Computational Models — Lecture 11[1]

Handout Mode

Ronitt Rubinfeld and Iftach Haitner.

Tel Aviv University.

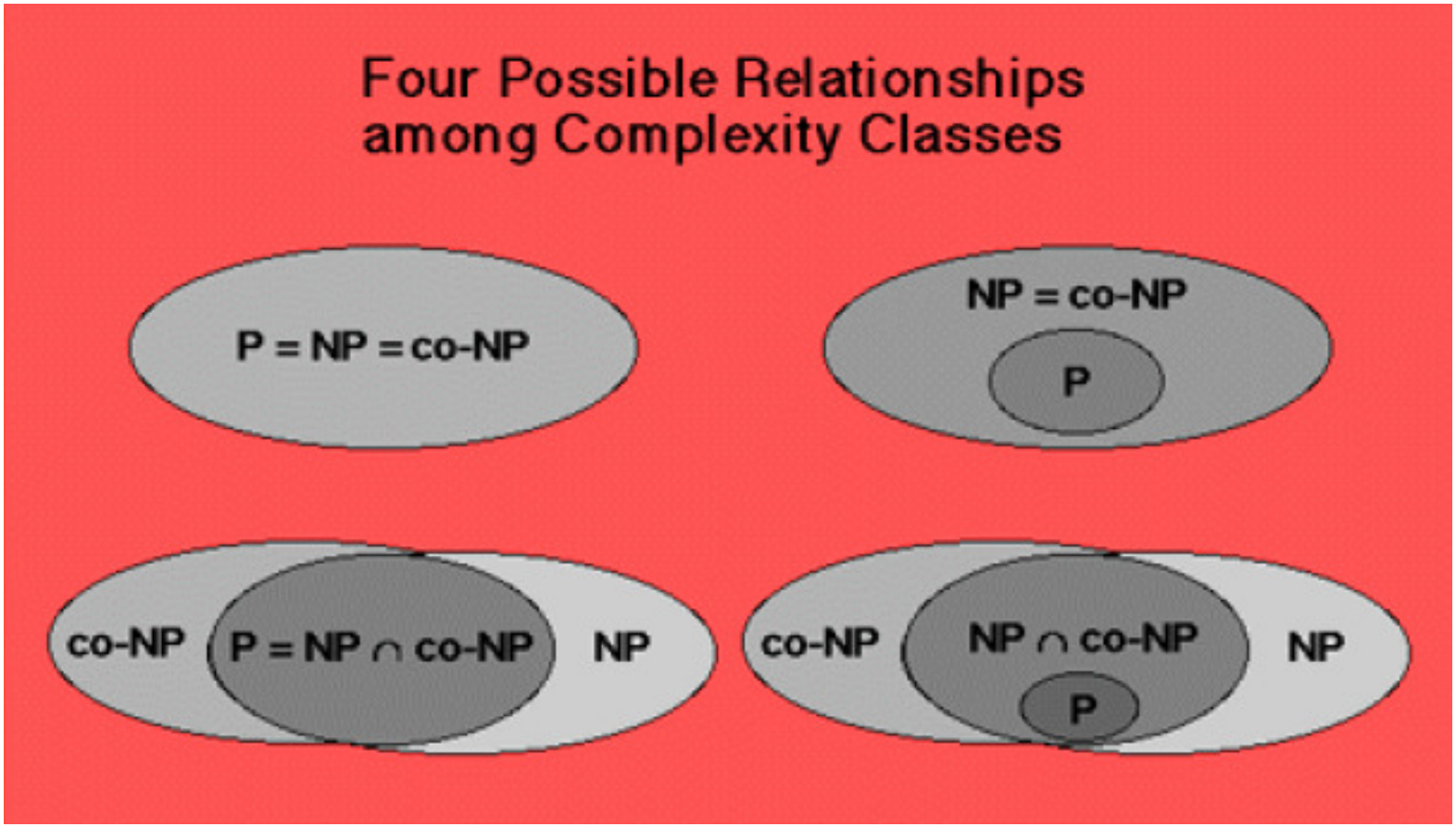
May 23/25, 2016

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University. With added modifications of Yishai Mansour.

Ladies and Gentlemen, Boys and Girls

We are about to begin the fourth part of the course:

Introduction to Computational Complexity



Talk Outline

- Introduction to time complexity
- The class $\mathcal{P}$
- The class $\mathcal{NP}$
- Verifiability

- Sipser's book 7.1-7.3

Part I

Introduction to Time Complexity

How long does it take to decide $L = \{0^m 1^m : m \geq 0\}$

Clearly $L \in \mathcal{R}$.

Question 1

How much time does a single-tape TM need to decide L?

Algorithm 2 (Decider M_1 for L)

On input string w:

1. Scan across tape and `Reject` if 0 is found to the right of a 1.
2. While both 0s and 1s appear on tape, repeat the following:

   Scan across tape, crossing of a single 0 and a single 1 in each pass.
3. `Accept` if no 0s and 1s remain; otherwise `Reject`.

We analyze the time it takes to perform each of the three steps separately. In the following let $n = |w|$.

Analyzing Step 1

Scan across tape and `Reject` *if* 0 *is found to the right of a* 1*. If not, return to starting point.*

- Scanning requires n steps.
- Re-positioning head requires n steps.
- Total is $2n = O(n)$ steps.

Analyzing Step 2

While both 0s and 1s appear on tape, repeat the following

Scan across tape, crossing of a single 0 and a single 1 in each pass.

- Each scan requires $O(n)$ steps.
- Since each scan crosses off two symbols, the number of scans is at most $n/2$.
- Total number of steps is $(n/2) \cdot O(n) = O(n^2)$.

Analyzing Step 3

If 0*s still remain after all* 1*s have been crossed out, or vice-versa,* `Reject`*. Otherwise, if the tape is empty,* `Accept`*.*

- Single scan requires $O(n)$ steps.
- Total is $O(n)$ steps.

Total Cost

Total cost for three steps

1. $O(n)$
2. $O(n^2)$
3. $O(n)$

which is $O(n^2)$

Deterministic Time

Definition 3 (deterministic Time)

Let M be a deterministic TM, and let $t : \mathbb{N} \mapsto \mathbb{N}$. We say that M runs in time $t(n)$, if For every input x of length n, the number of steps that $M(x)$ uses is at most $t(n)$.

Question 4

What is a “step"?

Definition 5 (DTIME)

For $t : \mathbb{N} \mapsto \mathbb{N}$, let $\mathrm{DTIME}(t(n)) =$
$\{L \subseteq \Sigma^* :$ L is decided by an $O(t(n))$-time single tape TM$\}$

Note that $t(n)$ running time, is also required for strings not in L.

Language Encoding

- Let $L = \{1^n : n \in \mathbb{N}\}$.

  Is L in DTIME(n)? in DTIME($n^{1/2}$)? Proof?

- Let PATH $= \{\langle G, s, t\rangle : G$ has directed path from s to $t\}$

  Is L in DTIME(n)? in DTIME($n^{1/2}$)?

- The questions are not well defined w/o defining the encoding of strings into triplets $\langle G, s, t\rangle$.
- Encoding may matter much — an exponential algorithm with respect to one encoding might turn to linear algorithm with respect to other encoding.

If not defined explicitly, we assume a “reasonable encoding":

- Graphs encoding: adjacency list or adjacency matrix
- Integers encoding: binary encoding (and not unary)

Encoding of integers: binary versus unary

- An algorithm for finding a factor of integer q:
  - Do for $i = 2$ to q:
    - ⋆ If i divides q output i and halt.
  - Output "none" and halt.
- What is the running time?
  - $O(q)$ (could even improve to $O(\sqrt{q})$)
  - Yay: poly time! ? Yes, if q represented in unary.
  - Uh oh... number of bits to represent q in binary is $\log q$, so $O(q)$ is EXPONENTIAL IN THE INPUT LENGTH
- Today's crypto systems assume that factoring 4K bit numbers takes a long time!

Relations among Time Classes

Let $t_1, t_2 : \mathbb{N} \mapsto \mathbb{N}$ be two functions.

Claim 6

If $t_1(n) = O(t_2(n))$, then $\text{DTIME}(t_1(n)) \subseteq \text{DTIME}(t_2(n))$.

Stated informally, more time does not hurt. But does it actually help?
Would like to say "if $t_1(n) = o(t_2(n))$ then $\text{DTIME}(t_1(n)) \subsetneq \text{DTIME}(t_2(n))$".
But this is what we can get:

Claim 7

(Assume that $t_1(n)$ and $t_2(n)$ are time constructible[a]) If $t_1(n) \in O(t_2(n)/\log(n))$, then $\text{DTIME}(t_1(n)) \subsetneq \text{DTIME}(t_2(n))$.

[a] t is *time constructible* if the function mapping 1^n to the binary representation of $t(n)$ is computable in time $O(t(n))$.

Informally, sufficiently more time does help

Proof – omitted.

Back to $L = \{0^m 1^m | m \geq 0\} \in DTIME(n^2)$

We have seen that $L \in DTIME(n^2)$. Can we do faster?

Algorithm 8 (Decider M_2 for L)

On input string w

1. Scan across tape and `Reject` if 0 is found to the right of a 1.
2. Repeat the following while both 0s and 1s appear on tape:
   - 2.1 Scan across tape, checking whether total number of 0s plus 1s is even or odd. If odd, `Reject`.
   - 2.2 Scan across tape, crossing off every other 0 (starting with the first), and every other 1 (starting with the first) in each pass.
3. `Accept` if all string is crossed off; Otherwise `Reject`.

Example: $w = 0^{13}1^{13}$

Correctness?

- Let binary representation of number of 0's and 1's be $a_k \ldots a_0, b_\ell \ldots b_0$ respectively.
- If total number of 0s plus 1s is even, they have same parity $\rightarrow a_0 = b_0$.
- Crossing off every other 0,1 has same effect as "shift right" on binary representation.
- So, next iteration checks $a_1 = b_1$...
- When finish, know that $a_i = b_i$ for all i!

Running Time Analysis of M_2

Algorithm 9 (Decider M_2 for L)

On input string w

1. Scan across tape and `Reject` if 0 is found to the right of a 1.
2. Repeat the following while both 0s and 1s appear on tape:
   - 2.1 Scan across tape, checking whether total number of 0s plus 1s is even or odd. If odd, `Reject`.
   - 2.2 Scan across tape, crossing off every other 0 (starting with the first), and every other 1 (starting with the first) in each pass.
3. `Accept` if all string is crossed off; Otherwise `Reject`.

- One pass in each step (1,2,3) takes $O(n)$ time.
- Steps 1,3: each executed once
- Step 2 executed $1 + \log_2 n$ times
- Total for Step 2 is $(1 + \log_2 n)O(n) = O(n \log n)$.
- Grand total: $O(n) + O(n \log n) = O(n \log n)$.

Further Improvements, Anybody?

Question 10

Can the running time be made $o(n \log n)$?

Answer: Not on a single tape TM...

Theorem 11

$L \notin \text{DTIME}(o(n \log n))$ *(i.e.,* $L \notin \text{DTIME}(g(n))$ *for any* $g(n) \in o(n \log n))$*.*

$L \notin DTIME(o(n \log n))$

Fix an $o(n \log n)$-time TM M.

Definition 12

For $w \in \{0,1\}^*$, the crossing sequence of $M(w)$ at location i, is the **sequence of states** used by the invocation of M on input w, when moving from i to $(i+1)$, and from $(i+1)$ to i.

Claim 13

For long enough $w \in \{0,1\}^*$, the run $M(w)$ has a pair of identical crossing sequences.

Claim 14 (Pumping Lemma for low running time TMs)

Let $w = xyz$, and assume that in $M(w)$ locations $|x|$ and $|x+y|$ have identical crossing sequences, then $M(w)$ and $M(w' = (x, y^2, z))$ halt in the same state.

- Assume M is a decider for L and let w be a long enough word in L.
- Hence, M accepts w, but also a pumped (up) version of w.
- But all pumped up versions of $w \in L$ are outside of L ...

Proof of Claim 13 (idea)

Claim 15 (Restating Claim 13)

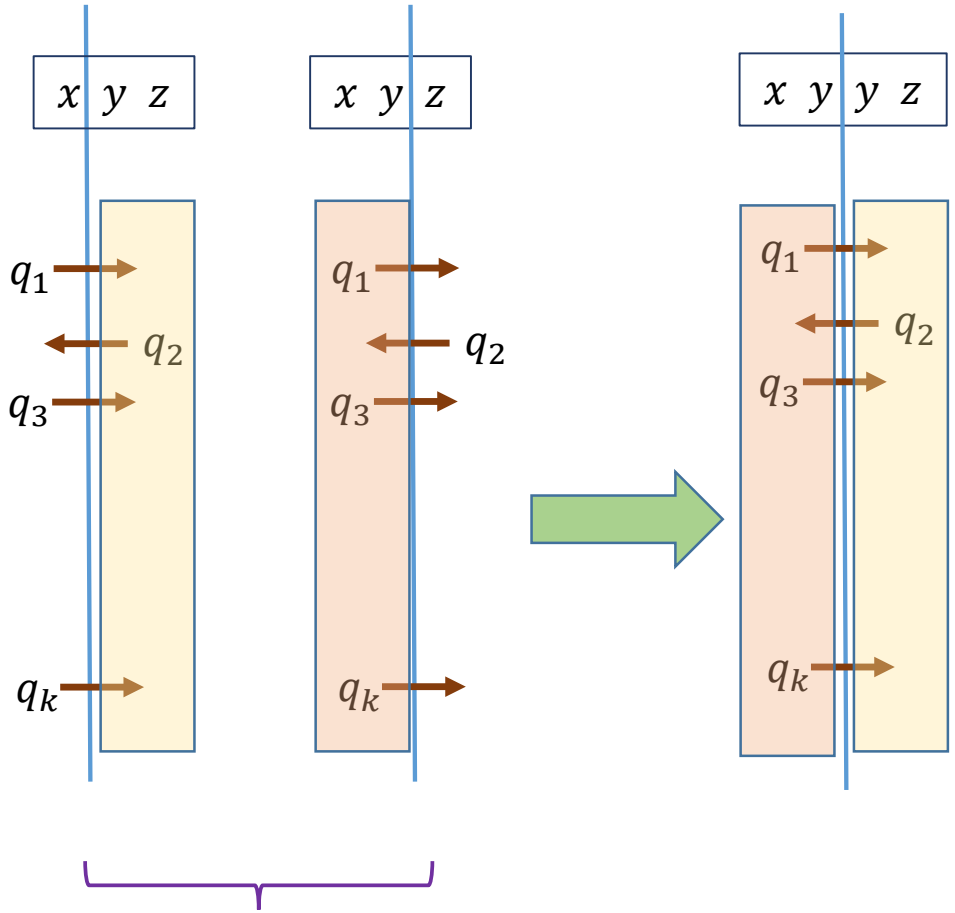
For long enough $w \in \{0,1\}^*$, the run $M(w)$ has a pair of identical crossing sequences.

Let $t(n) \in o(n \cdot \log n)$ be an upper bound on the running time of M.

Fix $w \in \{0,1\}^m$, and consider the run $M(w)$.

- Let C be the subset of the first m locations on the tape that are visited at most $2 \cdot t(m)/m$ (twice the average) times (and so crossing sequence is short – length $\leq 2t(m)/m$).
- Let C' be the subset of the first m locations on the tape that are visited strictly greater than $2 \cdot t(m)/m$ times.
- By Markov's inequality, $|C'| \leq m/2$. Since $|C| + |C'| = m$, we have that $|C| \geq m/2$:
- Note that $f(n) = t(n)/n$ is $o(\log n)$, so number of short crossing sequences is at most $(|Q|+1)^{2t(m)/m} = 2^{2f(m)\log|Q|} = m^{o(1)} << m/2$, so by pigeon hole principle, there is a pair $i, j \in C$ with the same crossing sequence.□

Proof by picture



2 identical crossing sequences

Proof idea for Claim 14

Claim 16 (Restating Claim 14)

Let $w = xyz$, and assume that in $M(w)$ locations $|x|$ and $|x + y|$ have identical crossing sequences, then $M(w)$ and $M(w' = (x, y^2, z))$ halt in the same state.

The augment content of a cell, contains its content (i.e., letter in Σ), plus the **machine state**, if currently at this cell, and $\perp$ otherwise.
Let $M(w)_i$ stands for the augmented input of $M(w)$ at location i.

Claim 17

If in the execution, $M(w)$ has spent k_1 steps in the first $|xy|$ locations, and k_2 steps in locations $|x| + 1, \ldots$. Then

- After $M(w')$ spends k_1 steps on the first $|xy|$ locations, $M(w)_{1,\ldots,|xy|} = M(w')_{1,\ldots,|xy|}$.
- After $M(w')$ spends k_2 steps on locations $|xy| + 1, \ldots$, $M(w)_{|x|+1,\ldots} = M(w')_{|xy|+1,\ldots}$.

Proof: by induction on $k_1 + k_2$ (and using Claim 14).(?)

More on the "fooling"

- The first time M goes over x, y border into z, it is in same state as when goes over x, y, y border into z, so when gets to z in both inputs, will behave the same while in z.
- When gets back to cross left border of what was originally z, will also be in the same state.
- Why will it work the second, third, fourth, ..., n^{th} time it crosses the border?? (hint: definition of crossing sequence!)

A Two-tape Decider for L

Algorithm 18 (Two-tape Decider M_3 for L)

On input string w:

1. Scan across tape and `Reject` if 0 is found to the right of a 1.
2. Scan across 0s to first 1, copying 0s to tape 2.
3. Scan across 1s on tape 1 until the end. For each 1, cross off a 0. If no 0s left, `Reject`.
4. If any 0s left, `Reject`; otherwise `Accept`.

Question 19

What is M_3 running time?

Complexity of Deciding $L = \{0^n 1^n\}$

- Single-tape M_1: $O(n^2)$.
- single-tape M_2: $O(n \log n)$ (fastest possible!).

  Hence $L \in \text{DTIME}O(n \log n)$, but not in $\text{DTIME}(O(f(n)))$ for $f(n) \in o(n \log n)$
- Two-tape M_3: $O(n)$.

Important difference between complexity and computability:

- Computability: all reasonable models equivalent (Church-Turing)
- Complexity: choice of model does affect running time.

Question 20

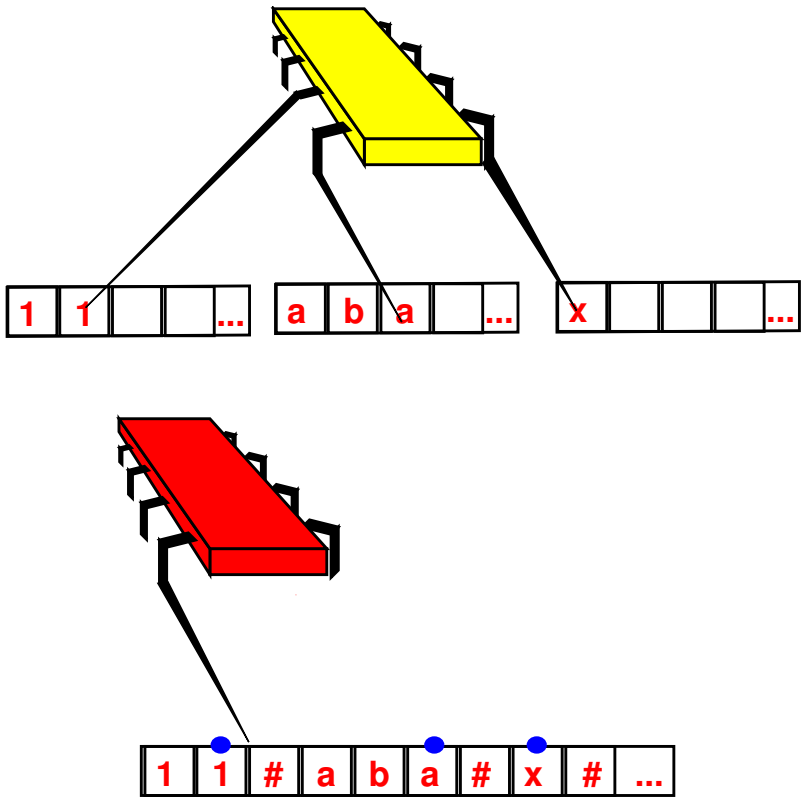
By how much does a model affect complexity?

Multitape Speedup

Let $t(n)$ be a function where , and let $L \subseteq \Sigma^*$ be a language.

Claim 21

Assume $\exists$ $t(n)$-time multitape TM that decides L with $t(n) \geq n$, then $\exists$ an $O(t(n)^2)$-time single-tape TM that decides L.



Reminder: Emulating Multi-Tape TMs

Algorithm 22 (Single tape emulator S for k-tape M)

On input $w = w_1 \cdots w_n$:

1. Write $\# \dot{w}_1 \; w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#$ on tape.
2. Scan tape from first $\#$ to $(k+1)$-st $\#$ to read symbols under virtual heads.
3. Rescan tape to write new symbols and move heads.
4. If need to move virtual head onto $\#$, shift tape content to the right.

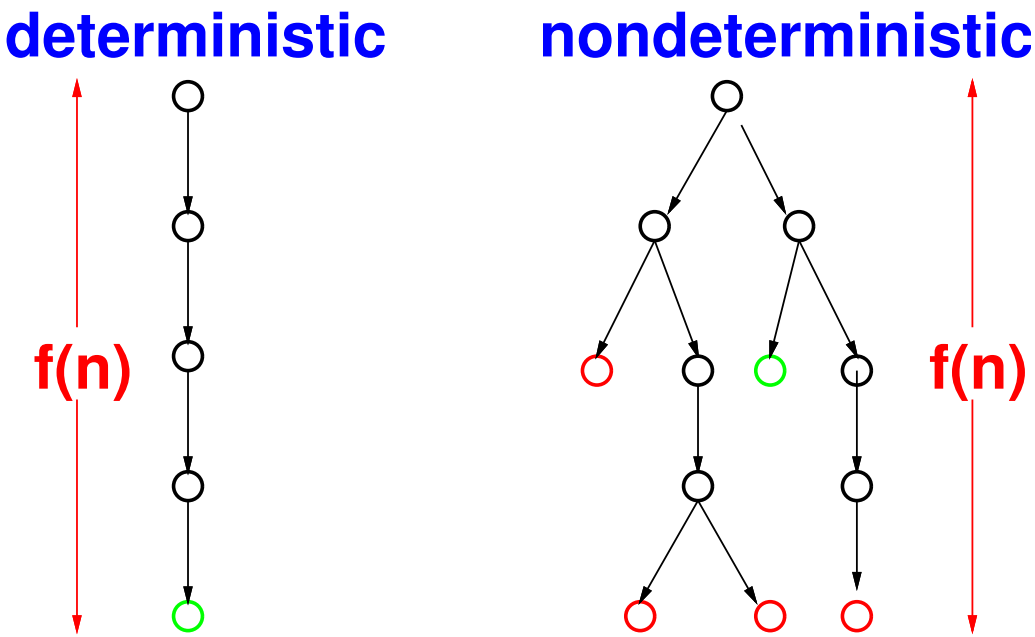
- For each step of M, the emulator S performs 2 scans and up to k rightward shifts
- On input of length n, M makes $O(t(n))$ many steps, so active portion of each tape is $O(t(n))$ long.
- Total number of steps S makes:
  - $O(k \cdot t(n)) = O(t(n))$ steps to simulate one step of M.
  - Initial tape arrangement $O(n)$.
  - Grand total: $O(n) + O(t(n)^2) = O(t(n)^2)$ steps (recall $t(n) > n$).

Nondeterministic Time

Definition 23 (nondeterministic Time)

A non-deterministic TM N runs in time $f(n)$, where $f : \mathbb{N} \mapsto \mathbb{N}$, if for every input x of length n, the maximum number of steps that N uses on any branch of its computation tree on x, is at most $f(n)$.

Notice that also non-accepting branches must reject within $f(n)$ many steps.



TAKE NOTE: the depth of the tree, not the size of the tree!!!

Nondeterminism Speedup

Claim 24

Suppose N is a nondeterministic TM that runs in time $t(n)$ and decides the language L. Then there exists an $2^{O(t(n))}$-time deterministic TM D that decided L.

Note contrast with multi-tape result.
Proof's idea: D emulates N. Reminder

- D tries all possible branches (say using BFS).
  - `Accept` if finds any accepting state.
  - `Reject` if all branches reject.
- Since N always stops, exactly one of two possibilities must occur.

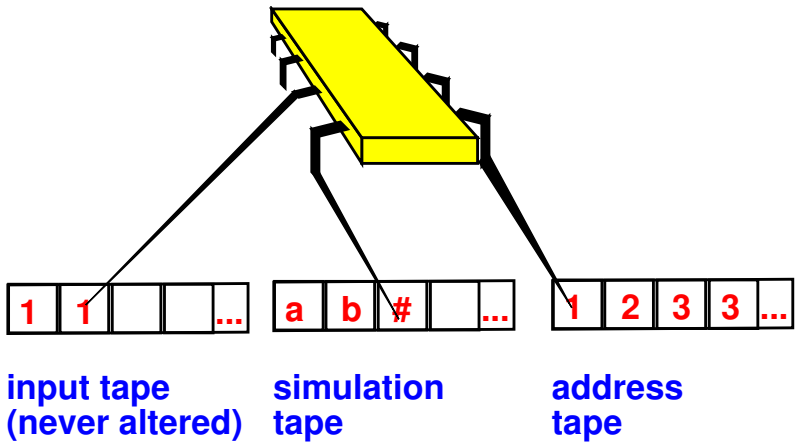
Emulation Details

We view the computation of N as a tree, whose nodes are configurations of the TM (i.e., state, head location and tape content).

- Root is starting configuration,
- Fanout is at most some constant b (?),
- Depth at most $\leq t(n)$,
- Total number of nodes $O(b^{t(n)})$,
- Emulation time is $O(b^{t(n)}) = O(2^{O(t(n))})$

Remarks

1. Breadth-first search used in emulation
   - Visit each node.
   - May be improved upon by using depth-first search (is it OK?) or other tree search strategies.
   - Still, doing this may save constants, but nothing substantial (?)
2. Simulation uses a three-tape machine.



Single-tape simulation: $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$.

Polynomial is Good, Exponential is Bad

	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.00001 second	.00004 second	.00009 second	.00016 second	.00025 second	.00036 second
n^3	.00001 second	.00008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minute	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	$2 \cdot 10^8$ centuries	$1.3 \cdot 10^{13}$ centuries

exponential time factoring algorithms

2008 claim on forum:

1 hr for 120 bits
10 hrs for 150 bits
100 hrs for 180 bits
1000 hrs for 210 bits

Gaps Between Models

- At most polynomial gap in time to perform tasks between different deterministic models (single- vs. multi-tape TMs, TM vs. RAM, etc.)
- Apparently exponential gap in time to perform tasks between deterministic and non-deterministic models.

Claim 25

All "reasonable" classical (not including quantum) models of computation are polynomially equivalent.

Any one can simulate another with only polynomial blowup in running time.

Fact 26

Is a given problem solvable in

- *Linear time? model-specific.*
- *Polynomial time? model-independent.*

Part II

The Class P

The Class $\mathcal{P}$

$\mathcal{P}$ is the set of languages decidable in polynomial time on deterministic TMs.

Definition 27 ($\mathcal{P}$)

$\mathcal{P} = \bigcup_{c \geq 0} \mathrm{DTIME}(n^c)$

The class $\mathcal{P}$ is important because:

- Invariant for all (deterministic) models of computation polynomially equivalent to TMs
  - not affected by particulars of model . . .
  - go ahead, have another tape, they're pretty small and inexpensive . . .
- Roughly corresponds to realistically solvable (tractable) problems.
  - actually depends on context
  - going from exponential to polynomial algorithm usually requires major insight,
  - if you find an inefficient polynomial algorithm, you can often find a more efficient one.

Known Problems in $\mathcal{P}$

- Integer arithmetic: Addition, subtraction, multiplication, division with remainder.
- Modular arithmetic: Exponentiation (RSA), inverse.
- Integer Algorithms: Greatest common divisor (gcd).
- Operations research: Maximum network flow, linear programming.
- Algebra: Matrix multiplication, computing determinants, matrix inversion, solving systems of linear equations, factoring polynomials.
- Graph algorithms: BFS and DFS in graphs, minimum spanning trees, finding Eulerian path.

PATH

$$\text{PATH} = \{\langle G, s, t\rangle : G \text{ has directed path from } s \text{ to } t\}$$

Theorem 28

$\text{PATH} \in \mathcal{P}$.

Algorithm 29 (M_1 – Naive algorithm for PATH)

Input: an m nodes graph G
For each path p in G of length $\leq m$: check if p goes from s to t.

Question 30

What is the (time) complexity of M_1?

- there are m^m possible paths
  $\Longrightarrow$ exponential in number of nodes
  $\Longrightarrow$ exponential in input size
- Oh, oh. Does not sound like $\mathcal{P}$ to me ...

Efficient Algorithm for PATH

Algorithm 31 (M_2 – efficient algorithm for PATH)

1. Place mark on s.
2. Repeat until no additional nodes marked:
   - 2.1 Scan edges of G.
   - 2.2 If edge (a, b) found from marked node a to unmarked node b, mark node b.
3. `Accept` if t is marked; otherwise `Reject`.

Question 32

What is the complexity of M_2?

- Steps 1 and 3 run once.
- Step 2 runs at most m times, because each time (except last) it marks at least one new node.

$\Longrightarrow$ total number of steps is polynomial.

Relative Primality

Two numbers are relatively prime if their *greater common divisor (gcd)* is 1 (i.e., the largest integer that divides them both).

- $\gcd(10, 21) = 1 \implies$ 10 and 21 are relatively prime
- $\gcd(10, 22) = 2 \implies$ 10 and 22 are not relatively prime

Definition 33

$\text{RELPRIME} = \{\langle x, y \rangle : \gcd(x, y) = 1\}$.

Theorem 34

$\text{RELPRIME} \in \mathcal{P}$.

Naive algorithm for RELPRIME

Algorithm 35 (Naive algorithm for RELPRIME)

Input: integers x, y:
Search through all possible divisors of x, y and test divisibility.

- If x, y are give in unary:
  - Size of $\langle x \rangle$ is x
  - Testing all potential divisors of x, y is polynomial
- If x, y are give in binary
  - Size of $\langle x \rangle$ is $\log x$
  - Testing all potential divisors of x, y is exponential
- To analyze the running time of an algorithm that decides a language L, one should first say what is the encoding of L.
- Yet, we sometimes ignore that, and assume “reasonable encoding"
- The above algorithm is sometimes called pseudo polynomial.

Euclid's Algorithm for Computing gcd

Algorithm 36 (E)

On input $\langle x, y \rangle$:

1. Repeat until $x = 0$:
   - **1.1** If $y > x$, swap x and y.
   - **1.2** $x \leftarrow x \bmod y$
2. Output y.

Claim 37

E runs in polynomial-time and correctly computes the gcd function.

Algorithm 38 (M for RELPRIME)

On input $\langle x, y \rangle$:
`Accept` iff $E(x, y) = 1$.

To prove that RELPRIME $\in \mathcal{P}$, we only need to prove Claim 37.

Analyzing Euclid's Algorithm

We will only prove the running time part.

Algorithm 39 (E)

On input $\langle x, y \rangle$:

1. Repeat until $x = 0$:
   - 1.1 If $y > x$, swap x and y.
   - 1.2 $x \leftarrow x \bmod y$
2. Output y.

- Each execution of Step 1 cuts x by at least half (case analysis with for $y < x/2$ and $x/2 \leq y < x$)
- After each two executions maximal value is cut in half

  $\Longrightarrow$ number of stages is $\min(\log_2 x, \log_2 y)$ $\Longrightarrow$ total running time is polynomial.

Consequently, RELPRIME $\in \mathcal{P}$. ♣

Part III

The Class NP

The Class $\mathcal{NP}$

Definition 40 (NTIME)

For $t \mapsto \mathbb{N} \mapsto \mathbb{N}$, let $\text{NTIME}(t(n)) =$
$\{L \subseteq \Sigma^* :$ L is decided by an $O(t(n))$-time single tape NTM$\}$

$\mathcal{NP}$ is the set of languages decidable in polynomial time on non-deterministic TMs.

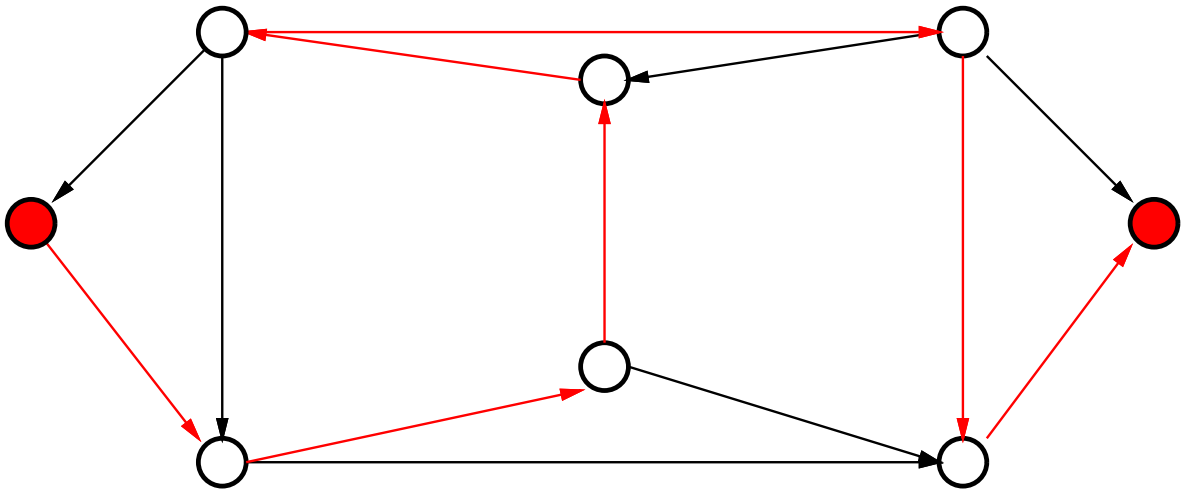
Definition 41 ($\mathcal{NP}$)

$\mathcal{NP} = \bigcup_{c \geq 0} \text{NTIME}(n^c)$

The class *NP* is important because:

- Insensitive to choice of reasonable non-deterministic computational model.
- Roughly corresponds to problems whose positive solutions are efficiently verified.

Hamiltonian Path



A Hamiltonian path in a directed G visits each node exactly once.

$$\text{HAMPATH} = \{\langle G, s, t\rangle : G \text{ has Hamiltonian path from } s \text{ to } t\}$$

Question 42

How hard is it to decide HAMPATH?

Easy to obtain exponential time algorithm:

- Generate each potential path
- Check whether it is Hamiltonian