CS21
Decidability and Tractability
Lecture 16-17
February 13 + 15, 2017
Outline

- Extended Church-Turing Thesis
- The complexity class $P$
  - examples of problems in $P$
- The complexity class $EXP$
- Time Hierarchy Theorem
- hardness and completeness
- an $EXP$-complete problem
Multitape TMs

• Convenient to “program” multitape TMs rather than single ones
  – equivalent when talking about decidability
  – not equivalent when talking about time complexity

**Theorem**: Let \( t(n) \) satisfy \( t(n) \geq n \). Every multi-tape TM running in time \( t(n) \) has an equivalent TM running in time \( O(t(n)^2) \).
Multitape TMs

simulation of k-tape TM by single-tape TM:

- add new symbol $x$ for each old $x$
- marks location of “virtual heads”

(input tape)
Multitape TMs

Repeat: $O(t(n))$ times

• scan tape, remembering the symbols under each virtual head in the state

• make changes to reflect 1 step of M;

• if hit #, shift to right to make room.

$O(k \ t(n)) = O(t(n))$

when M halts, erase all but 1st string

$O(t(n))$
Multitape TMs

• Moral: feel free to use k-tape TMs, but be aware of slowdown in conversion to TM
  – note: if $t(n) = O(n^c)$ then $t(n)^2 = O(n^{2c}) = O(n^{c'})$
  – note: if $t(n) = O(2^{n\delta})$ for $\delta > 0$ then $t(n)^2 = O(2^{2n\delta}) = O(2^{n\delta'})$ for $\delta' > 0$

• high-level operations you are used to using can be simulated by TM with only polynomial slowdown
  – e.g., copying, moving, incrementing/decrementing, arithmetic operations $+,-,\times,/$
Extended Church-Turing Thesis

• the belief that TMs formalize our intuitive notion of an efficient algorithm is:

The “extended” Church-Turing Thesis

everything we can compute in time \( t(n) \)
on a physical computer can be computed on a Turing Machine in time \( t(n)^{O(1)} \) (polynomial slowdown)

• quantum computers challenge this belief
Time Complexity

• interested in a coarse classification of problems. For this purpose,
  – treat any polynomial running time as “efficient” or “tractable”
  – treat any exponential running time as inefficient or “intractable”

Key definition: “P” or “polynomial-time” is

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

• Why polynomial-time?
  – insensitive to particular deterministic model of computation chosen
  – closed under modular composition
  – empirically: qualitative breakthrough to achieve polynomial running time is followed by quantitative improvements from impractical (e.g. \( n^{100} \)) to practical (e.g. \( n^3 \) or \( n^2 \))
Examples of languages in P

• Recall: positive integers x, y are relatively prime if their Greatest Common Divisor (GCD) is 1.

• will show the following language is in P:
  \[ \text{RELPRIME} = \{ <x, y> : x \text{ and } y \text{ are relatively prime} \} \]

• what is the running time of the algorithm that tries all divisors up to \( \min\{x, y\} \)?
Euclid’s Algorithm

- possibly earliest recorded algorithm

on input \( <x, y> \):
  - repeat until \( y = 0 \)
    - set \( x = x \mod y \)
    - swap \( x, y \)
  - \( x \) is the \( \text{GCD}(x, y) \). If \( x = 1 \), accept; otherwise reject

Example run on input \( <10, 22> \):

\[
\begin{align*}
x, y &= 10, 22 \\
x, y &= 22, 10 \\
x, y &= 10, 2 \\
x, y &= 2, 0
\end{align*}
\]

reject
Euclid’s Algorithm

• possibly earliest recorded algorithm

on input \(<x, y>\):
  • repeat until \(y = 0\)
    • set \(x = x \mod y\)
    • swap \(x, y\)
  • \(x\) is the GCD\((x, y)\). If \(x = 1\), accept; otherwise reject

Example run on input \(<24, 5>\):

\[
\begin{align*}
x, y &= 24, 5 \\
x, y &= 5, 4 \\
x, y &= 4, 1 \\
x, y &= 1, 0 \\
\text{accept}
\end{align*}
\]
Euclid’s Algorithm

on input \(<x, y>\):

1. repeat until \(y = 0\)
   2. set \(x = x \mod y\)
   3. swap \(x, y\)

\(x\) is the GCD\((x, y)\). If \(x = 1\), accept; otherwise reject

Claim: value of \(x\) reduced by \(\frac{1}{2}\) at every execution of (2) except possibly first one.

Proof:

1. after (2) \(x < y\)
2. after (3) \(x > y\)
3. if \(x/2 \geq y\), then \(x \mod y < y \leq x/2\)
4. if \(x/2 < y\), then \(x \mod y = x - y < x/2\)

\(\text{every 2 times through loop, (}x, y\) each reduced by \(1/2\)\)

\(\text{loops } \leq 2 \cdot \max\{\log_2 x, \log_2 y\} = O(n = |<x, y>|); \text{ poly time for each loop}\)
A puzzle

• Find an efficient algorithm to solve the following problem:
• Input: sequence of pairs of symbols
e.g. (A, b), (E, D), (d, C), (B, a)
• Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.
A puzzle

• Find an efficient algorithm to solve the following problem.
• Input: sequence of pairs of symbols
e.g. (A, b), (E, D), (d, C), (b, a)
• Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.
2SAT

- This is a disguised version of the language $2\text{SAT} = \{\text{formulas in Conjunctive Normal Form with 2 literals per clause for which there exists a satisfying truth assignment}\}$
  - CNF = “AND of ORs”
    
    \[(A, b), (E, D), (d, C), (b, a)\]
    \[(x_1 \lor \neg x_2) \land (x_5 \lor x_4) \land (\neg x_4 \lor x_3) \land (\neg x_2 \lor \neg x_1)\]

- satisfying truth assignment = assignment of TRUE/FALSE to each variable so that whole formula is TRUE
2SAT

**Theorem:** There is a polynomial-time algorithm deciding 2SAT ("2SAT $\in$ P").

Proof: algorithm described on next slides.
Algorithm for 2SAT

- Build a graph with separate nodes for each literal.
  - add directed edge \((x, y)\) iff formula includes clause \((\neg x \lor y)\) or \((y \lor \neg x)\) (equiv. to \(x \implies y\))

\[ (x_1 \lor \neg x_2) \land (x_5 \lor x_4) \land (\neg x_4 \lor x_3) \land (\neg x_2 \lor \neg x_1) \]
Algorithm for 2SAT

Claim: formula is unsatisfiable iff there is some variable \( x \) with a path from \( x \) to \( \neg x \) and a path from \( \neg x \) to \( x \) in derived graph.

• Proof (\( \equiv \))
  – edges represent implication \( \Rightarrow \). By transitivity of \( \Rightarrow \), a path from \( x \) to \( \neg x \) means \( x \Rightarrow \neg x \), and a path from \( \neg x \) to \( x \) means \( \neg x \Rightarrow x \).
Algorithm for 2SAT

• Proof ($\Rightarrow$)
  – to construct a satisfying assign. (if no x with a path from x to $\neg x$ and a path from $\neg x$ to x):
    • pick unassigned literal s with no path from s to $\neg s$
    • assign it TRUE, as well as all nodes reachable from it; assign negations of these literals FALSE
    • note: path from s to t and s to $\neg t$ implies path from $\neg t$ to $\neg s$ and t to $\neg s$, implies path from s to $\neg s$
    • note: path s to t (assigned FALSE) implies path from $\neg t$ (assigned TRUE) to $\neg s$, so s already assigned at that point.
Algorithm for 2SAT

• Algorithm:
  – build derived graph
  – for every pair $x, \neg x$ check if there is a path from $x$ to $\neg x$ and from $\neg x$ to $x$ in the graph

• Running time of algorithm (input length $n$):
  – $O(n)$ to build graph
  – $O(n)$ to perform each check
  – $O(n)$ checks
  – running time $O(n^2)$. 2SAT $\in$ P.
Another puzzle

• Find an efficient algorithm to solve the following problem.
• Input: sequence of *triples* of symbols e.g. \((A, b, C), (E, D, b), (d, A, C), (c, b, a)\)
• Goal: determine if it is possible to circle at least one symbol in each *triple* without circling upper and lower case of same symbol.
3SAT

• This is a disguised version of the language

\[3\text{SAT} = \{\text{formulas in Conjunctive Normal Form with 3 literals per clause for which there exists a satisfying truth assignment}\}\]

\[\text{e.g. } (A, b, C), (E, D, b), (d, A, C), (c, b, a)\]
\[(x_1 \lor \neg x_2 \lor x_3) \land (x_5 \lor x_4 \lor \neg x_2) \land (\neg x_4 \lor x_1 \lor x_3) \land (\neg x_3 \lor \neg x_2 \lor \neg x_1)\]

• observe that this language is in \(\text{TIME}(2^n)\)
Key definition: “P” or “polynomial-time” is

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

Definition: “EXP” or “exponential-time” is

\[ \text{EXP} = \bigcup_{k \geq 1} \text{TIME}(2^{n^k}) \]
\[ \text{EXP} \]

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

\[ \text{EXP} = \bigcup_{k \geq 1} \text{TIME}(2^{n^k}) \]

• Note: \( P \subseteq \text{EXP} \).

• We have seen 3SAT \( \in \text{EXP} \).
  
  – does not rule out possibility that it is in \( P \)

• Is \( P \) different from \( \text{EXP} \)?
Time Hierarchy Theorem

**Theorem:** For every proper complexity function \( f(n) \geq n: \)
\[
\text{TIME}(f(n)) \text{ } \text{ } \text{TIME}(f(2n)^3).
\]

- **Note:** \( P \subseteq \text{TIME}(2^n) \not\subseteq \text{TIME}(2^{(2n)^3}) \subseteq \text{EXP} \)
- Most natural functions (and \( 2^n \) in particular) are proper complexity functions. We will ignore this detail in this class.
Time Hierarchy Theorem

**Theorem:** For every proper complexity function \( f(n) \geq n \):

\[
\text{TIME}(f(n)) \not\subseteq \text{TIME}(f(2n)^3).
\]

- Proof idea:
  - use diagonalization to construct a language that is not in \( \text{TIME}(f(n)) \).
  - constructed language comes with a TM that decides it and runs in time \( f(2n)^3 \).
Recall proof for Halting Problem

Turing Machines

inputs

box

(M, x): does M halt on x?

The existence of H which tells us yes/no for each box allows us to construct a TM H' that cannot be in the table.
Proof of Time Hierarchy Theorem

Turing Machines

box \ (M, x): \ does \ M \ accept \ x \ in \ time \ f(n) ?

- TM SIM tells us yes/no for each box in time g(n)
- rows include all of \ TIME(f(n))
- construct TM D running in time g(2n) that is not in table

D : \ n \ Y \ n \ Y \ Y \ n \ Y
Proof of Time Hierarchy Theorem

• Proof:
  – SIM is TM deciding language
    \[ \{ <M, x> : M \text{ accepts } x \text{ in } \leq f(|x|) \text{ steps} \} \]
  – Claim: SIM runs in time \( g(n) = f(n)^3 \).
  – define new TM D: on input \(<M>\)
    • if SIM accepts \(<M, <M>>\), reject
    • if SIM rejects \(<M, <M>>\), accept
  – D runs in time \( g(2n) \)
Proof of Time Hierarchy Theorem

• Proof (continued):
  – suppose $M$ in $\text{TIME}(f(n))$ decides $L(D)$
    • $M(<M>) = \text{SIM}(<M, <M>>) \neq D(<M>)$
    • but $M(<M>) = D(<M>)$
  – contradiction.
Proof of Time Hierarchy Theorem

• Claim: there is a TM SIM that decides
  \[ \{<M, x> : M \text{ accepts } x \text{ in } \leq f(|x|) \text{ steps}\} \]
  and runs in time \( g(n) = f(n)^3 \).

• Proof sketch: SIM has 4 work tapes
  • contents and “virtual head” positions for M’s tapes
  • M’s transition function and state
  • \( f(|x|) \) “+”s used as a clock
  • scratch space
Proof of Time Hierarchy Theorem

- Proof sketch (continued): 4 work tapes
  - contents and “virtual head” positions for M’s tapes
  - M’s transition function and state
  - $f(|x|)$ “+”s used as a clock
  - scratch space
- initialize tapes
- simulate step of M, advance head on tape 3; repeat.
- can check running time is as claimed.
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)
Poly-time reductions

- function $f$ should be **poly-time computable**

**Definition:** $f : \Sigma^* \rightarrow \Sigma^*$ 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 $A$
Poly-time reductions

**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

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

• Recall:
  – a language L is a set of strings
  – a complexity class C is a set of languages

**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
Hardness and completeness

• Recall:
  – a language L is a set of strings
  – a complexity class C is a set of languages

**Definition:** a language L is C-complete if L is C-hard and L ∈ 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?
An EXP-complete problem

- \( \text{ATM}_B = \{<M, x, m> : M \text{ is a TM that accepts } x \text{ within at most } m \text{ steps}\} \)

- Proof that \( \text{ATM}_B \) is EXP-complete:
  - Part 1. Need to show \( \text{ATM}_B \in \text{EXP} \).
    - simulate \( M \) on \( x \) for \( m \) steps; accept if simulation accepts; reject if simulation doesn’t accept.
    - running time \( m^{O(1)} \).
    - \( n = \text{length of input} \geq \log_2 m \)
    - running time \( \leq m^k = 2^{(\log m)^k} \leq 2^{(kn)} \)
An EXP-complete problem

• ATM_B = {<M, x, m> : M is a TM that accepts x within at most m steps}

• Proof that ATM_B is EXP-complete:
  – Part 2. For each language A ∈ EXP, need to give poly-time reduction from A to ATM_B.
  – for a given language A ∈ EXP, we know there is a TM M_A that decides A in time g(n) ≤ 2^{n^k} for some k.
  – what should reduction f(w) produce?
An EXP-complete problem

- \( \text{ATM}_B = \{ <M, x, m> : M \text{ is a TM that accepts } x \text{ within at most } m \text{ steps} \} \)

- Proof that \( \text{ATM}_B \) is EXP-complete:
  - \( f(w) = <M_A, w, m> \) where \( m = 2^{|w|^k} \)
  - is \( f(w) \) poly-time computable?
    * hardcode \( M_A \) and \( k \)...
  - YES maps to YES?
    * \( w \in A \Rightarrow <M_A, w, m> \in \text{ATM}_B \)
  - NO maps to NO?
    * \( w \notin A \Rightarrow <M_A, w, m> \notin \text{ATM}_B \)
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?
An EXP-complete problem

• Can you find a poly-time alg for ATM_B?
• NO! we showed that P \subsetneq EXP.
• ATM_B is not tractable (intractable).
Back to 3SAT

• Remember $3\text{SAT} \in \text{EXP}$

$3\text{SAT} = \{\text{formulas in CNF with 3 literals per clause for which there exists a satisfying truth assignment}\}$

• It seems hard. Can we show it is intractable?
  – formally, can we show $3\text{SAT}$ is $\text{EXP}$-complete?
Back to 3SAT

• can we show 3SAT is $\text{EXP}$-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 $\text{EXP}$ (e.g. $\text{ATM}_B$):

\[
\text{3SAT is decidable in polynomial time by a nondeterministic TM}
\]
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:
  \[\delta : Q \times \Gamma \rightarrow \powerset(Q \times \Gamma \times \{L, R\})\]
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:** \( \text{TIME}(t(n)) = \{L : \text{there exists a TM } M \text{ that decides } L \text{ in time } O(t(n))\} \)

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

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

\[
\text{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k)
\]
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 \( \leq b^{nk} \) nodes
  - can traverse entire tree in \( O(b^{nk}) \) time

we do not know if either inclusion is proper