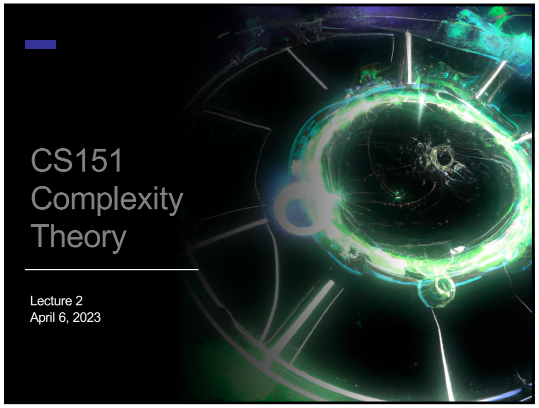


CS151 Complexity Theory

Lecture 2
April 6, 2023

1

Turing Machines

- Amazing fact: there exist (natural) **undecidable** problems

HALT = { (M, x) : M halts on input x }

- Theorem: HALT is undecidable.

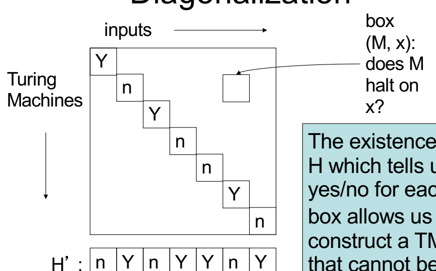
2

Turing Machines

- Proof:
  - Suppose TM H decides HALT
  - Define new TM H’ : on input <M>
    - if H accepts (M, <M>) then loop
    - if H rejects (M, <M>) then halt
  - Consider H’ on input <H’ >:
    - if it halts, then H rejects (H’ , <H’ >), which implies it cannot halt
    - if it loops, then H accepts (H’ , <H’ >) which implies it must halt
  - contradiction.

3

Diagonalization



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.

4

Turing Machines

- Back to complexity classes:
  - **TIME(f(n))** = languages decidable by a multi-tape TM in at most f(n) steps, where n is the input length, and f : $\mathbf{N} \rightarrow \mathbf{N}$
  - **SPACE(f(n))** = languages decidable by a multi-tape TM that touches at most f(n) squares of its work tapes, where n is the input length, and f : $\mathbf{N} \rightarrow \mathbf{N}$

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

5

Interlude

- In an ideal world, given language L
  - state an algorithm deciding L
  - **prove** that no algorithm does better
- we are pretty good at part 1
- we are currently **completely helpless** when it comes to part 2, for most problems that we care about

6

Interlude

- in place of part 2 we can
  - relate the difficulty of problems to each other via **reductions**
  - prove that a problem is a "hardest" problem in a complexity class via **completeness**
- powerful, successful surrogate for lower bounds

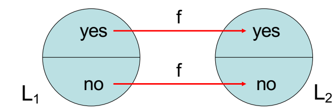
7

Reductions

- **reductions** are the main tool for relating problems to each other
- given two languages L_1 and L_2 we say "$\mathbf{L_1}$ **reduces to** $\mathbf{L_2}$" and we write "$L_1 \leq L_2$" to mean:
  - there exists an efficient (for now, poly-time) algorithm that computes a function f s.t.
    - $x \in L_1$ implies $f(x) \in L_2$
    - $x \notin L_1$ implies $f(x) \notin L_2$

8

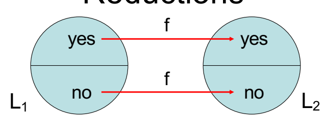
Reductions



- positive use: given new problem $\mathbf{L_1}$ reduce it to $\mathbf{L_2}$ that we know to be in **P**. Conclude $\mathbf{L_1}$ in **P** (how?)
  - e.g. bipartite matching ≤ max flow
  - formalizes "$\mathbf{L_1}$ as easy as $\mathbf{L_2}$"

9

Reductions



- **negative use**: given new problem $\mathbf{L_2}$ reduce $\mathbf{L_1}$ (that we believe not to be in **P**) to it. Conclude $\mathbf{L_2}$ *not* in **P** if $\mathbf{L_1}$ *not* in **P** (how?)
  - e.g. satisfiability ≤ graph 3-coloring
  - formalizes "$\mathbf{L_2}$ as hard as $\mathbf{L_1}$"

10

Reductions

- Example reduction:
  - 3SAT = { φ : φ is a 3-CNF Boolean formula that has a satisfying assignment }

    (3-CNF = AND of OR of ≤ 3 literals)
  - IS = { (G, k) | G is a graph with an independent set $V' \subseteq V$ of size $\geq k$ }

    (ind. set = set of vertices no two of which are connected by an edge)

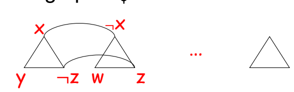
11

Ind. Set is NP-complete

The reduction f: given

$$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$$

we produce graph G_φ:

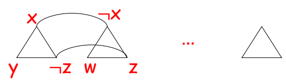


- one triangle for each of m clauses
- edge between every pair of contradictory literals
- set k = m

12

Reductions

$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$



- Claim: φ has a satisfying assignment if and only if G has an independent set of size at least k
  - proof?
- Conclude that 3SAT ≤ IS.

Completeness

- complexity class **C**
- language L is **C-complete** if
  - L is in **C**
  - every language in **C** reduces to L
- very important concept
- formalizes "L is hardest problem in complexity class **C**"

Completeness

- Completeness allows us to reason about the entire class by thinking about a single concrete problem
- related concept: language L is **C-hard** if
  - every language in **C** reduces to L

Completeness

- May ask: how to show *every* language in **C** reduces to L?
  - in practice, shown by reducing **known C-complete** problem to L
  - often not hard to find "1st" C-complete language, but it might not be "natural"

Completeness

- Example:
  NP = the set of languages L where
  $L = \{ x : \exists\, y, |y| \leq |x|^k, (x, y) \in R \}$
  and R is a language in **P.**
  one **NP**-complete language "bounded halting":
  BH = $\{(M, x, 1^k, 1^m) : \exists\, y, |y| \leq |x|^k$ s.t. M accepts (x, y) in at most m steps $\}$
- challenge is to find **natural** complete problem
- Cook 71 : 3-SAT **NP**-complete

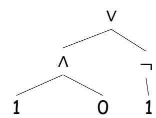
Summary

- problems
  - function, decision
  - language = set of strings
- complexity class = set of languages
- efficient computation identified with efficient computation on Turing Machine
  - single-tape, multi-tape
  - diagonalization technique: HALT undecidable
- TIME and SPACE classes
- reductions
- **C**-completeness, **C**-hardness

Time and Space

A motivating question:

– Boolean formula with n nodes
– evaluate using O(log n) space?

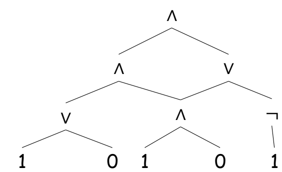


• depth-first traversal requires storing intermediate values

• idea: short-circuit ANDs and ORs when possible

Time and Space

- Can we evaluate an n node Boolean **circuit** using O(log n) space?



Time and Space

- Recall:
  - **TIME(f(n))**, **SPACE(f(n))**
- Questions:
  - how are these classes related to each other?
  - how do we define **robust** time and space classes?
  - what problems are contained in these classes? complete for these classes?

Outline

- Why big-oh? Linear Speedup Theorem
- Hierarchy Theorems
- Robust Time and Space Classes
- Relationships between classes
- Some complete problems

Linear Speedup

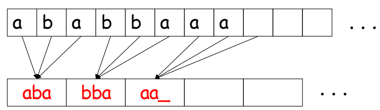
Theorem: Suppose TM M decides language L in time f(n). Then for any $\epsilon > 0$, there exists TM M' that decides L in time

$$\epsilon f(n) + n + 2.$$

- Proof:
  - simple idea: increase "word length"
  - M' will have
    - one more tape than M
    - m-tuples of symbols of M
      $\Sigma_{new} = \Sigma_{old} \cup \Sigma_{old}^m$
    - many more states

Linear Speedup

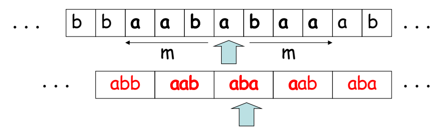
- part 1: compress input onto fresh tape



25

Linear Speedup

- part 2: simulate M, m steps at a time



  – 4 (L,R,R,L) steps to read relevant symbols, "remember" in state
  – 2 (L,R or R,L) to make M's changes

26

Linear Speedup

- accounting:
  – part 1 (copying): n + 2 steps
  – part 2 (simulation): 6 (f(n)/m)
  – set m = $6/\epsilon$
  – total: $\epsilon f(n) + n + 2$

Theorem: Suppose TM M decides language L in space f(n). Then for any $\epsilon > 0$, there exists TM M' that decides L in space $\epsilon f(n) + 2$.

- Proof: same.

27

Time and Space

- Moral: big-oh notation **necessary** given our model of computation
  – Recall: f(n) = O(g(n)) if there exists c such that f(n) ≤ c g(n) for all sufficiently large n.
  – TM model incapable of making distinctions between time and space usage that differs by a constant.
- In general: interested in course distinctions not affected by model
  – e.g. simulation of k-string TM running in time f(n) by single-string TM running in time $O(f(n)^2)$

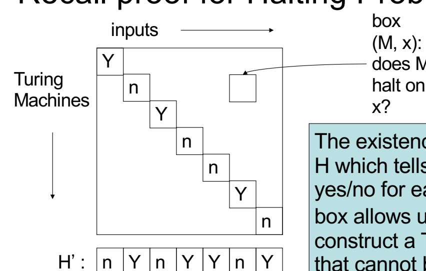
28

Hierarchy Theorems

- Does **genuinely** more time permit us to decide new languages?
- how can we construct a language L that is *not* in **TIME(f(n))…**
  - idea: same as "HALT undecidable" diagonalization and simulation

29

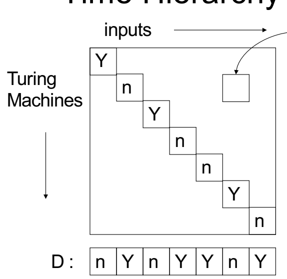
Recall proof for Halting Problem



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.

30

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

Time Hierarchy Theorem

__**Theorem**__ (Time Hierarchy Theorem): For every *proper complexity function* $f(n) \geq n$:

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

- more on "proper complexity functions" later…

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)

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.

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

Proof of Time Hierarchy Theorem

- 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.
- Important detail: need to initialize tape 3 in time O(f(n))

Proper Complexity Functions

- Definition: f is a **proper complexity function** if
  - f(n) ≥ f(n-1) for all n
  - there exists a TM M that outputs exactly f(n) symbols on input 1^n, and runs in time O(f(n) + n) and space O(f(n)).

CS151 Lecture 2 37

37

Proper Complexity Functions

- includes all reasonable functions we will work with
  - log n, $\sqrt{n}$, n^2, 2^n, n!, …
  - if f and g are proper then f + g, fg, f(g), f^g, 2^g are all proper
- can mostly ignore, but be aware it is a genuine concern:
- **Theorem**: ∃ non-proper f such that **TIME(f(n)) = TIME($2^{f(n)}$)**.

CS151 Lecture 2 38

38

Hierarchy Theorems

- Does **genuinely** more space permit us to decide new languages?

Theorem (Space Hierarchy Theorem): For every *proper complexity function* f(n) ≥ log n:

SPACE(f(n)) $\subsetneq$ SPACE(f(n)log f(n)).

- Proof: same ideas.

CS151 Lecture 2 39

39

Robust Time and Space Classes

- What is meant by "robust" class?
  - no formal definition
  - reasonable changes to model of computation shouldn't change class
  - should allow "modular composition" – calling subroutine in class (for classes closed under complement…)

CS151 Lecture 2 40

40

Robust Time and Space Classes

- Robust time and space classes:

L = SPACE(log n)

PSPACE = $\cup_k$ SPACE(n^k)

P = $\cup_k$ TIME(n^k)

EXP = $\cup_k$ TIME(2^{n^k})

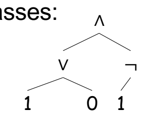
CS151 Lecture 2 41

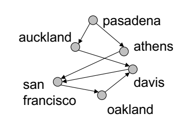
41

Time and Space Classes

- Problems in these classes:

L : FVAL, integer multiplication, most reductions…





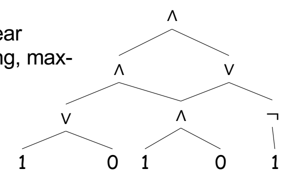
PSPACE : generalized geography, 2-person games…

CS151 Lecture 2 42

42

Time and Space Classes

P : CVAL, linear programming, max-flow…



EXP : SAT, all of NP and much more…

43

Relationships between classes

- How are these four classes related to each other?
- Time Hierarchy Theorem implies

  P ⊊ EXP

  – $P \subseteq TIME(2^n) \subsetneq TIME(2^{(2n)^3}) \subseteq EXP$
- Space Hierarchy Theorem implies

  L ⊊ PSPACE

  – $L = SPACE(\log n) \subsetneq SPACE(\log^2 n) \subseteq PSPACE$

44

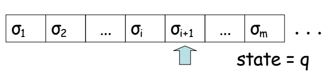
Relationships between classes

- Easy: **P ⊆ PSPACE**
- L vs. P, PSPACE vs. EXP ?

45

Relationships between classes

- Useful convention: Turing Machine configurations. Any point in computation



represented by string:

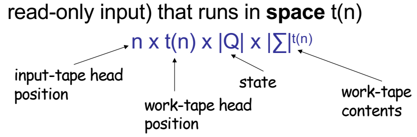
$C = \sigma_1 \sigma_2 \ldots \sigma_i \, \mathbf{q} \, \sigma_{i+1} \sigma_{i+2} \ldots \sigma_m$

- start configuration for single-tape TM on input x: $\mathbf{q_{start}} x_1 x_2 \ldots x_n$

46

Relationships between classes

- easy to tell if C **yields** C' in 1 step
- configuration graph: nodes are configurations, edge (C, C') iff C yields C' in one step
- # configurations for a 2-tape TM (work tape + read-only input) that runs in **space** t(n)



47

Relationships between classes

- if t(n) = c log n, at most

  $n \times (c \log n) \times c_0 \times c_1^{c \log n} \leq n^k$

  configurations.
- can determine if reach $\mathbf{q_{accept}}$ or $\mathbf{q_{reject}}$ from start configuration by exploring config. graph of size n^k (e.g. by DFS)
- Conclude: **L ⊆ P**

48

Relationships between classes

- if $t(n) = n^c$, at most
  $$n \times n^c \times c_0 \times c_1^{n^c} \leq 2^{n^k}$$
  configurations.
- can determine if reach $\mathbf{q_{accept}}$ or $\mathbf{q_{reject}}$ from start configuration by exploring config. graph of size 2^{n^k} (e.g. by DFS)
- Conclude: **PSPACE ⊆ EXP**

Relationships between classes

- So far:
  L ⊆ P ⊆ PSPACE ⊆ EXP
- believe all containments strict
- know **L ⊊ PSPACE, P ⊊ EXP**
- even before any mention of NP, two **major** unsolved problems:

$$\mathbf{L} \stackrel{?}{=} \mathbf{P} \qquad \mathbf{P} \stackrel{?}{=} \mathbf{PSPACE}$$