Logic, Data, and Incomplete Information

Phokion G. Kolaitis

UC Santa Cruz & IBM Research

Lecture 2

The Relational Data Model (E.F. Codd – 1970)

- The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.
- Question: What is a relation?
- Answer:
  - Formally, a relation is a subset of a cartesian product of sets
    $R \subseteq B_1 \times B_2 \times \ldots \times B_k$.
  - Informally, a relation is a "table" with rows and columns.

CHECKING Table

branch-name	**account-no**	**customer-name**	**balance**
Orsay	10991-06284	Abiteboul	$13,567.53
Hawthorne	10992-35671	Hull	$21,245.75
…	…	…	…

Database Queries

Definition: Let **S** be a relational database schema and let k ≥1.

A k-ary query on **S** is a function q defined on databases over **S** such that if D is a database over **S**, then q(D) is a k-ary relation with entries from the relations in D that is invariant under isomorphisms

Definition: Let **S** be a relational database schema.

A Boolean query on **S** is a function q defined on database instances over S such that if D is a database over S, then q(D) = 1 or q(D) = 0, and q(D) is invariant under isomorphisms.

Example 1: TEACHES(instructor, course), ENROLLS(student,course)

- Find all students enrolled in some course taught by Miller.

Example 2: **CHECKING**(branch-name, account-no, customer-name, balance)

- Is there a customer at the San Jose branch whose account has a balance of more than 1 Million dollars?

Query Languages for the Relational Data Model

Codd introduced two different query languages for the relational data model:

- Relational Algebra, which is a procedural language.
  - It is an algebraic formalism in which queries are expressed by applying a sequence of operations to relations.

- Relational Calculus, which is a declarative language.
  - It is a logical formalism in which queries are expressed as formulas of first-order logic.

Codd's Theorem: Relational Algebra and Relational Calculus are "essentially equivalent" in terms of expressive power.

The Five Basic Operations of Relational Algebra

- Group I: Three standard set-theoretic binary operations:
  - Union
  - Difference
  - Cartesian Product.

- Group II. Two special unary operations on relations:
  - Projection
  - Selection.

- Relational Algebra consists of all expressions obtained by combining these five basic operations in syntactically correct ways.

The Projection Operation - Example

SAVINGS

branch-name	acc-no	cust-name	balance
Aptos	153125	Vianu	3,450
Santa Cruz	123658	Hull	2,817
San Jose	321456	Codd	9,234
San Jose	334789	Codd	875

$\pi_{\text{cust-name,branch-name}}(\text{SAVINGS})$

cust-name	branch-name
Vianu	Aptos
Hull	Santa Cruz
Codd	San Jose

The Selection Operation

Example:

CHECKING(branch-name, account-no, customer-name, balance)

- Find the records at the Hawthorne branch in which the balance is less than $10000

- σ (branch-name='Hawthorne') $\wedge$ (balance < 10000) (**CHECKING**)

Relational Calculus (FO Logic for Databases)

- First-order variables: x, y, z, …, x_1, …,x_k,…
  - They range over values that may occur in tables.
- Relation symbols: R, S, T, … of specified arities (names of relations)
- Atomic (Basic) Formulas:
  - $R(x_1,\ldots,x_k)$, where R is a k-ary relation symbol
    (alternatively, $(x_1,\ldots,x_k) \in R$; the variables need not be distinct)
  - (x op y), where op is one of =, ≠, <, >, ≤, ≥
  - (x op c), where c is a constant and op is one of =, ≠, <, >, ≤, ≥.
- Relational Calculus Formulas:
  - Every atomic formula is a relational calculus formula.
  - If φ and ψ are relational calculus formulas, then so are:
    - $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\neg \psi$, $(\varphi \rightarrow \psi)$ (propositional connectives)
    - $(\exists x\ \varphi)$ (existential quantification)
    - $(\forall x\ \varphi)$ (universal quantification).

Relational Calculus as a Query Language

Definition:

- A relational calculus expression is an expression of the form

  $$\{ (x_1,\ldots,x_k) : \varphi(x_1,\ldots x_k) \},$$

  where $\varphi(x_1,\ldots,x_k)$ is a relational calculus formula with $x_1,\ldots,x_k$ as its free variables.

- When applied to a database D, this relational calculus expression returns the k-ary relation that consists of all k-tuples $(a_1,\ldots,a_k)$ that make the formula "true" on D.

Fact:

- Every relational calculus expression $\{ (x_1,\ldots,x_k) : \varphi(x_1,\ldots x_k) \}$ defines a k-ary query.

Relational Calculus as a Query Language

Example 1: TEACHES(instructor, course), ENROLLS(student,course)

- Find all students enrolled in some course taught by Miller.

  {s: ∃ t ∃ c (TEACHES(t, c) ∧ t = 'Miller' ∧ ENROLLS(s,c)) }

- Find all students who are enrolled in every course taught by Miller.

  {s: ∀ t ∀ c (TEACHES(t, c) ∧ t = 'Miller' → ENROLLS(s,c)) }

Example 2: FLIGHT(from, to)

- Find all pairs of cities such that one can fly to the other with at most one stop.

  {(x,y): FLIGHT(x,y) ∨ ∃ z (FLIGHT(x,z) ∧ FLIGHT(z,y)) }

Active Domain and Active Domain Interpretation

Definition:

- The active domain adom(D) of a relational database instance D is the set of all values that occur in the relations of D.

- Let $\varphi(x_1,\ldots,x_k)$ be a relational calculus formula and let D be a relational database instance. Then

$$\varphi^{adom}(D)$$

is the result of evaluating $\varphi(x_1,\ldots,x_k)$ over adom(D) and D, i.e.,

  - all variables and quantifiers are assumed to range over adom(D);
  - the relation symbols in φ are interpreted by the relations in D.

Equivalence of Relational Algebra and Calculus

Theorem: If q is a k-ary query, then the following statements are equivalent:

1. There is a relational algebra expression E such that
   q(D) = E(D), for every database instance D
   (in other words, q is expressible in relational algebra).

2. There is a relational calculus formula ψ such that
   q(D) = ψ^{adom} (D)
   (in other words, q is expressible in relational calculus under the active domain interpretation).

From Relational Algebra to Relational Calculus

Example: R(A,B), S(C,D)

Translate $\pi_{1,4}(\sigma_{R.B=S.C}(R \times S))$ to relational calculus

1. R translates to R(x,y)
2. S translates to S(z,w)
3. $R \times S$ translates to $R(x,y) \wedge S(z,w)$
4. $\sigma_{R.B=S.C}(R \times S)$ translates to $(y=z) \wedge R(x,y) \wedge S(z,w)$
5. $\pi_{1,4}(\sigma_{R.B=S.C}(R \times S))$ translates to

   $\exists y \exists z ((y=z) \wedge R(x,y) \wedge S(z,w))$

   or, simply, to

   $\exists y (R(x,y) \wedge S(y,w))$

From Relational Calculus to Relational Algebra

- In this translation, the most interesting part is the simulation of the universal quantifier $\forall$ in relational algebra.
  - It uses the logical equivalence $\forall y\psi \equiv \neg\exists y\neg\psi$
- As an illustration, consider $\forall y R(x,y)$.
  - $\forall y R(x,y) \equiv \neg\exists y\neg R(x,y)$
  - adom(D) = $\pi_1(R) \cup \pi_2(R)$

Rel.Calc. formula φ	Relational Algebra Expression for φ^{adom}
$\neg$ R(x,y)	$(\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R$
$\exists y \neg R(x,y)$	$\pi_1((\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R)$
$\neg\exists y \neg R(x,y)$	$(\pi_1(R) \cup \pi_2(R)) - (\pi_1((\pi_1(R) \cup \pi_2(R)) \times (\pi_1(R) \cup \pi_2(R)) - R))$

Conjunctive Queries

Definition:

- A conjunctive query (CQ) is a query expressible by a relational calculus formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\wedge$ and $\exists$ only.

$$\{ (x_1,\ldots,x_k): \exists z_1 \ldots \exists z_m \chi(x_1, \ldots,x_k, z_1,\ldots,z_m) \}$$

- A union of conjunctive queries (UCQ) is a query expressible by a disjunction of relational calculus formulas each expressing a conjunctive query.

Examples:

- Path of length 2: (CQ)

$$\{ (x,y): \exists z (E(x,z) \wedge E(z,y)) \}$$

- Path of length at most 3: (UCQ)

$$\{ (x,y): E(x,y) \vee \exists z (E(x,z) \wedge E(z,y)) \vee \exists z \exists w(E(x,z) \wedge E(z,w) \wedge E(w,y)) \}$$

Conjunctive Queries

Example: TEACHES(instructor, course), ENROLLS(student, course)

TAUGHT-BY(student, instructor) is a conjunctive query

$$\{ (s,t) : \exists c \, (\text{TEACHES}(t,c) \wedge \text{ENROLLS}(s,c)) \}$$

Example: Every relational join is a quantifier-free conjunctive query:

P(A,B,C), R(B,C,D) two relation symbol

Relational Join $P \bowtie R = \{ (x,y,z,w): P(x,y,z) \wedge R(y,z,w) \}$

Example: Is there a triangle? (Boolean conjunctive query)

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

Note: In general, every Boolean conjunctive query express the existence of a "pattern".

A Fundamental Algorithmic Problem

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

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

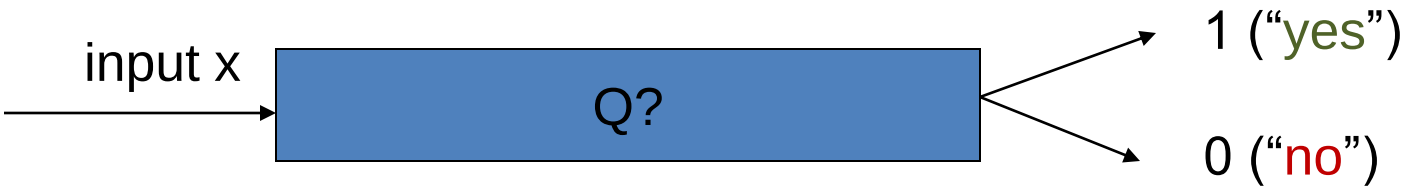
- The Boolean Query Evaluation Problem:
  Given a Boolean query q and a database D, find q(D).

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

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

Decision Problems and Languages

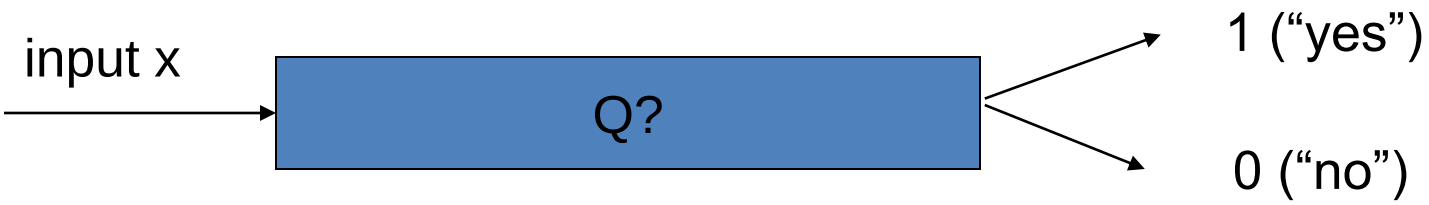
- Definition (informal): A decision problem Q consists of a set of inputs and a question with a "yes" or "no" answer for each input.



- Definition:
  - Σ^* is the set of all strings over a finite alphabet Σ.
  - A language over Σ is a set $L \subseteq \Sigma^*$
  - Every language L gives rise to the following decision problem:
    - Given $x \in \Sigma^*$, is $x \in L$?
  - Conversely, every decision problem can be thought of as arising from a language, namely,
    the language consisting of all inputs with a "yes" answer.

Decision Problems

- Definition (informal): A decision problem Q consists of a set of inputs and a question with a “yes” or “no” answer for each input.



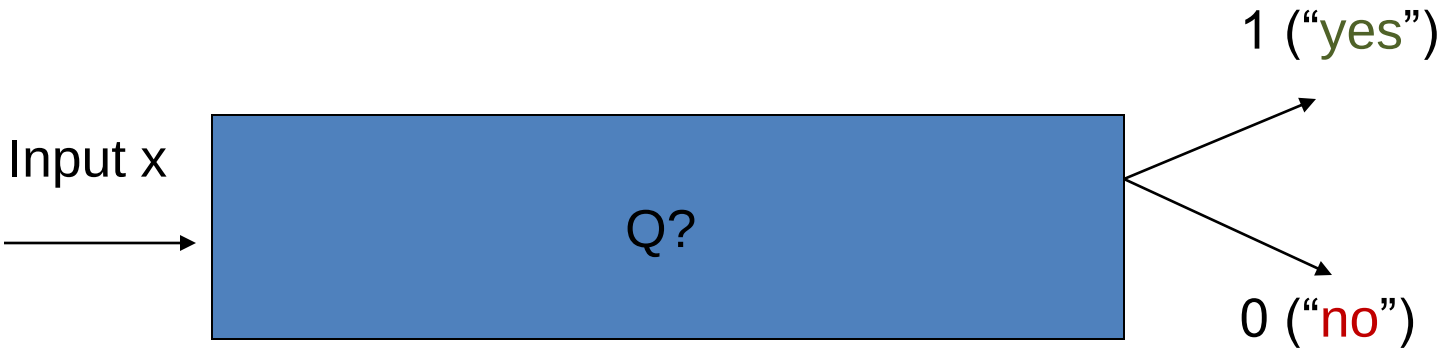
- Definition: (informal) A decision problem is
  - Decidable if there is an algorithm for solving it;
  - Undecidable if there is no algorithm for solving it.

Turing Computability

- Turing machine (TM)
- Turing computable (partial) functions f: $\Sigma^* \rightarrow \Sigma^*$
- Church's Thesis (aka Church-Turing Thesis): The following statements are equivalent for a (partial) function f: $\Sigma^* \rightarrow \Sigma^*$:
  - There is a Turing machine that computes f
  - There is an algorithm that computes f.
- Main Use of Church's Thesis: To show that there is no algorithm for computing a function f, it suffices to show that there is no Turing machine that computes f.

Decidable and Undecidable Problems

- Definition: Let Q be a decision problem.
  - Q is decidable (solvable) if there is a TM for solving the membership problem for the associated language.
  - Q is undecidable (unsolvable) if there no such TM.



Q is undecidable means that there is **no** algorithm for this problem

Undecidable Problems

Theorem: The following problems are undecidable:

- The Halting Problem (A. Turing – 1936): Given a Turing machine M and an input x, does M halt on x?

- The Finite Validity Problem (B. Trakhtenbrot – 1949): Given a first-order formula φ on graphs, is φ true on every finite graph?

Undecidable Problems

- The Finite Validity Problem (B. Trakhtenbrot – 1949): Given a first-order sentence φ on graphs, is φ true on every finite graph?
- Examples of Finitely Valid Formulas:
  - $\forall x\,(E(x,x) \rightarrow \exists y E(x,y))$
  - $\forall x \forall y\,(E(x,x) \wedge x = y \rightarrow E(y,y))$
  - "if E is a total order, then E has a biggest element"
- Examples of Non-Finitely Valid Formulas:
  - $\forall x \forall y\,(E(x,y) \rightarrow E(y,x))$
  - $(\forall x \exists y\, E(x,y)) \rightarrow (\exists y \forall x\, E(x,y))$
- The undecidability of the Finite Validity Problem means that there is **no** algorithm for telling formulas in the first group from formulas in the second group.

The Reduction Method

- By now there is a vast library of undecidable problems.
- The Reduction Method is the main technique for establishing undecidability.
- Reduction Method: To show that a problem L* is undecidable, it suffices to find an undecidable problem L and a computable function f such that for every input x, we have that

$$x \in L \quad \Leftrightarrow \quad f(x) \in L^*.$$

  - Such a function f is called a reduction of L to L*
  - $L \preccurlyeq L^*$ means that there is a reduction of L to L*.

The Reduction Method

- The Halting Problem was the first fundamental decision problem shown to be undecidable.

- The Finite Validity Problem was shown to be undecidable by showing that Halting Problem $\preccurlyeq$ Finite Validity Problem.

- Many database problems have been shown to be undecidable via reductions from
  - The Halting Problem

  or
  - The Finite Validity Problem

Computability and Complexity

- From the 1930s on, an extensive investigation of the boundary between decidability and undecidability has been carried out by mathematical logicians.
  - This investigation gave rise to the field of computability theory.

    (What can be automated?)

- From the 1960s on, an extensive investigation of decidable problems has been carried out by computer scientists aiming to identify the boundary between tractability and intractability.
  - This investigation gave rise to the field of computational complexity.

    (What can be efficiently automated?)

- Logic has played a major role in both computability theory and computational complexity.

Decidable Problems

Fact: The following problems are decidable.

- Given a propositional logic formula, is it satisfiable?
- Given a propositional logic formula, is it a tautology?
- Given a graph, is it 2-colorable?
- Given a graph, is it 3-colorable?
- Given a number, is it prime?
- Given a system of linear inequalities with integer coefficients, does it have a rational solution?
- Given a system of linear inequalities with integer coefficients, does it have an all-integer solution?

Question: Which of these problems have “efficient” algorithms?

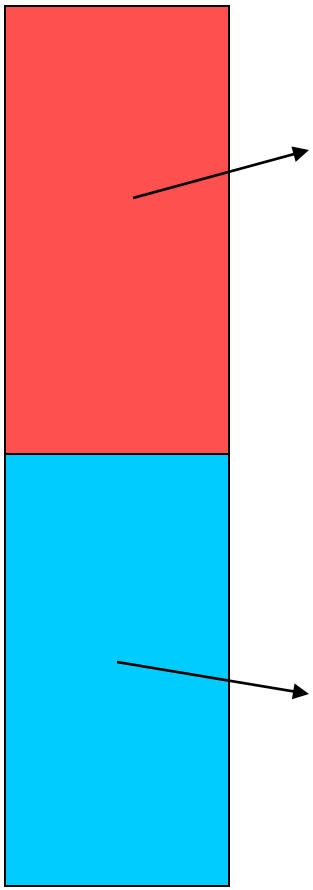
The Query Evaluation Problem

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

- The Query Evaluation Problem for relational calculus queries is decidable (why?), but, as we will see, it has high computational complexity.

- To understand the precise algorithmic difficulty of the Query Evaluation Problem, we need some basic notions and results from computational complexity.

Decidable Problems & Computational Complexity



- Computational Complexity is the quantitative study of decidable problems.

Computational Complexity Classes

- Decidable problems are grouped together in computational complexity classes.
- Each computational complexity class consists of all problems that can be solved in a computational model under certain restrictions on the resources used to solve the problem.
- Examples of computational models:
  - Turing Machine TM (deterministic Turing machine)
  - Non-deterministic Turing machine NTM
  - …
- Examples of resources:
  - Amount of time needed to solve the problem
  - Amount of space (memory) needed to solve the problem.
  - …

The Five Basic Computational Complexity Classes

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

More on the Class P

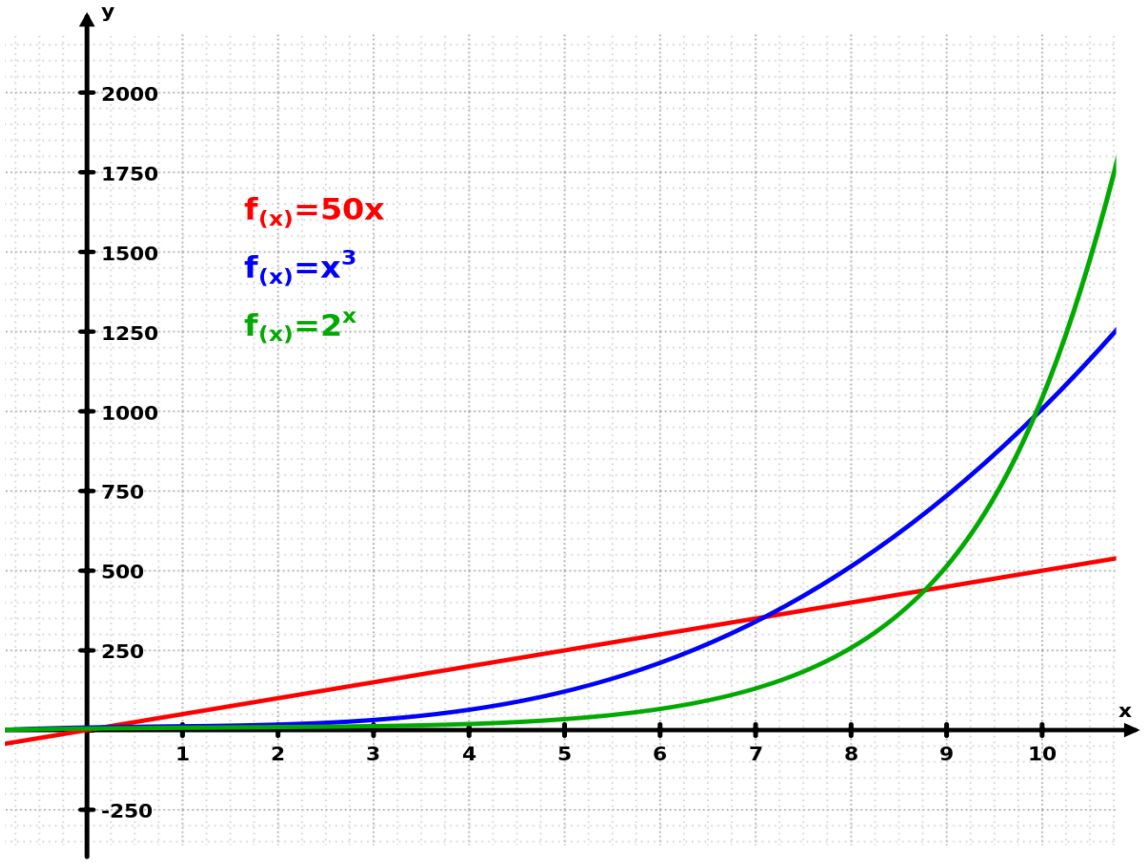
- The running time of an algorithm is the maximum number f(n) of steps the algorithm takes on inputs of size n.

- Informally, P is the collection of all problems that can be solved by an algorithm of polynomial running time
  (e.g., $f(n) = 50n$, $f(n) = 3n^2$, $f(n) = n^3$, …).

- Problems in P are also called tractable problems.

- Problems that are not in P are also called intractable problems.
  (e.g., if every algorithm for solving a problem has running time $f(n) \geq 2^n$, then the problem is intractable).

Polynomial vs. Exponential



Source: Wikipedia

Fact: Exponential functions grow much faster than polynomial functions

The Five Basic Computational Complexity Classes

Theorem:

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

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

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

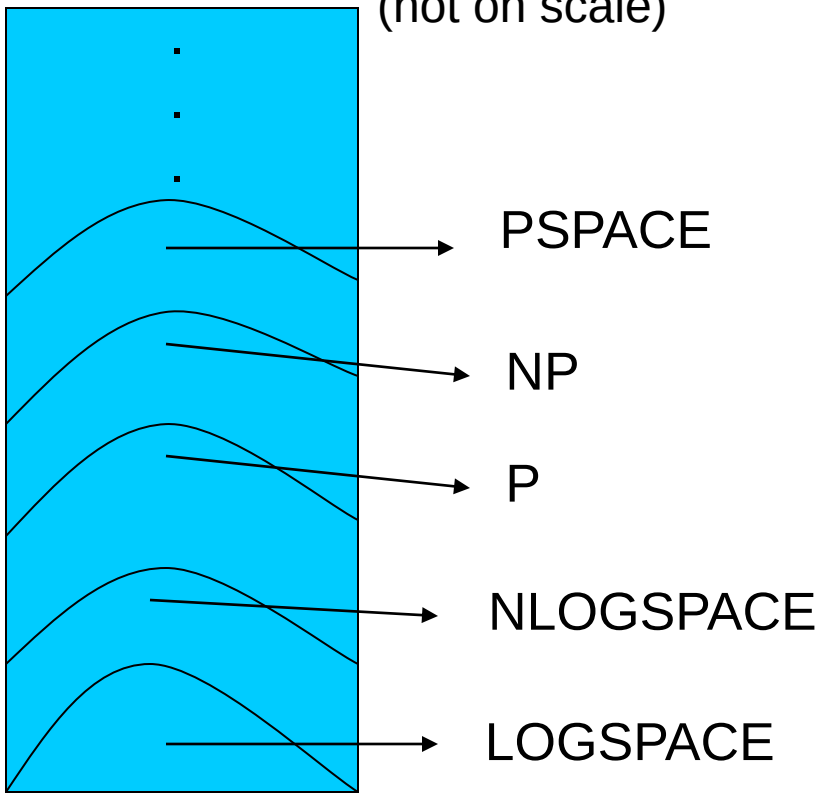
Note:

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

Computational Complexity Classes

Classification of Decidable Problems

(not on scale)



There are many other complexity classes.

For a comprehensive catalog, visit the

Complexity Zoo

or the less intimidating

Petting Zoo

Complete Problems

- A key property of most (but not all) complexity classes is that they possess complete problems.

- Intuitively, complete problems are the "hardest" problems in the class in the sense that every other problem can be reduced to it.

- Definition: Let ***C*** be a complexity class.

  A decision problem Q is ***C***-complete if
  - Q is in ***C***.
  - If Q' is in ***C*** , then there is a "suitable" total Turing computable function f such that for every input x, we have that

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

    - "suitable" means that f can be computed with fewer resources than those used to define ***C***.
    - So, f is a reduction of a restricted nature.

Complete Problems for Complexity Classes

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

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

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

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

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

Boolean Satisfiability

Definition: Conjunctive Normal Form (CNF): A formula written as a conjunction of disjunctions of variables / negated variables

$$(x \vee y \vee z) \wedge (\neg x \vee w) \wedge (y \vee \neg z \vee \neg w)$$

Definition: SAT is the following decision problem

Given a CNF-formula ψ, is ψ satisfiable?

(can we assign value 0/1 to each variable, so that ψ evaluates to 1?)

Example:

- $(x \vee y \vee z) \wedge (\neg x \vee w) \wedge (y \vee \neg z \vee \neg w)$ is satisfiable
- $(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$ is unsatisfiable.

NP-Completeness

Definition:

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

Theorem (S. Cook – 1971)

SAT is NP-complete

Hint of Proof:

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

NP-Completeness

Theorem (S. Cook – 1971)

SAT is NP-complete.

Note:

- Cook's work was motivated from automated theorem-proving.
- SAT was the first problem shown to be NP-complete.
- Cook's Theorem became the catalyst for the systematic investigation of NP-completeness.
- By now, thousands of problems from all areas of computer science have been shown to be NP-complete.
- NP-completeness is a pervasive phenomenon in computing.

NP-Complete Problems

Fact: The following problems are NP-complete:

- SAT
- 3SAT
- 3-Colorability
- 4-Colorability
- CLIQUE: Given a graph G and an integer k, does G contain a clique of size k?
- HAMILTON CYCLE: Given a graph G, does G contain a Hamilton cycle?
- Integer Linear Programming: Given a system of linear inequalities with integer coefficients, does it have an integer solution?
- …

Polynomial-Time Reductions

- 3-Satisfiability (3SAT): Given a 3CNF formula φ, is it satisfiable?
  (each clause has at most 3 literals)

- Theorem: 3SAT is NP-complete

  Proof: Show that SAT $\preccurlyeq_p$ 3SAT
  - Let φ be a CNF formula $c_1 \wedge c_2 \wedge \ldots \wedge c_m$
  - If a clause c_i has more than three literals, then we replace it with a set of clauses each with three literals and certain new variables.
  - For example, if c_i is $(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee x_5)$, then we replace c_i by $(x_1 \vee \neg x_2 \vee y_1)$, $(\neg y_1 \vee x_3 \vee y_2)$, $(\neg y_2 \vee x_4 \vee x_5)$.
  - Let φ^* be the resulting 3CNF formula. Then

    φ is satisfiable $\Leftrightarrow$ φ^* is satisfiable (check this).

Polynomial-Time Reductions

- 1-in-3SAT: Given a 3CNF formula φ, is there a truth assignment such that for every clause of φ, exactly one literal (variable or negated variable) becomes true?

  (in short, is φ 1-in-3 satisfiable?)

- Example:
  - The formula

    $$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$$

    is 1-in-3 satisfiable using $x \mapsto 1, y \mapsto 0, z \mapsto 0$.

  - The formula

    $$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

    is satisfiable but it is not 1-in-3 satisfiable.

Polynomial-Time Reductions

- 1-in-3SAT: Given a 3CNF formula φ, is there a truth assignment such that for every clause of φ, exactly one literal becomes true?
  (in short, is φ 1-in-3 satisfiable?)

- Theorem: 1-in-3SAT is NP-complete
  Proof: Show that 3SAT $\preccurlyeq_p$ 1-in-3-SAT
  - Let φ be a 3CNF formula $c_1 \wedge c_2 \wedge \ldots \wedge c_m$
  - For every clause $(x \vee y \vee z)$ of φ, introduce four new propositional variables p, q, r, u.
  - Replace the clause $(x \vee y \vee z)$ by the conjunction
    $$(\neg x \vee p \vee q) \wedge (q \vee y \vee r) \wedge (r \vee u \vee \neg z).$$
  - Let φ^* be the resulting 3CNF formula. Then
    φ is satisfiable $\Leftrightarrow$ φ^* is 1-in-3 satisfiable (check this).

Another Perspective on P vs. NP

- Intuitively, P is the class of all decision problems for which we can find a "solution" efficiently (where "efficiently" means in time bounded by a polynomial in the size of the input).

- Intuitively, NP is the class of all decision problems for which we can check efficiently whether a "candidate solution" is indeed a "solution".

  ("We can guess a "solution" and verify that it is indeed a solution in polynomial time")

Problem	Candidate Solution
SAT	An assignment of Boolean values to the variables
3-Colorability	An assignment of colors **B**, **R**, **G** to the nodes.

More on P vs. NP

- The “P = NP?” question is equivalent to the following:

- Question:

Is it true that every decision problem for which a “candidate solution” can be verified efficiently has the property that a “solution” can also be found efficiently?

  - In particular, this means that the “candidate solution” must be “small” (bounded by a polynomial in the size of the input).

- Note:

  The prevailing belief is that the answer to the above question is “**No**”, which means that P **≠** NP.

Why are NP-Complete Problems Special?

Proposition: The following statements are equivalent:

- P = NP.
- SAT is in P
- Your favorite NP-complete problem is in P.

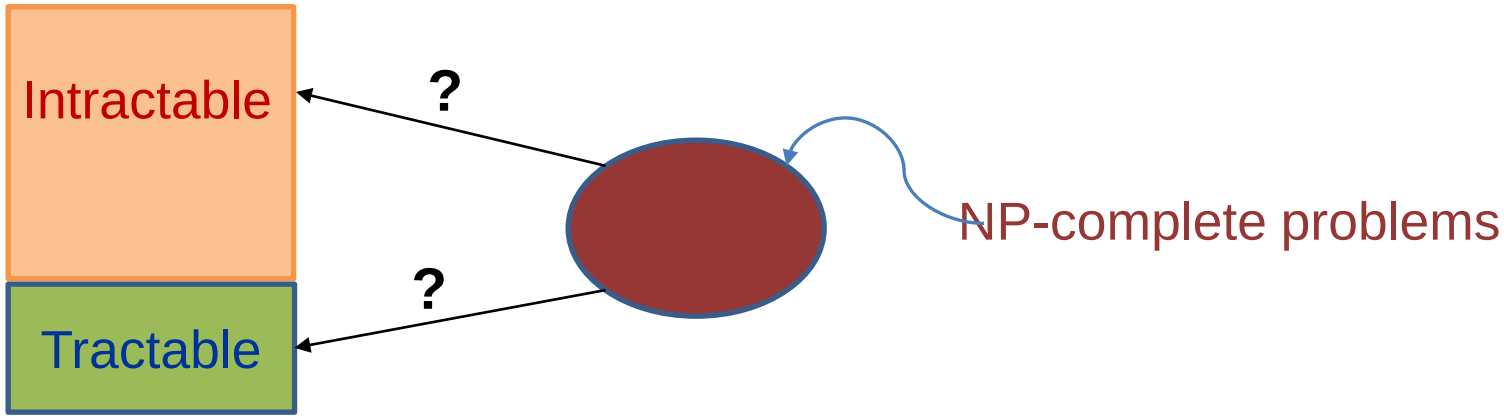
Conclusion:

- NP-complete problems hold the secret of whether or not P = NP.
- Either all NP-complete problems are in P or none is in P.
- Showing that a decision problem Q is NP-complete provides strong evidence that Q is not in P.

More on NP-Complete Problems

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

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



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

Complete Problems for PSPACE

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

Example:

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

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

Complexity of the Query Evaluation Problem

The Query Evaluation Problem for Relational Calculus:

Given a relational calculus formula φ and a database instance D, find $\varphi^{adom}(D)$.

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

Proof: We need to show that

- This problem is in PSPACE.
- This problem is PSPACE-hard.

We start with the second task.

Complexity of the Query Evaluation Problem

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

Proof: QBF – Quantified Boolean Formulas

Show that

QBF $\preccurlyeq_p$ Query Evaluation for Relational Calculus

Given QBF $\forall x_1 \exists x_2 \ldots \forall x_k \psi$

- Let V and P be two unary relation symbols
- Obtain ψ^* from ψ by replacing x_i by $P(x_i)$, and $\neg x_i$ by $\neg P(x_i)$
- Let D be the database instance with V = {0,1}, P = {1}.
- Then the following statements are equivalent:
  - $\forall x_1 \exists x_2 \ldots \forall x_k \psi$ is true
  - $\forall x_1 (V(x_1) \rightarrow \exists x_2 (V(x_2) \wedge (\ldots \forall x_k (V(x_k) \rightarrow \psi^*)) \ldots)$ is true on D.

Complexity of the Query Evaluation Problem

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

  Proof (Hint): Let φ be a relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$ and let D be a database instance.

  - Exponential Time Algorithm: We can find $\varphi^{adom}(D)$, by exhaustively cycling over all possible interpretations of the x_i's.

    This runs in time $O(n^m)$, where n = |D| (size of D).
  - A more careful analysis shows that this algorithm can be implemented in O(m·logn)-space.
    - Use m blocks of memory, each holding one of the n elements of adom(D) written in binary (so O(logn) space is used in each block).
    - Maintain also m counters in binary to keep track of the number of elements examined.

$\forall x_1$	$\exists x_2$	...	$\forall x_m$
a_1 in adom(D) written in binary	a_2 in adom(D) written in binary	...	a_m in adom(D) written in binary

Complexity of the Query Evaluation Problem

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

  Proof:

  The translation of relational calculus to relational algebra yields a polynomial-time reduction of the Query Evaluation Problem for Relational Calculus to the Query Evaluation Problem for Relational Algebra.

Complexity of Conjunctive Query Evaluation

Theorem: The query evaluation problem for conjunctive queries is NP-complete.

Proof:

Membership in NP:

Given a conjunctive query $\exists z_1 \cdots \exists z_n \chi(x_1, \ldots, x_n)$ and a database D, guess a tuple $(a_1, \ldots, a_n)$ and verify that D $\models \chi(a_1, \ldots, a_n)$.

NP-hardness: Reduction from the CLIQUE problem

Given a graph G = (V,E) and an integer k, does

$$G \models \exists z_1 \ldots \exists z_k \bigwedge_{i \neq j} E(z_i, z_j) \ ?$$

The Query Evaluation Problem Revisited

- Since the Query Evaluation Problem for Relational Calculus is PSPACE-hard, there are **no** polynomial-time algorithms for this problem, unless PSPACE = P (which is considered highly unlikely).
- Let's take another look at the exponential-time algorithm for this problem:
  - Let φ be a relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$ and let D be a database instance.
  - Exponential Time Algorithm: We can find $\varphi^{adom}(D)$, by exhaustively cycling over all possible interpretations of the x_i's.

    This runs in time $O(n^m)$, where $n = |D|$).
  - So, the running time is $O(|D|^{|\varphi|})$, where $|D|$ is the size of D and $|\varphi|$ is the size of the relational calculus formula φ.
  - This tells that the source of exponentiality is the formula size.

The Query Evaluation Problem Revisited

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

Proof:

Let φ be a fixed relational calculus formula $\forall x_1 \exists x_2 \ldots \forall x_m \psi$

- The previous algorithm has running time $O(|D|^{|\varphi|})$, which is a polynomial, since now $|\varphi|$ is a constant.
- Moreover, the algorithm can now be implemented using logarithmic-space only, since we need only maintain a constant number of memory blocks, each of logarithmic size

$\forall\ x_1$	$\exists\ x_2$	...	$\forall\ x_m$
a_1 in adom(D) written in binary	a_2 in adom(D) written in binary	...	a_m in adom(D) written in binary

Vardi's Taxonomy of the Query Evaluation Problem

M.Y Vardi, "The Complexity of Relational Query Languages" - 1982

Definition: Let L be a database query language and ***C*** a complexity class.

- The combined complexity of L is the decision problem:
  given an L-sentence and a database instance D, does $D \models \varphi$?
- The data complexity of L is the family of the following decision problems P_φ, where φ is an L-sentence:
  given a database instance D, does $D \models \varphi$?
- The data complexity of L is ***C***-complete if
  - P_φ is in ***C***, for every φ in L;
  - there at least one φ in L such that P_φ is ***C***-complete.

Vardi's "empirical" discovery:

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

Combined Complexity vs. Data Complexity

Vardi's "empirical" discovery:

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

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