Lecture 9: Query Complexity

Tuesday, January 30, 2001

Outline

- Properties of queries
- Relational Algebra v.s. First Order Logic
- Classical Logic v.s. Logic on Finite Models
- Query Complexity
  - start today, finish Thursday
- Reading assignment:
  - Sections 1-3 from the paper

A Note on Notation

- Used to denote models $D = (D, R_1, ..., R_k)$
- New notation: $\mathbf{D} = (D, R_1, ..., R_k)$
  - model is in boldface, domain is in normal font

Properties of Queries

- Decidable
- Generic
- Domain-independent
- They make more sense if we think of queries in general, not just FO queries
- Define next general queries

Queries

- A ***query,*** q, is a function from models to relations, s.t. for every model $\mathbf{D} = (D, R_1, ..., R_k)$:
  - $q(\mathbf{D}) = R$, s.t. $R \subseteq D^n$
- Here n is called the *arity* of q; when n=0, q is called a *boolean* query

Property 1: Decidable Queries

- q is *decidable* if there exists a Turing Machine that, for some encoding of **D**, given $R_1, ..., R_k$ on its input tape, computes $q(\mathbf{D})$

Property 2: Domain Independence

- In English
  - q only depends on R_1, ..., R_k, not on D !
  - Intuition: a database consists only of R_1, ..., R_k, not on D.
- Formally: a query q is *domain independent* if
  - for any model (D, R_1, ..., R_k)
  - for any set D' s.t. $R_1 \subseteq (D')^{ar(R1)}$, ..., $R_k \subseteq (D')^{ar(Rk)}$
  - the following holds
    - $q(D , R_1, ..., R_k) = q(D', R_1, ..., R_k)$

Property 2: Domain Independence

Examples:

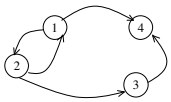
- Queries that are domain independent:
  - "Find pairs of nodes connected by a path of length 2"
  - "Find the manager of Smith"
  - "Find the largest salary in the database"
- Queries that are not domain independent:
  - "Find all nodes that are not in the graph"
  - "Find the average salary"

Property 3: Genericity

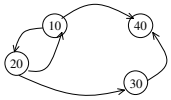
- In English:
  - q does not depend on the particular encoding of the database
- Formally:
  - for every $h:(D,R_1, ...,R_k) \rightarrow (D',R'_1, ...,R'_k)$
  - s.t. h=bijective, $h(D) = D'$, $h(R_1)=R'_1,..., h(R_k)=R'_k$
  - It follows: $h(q(D ,R_1, ...,R_k)) = q(D',R'_1, ...,R'_k)$

Property 3: Genericity

Example:

D = 

q(D)={1,3}

D'= 

q(D')= ??

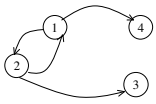
Property 3: Genericity

Examples:

- Queries that are generic:
  - "Find pairs of nodes connected by a path of length 2"
  - "Find all employees having the same office as their manager"
  - "Find all nodes that are not in the graph"
- Queries that are not generic:
  - "Find the manager of Smith"
    - we often relax the definition to allow this to be generic
    - *C-genericity*, for a set of constants C
  - "Find the largest salary in the database"

Property 3: Genericity

More example:

D = 

q(D)={4}

This query cannot be generic (why ?)

Back to FO Queries

1. All FO queries are computable
2. NOT All FO queries are domain independent
   - Why ? Next...
3. All FO queries are generic
   - In particular query on previous slide not expressible in FO

FO Queries and Domain Independence

- Find all nodes that are not in the graph:

$$q(x) = \neg\exists y.R(x, y) \wedge \neg\exists z.R(z, x)$$

- Find all nodes that are connected to "everything":

$$q(x) = \forall y.R(x, y)$$

- Find all pairs of employees or offices:

$$q(x, y) = Emp(x) \vee Office(y)$$

- We don't want such queries !

FO Queries and Domain Independence

- Domain independent FO queries are also called *safe* queries
- Definition. The ***active domain*** of $(D, R_1, ..., R_k)$ is D_a = the set of all constants in $R_1, ..., R_k$
- E.g. for graphs, $D_a = \{x \mid \exists y.R(x, y) \vee \exists z.R(z, x)\}$
- Very important:
  - If a query is safe, it suffices to range quantifiers only over the active domain (why ?)

FO Queries and Domain Independence

- The bad news:
  - **Theorem** It is undecidable if a given a FO query is safe.
- The good news:
  - no big deal
  - can define a subset of FO queries that we *know* are safe = range restricted queries (rr-query)
  - Any safe query is equivalent to some rr-query

Range-restriction

- Syntactic, rather ad-hoc definition (several exists):
- $S(x) \wedge \neg R(x, x)$ OK, $\neg R(x, x)$ not OK
- $\exists y.(S(y) \wedge \neg R(y, y))$ OK, $\exists y.(\neg R(y, y))$ not OK
- $\forall y.(S(y) \Rightarrow \neg R(y, y))$ OK, $\forall y.(\neg R(y, y))$ not OK

- If a query q is safe, it is equivalent to a rr-query:

$\neg R(x, y,...)$ to $(x \in D_a \wedge y \in D_a \wedge ... \wedge \neg R(x, y,...))$

$\exists y.\varphi$ to $\exists y.(y \in D_a \wedge \varphi)$ $\forall y.\varphi$ to $\forall y.(y \in D_a \Rightarrow \varphi)$

Safe-FO = Relational Algebra

- Recall the 5 operators in the relational algebra:
  - U, -, x, σ, Π
- **Theorem**. A query is expressible in safe-FO iff it is expressible in the relational algebra

Proof

RA query E ➔ safe FO query ϕ

$R \Rightarrow R(x_1,...,x_n)$

$E \cup E' \Rightarrow \varphi(x_1,...,x_n) \vee \varphi'(x_1,...,x_n)$

$E - E' \Rightarrow \varphi(x_1,...,x_n) \wedge \neg\varphi'(x_1,...,x_n)$

$E \times E' \Rightarrow \varphi(x_1,...,x_n) \wedge \varphi'(y_1,...,y_m)$

$\sigma_{x=a}(E) \Rightarrow \varphi(...) \wedge (x = a)$

$\Pi_{x_1,...,x_n}(E) \Rightarrow \exists x_{n+1},...,\exists x_m.\varphi(x_1,...,x_n, x_{n+1},...,x_m)$

Proof

Define: Active domain formula:

$D_a = (\Pi_1(R) \cup \Pi_2(R) \cup ...) \cup (\Pi_1(S) \cup \Pi_2(S) \cup ...) \cup ...$

safe FO query ϕ ➔ RA query E

$R(x_1,...,x_n) \Rightarrow R$

$\varphi(x_1,...,x_n, y_1,...,y_m) \wedge \varphi'(x_1,...,x_n, z_1,...,z_p) \Rightarrow E \bowtie E'$

$\varphi(x_1,...,x_n, y_1,...,y_m) \vee \varphi'(x_1,...,x_n, z_1,...,z_p)$

$\Rightarrow E \times (D_a)^m \cup E' \times (D_a)^p$

$\neg\varphi(x_1,...,x_n) \Rightarrow (D_a)^n - E$

$\exists x_1.\varphi(x_1,...,x_n) \Rightarrow \Pi_{2,3,...,n}(E)$

No need for $\forall$ (why ?)

Examples

- Vocabulary (= schema):
  - Employee(name, office, mgr), Manager(name, office)
- Find offices:

$q_1(x) = \exists y.\exists z.(E(y, x, z)) \Rightarrow \Pi_2(\Pi_{1,2}(E)) = \Pi_2(E)$

Factoid: existential quantifiers ARE projections, and vice versa

Examples (cont'd)

- Find the manager of all employees:

$q_2(x) = (\exists y.M(x, y)) \wedge \forall u.(\exists v.\exists w.E(u, v, w) \rightarrow \exists v.E(u, v, x))$

Discussion

- (safe)-FO and RA:
  - (safe)-FO: for *declarative query.*
  - RA: for *query plan.*
  - Theorem says: translate (safe)-FO to RA
  - In practice: need to consider "best" RA
- Query languages
  - (safe)-FO is just one instance; will discuss smaller and larger languages
  - All will express only computable, generic, and domain independent queries

Classical Logic v.s. Logic on Finite Models

- Recall:
  - given a model $\mathbf{D}=(D,R_1,...,R_k)$
  - and given a closed FO formula ϕ
  - we have defined what $\mathbf{D} \models \phi$ means
- A formula is *valid* if, for every $\mathbf{D}$, $\mathbf{D} \models \phi$
  - It is *finitely valid* if for every finite $\mathbf{D}$, $\mathbf{D} \models \phi$
- A formula is *satisfiable* if there exists $\mathbf{D}$ s.t. $\mathbf{D} \models \phi$
  - It is *finitely satisfiable* if there exists a finite $\mathbf{D}$ s.t. $\mathbf{D} \models \phi$
- Obviously: ϕ is valid iff not(ϕ) is not satisfiable

Classical Logic

- Notation: $\models \phi$ means ϕ is valid
- Notation: $\vdash \phi$ means ϕ is "provable"

Godel's Completeness Theorem: $\models \phi$ iff $\vdash \phi$

Corollary. The set of valid formulas is r.e.
  - Idea: enumerate all proofs

Church's Theorem: if $ar(R_i) > 1$ for some i, then the set of valid formulas is not decidable.

Corollary. The set of satisfiable formulas is not r.e.

Logic on Finite Models

Simple Fact: the set of finitely satisfiable formulas is r.e.
  - Idea: enumerate all finite models $\mathbf{D}$, and all formulas ϕ s.t. $\mathbf{D} \models \phi$

Trakhtenbrot's Theorem: if $ar(R_i) > 1$ for some i, then the set of finitely satisfiable formulas is not decidable

Corollary: the set of finitely valid formulas is not r.e.

An Example Where Finite/Infinite Differ

A formula ϕ that is satisfiable but not finitely satisfiable
  - "$<$ is a total order and has no maximal element"

$$\varphi = \forall x.\neg(x < x) \wedge$$
$$\forall x.\forall y.(x < y \vee y < x) \wedge \neg(x < y \wedge y < x) \wedge$$
$$\forall x.\forall y.\forall z.(x < y \wedge y < z) \rightarrow (x < z) \wedge$$
$$\forall x.\exists y.x < y$$

- It has an infinite model, but no finite one

Applications of Trakhtenbrot's Theorem

- Given a FO query ϕ, it is undecidable if ϕ is safe
  - Proof: the query $\neg R(x) \wedge \varphi$ is unsafe iff ϕ is finitely satisfiable
- Given two FO queries ϕ, ϕ', it is undecidable if they are equivalent, i.e. $\phi \Leftrightarrow \phi$'
  - Proof the queries $R(x) \wedge \varphi$ and $\neg R(x) \wedge \varphi$ are equivalent iff ϕ is not finitely satisfiable
- Trakhtenbrot's theorem for FO queries = like Rice's theorem for programs

More of That

- **Definition**. A query q is monotone if, for any two finite models
  $\mathbf{D} = (D, R_1, ..., R_k)$ and $\mathbf{D'} = (D', R_1', ..., R_k')$
  s.t. $D \subseteq D'$, $R_1 \subseteq R_1'$, ..., $R_k \subseteq R_k'$
  we have $q(\mathbf{D}) \subseteq q(\mathbf{D'})$.
- **Proposition**. It is undecidable if a query q in FO is monotone.
- **Proof**: why ?

Complexity of Query Languages

- All queries in a query language L are computable
- Converse false: usually L does not express *all* computable queries. Limited expressive power.
- Why do we care about such languages ?
  - Typically queries always terminate (e.g. FO)
  - Typically queries have a *low complexity* (next)

Complexity of Query Languages

For a query language L, define:

- **Data complexity**: fix a query q, how complex is it to evaluate q(**D**), for finite models **D**.
- **Expression complexity**: fix a finite model **D**, how complex is it to evaluate q(**D**), for queries q in L
- **Combined complexity**: how complex is it to evaluate q(**D**), for finite models **D** and queries q in L

Complexity of Query Languages

Formally:

- **Data complexity** of L is the complexity of deciding the set:

$$S = \{(D, \bar{a}) \mid D = \text{finite model, and } \bar{a} \in q(D)\}$$

  for some q in L
- **Combined complexity** of L is the complexity of deciding the set:

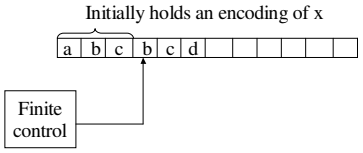
$$S = \{(D, \bar{a}) \mid D = \text{finite model, and } \bar{a} \in q(D),\ q \in L\}$$

Who Cares About What

- Users: care about data complexity:
  - the query q is fixed; the database **D** is variable
- Database Systems: care about combined complexity:
  - both the query q and the database **D** are variable
- Database Theoreticians:
  - care about expression complexity, when they need to publish more papers ☺

Crash Course in Complexity Classes

- Fix a problem, i.e. a set S. Given a value x, how difficult is it for a Turing Machine to decide whether x ∈ S



- Let n = |x|
- **Definition**. S is in PTIME if there exists a Turing machine that on every input x takes $n^{O(1)}$ steps (i.e. $O(n^k)$, for some $k > 0$).
- **Definition**. S is in PTIME if there exists a Turing machine for S that on every input x takes $n^{O(1)}$ space. Note: may take A LOT of time.
- **Definition**. S is LOGSPACE if there exists a Turing machine for S that on every input takes O(log n) space. OOPS !?!