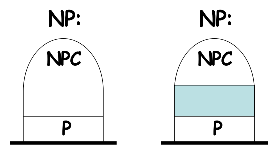
CS151 Complexity Theory

Lecture 4
April 13, 2023

1

Ladner's Theorem

- Assuming $\mathbf{P} \neq \mathbf{NP}$, what does the world (inside **NP**) look like?



2

Ladner's Theorem

Theorem (Ladner): If $\mathbf{P} \neq \mathbf{NP}$, then there exists $L \in \mathbf{NP}$ that is neither in **P** nor **NP**-complete.

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

3

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

4

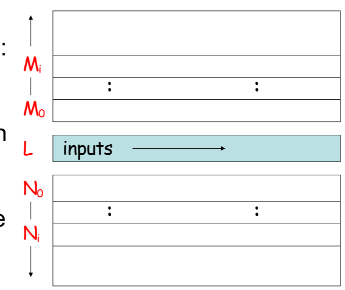
Ladner's Theorem

- 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…)

5

Ladner's Theorem

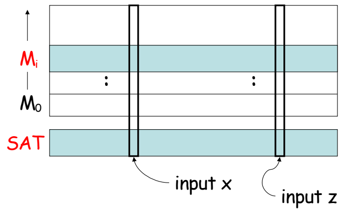
- Our goal: $L \in \mathbf{NP}$ that is neither in **P** nor **NP**-complete



6

Ladner's Theorem

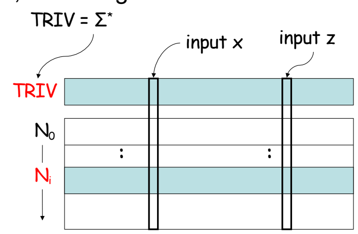
- Top half, assuming **P** ≠ **NP**:
  - focus on M_i
  - for any x, can always find some z ≥ x on which M_i and SAT differ (why?)



7

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 TRIV differ (why?)



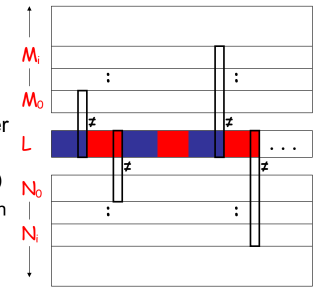
8

Ladner's Theorem

- Try to "merge":

SAT TRIV

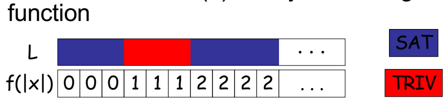
- on input x, either
  - answer SAT(x)
  - answer TRIV(x)
- if can decide which one in **P**, L ∈ **NP**



9

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

10

Ladner's Theorem

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

11

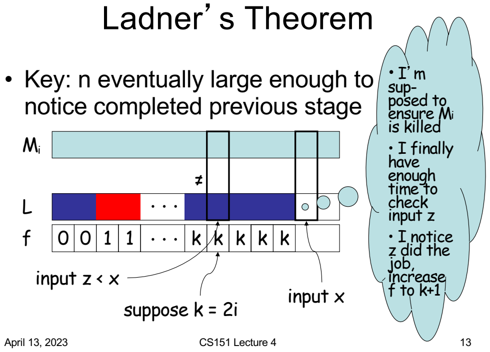
Ladner's Theorem

- 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…)

12

Ladner's Theorem

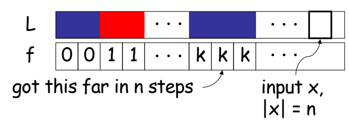
- Key: n eventually large enough to notice completed previous stage



13

Ladner's Theorem

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



14

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)

15

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

16

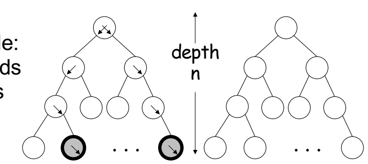
Ladner's Theorem

- 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…)

17

A puzzle

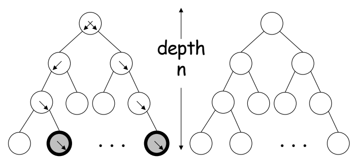
A puzzle: two kinds of trees



- cover up nodes with c colors
- promise: never color "arrow" same as "blank"
- determine which kind of tree in poly(n, c) steps?

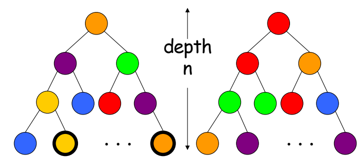
18

A puzzle



19

A puzzle



20

Introduction

- Ideas
  - depth-first-search; stop if see
  - how many times may we see a given “arrow color”?
    - at most n+1
  - pruning rule?
    - if see a color > n+1 times, it can’t be an arrow node; prune
  - \# nodes visited before know answer?
    - at most c(n+2)

21

Sparse languages and **NP**

- We often say **NP**-compete languages are “hard”
- More accurate: **NP**-complete languages are “expressive”
  - lots of languages reduce to them

22

Sparse languages and **NP**

- Sparse language: one that contains at most poly(n) strings of length ≤ n
- not very expressive – can we show this cannot be **NP**-complete (assuming **P** ≠ **NP**) ?
  - yes: Mahaney ’82 (homework problem)
- Unary language: subset of 1* (at most n strings of length ≤ n)

23

Sparse languages and **NP**

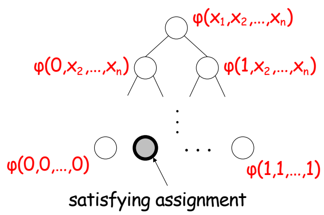
Theorem (Berman ’78): if a unary language is **NP**-complete then **P** = **NP**.

- Proof:
  - let $U \subseteq 1^*$ be a unary language and assume SAT ≤ U via reduction R
  - $\varphi(x_1,x_2,\ldots,x_n)$ instance of SAT

24

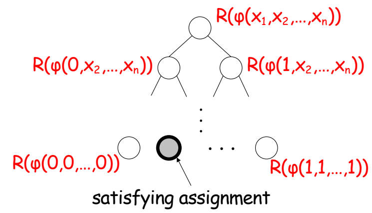
Sparse languages and **NP**

– self-reduction tree for φ:



Sparse languages and **NP**

– applying reduction R:



Sparse languages and **NP**

- on input of length $m = |\varphi(x_1,x_2,\ldots,x_n)|$, R produces string of length $\leq p(m)$
- R's different outputs are "colors"
  - 1 color for strings not in 1^*
  - at most p(m) other colors
- puzzle solution ⇒ can solve SAT in poly(p(m)+1, n+1) = poly(m) time!

Summary

- nondeterministic time classes:

  NP, coNP, NEXP
- NTIME Hierarchy Theorem:

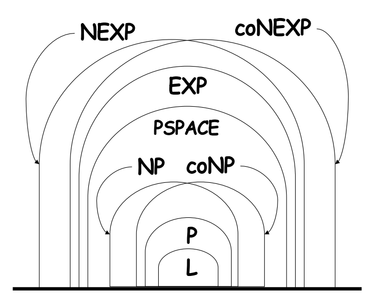
  NP ≠ NEXP
- major open questions:

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

Summary

- **NP**-"intermediate" problems (unless **P** = **NP**)
  - technique: delayed diagonalization
- unary languages not **NP**-complete (unless **P** = **NP**)
  - true for sparse languages as well (homework)
- complete problems:
  - circuit SAT is **NP**-complete
  - UNSAT is **coNP**-complete
  - succinct circuit SAT is **NEXP**-complete

Summary



Remainder of lecture

- nondeterminism applied to space
- reachability
- two surprises:
  - Savitch's Theorem
  - Immerman/Szelepcsényi Theorem

31

Nondeterministic space

- **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 :**N** → **N**

32

Nondeterministic space

- Robust nondeterministic space classes:

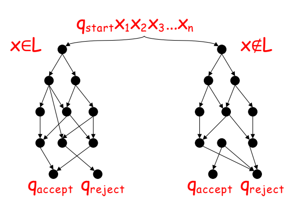
NL = NSPACE(log n)

NPSPACE = $\cup_k$ **NSPACE(n^k)**

33

Reachability

- Recall: at most n^k configurations of given NTM M running in **NSPACE**(log n).
  - easy to determine if C yields C' in one step
  - configuration graph for M on input x:



34

Reachability

- Conclude: **NL ⊆ P**
  - and **NPSPACE ⊆ EXP**
- S-T-Connectivity (STCONN): given directed graph G = (V, E) and nodes s, t, is there a path from s to t ?

<u>Theorem</u>: STCONN is **NL**-complete under logspace reductions.

35

Reachability

- Proof:
  - in **NL**: guess path from s to t one node at a time
  - given L ∈ **NL** decided by NTM M construct configuration graph for M on input x (can be done in logspace)
  - s = starting configuration; t = q_{accept}

36

Two startling theorems

- Strongly believe **P ≠ NP**
- nondeterminism seems to add enormous power
- for space: Savitch '70:

NPSPACE = PSPACE

and

NL ⊆ SPACE($\log^2 n$)

Two startling theorems

- Strongly believe **NP ≠ coNP**
- seems impossible to convert existential into universal
- for space: Immerman/Szelepscényi '87/'88:

NL = coNL