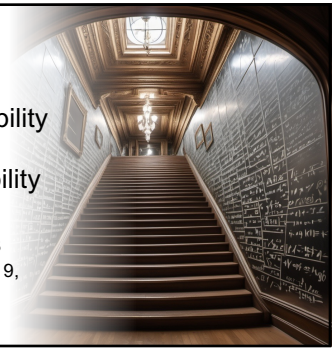
CS21 Decidability and Tractability



Lecture 18
February 19, 2025

1

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.

2

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.

3

2SAT

- This is a disguised version of the language
  2SAT = {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)
  $(x_1 \vee \neg x_2) \wedge (x_5 \vee x_4) \wedge (\neg x_4 \vee x_3) \wedge (\neg x_2 \vee \neg x_1)$
  – satisfying truth assignment = assignment of TRUE/FALSE to each variable so that whole formula is TRUE

4

2SAT

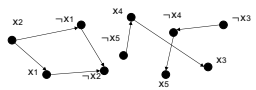
Theorem: There is a polynomial-time algorithm deciding 2SAT ("2SAT ∈ P").

Proof: algorithm described on next slides.

5

Algorithm for 2SAT

- Build a graph with separate nodes for each literal.
  – add directed edge (x, y) iff formula includes clause (¬x ∨ y) (equiv. to x ⇒ y)



e.g. $(x_1 \vee \neg x_2) \wedge (x_5 \vee x_4) \wedge (\neg x_4 \vee x_3) \wedge (\neg x_2 \vee \neg x_1)$

6

Algorithm for 2SAT

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

- Proof (⇐)
  - edges represent implication ⇒. By transitivity of ⇒, a path from x to ¬x means x ⇒ ¬x, and a path from ¬x to x means ¬x ⇒ x.

7

Algorithm for 2SAT

- Proof (⇒)
  - to construct a satisfying assign. (if no x with a path from x to ¬x and a path from ¬x to x):
    - pick unassigned literal s with no path from s to ¬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 ¬t implies path from ¬t to ¬s and t to ¬s, implies path from s to ¬s
    - note: path s to t (assigned FALSE) implies path from ¬t (assigned TRUE) to ¬s, so s already assigned at that point.

8

Algorithm for 2SAT

- Algorithm:
  - build derived graph
  - for every pair x, ¬x check if there is a path from x to ¬x and from ¬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 ∈ P.

9

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.

10

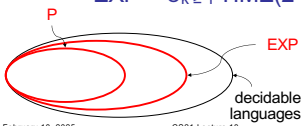
3SAT

- This is a disguised version of the language
  3SAT = {formulas in Conjunctive Normal Form with 3 literals per clause for which there exists a satisfying truth assignment}
  e.g. (A, b, C), (E, D, b), (d, A, C), (c, b, a)
  $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_5 \vee x_4 \vee \neg x_2) \wedge (\neg x_4 \vee x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_2 \vee \neg x_1)$
- observe that this language is in $TIME(2^n)$

11

Time Complexity

Key definition: "P" or "polynomial-time" is
$P = \cup_{k \geq 1} TIME(n^k)$
Definition: "EXP" or "exponential-time" is
$EXP = \cup_{k \geq 1} TIME(2^{nk})$



12

EXP

$P = \cup_{k \geq 1} TIME(n^k)$

$EXP = \cup_{k \geq 1} TIME(2^{n^k})$

- Note: $P \subseteq EXP$.
- We have seen 3SAT $\in$ EXP.
  - **does not rule out possibility that it is in P**
- Is P different from EXP?

13

Time Hierarchy Theorem

Theorem: for every proper complexity function $f(n) \geq n$:

$TIME(f(n)) \subsetneq TIME(f(2n)^3)$.

- Note: $P \subseteq TIME(2^n) \subsetneq TIME(2^{(2n)^3}) \subseteq EXP$
- Most natural functions (and 2^n in particular) are proper complexity functions. We will ignore this detail in this class.

14

Time Hierarchy Theorem

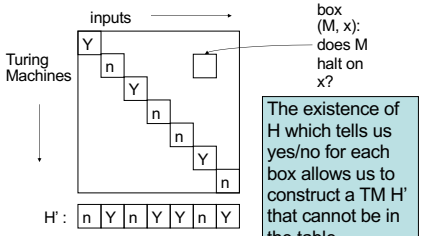
Theorem: for every proper complexity function $f(n) \geq n$:

$TIME(f(n)) \subsetneq TIME(f(2n)^3)$.

- Proof idea:
  - use diagonalization to construct a language that is not in TIME(f(n)).
  - constructed language comes with a TM that decides it and runs in time $f(2n)^3$.

15

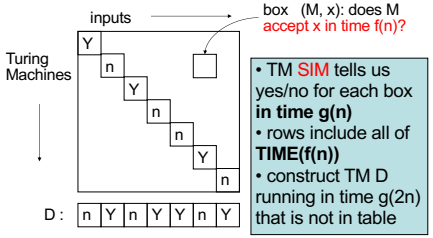
Recall proof for Halting Problem



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.

16

Proof of Time Hierarchy Theorem



- 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

17

Proof of Time Hierarchy Theorem

- Proof:
  - SIM is TM deciding language
    { <M, x> : M accepts x in ≤ f(|x|) 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)

18

Proof of Time Hierarchy Theorem

- Proof (continued):
  - suppose M in **TIME(f(n))** decides L(D)
    - M(<M>) = SIM(<M, <M>>) ≠ D(<M>)
    - but M(<M>) = D(<M>)
  - contradiction.

19

Proof of Time Hierarchy Theorem

- Claim: there is a TM SIM that decides
  {<M, x> : M accepts x in ≤ f(|x|) 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

20

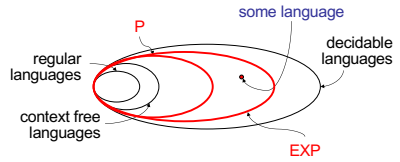
Proof of Time Hierarchy Theorem

- 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.

21

So far…

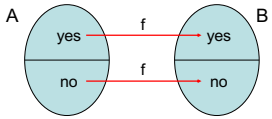
- We have defined the complexity classes P (polynomial time), EXP (exponential time)



22

Poly-time reductions

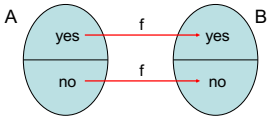
- Type of reduction we will use:
  - "many-one" poly-time reduction (commonly)
  - "mapping" poly-time reduction (book)



reduction from language A to language B

23

Poly-time reductions



- function f should be poly-time computable

Definition: f : Σ*→ Σ* 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 ∈ Σ*, M_f halts with f(w) on its tape.

24

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

$$w \in A \Leftrightarrow f(w) \in B$$

- 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

25

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"

26

Example

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

27