CS21
Decidability and Tractability
Lecture 19
February 16, 2022

Outline
• hardness and completeness
  – an EXP-complete problem
• The class NP
  – alternate characterization of NP

So far…
• We have defined the complexity classes P (polynomial time), EXP (exponential time)

Poly-time reductions
• Type of reduction we will use:
  – “many-one” poly-time reduction (commonly)
  – “mapping” poly-time reduction (book)

Definition: $f : \Sigma^* \to \Sigma^*$ is poly-time computable if for some $g(n) = n^{O(1)}$ there exists a $g(n)$-time TM $M_f$ such that on every $w \in \Sigma^*$, $M_f$ halts with $f(w)$ on its tape.
Poly-time reductions

**Definition:** A \( \leq_p B \) (“A reduces to B”) if there is a poly-time computable function \( f \) such that for all \( w \)

\[ w \in A \iff f(w) \in B \]

- as before, condition equivalent to:
  - YES maps to YES and NO maps to NO
- as before, meaning is:
  - B is at least as “hard” (or expressive) as A

**Theorem:** if \( A \leq_p B \) and \( B \in P \) then \( A \in P \).

**Proof:**
- a poly-time algorithm for deciding \( A \):
  - on input \( w \), compute \( f(w) \) in poly-time.
  - run poly-time algorithm to decide if \( f(w) \in B \)
  - if it says “yes”, output “yes”
  - if it says “no”, output “no”

Example

- \( 2\text{SAT} = \{ \text{CNF formulas with 2 literals per clause for which there exists a satisfying truth assignment} \} \)
- \( L = \{ \text{directed graph } G, \text{and list of pairs of vertices } (u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k), \text{such that there is no i for which } [u_i \text{ is reachable from } v_i \text{ in G and } v_i \text{ is reachable from } u_i \text{ in G}] \} \)
- We gave a poly-time reduction from \( 2\text{SAT} \) to \( L \).
- determined that \( 2\text{SAT} \in P \) from fact that \( L \in P \)

Hardness and completeness

- Reasonable that can efficiently transform one problem into another.
- Surprising:
  - can often find a special language \( L \) so that every language in a given complexity class reduces to \( L \! \)
  - powerful tool

**Definition:** a language \( L \) is C-hard if for every language \( A \in C \), A poly-time reduces to \( L \); i.e., \( A \leq_p L \).

- meaning: \( L \) is at least as “hard” as anything in \( C \)

**Definition:** a language \( L \) is C-complete if \( L \) is C-hard and \( L \in C \)

- meaning: \( L \) is a “hardest” problem in \( C \)
An EXP-complete problem

- Version of $A_{TM}$ with a time bound:
  $$ATM_B = \{<M, x, m> : M \text{ is a TM that accepts } x \text{ within at most } m \text{ steps}\}$$

**Theorem**: $ATM_B$ is EXP-complete.

**Proof**:
- what do we need to show?

\[m \in O(1)\]

\[n = \log m\]

\[m \leq 2^{kn}\]

An EXP-complete problem

- $ATM_B = \{<M, x, m> : M \text{ is a TM that accepts } x \text{ within at most } m \text{ steps}\}$
- Proof that $ATM_B$ is EXP-complete:
  - Part 2. For each language $A \in EXP$, need to give poly-time reduction from $A$ to $ATM_B$.
  - For a given language $A \in EXP$, we know there is a TM $M_A$ that decides $A$ in time $g(n) \leq 2^k$ for some $k$.
  - what should reduction $f(w)$ produce?

An EXP-complete problem

- A $C$-complete problem is a surrogate for the entire class $C$.
- For example: if you can find a poly-time algorithm for $ATM_B$ then there is automatically a poly-time algorithm for every problem in EXP (i.e., EXP = P).

- Can you find a poly-time alg for $ATM_B$?
Back to 3SAT

- Remember 3SAT ∈ EXP
  3SAT = \{formulas in CNF with 3 literals per clause for which there exists a satisfying truth assignment\}

- It seems hard. Can we show it is intractable?
  - formally, can we show 3SAT is EXP-complete?

Back to 3SAT

- can we show 3SAT is EXP-complete?

- Don’t know how to. Believed unlikely.
- One reason: there is an important positive feature of 3SAT that doesn’t seem to hold for problems in EXP (e.g. ATM_b):

  3SAT is decidable in polynomial time by a nondeterministic TM

Nondeterministic TMs

- Recall: nondeterministic TM
  - informally, TM with several possible next configurations at each step
  - formally, A NTM is a 7-tuple \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\) where:
    - everything is the same as a TM except the transition function:
      \[\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R}\)\]

Nondeterministic TMs

visualize computation of a NTM M as a tree

- nodes are configurations
- leaves are accept/reject configurations
- M accepts if and only if there exists an accept leaf
- M is a decider, so no paths go on forever
- running time is max. path length

The class NP

**Definition:** \(\text{TIME}(t(n)) = \{L : \text{there exists a TM M that decides L in time } O(t(n))\}\)

\[P = \bigcup_{k \geq 1} \text{TIME}(n^k)\]

**Definition:** \(\text{NTIME}(t(n)) = \{L : \text{there exists a NTM M that decides L in time } O(t(n))\}\)

\[NP = \bigcup_{k \geq 1} \text{NTIME}(n^k)\]

NP in relation to P and EXP

- \(P \subseteq NP\) (poly-time TM is a poly-time NTM)
- \(NP \subseteq EXP\)
  - configuration tree of \(n^k\)-time NTM has \(\leq b^{n^k}\) nodes
  - can traverse entire tree in \(O(b^n)\) time
  - we do not know if either inclusion is proper
Poly-time verifiers

- NP = \{ L : \text{L decided by poly-time NTM} \}
- Very useful alternate definition of NP:
  **Theorem:** language L is in NP if and only if it is expressible as:
  \[ L = \{ x \mid \exists y, |y| \leq |x|^k, (x, y) \in R \} \]
  where R is a language in P.
- poly-time TM \( M_R \) deciding R is a "verifier"

---

Example: 3SAT expressible as
3SAT = \{ \phi : \phi \text{ is a 3-CNF formula for which } \exists \text{ assignment } A \text{ for which } (\phi, A) \in R \}
\[ R = \{ (\phi, A) : A \text{ is a sat. assign. for } \phi \} \]
- satisfying assignment A is a "witness" of the satisfiability of \( \phi \) (it "certifies" satisfiability of \( \phi \))
- R is decidable in poly-time

---

Cook-Levin Theorem

- Gateway to proving lots of natural, important problems NP-complete is:
  **Theorem** (Cook, Levin): 3SAT is NP-complete.
  - Recall: 3SAT = \{ \phi : \phi \text{ is a CNF formula with 3 literals per clause for which there exists a satisfying truth assignment} \}

---

Proof outline
- show CIRCUIT-SAT is NP-complete
  CIRCUIT-SAT = \{ C : C \text{ is a Boolean circuit for which there exists a satisfying truth assignment} \}
  - show 3SAT is NP-complete (reduce from CIRCUIT SAT)
Boolean Circuits

- Boolean circuit $C$
  - directed acyclic graph
  - nodes: AND ($\land$); OR ($\lor$); NOT ($\neg$); variables $x_i$
- $C$ computes function $f: \{0,1\}^n \rightarrow \{0,1\}$ in natural way
  - identify $C$ with function $f$ it computes
- size = # nodes

Boolean Circuits

- every function $f: \{0,1\}^n \rightarrow \{0,1\}$ computable by a circuit of size at most $O(n2^n)$
  - AND of $n$ literals for each $x$ such that $f(x) = 1$
  - OR of up to $2^n$ such terms

CIRCUIT-SAT is NP-complete

**Theorem:** CIRCUIT-SAT is NP-complete

$CIRCUIT\text{-SAT} = \{C : C$ is a Boolean circuit for which there exists a satisfying truth assignment$\}$

**Proof:**
- Part 1: need to show $CIRCUIT\text{-SAT} \in \text{NP}$.
  - can express $CIRCUIT\text{-SAT}$ as:
    $CIRCUIT\text{-SAT} = \{C : C$ is a Boolean circuit for which $\exists x$ such that $(C, x) \in R\}$
    $R = \{(C, x) : C$ is a Boolean circuit and $C(x) = 1\}$