}$$

Conjunctive Queries

Definition

A Conjunctive Query (CQ) is an expression of the form:

$$\boxed{Q(\boldsymbol{x}_0) = \exists \boldsymbol{y} \bigwedge_i R_i(\boldsymbol{x}_i)}$$

- E.g. $Q(x, y) = \exists z(E(x, z) \wedge E(z, y))$.
- Equivalently: a CQ is an FO formula restricted to $=, \wedge, \exists$
- CQ has the same expressive power as RA restricted to $\sigma, \Pi, \bowtie$.
- These correspond to SELECT-FROM-WHERE queries in SQL (but we have to be careful what we allow in each clause).

Unions of Conjunctive Queries

Definition

A Union of Conjunctive Queries (UCQ) is a formula of the form:

$$\boxed{Q(\boldsymbol{x}) = \bigvee_i Q_i(\boldsymbol{x})}$$

where all Q_i's are CQs, and have the same sets of free variables.

Unions of Conjunctive Queries

Definition

A Union of Conjunctive Queries (UCQ) is a formula of the form:

$$\boxed{Q(\boldsymbol{x}) = \bigvee_i Q_i(\boldsymbol{x})}$$

where all Q_i's are CQs, and have the same sets of free variables.

- E.g. $Q(x, y) = E(x, y) \bigvee \exists z(E(x, z) \wedge E(z, y))$.
- Equivalently, UCQs are FO formulas restricted to $=, \wedge, \exists, \vee$.
- UCQ has the same expressive power as RA restricted to $\sigma, \Pi, \bowtie, \cup$.

Monotone Queries

Given two databases $\boldsymbol{D}, \boldsymbol{D}'$ over the same schema, we write $\boldsymbol{D} \subseteq \boldsymbol{D}'$ if $R_i^D \subseteq R_i^{D'}$ for every relation R_i in the schema.

Definition

A query Q is monotone if $\boldsymbol{D} \subseteq \boldsymbol{D}'$ implies $Q(\boldsymbol{D}) \subseteq Q(\boldsymbol{D}')$.

All UCQ queries are monotone. Exercise

The only non-monotone operators are:

- negation $\neg$ in FO.
- difference $-$ in RA.

Other Ways to Restrict the Query Language (1/2)

Adding $\neq, <, \leq$ to CQ, UCQ:

- By default they are not allowed in CQ, UCQ.
- If we want them, we write $CQ^{\neq}$ or $UCQ^{\leq}$.
- $Q(x, y) = \exists u \exists v \, (E(x, u) \wedge E(u, v) \wedge E(v, y) \wedge x < u < v < y)$.

Other Ways to Restrict the Query Language (1/2)

Adding $\neq, <, \leq$ to CQ, UCQ:

- By default they are not allowed in CQ, UCQ.
- If we want them, we write $\text{CQ}^{\neq}$ or $\text{UCQ}^{\leq}$.
- $Q(x, y) = \exists u \exists v\, (E(x, u) \wedge E(u, v) \wedge E(v, y) \wedge x < u < v < y)$.
  Is this query monotone?

Other Ways to Restrict the Query Language (1/2)

Adding $\neq, <, \leq$ to CQ, UCQ:

- By default they are not allowed in CQ, UCQ.
- If we want them, we write $\mathrm{CQ}^{\neq}$ or $\mathrm{UCQ}^{\leq}$.
- $Q(x, y) = \exists u \exists v\, (E(x, u) \wedge E(u, v) \wedge E(v, y) \wedge x < u < v < y)$.
  Is this query monotone?

  YES!

Other Ways to Restrict the Query Language (2/2)

Restricting the number of variables in FO:

- FO^k: restricted to using only k variables.
- E.g. check in FO^2 if there a path of length ≥ 5:
  $\exists x \exists y(E(x, y) \wedge \exists x(E(y, x) \wedge \exists y(E(x, y) \wedge \exists x(E(y, x)))))$

Other Ways to Restrict the Query Language (2/2)

Restricting the number of variables in FO:

- FO^k: restricted to using only k variables.
- E.g. check in FO^2 if there a path of length ≥ 5:
  $\exists x \exists y(E(x,y) \wedge \exists x(E(y,x) \wedge \exists y(E(x,y) \wedge \exists x(E(y,x)))))$

Theorem ([Grädel et al., 1997])

If $\varphi \in FO^2$ has any model (possibly infinite), then it has a model of size at most exponential in $|\varphi|$. Thus, $\mathtt{SAT}_{fin}(FO^2)$ is decidable.

Other Ways to Restrict the Query Language (2/2)

Restricting the number of variables in FO:

- FO^k: restricted to using only k variables.
- E.g. check in FO^2 if there a path of length ≥ 5:
  $\exists x \exists y(E(x, y) \wedge \exists x(E(y, x) \wedge \exists y(E(x, y) \wedge \exists x(E(y, x)))))$

Theorem ([Grädel et al., 1997])

If $\varphi \in FO^2$ has any model (possibly infinite), then it has a model of size at most exponential in $|\varphi|$. Thus, $\mathtt{SAT}_{fin}(FO^2)$ is decidable.

Suggested research: what are the implications for a query optimizer?

Other Ways to Restrict the Query Language (2/2)

Restricting the number of variables in FO:

- FO^k: restricted to using only k variables.
- E.g. check in FO^2 if there a path of length ≥ 5:
  $\exists x \exists y(E(x,y) \wedge \exists x(E(y,x) \wedge \exists y(E(x,y) \wedge \exists x(E(y,x)))))$

Theorem ([Grädel et al., 1997])

If $\varphi \in FO^2$ has any model (possibly infinite), then it has a model of size at most exponential in $|\varphi|$. Thus, $\mathtt{SAT}_{fin}(FO^2)$ is decidable.

Suggested research: what are the implications for a query optimizer?

What about FO^3?

Other Ways to Restrict the Query Language (2/2)

Restricting the number of variables in FO:

- FO^k: restricted to using only k variables.
- E.g. check in FO^2 if there a path of length ≥ 5:
  $\exists x \exists y (E(x,y) \wedge \exists x (E(y,x) \wedge \exists y (E(x,y) \wedge \exists x (E(y,x)))))$

Theorem ([Grädel et al., 1997])

If $\varphi \in FO^2$ has any model (possibly infinite), then it has a model of size at most exponential in $|\varphi|$. Thus, $\mathtt{SAT}_{fin}(FO^2)$ is decidable.

Suggested research: what are the implications for a query optimizer?

What about FO^3?

To watch how many variables we need to prove Trakhtenbrot's theorem

Conjunctive Queries

Are the most important and most studied fragment. Terminology:

- Boolean query: no head vars: $Q() = \exists x \exists y \exists z (E(x, y) \wedge E(y, z))$.
- Full query: no existential vars: $Q(x, y, z) = E(x, y) \wedge E(y, z)$.
- Without selfjoins: every relation name occurs at most once.

$$Q(x) = \exists y \exists z (R(x, y) \wedge S(y, z) \wedge T(z, x)).$$

- We often omit the existential quantifiers, and write for example:

$$Q(x) = R(x, y) \wedge S(y, z) \wedge T(z, x).$$

Summary

- Most of our discussion will be focused on CQ's.
- UCQs come almost for free, or with very little additional effort.
- Let's re-examine query evaluation when the query is restricted to a CQ.

Motivation

We already know that the data complexity is in AC^0.

What is the expression complexity? The combined complexity?

Will answer both, and also discuss the expression/combined complexity for FO (which we left out).

Importantly: we will define query evaluation for CQ in terms of Homomorphisms

Equivalent Concepts

- A Conjunctive Query:
$R(x, y, z) \wedge S(x, u) \wedge S(y, v) \wedge S(z, w) \wedge R(u, v, w)$
- A database instance:

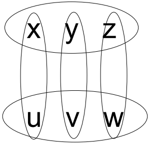
$R(A, B, C) =$

A	B	C
x	y	z
u	v	w

$S(D, E) =$

D	E
x	u
y	v
z	w

- A labeled hypergraph, $G = (V, E)$, where
$V = \{x, y, z, u, v, w\}$, $E = \{\{x, y, z\}, \{u, v, w\}, \{x, u\}, \{y, v\}, \{z, w\}\}$
(hyperedges are labeled with R, S respectively).



Equivalent Concepts

- A Conjunctive Query:
  $R(x, y, z) \wedge S(x, u) \wedge S(y, v) \wedge S(z, w) \wedge R(u, v, w)$
- A database instance:

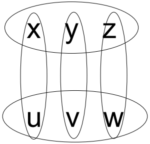
$R(A, B, C) =$

A	B	C
x	y	z
u	v	w

$S(D, E) =$

D	E
x	u
y	v
z	w

- A labeled hypergraph, $G = (V, E)$, where
  $V = \{x, y, z, u, v, w\}$, $E = \{\{x, y, z\}, \{u, v, w\}, \{x, u\}, \{y, v\}, \{z, w\}\}$
  (hyperedges are labeled with R, S respectively).



We will often switch back-and-forth between these equivalent notions

Homomorphisms

$$Q(\boldsymbol{x}_0) = R_1(\boldsymbol{x}_1) \wedge \cdots \wedge R_m(\boldsymbol{x}_m),\ Q'(\boldsymbol{y}_0) = S_1(\boldsymbol{y}_1) \wedge \cdots \wedge S_n(\boldsymbol{y}_n).$$

Definition

A homomorphism $h : Q' \to Q$ is a function
$h : \mathtt{Const}(Q') \cup \mathtt{Vars}(Q') \to \mathtt{Const}(Q) \cup \mathtt{Vars}(Q)$ s.t.:

- $\forall c \in \mathtt{Const}(Q'),\ h(c) = c$.
- $S_j(\boldsymbol{y}_j) \in \mathtt{Atoms}(Q'),\ \exists R_i(\boldsymbol{x}_i) \in \mathtt{Atoms}(Q)$ such that $R_i = S_j$ (the are the same relation name) and $h(\boldsymbol{y}_j) = \boldsymbol{x}_i$.
- h maps head vars to head vars: $h(\boldsymbol{y}_0) = \boldsymbol{x}_0$.

Homomorphisms

$$Q(\boldsymbol{x}_0) = R_1(\boldsymbol{x}_1) \wedge \cdots \wedge R_m(\boldsymbol{x}_m), \; Q'(\boldsymbol{y}_0) = S_1(\boldsymbol{y}_1) \wedge \cdots \wedge S_n(\boldsymbol{y}_n).$$

Definition

A homomorphism $h : Q' \to Q$ is a function
$h : \texttt{Const}(Q') \cup \texttt{Vars}(Q') \to \texttt{Const}(Q) \cup \texttt{Vars}(Q)$ s.t.:

- $\forall c \in \texttt{Const}(Q'), \; h(c) = c$.
- $S_j(\boldsymbol{y}_j) \in \texttt{Atoms}(Q'), \exists R_i(\boldsymbol{x}_i) \in \texttt{Atoms}(Q)$ such that $R_i = S_j$ (the are the same relation name) and $h(\boldsymbol{y}_j) = \boldsymbol{x}_i$.
- h maps head vars to head vars: $h(\boldsymbol{y}_0) = \boldsymbol{x}_0$.

E.g. graph homomorphism $h : G' \to G$ is $h : V \to V'$ s.t. $\forall e \in E'$, $h(e) \in E$.

Query Evaluation for CQ and Homomorphisms

Computing $Q(\boldsymbol{D})$ consists of finding all homomorphisms $h : Q \to D$ and returning $h(\texttt{Head}(Q))$.

$Q(x) = R(x) \wedge S(x, y) \wedge T(y, 'a')$

$R =$

x
1
2

$S =$

x	y
1	10
1	20
2	20

$T =$

y	z
10	a
10	b
20	a

We list all homomorphisms:

$h =$

$x(= \texttt{Head}(Q))$	y	a
1	10	a
1	20	a
2	20	a

Final answer after duplicate elimination: $Q(\boldsymbol{D}) = \{1, 2\}$.

The Combined Complexity for UCQ is in NP

Theorem

The combined complexity for UCQ is in NP.

Proof: Fix a UCQ $Q = Q_1 \vee Q_2 \vee \cdots$ and a database $\boldsymbol{D}$.

To check $\boldsymbol{D} \models Q$:

- "guess" a CQ Q_i, and
- "guess" a homomorphism $h : Q_i \to \boldsymbol{D}$

The Expression Complexity for CQ is NP-hard

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of CQ queries is NP complete.

Thus, the expression complexity is also NP-complete.

The Expression Complexity for CQ is NP-hard

Theorem

There exists a database $\boldsymbol{D}$ *for which the expression complexity of CQ queries is NP complete.*

Thus, the expression complexity is also NP-complete.

Proof Many proofs are possible (will explain shortly why). We will use reduction from 3SAT, because we will reuse it a few times.

The Expression Complexity for CQ is NP-hard

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of CQ queries is NP complete.

Thus, the expression complexity is also NP-complete.

Proof Many proofs are possible (will explain shortly why). We will use reduction from 3SAT, because we will reuse it a few times.

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

The Expression Complexity for CQ is NP-hard

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of CQ queries is NP complete.

Thus, the expression complexity is also NP-complete.

Proof Many proofs are possible (will explain shortly why). We will use reduction from 3SAT, because we will reuse it a few times.

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Notice that D is independent of Φ.

Details next.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \rightarrow \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q contains $A(x_i, x_j, x_k)$.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \rightarrow \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q contains $B(x_i, x_j, x_k)$.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q contains $D(x_i, x_j, x_k)$.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \rightarrow \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q contains $D(x_i, x_j, x_k)$.

$\boldsymbol{D}$ has 4 tables with 7 tuples each which tuple is missing?

$A =$

0	0	1
	⋮	
1	1	1

$B =$

0	0	0
0	1	0
	⋮	
1	1	1

$C = \dots$ $D = \dots$

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \rightarrow \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q contains $D(x_i, x_j, x_k)$.

$\boldsymbol{D}$ has 4 tables with 7 tuples each which tuple is missing?

$$A = \begin{array}{|c|c|c|} 0 & 0 & 1 \\ & \vdots & \\ 1 & 1 & 1 \\ \end{array} \qquad B = \begin{array}{|c|c|c|} 0 & 0 & 0 \\ 0 & 1 & 0 \\ & \vdots & \\ 1 & 1 & 1 \\ \end{array} \qquad C = \ldots \qquad D = \ldots$$

In class: Φ is satisfiable iff $\exists h : Q \rightarrow \boldsymbol{D}$.

Combined Complexity for FO

Recall that the combined complexity of FO is in PSPACE.

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of FO queries is PSPACE complete.

Thus, the combined complexity is also PSPACE-complete.

Combined Complexity for FO

Recall that the combined complexity of FO is in PSPACE.

Theorem

There exists a database $\boldsymbol{D}$ *for which the expression complexity of FO queries is PSPACE complete.*

Thus, the combined complexity is also PSPACE-complete.

Proof: Reduction from the Quantified Boolean Formula Satfiability:

$$\boxed{Q_1 X_1 \; Q_2 X_2 \; \cdots Q_n X_n \; \Phi}$$

where Φ is 3CNF.

Use the same $Q_\Phi, \boldsymbol{D}$ before, but add appropriate quantifiers to Q_Φ:

$$\boxed{Qx_1 \; Qx_2 \; \cdots Q_n x_n \; Q_\Phi(x_1, \ldots, x_n)}$$

Discussion: CQ and CSP

The generalized Constraint Satisfaction Problem is:

Definition ([Kolaitis and Vardi, 1998])

Given two classes of finite structures $\mathcal{A}, \mathcal{B}$, the $CSP(\mathcal{A}, \mathcal{B})$ problem is:
Given $A \in \mathcal{A}, B \in \mathcal{B}$, is there a homomorphism $h : A \to B$?

Discussion: CQ and CSP

The generalized Constraint Satisfaction Problem is:

Definition ([Kolaitis and Vardi, 1998])

Given two classes of finite structures $\mathcal{A}, \mathcal{B}$, the $CSP(\mathcal{A}, \mathcal{B})$ problem is:
Given $A \in \mathcal{A}, B \in \mathcal{B}$, is there a homomorphism $h : A \to B$?

Standard CSP restricts the right-hand side, $CSP(-, B)$.
What is B for 3SAT? For 3-colorability? For Hamiltonean path?

Discussion: CQ and CSP

The generalized Constraint Satisfaction Problem is:

Definition ([Kolaitis and Vardi, 1998])

Given two classes of finite structures $\mathcal{A}, \mathcal{B}$, the $CSP(\mathcal{A}, \mathcal{B})$ problem is:
Given $A \in \mathcal{A}, B \in \mathcal{B}$, is there a homomorphism $h : A \to B$?

Standard CSP restricts the right-hand side, $CSP(-, B)$.
What is B for 3SAT? For 3-colorability? For Hamiltonean path?

Query evaluation restricts the left-hand side, $CSP(Q, -)$

"Query evaluation is CSP from the other side."

Summary

- Evaluating $Q(\boldsymbol{D})$ consists of finding homomorphisms $h : Q \rightarrow \boldsymbol{D}$.
- This problem is in NP, in fact it is the very definition of NP.
- If Q is fixed, then the problem is in PTIME in $|\boldsymbol{D}|$. Data complexity
- If Q is part of the input (i.e. can be huge) then NP-complete. Expression complexity

Abiteboul, S., Hull, R., and Vianu, V. (1995).
Foundations of Databases.
Addison-Wesley.

Arenas, M., Barceló, P., Libkin, L., Martens, W., and Pieris, A. (2022).
Database Theory.
Open source at https://github.com/pdm-book/community.
See also
https://www.theoinf.uni-bayreuth.de/en/news/2021/Book-on-Database-Theory-in-the-works/index.html.

Bailis, P., Hellerstein, J. M., and Stonebraker, M., editors (2015).
Readings in Database Systems, 5th Edition.

Börger, E., Grädel, E., and Gurevich, Y. (1997).
The Classical Decision Problem.
Perspectives in Mathematical Logic. Springer.

Grädel, E., Kolaitis, P. G., and Vardi, M. Y. (1997).
On the decision problem for two-variable first-order logic.
Bull. Symb. Log., 3(1):53–69.

Kolaitis, P. G. and Vardi, M. Y. (1998).
Conjunctive-query containment and constraint satisfaction.
In Mendelzon, A. O. and Paredaens, J., editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 205–213. ACM Press.

Libkin, L. (2004).
Elements of Finite Model Theory.
Texts in Theoretical Computer Science. An EATCS Series. Springer.

Libkin, L. (2009).
The finite model theory toolbox of a database theoretician.

In Paredaens, J. and Su, J., editors, *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pages 65–76. ACM.

Ullman, J. D. (1980).
Principles of Database Systems, 1st Edition.
Computer Science Press.

Vardi, M. Y. (1982).
The complexity of relational query languages (extended abstract).
In Lewis, H. R., Simons, B. B., Burkhard, W. A., and Landweber, L. H., editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM.