DD2445 LECTURE 5

Last time

(1) Diagonalization

Table of all TMs running within specified resource bounds and all inputs

	1	2	3	4	5	6
M_1	$\neq$					
M_2		$\neq$				4
M_3			$\neq$			
M_4				$\neq$		
M_5					$\neq$	

cell (i,j) = output of M_i on j

Prove that certain language L is not decidable within resource bound by showing that in each row one cell contains wrong answer

Namely on the diagonal – DIAGONALIZATION

(2) Time hierarchy theorem
More computation time $\Rightarrow$ more problems solved

(3) Ladner's theorem
If $P \neq NP$, then there are (infinitely many) complexity classes in between

(Only sketched proof in class – details on problem set)

What exactly is "proof by diagonalization"?

How far can diagonalizing techniques take us? Maybe be prove $P \neq NP$ if work hard enough?!

Provably NO, unfortunately...

Our diagonalization proofs relied on:

(I) Representation of TMs by strings (every integer is a TM)

(II) Efficient simulation of TM by other (universal) TM without much overhead (in time or space)

Such an approach works even if we give TM access to certain subroutines for free and don't charge for time spent in such calls during running time analysis

Such TMs are known as "oracle Turing machines"

Might sound very strange - for us it will just be a program with a subroutine that can be called free of charge

DEF Oracle TM (informal - see textbook)

An oracle Turing machine is a usual TM except

- has special read-write oracle tape
- has special oracle state

To execute M, specify oracle language $O \subseteq \{0,1\}^*$

At any time, M can

- write string y on oracle tape (takes several steps)
- jump to oracle state (one time step)
- get answer whether $y \in O$ or not written as bit 1 if $y \in O$, 0 if $y \notin O$ on oracle tape (one time step)

Output of M on x when run with oracle O denoted $M^O(x)$

Nondeterministic oracle machines defined analogously

$$P^O = \left\{ \begin{array}{l} \text{all languages decidable by poly-time} \\ \text{deterministic TM with oracle } O \end{array} \right\}$$

$$NP^O = \left\{ \begin{array}{l} \text{all languages decidable by} \\ \text{poly-time nondeterministic TM} \\ \text{with oracle } O \end{array} \right\}$$

Also say that TM M has "oracle access" to language O.

Examples

(1) $UNSAT \in P^{SAT}$

Write down formula on oracle tape

Make query to SAT

Give the opposite answer as output

(2) If $O \in P$ then $P^O = P$

Oracle calls are not needed

Can compute answer by simulating machine deciding O in poly time

(3) $\mathcal{L}EXPCOM = \{ \langle M, x, 1^n \rangle : M$ outputs 1 on x within 2^n steps $\}$

Then $\boxed{P^{EXPCOM} = NP^{EXPCOM} = EXP}$

Recall $EXP = \bigcup_{c \in N} DTIME(2^{n^c})$

Proof Suppose $\mathcal{L} \in DTIME(2^{n^c})$ decided by M

Write down M, x, and then 1^{n^c} (i.e. n^c 1's) on oracle tape

Can be done in poly time

Q~~uery~~ EXPCOM and answer the same

$\Rightarrow \mathcal{L}EXP \subseteq P^{EXPCOM}$

$P^O \subseteq NP^O$ for any oracle language O (why?)

Suppose $\mathcal{L}' \in NP^{EXPCOM}$ decided by M' running in time $O(n^k)$

At most $2^{O(n^k)}$ nondeterministic choices which is exponential

At most that many ~~oracles~~ oracle calls — can also be computed in exponential time

Exponential × exponential = exponential, so $\mathcal{L}' \in EXP$.

Regardless of what the oracle O is

(I) and (II) holds for oracle TMs (if simulating machine is also given the oracle O)

O III

Hence, any theorem about TMs that uses only (I) + (II) holds for oracle TMs (the theorem RELATIVIZES).

The answer to $P \stackrel{?}{=} NP$ can't be a relativizing theorem, since there are oracles to flip the answer both ways!

$$\boxed{\begin{array}{l}\text{THEOREM (Baker, Gill, Solovay '75)} \\ \text{There exist oracles } A \text{ and } B \text{ s.t.} \\ \quad P^A = NP^A \text{ and } P^B \neq NP^B\end{array}}$$

Proof Set A to be EXPCOM

For any language B, let
$$U_B = \{ 1^n : \exists x \text{ s.t. } |x| = n \text{ and } x \in B \}$$

For any B, $U_B \in NP^B$
On input 1^n, guess x of length n, write on oracle tape, query B.

Want to build B s.t. $U_B \notin P^B$.
If so, proof finished

OIV

High-level intuition:

Any TM for U_B has to run in subexponential time.
Can only query vanishing small part of strings of length $\{0,1\}^n$ - exponentially many. Make sure any TM "queries the wrong strings."

<u>Construction of B</u>

M_i : Turing machine encoded by (binary expansion of) integer i

Construct B in stages. Stage i will make sure M_i doesn't solve U_B in time $\leq 2^n / 10$.
Initially $B := \emptyset$, $i := 1$.

<u>stage i:</u>

B contains finite # strings so far
Fix n s.t. no string of length $\geq n$ has had its status wrt B decided.
Run M_i on 1^n for $2^n/10$ steps.

Oracle <u>queries</u> y

<u>case 1</u> status of y decided in previous stages — answer accordingly

<u>case 2</u> status of y undecided — answer $y \notin B$

~~[illegible]~~

Suppose M_i finishes on 1^n and answers b

<u>Note</u> – Have only decided status of $\le 2^n/10$ strings in $\{0,1\}^n$
– For all of them, answered no.

If $b=1$, decide that <u>no string</u> in $\{0,1\}^n$ is in B
$\Rightarrow 1^n \notin U_B$, and M_i is wrong on 1^n

If $b=0$, pick some string $y \in \{0,1\}^n$ not queried and decide that $y \in B$
$\Rightarrow 1^n \in U_B$, so M_i is wrong on 1^n

Only remaining worry. What if we didn't allow M_i to finish? What if it runs in polynomial time p s.t. $p(n) \geq 2^n/10$ for this n?

The TM M_i will be repeated infinitely often for larger and larger i. Finally, will get some n' s.t. $p(n') \ll 2^n/10$, and for this n' the proof will work.

Recall our TM encoding has "stop marker" after which junk allowed, so each TM encoded by infinitely many integers

SUMMING UP

- Diagonalization can be used to separate cplx classes.
- In particular, $P \neq EXP$
- If $P \neq NP$, then there is infinite hierarchy of cplx classes between P & NP
- But diagonalization not enough to settle $P \stackrel{?}{=} NP$, since any such proof works for oracle TMs and different ~~TMs~~ oracles give different answers to $P \stackrel{?}{=} NP$...

NEXT ON THE AGENDA

- Memory consumption as the limiting factor
- Space-bounded cplx classes

So far focused on <u>running time</u> as limited resource.
At the end of the day, most interesting measure [at least a lot of the time]

But also interesting to consider <u>memory usage</u>
Arguably second most fundamental resource

Get new complexity classes and complete problems for them
Some similarities with time-bounded computation, but also some striking differences

<u>DEF 1</u> A language $L \subseteq \{0,1\}^*$ is in SPACE $(s(n))$ if $\exists$ constant c & Turing machine M such that

1) M correctly decides L
2) M's heads never visit more than $c \cdot s(n)$ distinct locations ON READ-WRITE TAPES EXCLUDING INPUT TAPE for any input of length n

$L \in$ NSPACE $(s(n))$ if $\exists$ non-deterministic TM deciding L s.t. at most $c \cdot s(n)$ read-write locations visited for any input of length n and <u>any non-deterministic choices</u>

<u>Note</u> that we have NDTM here and not certificate and verifier TM

Important points

- Input stored on read-only input tape
  Doesn't count towards memory
  $\Rightarrow$ Possible to do computations in <u>sublinear space</u>

- Decision problem: Need not use output tape other than for answer yes/no (0/1).
  Focus on work tape(s)

- Look at only <u>space-constructible $s(n)$</u>
  $\exists$ TM that computes $s(|x|)$ in $O(s(|x|))$ space given input x. (Technical condition that we will ignore.)

- Space bounds of interest $\geq \log n$ — want TM to be able to remember positions on input tape.

Clearly $DTIME(s(n)) \subseteq SPACE(s(n))$
  Can visit at most one tape position per time step
  But space can be <u>reused</u>
  Use space $S(n)$ to count from 0 to $2^{s(n)}-1$
  This is (almost) all that we know.

<u>THM 2</u>

$$DTIME(s(n)) \subseteq SPACE(s(n)) \subseteq NSPACE(s(n)) \subseteq DTIME(2^{O(s(n))}).$$

(Will be proven shortly.)

In fact, can do slightly better

<u>THM 3</u> (Hopcroft, Paul, Valiant '77)

$$DTIME(s(n)) \subseteq SPACE(s(n)/\log s(n))$$

So space is strictly more powerful than time as resource. (Probably won't do anything close to proving Thm 3.)

One more point:

- What about termination in Def 1?
  Can require it. Not necessary, really.
  After $2^{O(s(n))}$ steps, workspaces has looked exactly the same, read/ worktape heads have been exactly the same place, (ND)TM state has been exactly the same, etc at two time steps $t_1 < t_2$.
  So can ignore computation during interval $[t_1 + 1, t_2]$.
  If $\exists$ accepting computation, only $2^{O(s(n))}$ steps needed. Can equip any TM with "clock" that terminates after $2^{O(s(n))}$ steps. Only $O(s(n))$ space needed to count the time.

Back to proof of Thm 2 ...

DEF 4 Configuration graph of TM M

Configuration of M | consists of | at time t |

- program counter / state
- head positions
- contents of all tape positions that can possibly be visited (for same input length)

Configuration graph of M on input $x \in \{0,1\}^*$ | space $s(n)$ TM |

Directed graph $G_{M,x}$

Vertices: all possible configs C with input $= x$ and $c \cdot s(|x|)$ worktape cells

Edges: (C, C') if C' can be reached from C in one step acc to M's transition function.

Deterministic TM:	out-degree 1
Nondeterministic TM	out-degree 2

Assume M has "clean-up phase" erasing all worktapes before halting $\Rightarrow$ one unique accepting config C_{accept}.

$$\boxed{M \text{ accepts } x \iff \exists \text{ path in } G_{M,x} \text{ from } C_{start} \text{ to } C_{accept}}$$

CLAIM 5

1. Every vertex in $G_{M,x}$ can be described using $K \cdot s(n)$ bits ($K = O(1)$ depending on alphabet, # tapes, # states)
2. $G_{M,x}$ has at most $2^{O(s(n))}$ vertices
3. $\exists$ $O(s(n))$-size CNF formula $\varphi_{M,x}$ s.t. $\varphi_{M,x}(C, C') = 1$ iff (C, C') edge in $G_{M,x}$

Proof

1. Sort of by description in Def 4
2. Follows from 1.
3. Use Cook–Levin–style reasoning

Formula contains lots of local consistency checks

- tape contents are correct one time step later
- jump to correct state given read bits and previous state
- et cetera

$O(s(n))$ checks

Each check involves constant # bits = variables. ☒

Proof of Thm 2

Only need to show $NSPACE(s(n)) \subseteq DTIME(2^{O(s(n))})$.
Construct $G_{M,x}$ in $2^{O(s(n))}$ time.
Do BFS to check if C_{accept} reachable from C_{start}. ☒

DEF 6 Some cplx classes of particular interest

$$PSPACE = \bigcup_{c \in \mathbb{N}^+} SPACE(n^c)$$
$$NPSPACE = \bigcup_{c \in \mathbb{N}^+} NSPACE(n^c)$$

$$L = SPACE(\log n)$$
$$NL = NSPACE(\log n)$$

Next time

Talk about PSPACE, NPSPACE, L, NL

Relate to other complexity classes such as NP

Leads to discussion of complete problems

And more ...