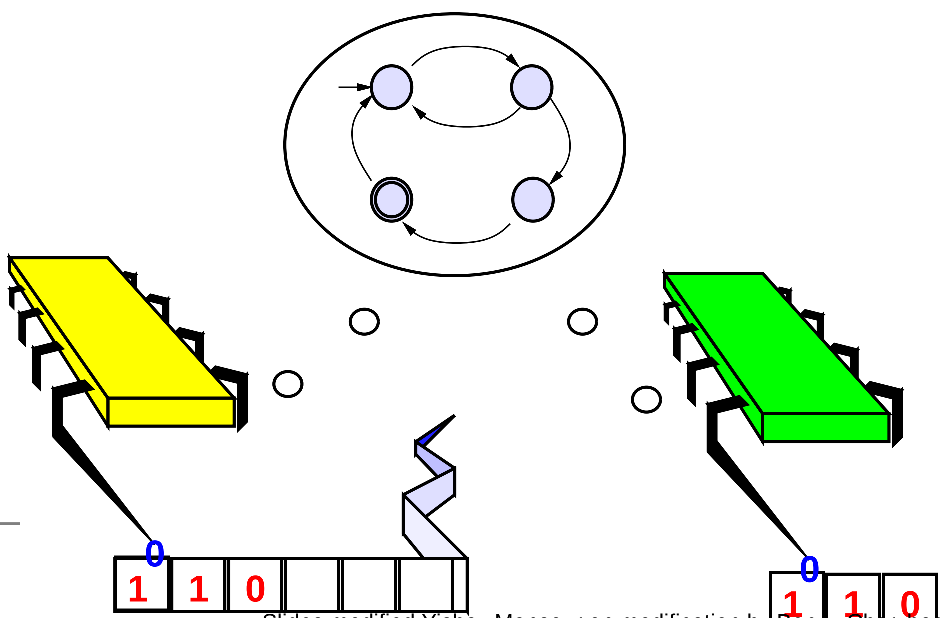
- Linear Bounded Automata
- Unrestricted Grammar
- Introduction to Time Complexity

- Sipser's book, Chapter 5, Sections 5.3 and Chapter 7, Section 7.1

Linear Bounded Automata

- A restricted form of TM.
- Cannot move off portion of tape containing input
  - Details: Has special letters for start and end input, which it can not modify.
  - Can not move left (right) of the left (right) mark.
- Size of input determines size of memory



Linear Bounded Automata

Question: Why linear?

Answer: Using a tape alphabet larger than the input alphabet increases memory by a constant factor.

Linear Bounded Automata

Believe it or not, LBAs are quite powerful.

The deciders for

- A_{DFA} (does a DFA accept a string?)
- A_{CFG} (is string in a CFG?)
- $\text{EMPTY}_{\text{DFA}}$ (is a DFA trivial?)
- $\text{EMPTY}_{\text{CFG}}$ (is a CFL empty?)

are all LBAs.

Every CFL can be decided by a LBA.

Not too easy to find a natural, decidable language that cannot be decided by an LBA.

Acceptance for LBAs

Define

$$A_{\text{LBA}} = \{\langle M, w\rangle | M \text{ is an LBA that accepts } w\}$$

Question: Is A_{LBA} decidable?

Answer: Unlike A_{TM}, the language A_{LBA} is decidable!

Lemma:

Let M be a LBA with

- q states
- g symbols in tape alphabet

On an input of size n, LBA has exactly qng^n distinct configurations, because a configuration involves:

- control state (q possibilities)
- head position (n possibilities)
- tape contents (g^n possibilities)

Theorem: A_{LBA} is decidable

Idea:

- Check that M is a valid LBA, if not reject.
- Simulate M on w
- But what do we do if M loops?
- Must detect looping and reject.
- M loops if and only if it repeats a configuration.
- Why? And is this also true of “regular” TMs?
- By pigeon hole, if our LBA M runs long enough, it must repeat a configuration!

Theorem: A_{LBA} is decidable

On input $\langle M, w\rangle$, where M is an LBA and $w \in \Sigma^*$,

1. Simulate M on w,
2. while maintaining a counter.
3. Counter incremented by 1 per each simulated step (of M).
4. Keep simulating M for qng^n steps, or until it halts (whichever comes first)
5. If M has halted, accept w if it was accepted by M, and reject w if it was rejected by M.
6. reject w if counter limit reached (M has not halted).

More LBAs

Surprisingly though, LBAs do have undecidable problems too!

Here is a related problem.

$$\text{Non-EMPTY}_{\text{LBA}} = \{\langle M \rangle | M \text{ is an LBA and } L(M) \neq \emptyset\}$$

Question: Is $\text{Non-EMPTY}_{\text{LBA}}$ decidable?

Non-EMPTY$_{\text{LBA}}$

Non-EMPTY$_{\text{LBA}} = \{\langle M\rangle | M$ is an LBA and $L(M) \neq \emptyset\}$

Theorem: Non-EMPTY$_{\text{LBA}}$ is undecidable.

Proof by reduction from A_{TM}, using computation histories.

More LBAs

Given M and w, we will construct an LBA, B.

- $L(B)$ will contain exactly all accepting computation histories for M on w.
- M accepts w iff $L(B) \neq \emptyset$.

More LBAs

It is not enough to show that B exists.

We must show that the mapping from $\langle M, w\rangle$ to $\langle B\rangle$ is computable.

We are now going to describe the linear bounded machine, $\langle B\rangle$. It will be clear that indeed $\langle B\rangle$ is computable from $\langle M, w\rangle$.

Assume an accepting computation history is presented as a string:

$$\#\underbrace{\qquad\qquad}_{C_1}\#\underbrace{\qquad\qquad}_{C_2}\#\underbrace{\qquad\qquad}_{C_3}\#\cdots\#\underbrace{\qquad\qquad}_{C_\ell}\#,$$

with descriptions of configurations separated by $\#$ delimiters.

The LBA

The LBA, B, works as follows:

On input x, the LBA B:

- breaks x according to the $\#$ delimiters
- identifies strings $C_1, C_2, \ldots, C_\ell$.
- then checks that all the following conditions hold:
  - Each C_i are a configuration of M
  - C_1 is the start configuration of M on w
  - Every C_{i+1} follows from C_i according to M
  - C_ℓ is an accepting configuration

The LBA

- Checking that each C_i is a configuration of M is easy: All it means is that C_i includes exactly one q symbols.
- Checking that C_1 is the start configuration on w, $q_0 w_1 w_2 \cdots w_n$, is easy, because the string w is "wired into" B.
- Checking that C_ℓ is an accepting configuration is easy, because C_ℓ must include the accepting state q_a.
- The only hard part is checking that each C_{i+1} follows from C_i by M's transition function.

The Hard Part

Checking that for all i, C_{i+1} follows from C_i by M's transition function:

- C_i and C_{i+1} almost identical, except for positions under head and adjacent to head.
- These positions should updated according to transition function.

Do this verification by

- zig-zagging between corresponding positions of C_i and C_{i+1}.
- use "dots" on tape to mark current position
- all this can be done inside space allocated by input x. Thus B is indeed a LBA.

Important!

The LBA, B, accepts the string x if and only if x equals an accepting computation history of M on w.

Therefore $L(B)$ is either empty or a singleton $\{x\}$.

We construct B so that $L(B)$ is non-empty iff M accepts w.

Thus $\langle M, w\rangle \in A_{\text{TM}}$ iff $\langle B\rangle \in$ Non-EMPTY$_{\text{LBA}}$.

Namely $A_{\text{TM}} \leq_m$ Non-EMPTY$_{\text{LBA}}$, so Non-EMPTY$_{\text{LBA}} \notin \mathcal{R}$. ♠

BTW, is Non-EMPTY$_{\text{LBA}} \in \mathcal{RE}$?

Unrestricted Grammars

Unrestricted grammars are similar to context free ones, except left hand side of rules can be strings of variables and terminal with at least one variable.

- To non-deterministically generate a string according to a given unrestricted grammar:
  - Start with the initial symbol
  - While the string contains at least one non-terminal:
    - Find a substring that matches the LHS of some rule
    - Replace that substring with the RHS of the rule

Unrestricted Grammars: $\{a^n b^n c^n\}$

- Generate the variable sequence $L(ABC)^n$
  $S \rightarrow LT|\epsilon$;
  $T \rightarrow ABCT|\epsilon$;
- Sort the $\{A, B, C\}$ and get $LA^k B^k C^k$.
  $BA \rightarrow AB$;
  $CB \rightarrow BC$;
  $CA \rightarrow AC$;
- Replace the variables by terminals. $LA \rightarrow a$;
  $aA \rightarrow aa$;
  $aB \rightarrow ab$;
  $bB \rightarrow bb$;
  $bC \rightarrow bc$;
  $cC \rightarrow cc$;

Unrestricted Grammars

- Let $\mathcal{UG}$ be the set of languages that can be described by an Unrestricted Grammar:
- $\mathcal{UG} = \{L : \exists$ Unrestricted Grammar G such that $L[G] = L\}$
- Claim: $\mathcal{UG} = \mathcal{RE}$
- To Prove:
  - Show $\mathcal{UG} \subseteq \mathcal{RE}$
  - Show $\mathcal{RE} \subseteq \mathcal{UG}$

$\mathcal{UG} \subseteq \mathcal{RE}$

- Given any Unrestricted Grammar G, we create a 2-tape non-deterministic Turing Machine M that accepts $L[G]$.
- M maintains the input w on tape 1.
- M initializes tape 2 to the initial symbol S.
- In each iteration M does:
  - moves (non-deterministically) to some location on tape 2
  - Selects non-deterministically a rule R.
  - Tries to apply rule R to that location.
  - If successful, tests if tape 1 and tape 2 are identical.
  - If identical, terminates and accepts.
  - Otherwise, starts a new iteration.

$\mathcal{RE} \subseteq \mathcal{UG}$

- Given any language $L \in \mathcal{RE}$, let M be a deterministic Turing Machine that accepts it. We can create an Unrestricted Grammar G such that $L[G] = L$
  - Grammar: Generates a string
  - Turing Machine: Works from string to accept state
  - Two formalisms work in different directions
- Simulating Turing Machine with a Grammar by maintaining the TM configuration.
- Idea: variables of G are the states Q
  - Maintain $w[c]$, where w is the input and c is the current configuration.
  - if c is an accepting configuration, replace it by ϵ.

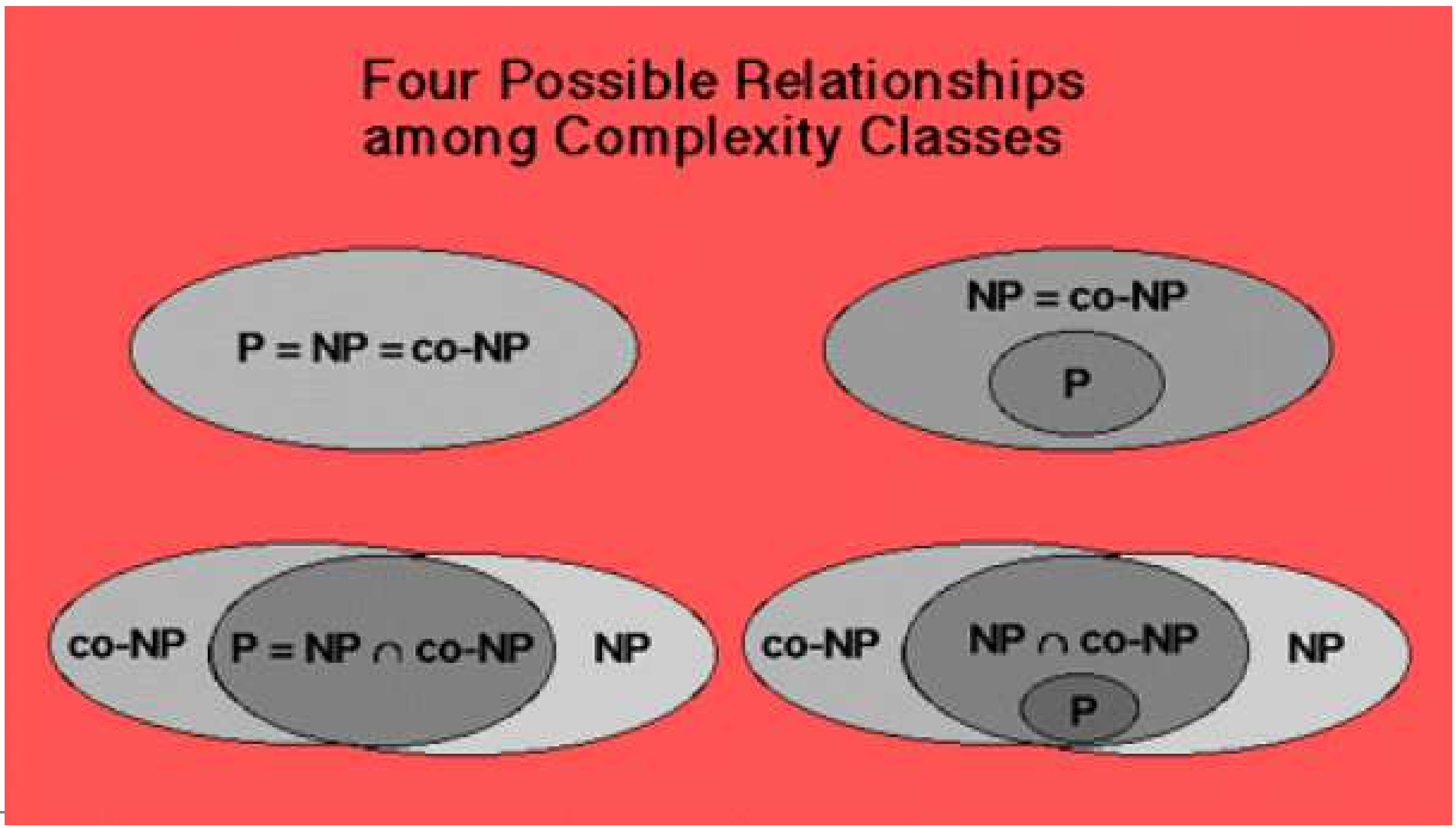
$\mathcal{RE} \subseteq \mathcal{UG}$

- Generate the string $w[q_0 w]$.
  $S \to T[q_0]$; $T \to aTA_a | \epsilon$
  $A_a[q_0 \to [q_0 A_a$; $A_a b \to bA_a$; $A_a] \to a]$
- Simulate TM M:
  $\delta(q,a) = (q', b, R) \Rightarrow qa \to bq'$;
  $\delta(q,a) = (q', b, L) \Rightarrow cqa \to q'cb$; $[qa \to [q'b$
  $q] \to q\sqcup]$
- Accepting: derive from $w[uq_a v]$ only w.
  $q_a \to E_L E_R$
  $aE_L \to E_L$; $[E_L \to \epsilon$
  $E_R a \to E_R$; $E_R] \to \epsilon$

Ladies and Gentlemen, Boys and Girls

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

Introduction to Computational Complexity



Time Complexity

Consider

$$A = \{0^m 1^m \,|\, m \geq 0\}$$

Clearly this language is decidable.

Question: How much time does a single-tape TM need to decide it?

Time Complexity

M_1: 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. If no 0s and 1s remain, accept, otherwise reject.

Analysis (1)

We consider the three stages separately. Let n denote the input length.

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.

Analysis (2)

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 $(n/2) \cdot O(n) = O(n^2)$.

Analysis (3)

3. If 0s still remain after all 1s 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.

Final Analysis

Total cost for stages

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

which is $O(n^2)$

Deterministic Time

Let M be a deterministic TM, and let

$$t : \mathcal{N} \longrightarrow \mathcal{N}$$

We say that M runs in time $t(n)$ if

- For every input x of length n,
- the number of steps that M uses,
- is at most $t(n)$.

Time Classes Definition

Let

$$t : \mathcal{N} \longrightarrow \mathcal{N}$$

be a function.

Definition:

$$\text{DTIME}(t(n)) = \{L \mid L \text{ is a language, decided by an } O(t(n))\text{-time DTM}\}$$

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

Do It Faster, Please

We have seen that

- $A = \{0^m 1^m | m \geq 0\}$,
- $A \in \text{DTIME}(n^2)$.

Can we do better, *i.e.* faster?

Home Improvement

M_2: 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. If no 0s or 1s remain, accept, otherwise reject.

Analysis

First, we verify that M_2 indeed halts.

- on each scan in step 2.2,
  - The total number of 0s is cut in half,
  - and if there was a remainder, it is discarded.
  - Same for 1s.
- Example: start with 13 0s and 13 1s,
  - first pass: 6 0s and 6 1s are left
  - second pass: 3 0s and 3 1s are left
  - third pass: one 0s and one 1s are left
  - then no 0s and 1s are left.

Analysis

We now verify that M_2 is correct.

- Consider parity of 0s and 1s in 2.1,
- example: start with 13 0s and 13 1s
  - odd, odd (13)
  - even, even (6)
  - odd, odd (3)
  - odd, odd (1)
- The result, written right to left, is 1101, which is the binary representation of 13.
- Each pass checks equality of the next bit.
- Inequality in any specific bit will be detected (total number of 0s plus 1s will be odd).

Running Time Analysis

M_2: On input string w,

1. Scan across tape and reject if 0 is found to the right of a 1.
2. Repeat the following if 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).
3. If no 0s or 1s remain, accept, otherwise reject.

Running Time Analysis (cont.)

M_2: On input string w,

1. Scan across tape and reject if 0 is found to the right of a 1.
2. Repeat the following if both 0s and 1s appear on tape
   - 2.1 scan across tape, checking whether total number of 0s and 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).
3. If no 0s or 1s remain, accept, otherwise reject.

- One pass in each stage (1, 2.1, 2.2, 3) takes $O(n)$ time.
- stage 1 and 3: each executed once
- 2.2 eliminates half of 0s and 1s: $1 + \log_2 n$ times
- total for 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: Can the running time be made $o(n \log n)$?

Answer: Not on a single tape TM (proof on board).
Question:

But why do we have to stick with
single tape TMs?

Answer: We don't!

A Two Tape TM

M_3: 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: What is the running time?

Complexity

Deciding $\{0^n 1^n\}$:

- single-tape M_1: $O(n^2)$.
- single-tape M_2: $O(n \log n)$ (fastest possible!).
- two-tape M_3: $O(n)$.

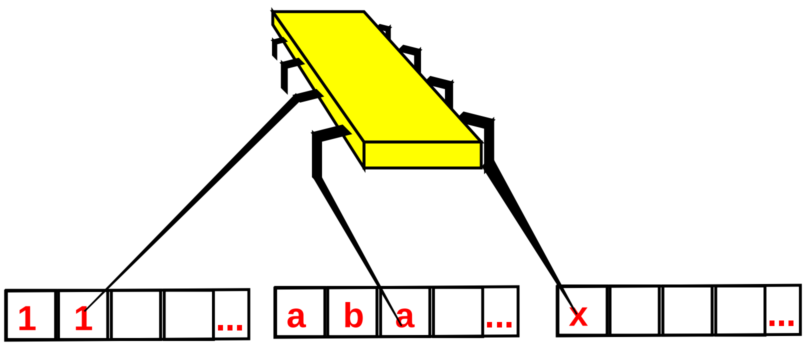
Important difference between complexity and computability:

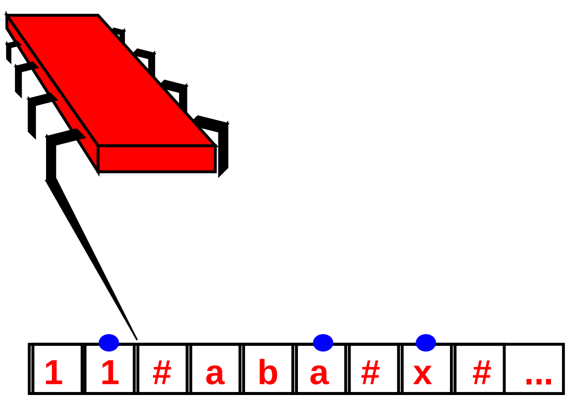
- Computability: all reasonable models equivalent (Church-Turing)
- Complexity: choice of model does affect run-time.

Q: By how much does model affect complexity?

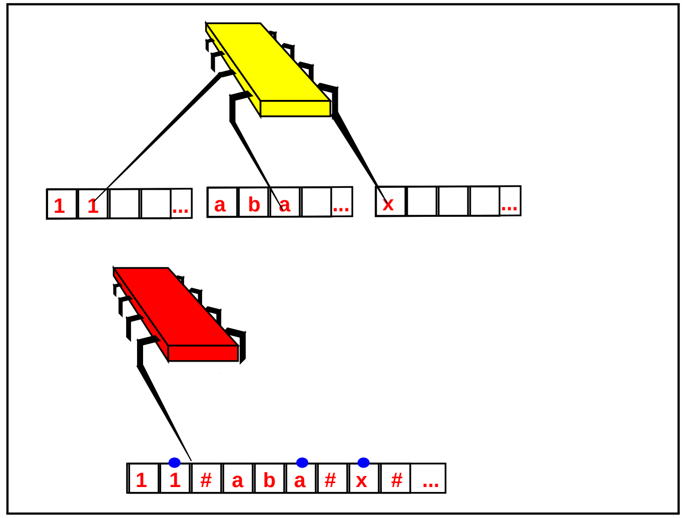
Models and Complexity

Let $t(n)$ be a function where $t(n) \geq n$, and let $L \subseteq \Sigma^*$ be a language.
Claim: If a $t(n)$-time multitape TM decides L, then $\exists$ an $O(t^2(n))$-time single tape TM that decided L.





Reminder: Simulating MultiTape TMs



On input $w = w_1 \cdots w_n$, single tape S:

- puts on its tape $\# \, \dot{w}_1 \, w_2 \cdots w_n \# \, \dot{\sqcup} \, \# \, \dot{\sqcup} \, \# \cdots \#$
- scans its tape from first $\#$ to $k+1$-st $\#$ to read symbols under "virtual" heads.
- rescans to write new symbols and move heads
- if S tries to move virtual head onto $\#$, then M takes "tape fault" and re-arranges tape.

Complexity of Simulation

For each step of M, S performs

- two scans
- 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:

- $O(t(n))$ steps to simulate one step of M.
- Total simulation $O(t(n)) \times O(t(n)) = O(t^2(n))$.
- Initial tape arrangement $O(n)$.
- Grand total: $O(n) + O(t^2(n)) = O(t^2(n))$ steps,
  - under the reasonable assumption **(why?)** that $t(n) > n$.

Time Classes Definition, Again

Let

$$t : \mathcal{N} \longrightarrow \mathcal{N}$$

be a function.

Definition:

$$\text{DTIME}(t(n)) \;=\; \{L \mid L \text{ is a language, decided by an } O(t(n))\text{-time TM}\}$$

Relations among Time Classes

Let $t_1, t_2 : \mathcal{N} \longrightarrow \mathcal{N}$ be two functions.

- Claim: 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?
- Claim: If $t_1(n) = O(t_2(n)/\log(n))$ then

$$\text{DTIME}(t_1(n)) \subsetneq \text{DTIME}(t_2(n)) .$$

- Informally, sufficiently more time does help.
- Proofs – sophisticated diagonalizations (omitted).

Non-Deterministic Time

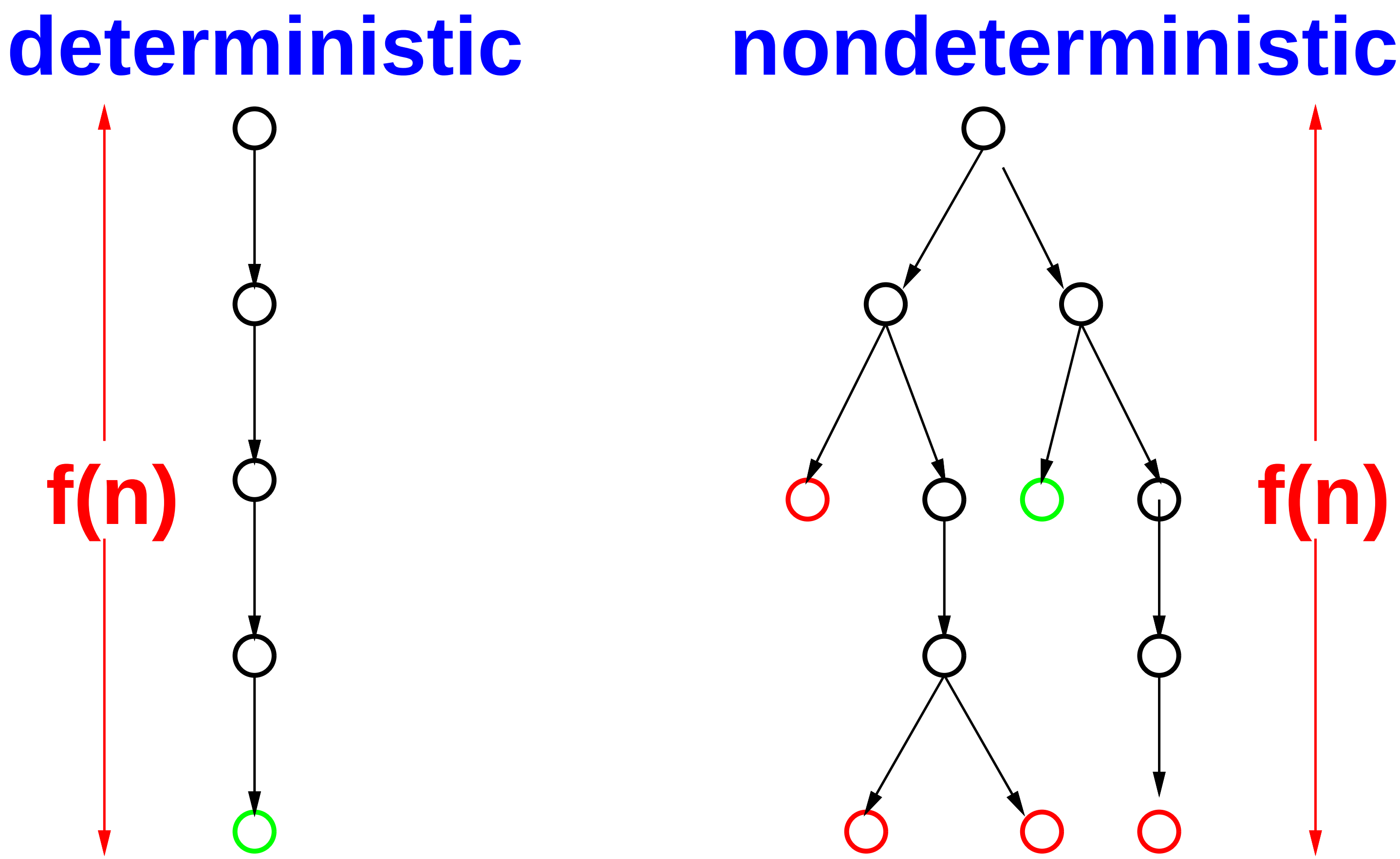
Let N be a non-deterministic TM, and let

$$f : \mathcal{N} \longrightarrow \mathcal{N}$$

We say that N runs in time $f(n)$ 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 $f(n)$.

Deterministic vs. Non-Deterministic



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

Models and Complexity

Claim: Suppose N is a nondeterministic TM that runs in time $t(n)$ and decides the language L.

Then there is an $2^{O(t(n))}$-time deterministic TM, D, that decided L.

Note contrast with multi-tape result.

Simulation

Let N be a non-deterministic TM running in $t(n)$ time. Want to describe the deterministic TM, D, simulating N.

Basic idea of simulation:

- D tries all possible branches.
- If D finds any accepting state, it accepts.
- If all branches reject, D rejects.
- Notice N has no looping branches, so exactly one of two possibilities must occur.

Simulation Details

N's computation is a tree:

- root is starting configuration,
- each node has bounded fanout $\leq b$ (why?),
- each branch has length $\leq t(n)$,
- total number of leaves at most $b^{t(n)}$,
- total number of nodes in tree $O\left(b^{t(n)}\right)$,
- time to arrive from root to any node is $O(t(n))$.
- $\Longrightarrow$ Time to visit all nodes is

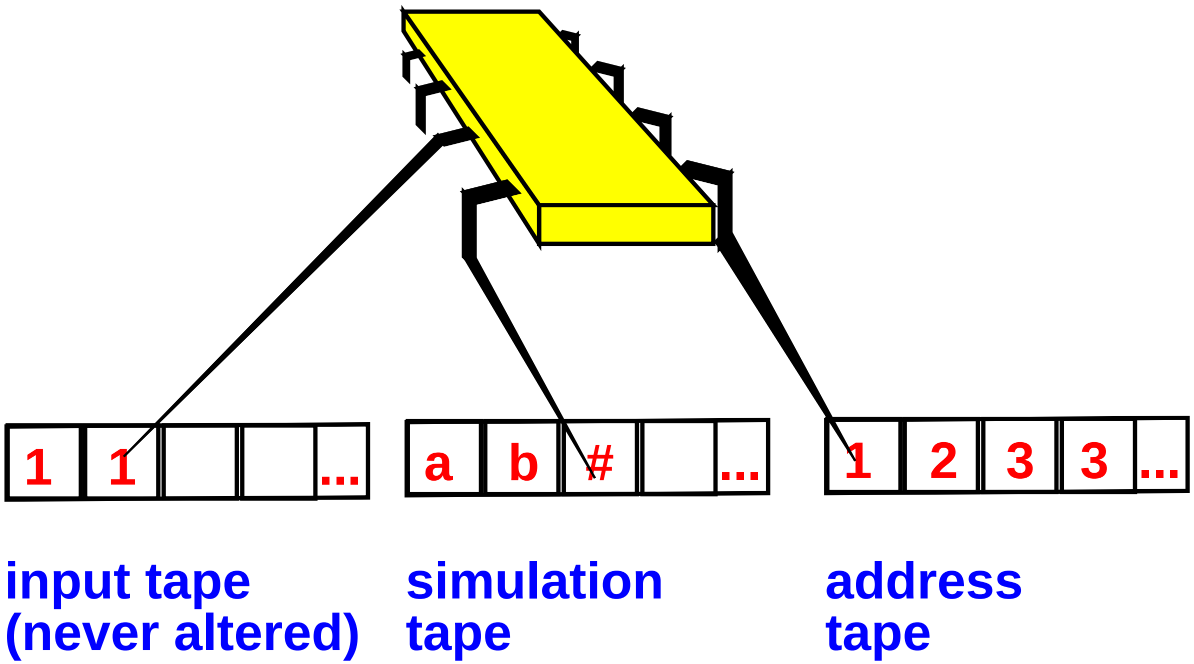
$$O\left(t(n) \times b^{t(n)}\right) = O\left(2^{O(t(n))}\right) .$$

Remark

Breadth-first search used in simulation

- Inefficiently traverses from root to visit each node.
- Can be improved upon by using depth-first search (why is it OK now?) or other tree search strategies.
- Still, doing this may save constants, but nothing substantial (why?)

Remark



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

Important Distinction

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

The Good, the Bad, and the Ugly

Complexity differences:
Polynomial is small; Exponential is large

	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

Polynomial is Good, Exponential is Bad

Claim: All “reasonable” models of computation are polynomially equivalent.

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

Question: Is a given problem solvable in

- Linear time? model-specific.
- Polynomial time? model-independent.
- We are interested in computation, not in models *per se*!