CS151 Complexity Theory

Lecture 5
April 18, 2023

1

Two startling theorems

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

NPSPACE = PSPACE

and

NL ⊆ SPACE($\log^2 n$)

2

Two startling theorems

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

NL = coNL

3

Savitch's Theorem

Theorem: STCONN ⊆ **SPACE($\log^2$ n)**

- Corollary: **NL ⊆ SPACE($\log^2$n)**
- Corollary: **NPSPACE = PSPACE**

4

Proof of Theorem

– input: G = (V, E), two nodes s and t
– recursive algorithm:

```
/* return true iff path from x to y of length at most 2^i */
PATH(x, y, i)
 if i = 0 return ( x = y or (x, y) ∈ E )          /* base case */
 for z in V
    if PATH(x, z, i-1) and PATH(z, y, i-1) return(true);
 return(false);
end
```


5

Proof of Theorem

– answer to STCONN: PATH(s, t, log n)
– space used:
  - (depth of recursion) x (size of "stack record")
– depth = log n
– claim stack record: "(x, y, i)" sufficient
  - size O(log n)
– when return from PATH(a, b, i) can figure out what to do next from record (a, b, i) and previous record

6

Nondeterministic space

- Robust nondeterministic space classes:

NL = NSPACE(log n)

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

7

Second startling theorem

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

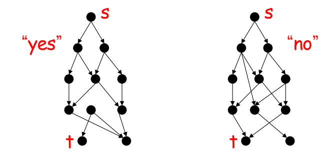
NL = coNL

8

I-S Theorem

<u>**Theorem**</u>: ST-NON-CONN ∈ **NL**

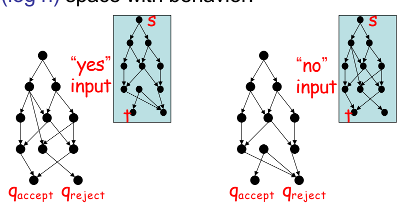
- Proof: slightly tricky setup:
  - input: G = (V, E), two nodes s, t



9

I-S Theorem

- want nondeterministic procedure using only O(log n) space with behavior:



10

I-S Theorem

- observation: given **count** of # nodes reachable from s, can solve problem
  - for each v ∈ V, *guess* if it is reachable
  - if yes, *guess* path from s to v
    - if guess doesn’t lead to v, reject.
    - if v = t, reject.
    - else counter++
  - if counter = **count** accept

11

I-S Theorem

- every computation path has sequence of guesses…
- only way computation path can lead to accept:
  - correctly guessed reachable/unreachable for each node v
  - correctly guessed path from s to v for each reachable node v
  - saw *all* reachable nodes
  - t not among reachable nodes

12

I-S Theorem

- R(i) = # nodes reachable from s in at most i steps
- R(0) = 1: node s
- we will compute R(i+1) from R(i) using O(log n) space and nondeterminism
- computation paths with “bad guesses” all lead to reject

13

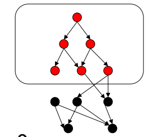
I-S Theorem

- Outline: in n phases, compute
  R(1), R(2), R(3), … R(n)
- only O(log n) bits of storage between phases
- in end, lots of computation paths that lead to reject
- only computation paths that survive have computed correct value of R(n)
- apply observation.

14

I-S Theorem

- computing R(i+1) from R(i):



R(i) = R(2) = 6

- Initialize R(i+1) = 0
- For each v ∈ V, *guess* if v reachable from s in at most i+1 steps

15

I-S Theorem

- if “yes”, *guess* path from s to v of at most i+1 steps. Increment R(i+1)
- if “no”, visit R(i) nodes reachable in at most i steps, check that none is v or adjacent to v
  - for u ∈ V *guess* if reachable in ≤ i steps; *guess* path to u; counter++
  - KEY: if counter ≠ R(i), reject
  - at this point: **can be sure v not reachable**

16

I-S Theorem

- correctness of procedure:
- two types of errors we can make
- (1) might guess v is reachable in at most i+1 steps when it is not
  - won’t be able to guess path from s to v of correct length, so we will reject.

“easy” type of error

17

I-S Theorem

- (2) might guess v is **not** reachable in at most i+1 steps when it is
  - then must **not** see v or neighbor of v while visiting nodes reachable in i steps.
  - but forced to visit R(i) distinct nodes
  - therefore must try to visit node v that is **not** reachable in ≤ i steps
  - won’t be able to guess path from s to v of correct length, so we will reject.

“easy” type of error

18

Summary

- nondeterministic space classes
  NL and **NPSPACE**

- ST-CONN **NL**-complete

Summary

- Savitch: **NPSPACE = PSPACE**
  - Proof: ST-CONN ∈ **SPACE($\log^2$ n)**
  - open question:

    NL = L?

- Immerman/Szelepcsényi : **NL = coNL**
  - Proof: ST-NON-CONN ∈ **NL**

Introduction

Power from an unexpected source?

- we know **P ≠ EXP**, which implies no poly-time *algorithm* for Succinct CVAL
- poly-size Boolean *circuits* for Succinct CVAL ??

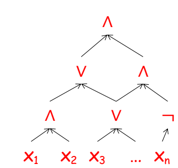
Does **NP** have linear-size, log-depth Boolean circuits ??

Outline

- Boolean circuits and formulas
- uniformity and advice
- the **NC** hierarchy and parallel computation
- the quest for circuit lower bounds
- a lower bound for formulas

Boolean circuits

- **circuit** C
  - directed acyclic graph
  - nodes: AND (∧); OR (∨); NOT (¬); variables x_i



- C computes function $f:\{0,1\}^n \rightarrow \{0,1\}$ in natural way
  - identify C with function f it computes

Boolean circuits

- **size** = # gates
- **depth** = longest path from input to output
- **formula (or expression)**: graph is a tree

- every function $f:\{0,1\}^n \rightarrow \{0,1\}$ computable by a circuit of size at most $O(n2^n)$
  - AND of n literals for each x such that f(x) = 1
  - OR of up to 2^n such terms

Circuit families

- circuit works for specific input length
- we're used to $f:\Sigma^* \rightarrow \{0,1\}$
- circuit **family** : a circuit for each input length $C_1, C_2, C_3, \ldots$ = "$\{C_n\}$"
- "$\{C_n\}$ computes f" iff for all x

$$C_{|x|}(x) = f(x)$$

- "$\{C_n\}$ decides L", where L is the language associated with f

Connection to TMs

- given TM M running in time t(n) decides language L
- can build circuit family $\{C_n\}$ that decides L
  - size of $C_n = O(t(n)^2)$
  - Proof: CVAL construction

- Conclude: $L \in$ **P** implies family of polynomial-size circuits that decides L

Connection to TMs

- other direction?

- A poly-size circuit family:
  - $C_n = (x_1 \vee \neg x_1)$ if M_n halts
  - $C_n = (x_1 \wedge \neg x_1)$ if M_n loops

- decides (unary version of) HALT!
- oops…

Uniformity

- Strange aspect of circuit family:
  - can "encode" (potentially uncomputable) information in family specification
- solution: uniformity – require specification is simple to compute

Definition: circuit family $\{C_n\}$ is **logspace uniform** iff TM M outputs C_n on input 1^n and runs in O(log n) space

Uniformity

Theorem: **P** = languages decidable by logspace uniform, polynomial-size circuit families $\{C_n\}$.

- Proof:
  - already saw ($\Rightarrow$)
  - ($\Leftarrow$) on input x, generate $C_{|x|}$, evaluate it and accept iff output = 1

TMs that take advice

- family $\{C_n\}$ without uniformity constraint is called "non-uniform"
- regard "non-uniformity" as a limited resource just like time, space, as follows:
  - add read-only "advice" tape to TM M
  - M "decides L with advice A(n)" iff

    $M(x, A(|x|))$ accepts $\Leftrightarrow x \in L$
  - note: A(n) depends only on |x|

31

TMs that take advice

- Definition: **TIME(t(n))/f(n)** = the set of those languages L for which:
  - there exists A(n) s.t. $|A(n)| \leq f(n)$
  - TM M decides L with advice A(n) in time t(n)
- most important such class:

  $$\textbf{P/poly} = \cup_k \textbf{TIME}(n^k)/n^k$$

32

TMs that take advice

Theorem: $L \in$ **P/poly** iff L decided by family of (non-uniform) polynomial size circuits.

- Proof:
  - ($\Rightarrow$) C_n from CVAL construction; hardwire advice A(n)
  - ($\Leftarrow$) define A(n) = description of C_n; on input x, TM simulates $C_{|x|}(x)$

33

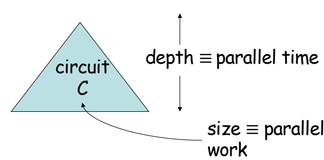
Approach to P/NP

- Believe **NP** $\neq$ **P**
  - equivalent: "**NP** does not have uniform, polynomial-size circuits"
- *Even believe* **NP** $\not\subset$ **P/poly**
  - equivalent: "**NP** (or, e.g. SAT) does not have polynomial-size circuits"
  - implies **P** $\neq$ **NP**
  - many believe: best hope for **P** $\neq$ **NP**

34

Parallelism

- uniform circuits allow refinement of polynomial time:



35

Parallelism

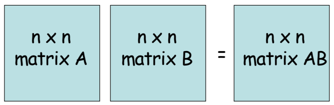
- the **NC** ("**N**ick's **C**lass") Hierarchy (of logspace uniform circuits):

  $\mathbf{NC_k}$ = $O(\log^k n)$ depth, poly(n) size

  $\mathbf{NC} = \cup_k \mathbf{NC_k}$
- captures "efficiently parallelizable problems"
- not realistic? overly generous
- OK for proving non-parallelizable

36

Matrix Multiplication



- what is the parallel complexity of this problem?
  - work = poly(n)
  - time = $\log^k(n)$? (which k?)

Matrix Multiplication

- two details
  - arithmetic matrix multiplication…

    $A = (a_{i,k})$ $B = (b_{k,j})$ $(AB)_{i,j} = \Sigma_k (a_{i,k} \times b_{k,j})$

    … vs. Boolean matrix multiplication:

    $A = (a_{i,k})$ $B = (b_{k,j})$ $(AB)_{i,j} = \vee_k (a_{i,k} \wedge b_{k,j})$
  - single output bit: to make matrix multiplication a language: on input A, B, (i, j) output $(AB)_{i,j}$

Matrix Multiplication

- Boolean Matrix Multiplication is in **NC_1**
  - level 1: compute n ANDS: $a_{i,k} \wedge b_{k,j}$
  - next log n levels: tree of ORS
  - n^2 subtrees for all pairs (i, j)
  - select correct one and output

Boolean formulas and NC_1

- Previous circuit is actually a formula. This is no accident:

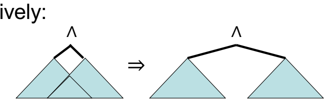
Theorem: L ∈ **NC_1** iff decidable by polynomial-size uniform* family of Boolean formulas.

* DSPACE($\log^2$ n)-uniform

Note: we measure formula size by leaf-size.

Boolean formulas and NC_1

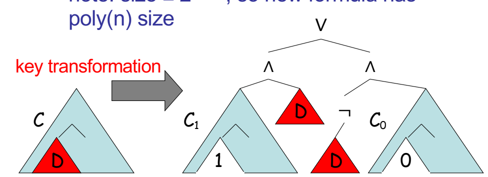
- Proof:
  - (⇒) convert **NC_1** circuit into formula
    - recursively:



    - note: logspace transformation (stack depth log n, stack record 1 bit – "left" or "right")

Boolean formulas and NC_1

- (⇐) convert formula of size n into formula of depth O(log n)
  - note: size ≤ 2^{depth}, so new formula has poly(n) size



Boolean formulas and $\mathbf{NC_1}$

- D any minimal subtree with size at least n/3
  - implies size(D) ≤ 2n/3
- define T(n) = maximum depth required for any size n formula
- C_1, C_0, D all size ≤ 2n/3

$$T(n) \leq T(2n/3) + 3$$

$$\text{implies } T(n) \leq O(\log n)$$

Relation to other classes

- Clearly **NC** ⊆ **P**
  - recall **P** ≡ uniform poly-size circuits
- $\mathbf{NC_1} \subseteq \mathbf{L}$
  - on input x, compose logspace algorithms for:
    - generating $C_{|x|}$
    - converting to formula
    - FVAL

Relation to other classes

- $\mathbf{NL} \subseteq \mathbf{NC_2}$: S-T-CONN ∈ $\mathbf{NC_2}$
  - given G = (V, E), vertices s, t
  - A = adjacency matrix (with self-loops)
  - $(A^2)_{i,j} = 1$ iff path of length ≤ 2 from node i to node j
  - $(A^n)_{i,j} = 1$ iff path of length ≤ n from node i to node j
  - compute with depth log n tree of Boolean matrix multiplications, output entry s, t
  - $\log^2 n$ depth total

NC vs. P

- can every efficient algorithm be efficiently parallelized?

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

- **P**-complete problems least-likely to be parallelizable
  - if **P**-complete problem is in **NC**, then **P** = **NC**
  - Why?

    we use logspace reductions to show problem **P**-complete; **L** in **NC**

NC vs. P

- can every uniform, poly-size Boolean circuit family be converted into a uniform, poly-size Boolean formula family?

$$\mathbf{NC_1} \stackrel{?}{=} \mathbf{P}$$