CS151 Complexity Theory

Lecture 3

April 6, 2004

Introduction

A motivating question:

- Can computers replace mathematicians?

L = { (x, 1^k) : statement x has a proof of length at most k }

Introduction

- This lecture:
  - nondeterminism
  - nondeterministic time classes
  - **NP**, **NP**-completeness, **P** vs. **NP**
  - **coNP**
  - NTIME Hierarchy
  - Ladner's Theorem

Nondeterminism

- Recall deterministic TM
  - **Q** finite set of states
  - ∑ alphabet including blank: "_"
  - $\mathbf{q_{start}}$, $\mathbf{q_{accept}}$, $\mathbf{q_{reject}}$ in Q
  - transition function:

$$\boldsymbol{\delta} : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, -\}$$

Nondeterminism

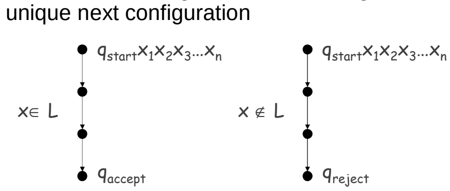
- **nondeterministic** Turing Machine:
  - **Q** finite set of states
  - ∑ alphabet including blank: "_"
  - $\mathbf{q_{start}}$, $\mathbf{q_{accept}}$, $\mathbf{q_{reject}}$ in Q
  - transition **relation**

$$\Delta \subset (Q \times \Sigma) \times (Q \times \Sigma \times \{L, R, -\})$$

- given current state and symbol scanned, several choices of what to do next.

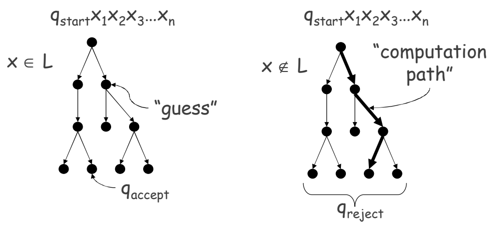
Nondeterminism

- deterministic TM: given current configuration, unique next configuration



- nondeterministic TM: given current configuration, several possible next configurations

Nondeterminism



- asymmetric accept/reject criterion

Nondeterminism

- all paths terminate
- time used: maximum length of paths from root
- space used: maximum # of work tape squares touched on any path from root

Nondeterminism

- **NTIME(f(n))** = languages decidable by a multi-tape NTM that runs for at most f(n) steps *on any computation path*, where n is the input length, and $f : \mathbf{N} \rightarrow \mathbf{N}$
- **NSPACE(f(n))** = languages decidable by a multi-tape NTM that touches at most f(n) squares of its work tapes *along any computation path*, where n is the input length, and $f : \mathbf{N} \rightarrow \mathbf{N}$

Nondeterminism

- Focus on time classes first:

$$\mathbf{NP} = \cup_k \mathbf{NTIME}(n^k)$$

$$\mathbf{NEXP} = \cup_k \mathbf{NTIME}(2^{n^k})$$

Poly-time verifiers

Very useful alternat[ive definition o]f NP: ["witness" or "certificate"]

Theorem: language L is in NP if a[nd only if] it is expressible as: [efficiently verifiable]

$$L = \{ x \mid \exists\, y, |y| \le |x|^k, (x, y) \in R \}$$

where R is a language in P.

- poly-time TM M_R deciding R is a "verifier"

Poly-time verifiers

- Example: 3SAT expressible as

3SAT = {φ : φ is a 3-CNF formula for which ∃ assignment A for which (φ, A) ∈ R}

R = {(φ, A) : A is a sat. assign. for φ}

  – satisfying assignment A is a "witness" of the satisfiability of φ (it "certifies" satisfiability of φ)
  – R is decidable in poly-time

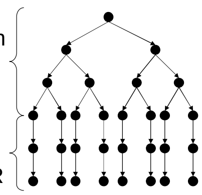
Poly-time verifiers

$L = \{ x \mid \exists\, y, |y| \le |x|^k, (x, y) \in R \}$

Proof: ($\Leftarrow$) give poly-time NTM deciding L

phase 1: "guess" y with $|x|^k$ nondeterministic steps

phase 2: decide if $(x, y) \in R$



Poly-time verifiers

Proof: ($\Rightarrow$) given $L \in$ NP, describe L as:

$L = \{ x \mid \exists\, y, |y| \le |x|^k, (x, y) \in R \}$

- L is decided by NTM M running in time n^k
- define the language
  - R = { (x, y) : y is an accepting computation history of M on input x}
- check: accepting history has length $\le |x|^k$
- check: R is decidable in polynomial time
- check: M accepts x iff $\exists y, |y| \le |x|^k, (x, y) \in R$

Why NP?

problem requirements

object we are seeking

- stic model of
- but, captures important computational feature of many problems:
  - exhaustive search wo
- contains **huge** number of natu problems
- many problems have form:
  - $L = \{ x \mid \exists\, y \text{ s.t. } (x, y) \in R \}$

efficient test: does y meet requirements?

Why NP?

- Why not **EXP**?
  - too strong!
  - important problems not complete.

Relationships between classes

- Easy: **P** $\subset$ **NP**, **EXP** $\subset$ **NEXP**
  - TM special case of NTM
- Recall: $L \in$ **NP** iff expressible as
  - $L = \{ x \mid \exists\, y, |y| \le |x|^k, (x, y) \in R \}$
- **NP** $\subset$ **PSPACE** (try all possible y)
- The central question:

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$$

recognizing a solution vs. finding a solution

NP-completeness

- Circuit SAT: given a Boolean circuit (gates $\wedge, \vee, \neg$), with variables $y_1, y_2, \ldots, y_m$ is there some assignment that makes it output 1?

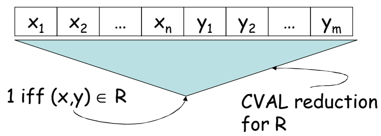
Theorem: Circuit SAT is **NP**-complete.

- Proof:
  - clearly in **NP**

NP-completeness

– Given $L \in$ **NP** of form

$L = \{ x \mid \exists y \text{ s.t. } (x, y) \in R \}$



– hardwire input x; leave y as variables

NEXP-completeness

- Succinct Circuit SAT: given a **succinctly encoded** Boolean circuit (gates $\wedge, \vee, \neg$), with variables $y_1, y_2, \ldots, y_m$ is there some assignment that makes it output 1?

Theorem: Succinct Circuit SAT is **NEXP**-complete.

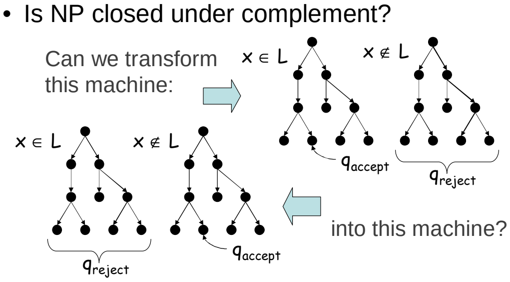
- Proof:
  – same trick as for Succinct CVAL **EXP**-complete.

Complement classes

- In general, if **C** is a complexity class
- **co-C** is the complement class, containing all complements of languages in C
  – $L \in$ **C** implies $(\Sigma^* - L) \in$ **co-C**
  – $(\Sigma^* - L) \in$ **C** implies $L \in$ **co-C**
- Some classes closed under complement:
  – e.g. co-P = P

coNP

- Is NP closed under complement?



coNP

- "proof system" interpretation:
- Recall: $L \in$ **NP** iff expressible as

$L = \{ x \mid \exists y, |y| \leq |x|^k, (x, y) \in R \}$

"proof" "proof verifier"

- languages in **NP** have "short proofs"
- **coNP** captures (in its complete problems) problems least likely to have "short proofs".
  – e.g., UNSAT is **coNP**-complete

coNP

- **P** = **NP** implies **NP** = **coNP**
- Belief:

NP ≠ coNP

– another major open problem

NTIME Hierarchy Theorem

Theorem (Nondeterministic Time Hierarchy Theorem):

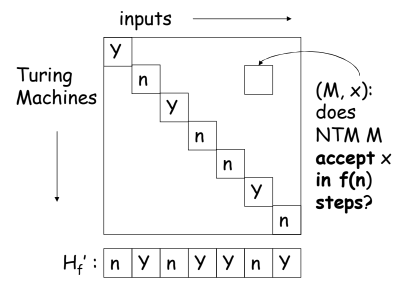
For every *proper complexity function* f(n) ≥ n, and $g(n) = \omega(f(n+1))$,

$$\mathbf{NTIME(f(n))} \subsetneq \mathbf{NTIME(g(n))}.$$

NTIME Hierarchy Theorem

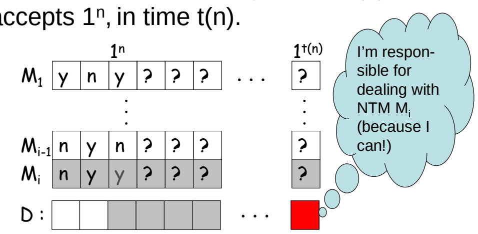
Proof attempt #1:

(what's wrong?)



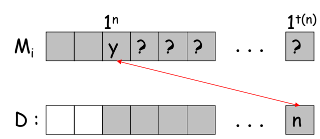
NTIME Hierarchy Theorem

- Let t(n) be large enough so that can decide if NTM M running in time f(n) accepts 1^n, in time t(n).



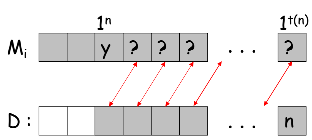
NTIME Hierarchy Theorem

- Enough time on input $1^{t(n)}$ to do the *opposite* of $M_i(1^n)$:



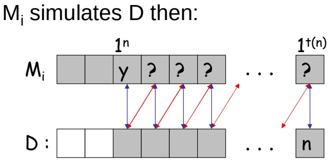
NTIME Hierarchy Theorem

- For k in [n…t(n)] can to do *same* as $M_i(1^{k+1})$ on input 1^k



NTIME Hierarchy Theorem

- Did we diagonalize against M_i?
  - if M_i simulates D then:



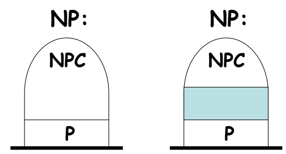
  - equality along all arrows.
  - contradiction.

NTIME Hierarchy Theorem

- General scheme:
  - interval [1...t(1)] kills M_1
  - interval [t(1)…t(t(1))] kills M_2
  - interval $[t^{i-1}(1)…t^i(1)]$ kills M_i
- Running time of D on 1^n: f(n+1) + time to compute interval containing n
- conclude D in **NTIME(g(n))**

Ladner's Theorem

- Assuming **P** ≠ **NP**, what does the world (inside **NP**) look like?



Ladner's Theorem

<u>Theorem</u> (Ladner): If **P** ≠ **NP**, then there exists L ∈ **NP** that is neither in **P** nor **NP**-complete.

- Proof: "lazy diagonalization"
  - deal with similar problem as in NTIME Hierarchy proof

Ladner's Theorem

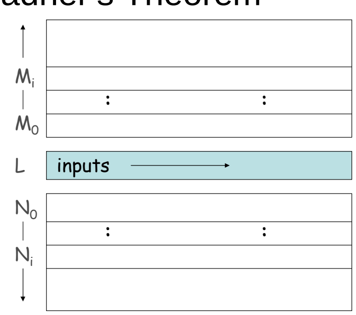
- Can enumerate (TMs deciding) all languages in **P**.
  - enumerate TMs so that each machine appears infinitely often
  - add clock to M_i so that it runs in at most n^i steps

Ladner's Theorem

- Can enumerate (TMs deciding) all **NP**-complete languages.
  - enumerate TMs f_i computing all polynomial-time functions
  - machine N_i decides language SAT reduces to via f_i *if f_i is reduction*, else SAT (details omitted…)

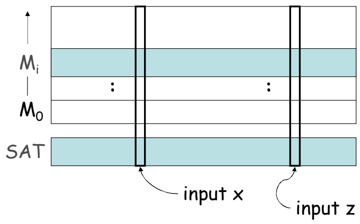
Ladner's Theorem

- Our goal: L ∈ **NP** that is neither in **P** nor **NP**-complete



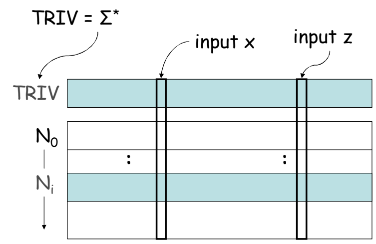
Ladner's Theorem

- Top half, assuming **P ≠ NP:**
  - focus on M_i
  - for any x, can always find some z ≥ x on which M_i and SAT differ (why?)



Ladner's Theorem

- Bottom half, assuming **P ≠ NP:**
  - focus on N_i
  - for any x, can always find some z ≥ x on which N_i and SAT differ (why?)

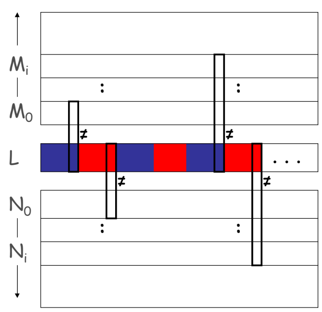


Ladner's Theorem

- Try to "merge":

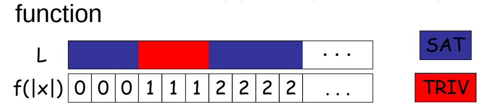
SAT TRIV

- on input x, either
  - answer SAT(x)
  - answer TRIV(x)
- if can decide which one in **P**, L ∈ **NP**



Ladner's Theorem

- General scheme: f(n) slowly increasing function



  - f(|x|) even: answer SAT(x)
  - f(|x|) odd: answer TRIV(x)
- notice choice only depends on length of input… that's OK

Ladner's Theorem

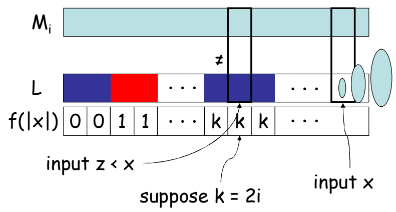
- 1^{st} attempt to define f(n)
- "eager f(n)": increase at 1^{st} opportunity
- Inductive definition: f(0) = 0; f(n) =
  - if f(n-1) = 2i, trying to kill M_i
    - if $\exists$ z < 1^n s.t. $M_i(z) \neq$ SAT(z), then f(n) = f(n-1) + 1; else f(n) = f(n-1)
  - if f(n-1) = 2i+1, trying to kill N_i
    - if $\exists$ z < 1^n s.t. $N_i(z) \neq$ TRIV(z), then f(n) = f(n-1) + 1; else f(n) = f(n-1)

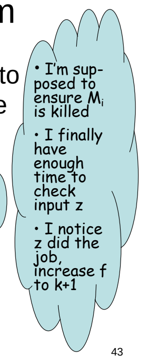
Ladner's Theorem

- Problem: eager f(n) too difficult to compute
- on input of length n,
  - look at all strings z of length < n
  - compute SAT(z) or N_i(z) for each !
- Solution: "lazy" f(n)
  - on input of length n, only run for 2n steps
  - if enough time to see should increase (over f(n-1)), do it; else, stay same
  - (alternate proof: give explicit f(n) that grows slowly enough…)

Ladner's Theorem

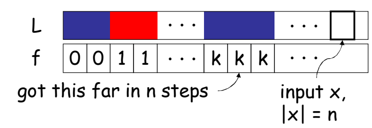
- Key: n eventually large enough to notice completed previous stage





Ladner's Theorem

- Inductive definition of f(n)
  - f(0) = 0
  - f(n): for n steps compute f(0), f(1), f(2),…



Ladner's Theorem

- if k = 2i:
  - for n steps try (lex order) to find z s.t. SAT(z) ≠ M_i(z) and f(|z|) even
  - if found, f(n) = f(n-1)+1 else f(n-1)
- if k = 2i + 1:
  - for n steps try (lex order) to find z s.t. TRIV(z) ≠ N_i(z) and f(|z|) odd
  - if found, f(n) = f(n-1)+1 else f(n-1)

Ladner's Theorem

- Finishing up:

  L = { x | x ∈ SAT if f(|x|) even,
  x ∈ TRIV if f(|x|) odd }

- L ∈ **NP** since f(|x|) can be computed in O(n) time

Ladner's Theorem

- suppose M_i decides L
  - f gets stuck at 2i
  - L ≡ SAT for z : |z| > n_o
  - implies SAT ∈ **P**. Contradiction.
- suppose N_i decides L
  - f gets stuck at 2i+1
  - L ≡ TRIV for z : |z| > n_o
  - implies L(N_i) ∈ **P**. Contradiction.
- (last of diagonalization…)

Summary

- nondeterminism
- nondeterministic time classes:

  NP, coNP, NEXP

- NTIME Hierarchy Theorem:

  NP ≠ NEXP

- major open questions:

  P $\stackrel{?}{=}$ NP **NP $\stackrel{?}{=}$ coNP**

Summary

- **NP**-"intermediate" problems
  - technique: delayed diagonalization
- complete problems:
  - circuit SAT is **NP**-complete
  - UNSAT is **coNP**-complete
  - succinct circuit SAT is **NEXP**-complete

Summary