CSCI 361 Lecture 18:

Classes P and NP

Shikha Singh

Announcements & Logistics

- Hand **reading assignment # 12**
- Pick up **reading assignment # 13**
- **HW 7** due tomorrow 10 pm
- Office hours today:
    - **2.15 to 3.45 pm** (15 early)

Last Time

- Wrapped up Computability Theory
- Started discussion of time complexity
  - Zoom in on decidable problems
  - How long does it take to decide/solve them?
  - Extended Church-Turing thesis
    - Polynomial time in input: decidable in "reasonable time"

Today

- Time complexity comparison of multi-tape and nondeterministic TMs
- Revisit classes P and NP using Turing machine terminology

Time Complexity Class

Definition. Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The time complexity class, $\text{TIME}(t(n))$, is

$$\text{TIME}(t(n)) = \{L \mid L \text{ is decided by a TM in } O(t(n)) \text{ steps}\}$$

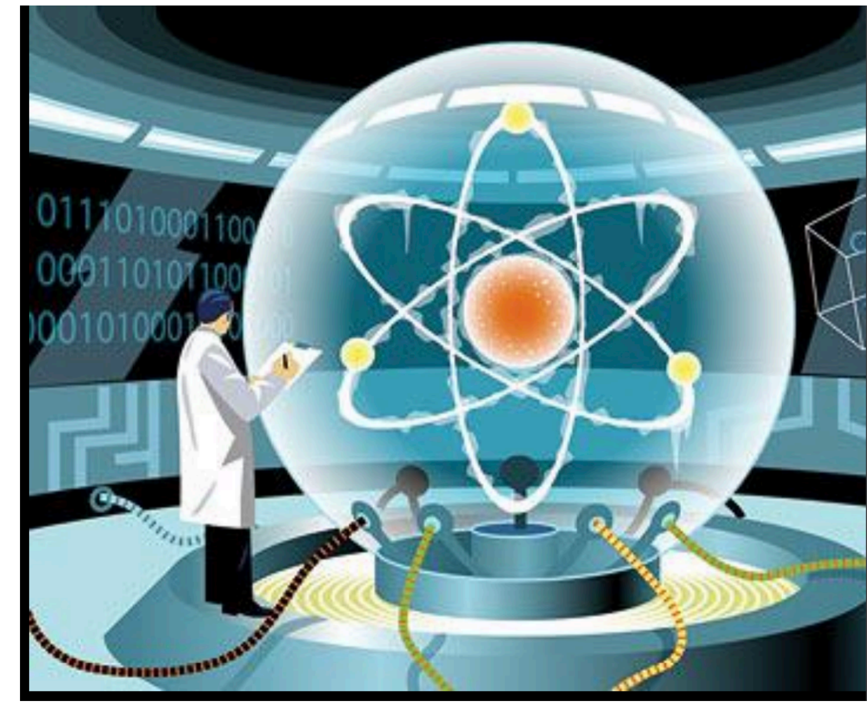
Complexity Class P

Definition. P is the class of languages that are decidable in polynomial time on a single-tape Turing machine. That is,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

Extended Church Turing Thesis

Everyone's intuitive notion of
efficient algorithms
= polynomial-time algorithms



- Much more controversial:
  - Is $O(n^{10})$ efficient?
  - Randomized algorithms/ quantum algorithms can do much better

Extended Church Turing Thesis

Everyone's intuitive notion of efficient algorithms = polynomial-time algorithms

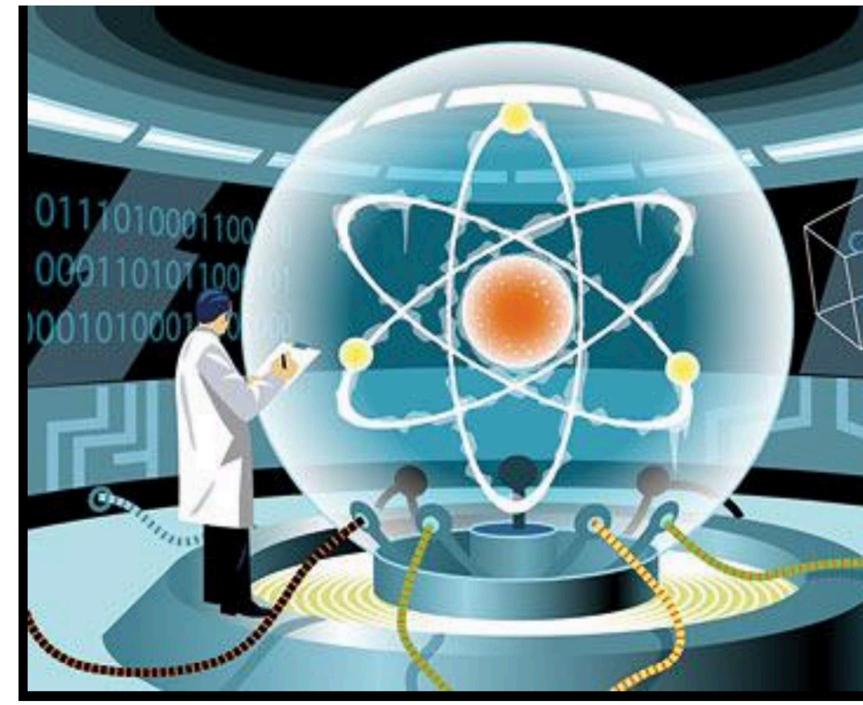


Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

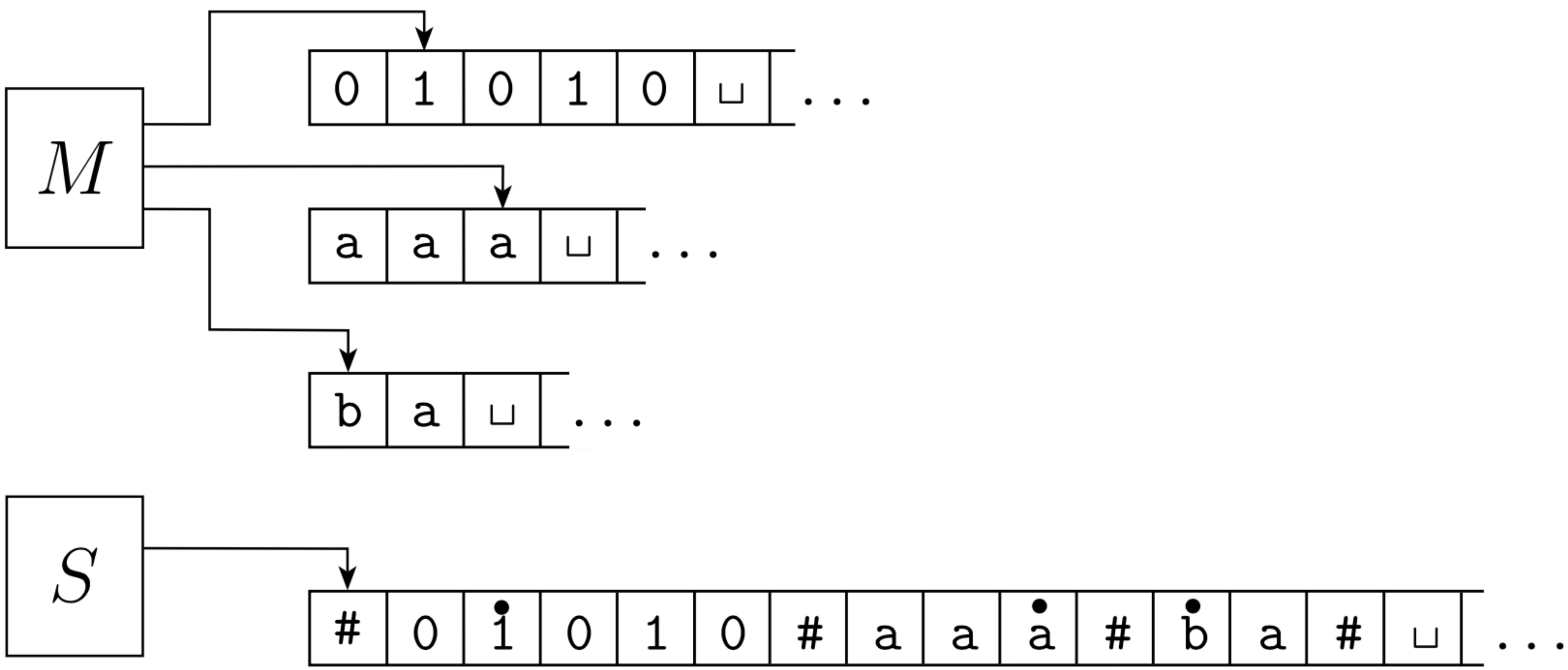
	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Two Tapes Can be More Efficient

- How quickly can we decide the language $A = \{0^n1^n \mid n \geq 0\}$ on a two tape TM?
    - Can do this in $O(n)$ time
- **Takeaway:** Different models of computation can yield different running times for the same language!
- Let's revisit multi-tape TM to single tape reduction with the lens of complexity theory

Multitape TM to Single Tape TM

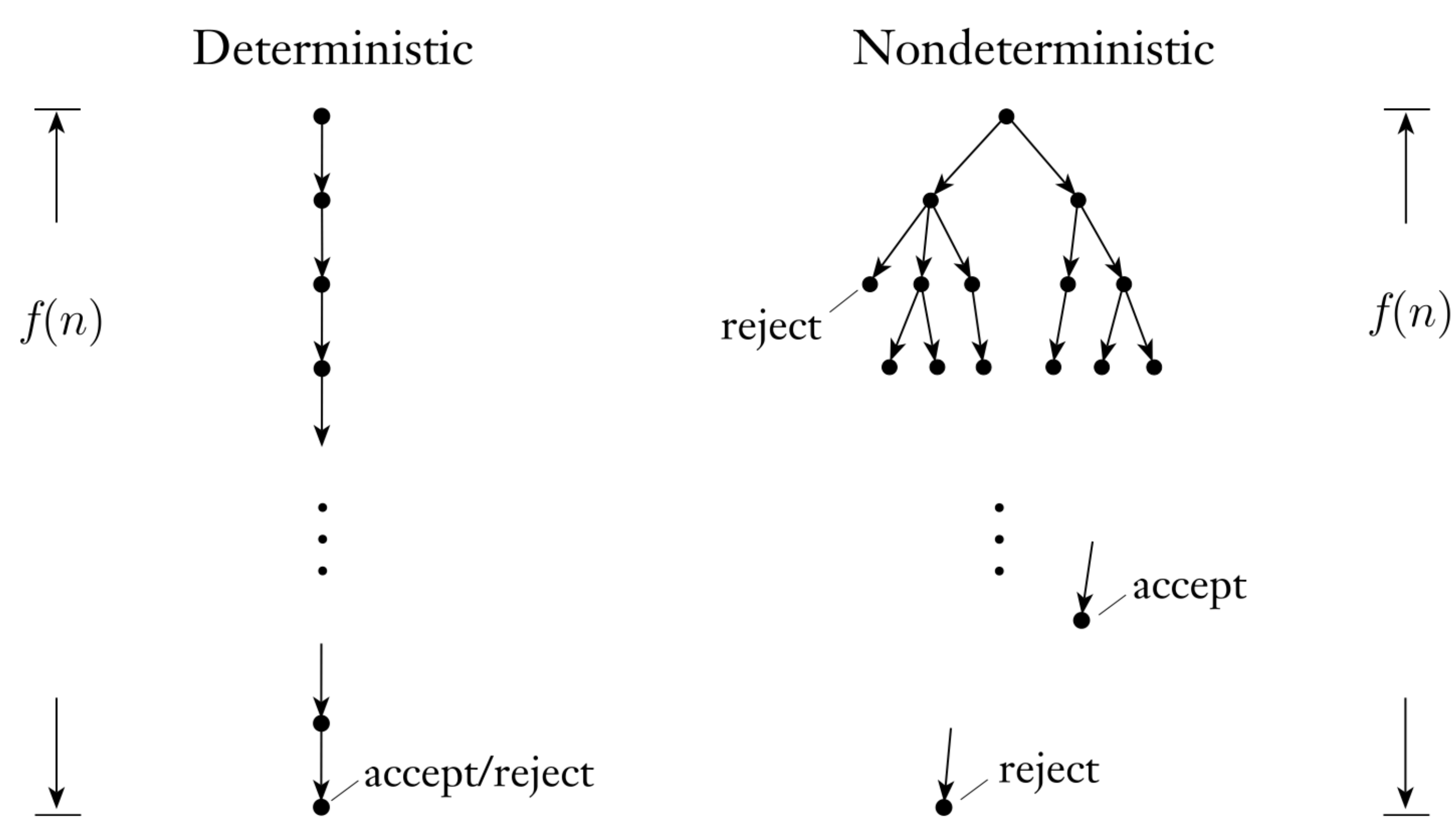
- **Theorem.** Every $t(n)$-time multi-tape TM has an equivalent $O(t^2(n))$-time single-tape TM, where $t(n) \geq n$.



- **Takeaway:** Both models are polynomially-equivalent.

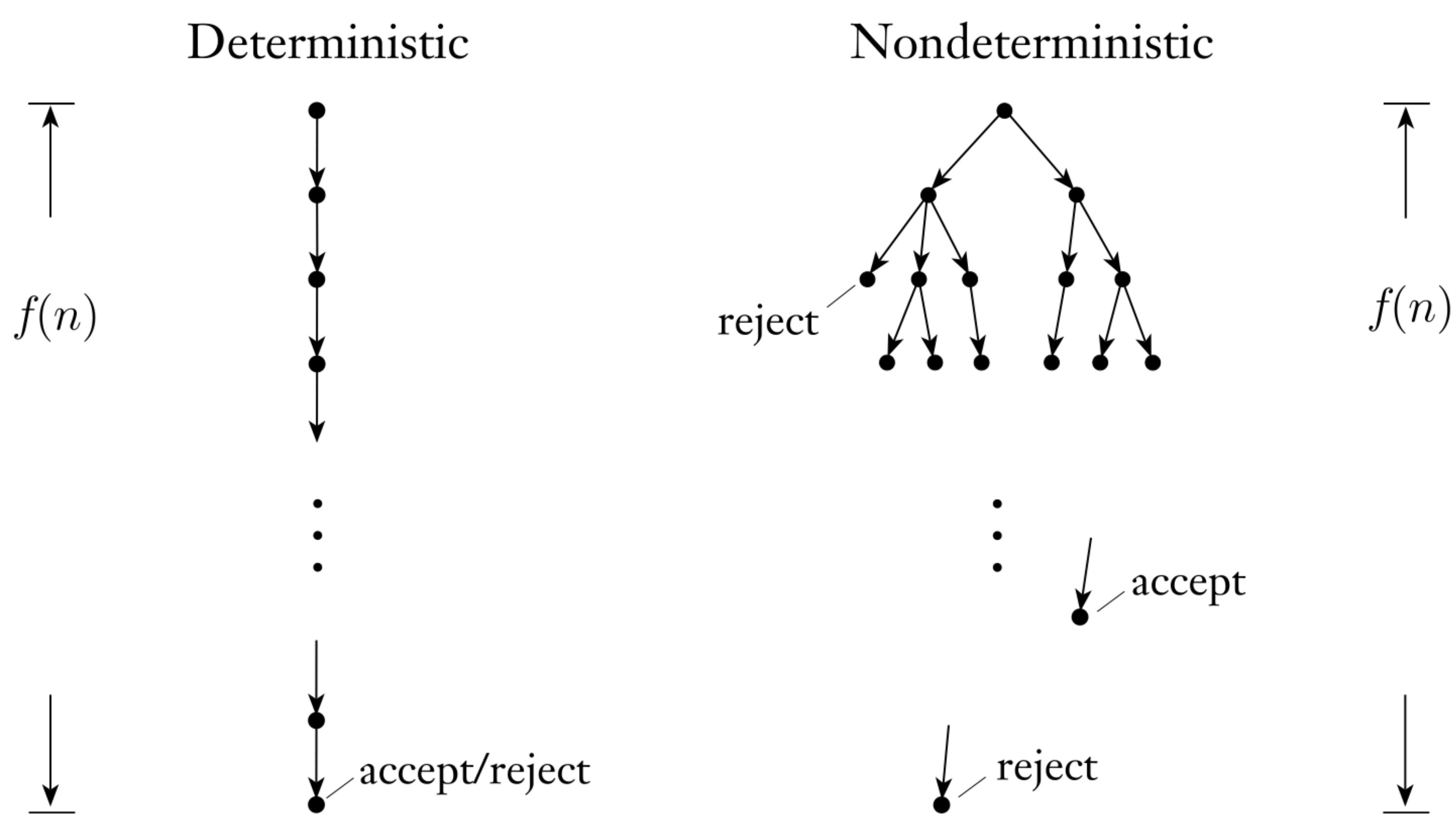
How About Non-Determinism?

- **Definition.** Let M be a non-deterministic TM that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M takes on any branch of its computation on any input of length n.



How About Non-Determinism?

- **Theorem.** Every $t(n)$-time non-deterministic TM has an equivalent $2^{O(t(n))}$-time deterministic TM, where $t(n) \geq n$.



- **Takeaway:** NTM is not polynomially-equivalent to a DTM.

Problems in P

- Studied extensively in CSCI 256, but will use "language terminology"
- Examples in the book:
  - PATH $= \{\langle G, s, t\rangle \mid$ Given graph G and nodes s, t there is a path from $s \to t\}$
  - RELPRIME $= \{\langle x, y\rangle \mid x, y$ are relatively prime $\}$
  - $A_{CFG} = \{\langle G, w\rangle \mid G$ is a CFG and $w \in L(G)\}$
    - Parsing problem for CFGs
- Let's look at the last one: discuss a common parsing algorithm
- One-off example of a dynamic program

Chomsky Normal Form

- Algorithm described in book: CYK Parsing Algorithm (by John **C**ocke, Daniel **Y**ounger, and Tadao **K**asami)
- Assumes G is in CNF:
  - All rules are of the form $A \rightarrow BC$, $A \rightarrow b$
  - Additionally allow $S \rightarrow \varepsilon$
- Converting a grammar to CNF incurs constant-factor blow up in size

CYK Parsing Algorithm

- Let the input $w = w_1 \ldots w_n$. Goal: Does there exists a derivation $S \rightarrow \cdots \rightarrow w_n$ using the rules of G
- table[i, j] = variables of G that generate substring $w_i w_{i+1} \ldots w_j$
  - How do we find out if w is in $L(G)$?
  - Check if $S \in$ table $[1\ ,\ n]$
- Base case?
  - Handle $w = \varepsilon$ by checking if $s \rightarrow \varepsilon$
  - Fill out the diagonal: table $[i, i] = A$ if $A \rightarrow w_i$

CYK Parsing Algorithm

- Next step: all substrings of length 2
  - for $i = 1, \ldots, n-1$
    - For each rule $A \to BC$, if table$[i,\ i]$ contains B and $[i+1,\ i+1]$ contains C, then add A to $[i,\ i+1]$
- Substring of length 3 and so on,
  - Need a "split" point k such that if $w[i,k]$ is generated by B and $w[k+1,\ j]$ is generated by C and $A \to BC$, add A to table$[i,j]$

CYK Parsing Algorithm

$D =$ "On input $w = w_1 \cdots w_n$:

1. For $w = \boldsymbol{\varepsilon}$, if $S \to \boldsymbol{\varepsilon}$ is a rule, *accept*; else, *reject*. $[\![w = \boldsymbol{\varepsilon}$ case $]\!]$

2. For $i = 1$ to n: $[\![$ examine each substring of length 1 $]\!]$

3. For each variable A:

4. Test whether $A \to \mathtt{b}$ is a rule, where $\mathtt{b} = w_i$.

5. If so, place A in $table(i, i)$.

6. For $l = 2$ to n: $[\![l$ is the length of the substring $]\!]$

7. For $i = 1$ to $n - l + 1$: $[\![i$ is the start position of the substring $]\!]$

8. Let $j = i + l - 1$. $[\![j$ is the end position of the substring $]\!]$

9. For $k = i$ to $j - 1$: $[\![k$ is the split position $]\!]$

10. For each rule $A \to BC$:

11. If $table(i, k)$ contains B and $table(k + 1, j)$ contains C, put A in $table(i, j)$.

12. If S is in $table(1, n)$, *accept*; else, *reject*."

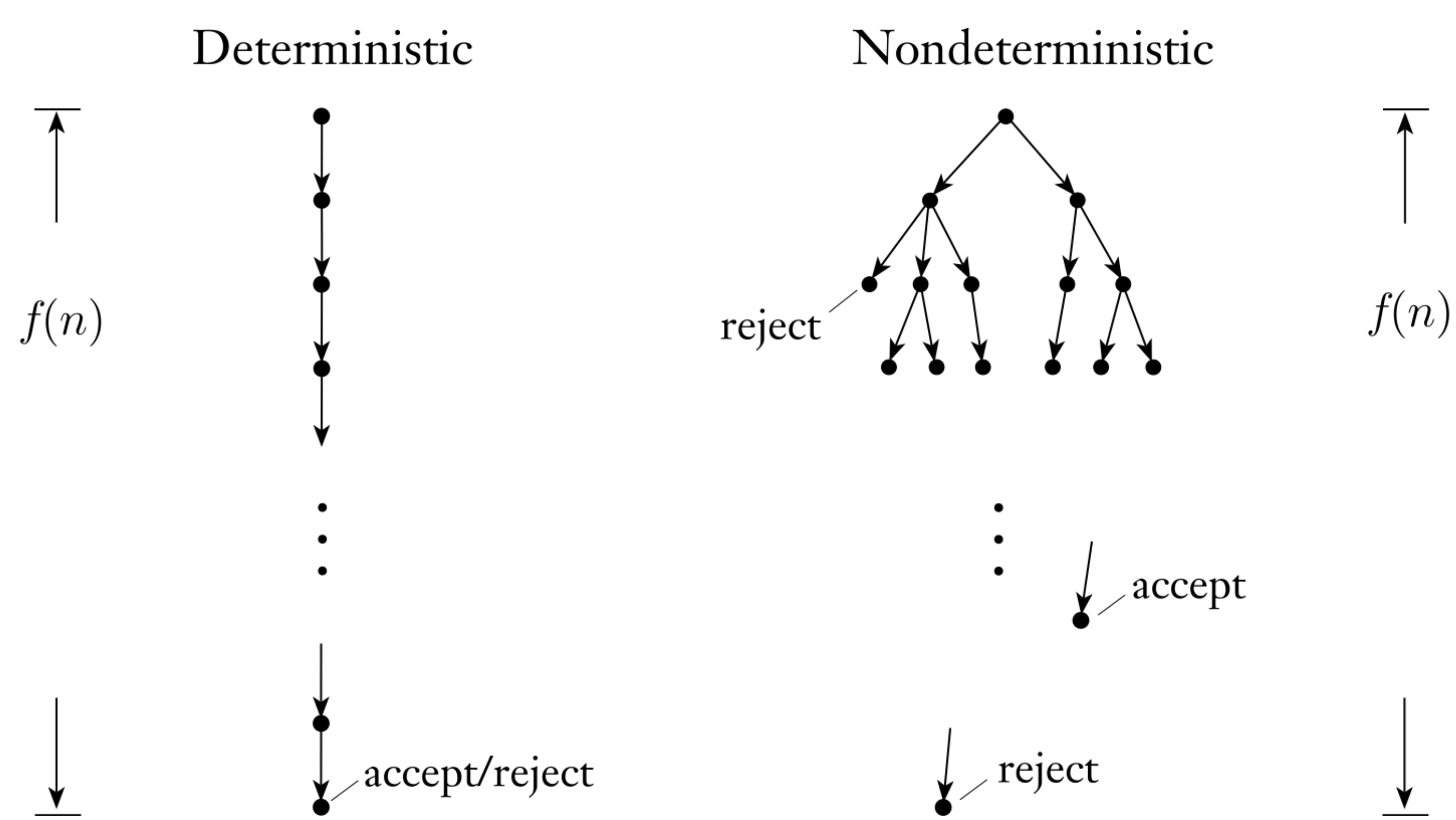
CYK Parsing is in **P**

- Running time of CYK parsing is $O(n^3)$
- Thus, verifying if a given CFG generates a given string is in **P**

Towards NP

- **Definition.** Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The time complexity class, NTIME$(t(n))$, is

$$\text{NTIME}(t(n)) = \{L \mid L \text{ is decided by an NTM in } O(t(n)) \text{ steps}\}$$



Complexity Class NP: Definition 1

Definition. **NP** is the class of languages that are decidable in polynomial time on non-deterministic Turing machine. That is,

$$\mathrm{NP} = \cup_k \, \mathrm{NTIME}(n^k)$$

Complexity Class NP: Definition 2

(Algorithms analog.) NP is the class of languages that have "polynomial-time verifiers"

Definition. A verifier for a language A is an algorithm V such that

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$$

- For each $w \in A$, there exists a string c s.t. V accepts $\langle w, c \rangle$ iff $w \in A$
- A polynomial-time verifier V runs in polynomial time in $|w|$
- Here c is a **certificate**: polynomial-length string, $|c| = \text{poly}(|w|)$
- Eg.
  HAMPATH $= \{\langle G, s, t \rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$

HAMPATH in NP

- HAMPATH $= \{\langle G, s, t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$
- For each "yes" instance $\langle G, s, t\rangle$, a certificate c is just a Hampath from s to t
- Following is a polynomial-time verifier:
    - On input $\langle\langle G, s, t\rangle, c\rangle$,
        - Check if c is a valid permutation of the nodes of G: that is, every node is present with no repetitions; reject if not
        - Check if c starts with s and ends with t; reject if not
        - Check if each adjacent pair of nodes correspond to an edge in G; reject if not
        - If all checks pass, c represents a valid Hamiltonian path from s to t in G and so accept

Hamiltonian Path

- Non-deterministic Turing machine?

Equivalent Definitions

- **Theorem.** A language can be decided by a NTM in polynomial time if and only if it has a polynomial time verifier.
- Proof outline.
  - Suppose it can be decided by a NTM, what is the certificate that an input $w \in L$?
  - Suppose it has a polynomial-time verifier, what should a NTM "guess" to show $w \in L$
- Takeaway: Class **NP** is the "one-sided" analog of Turing recognizable.