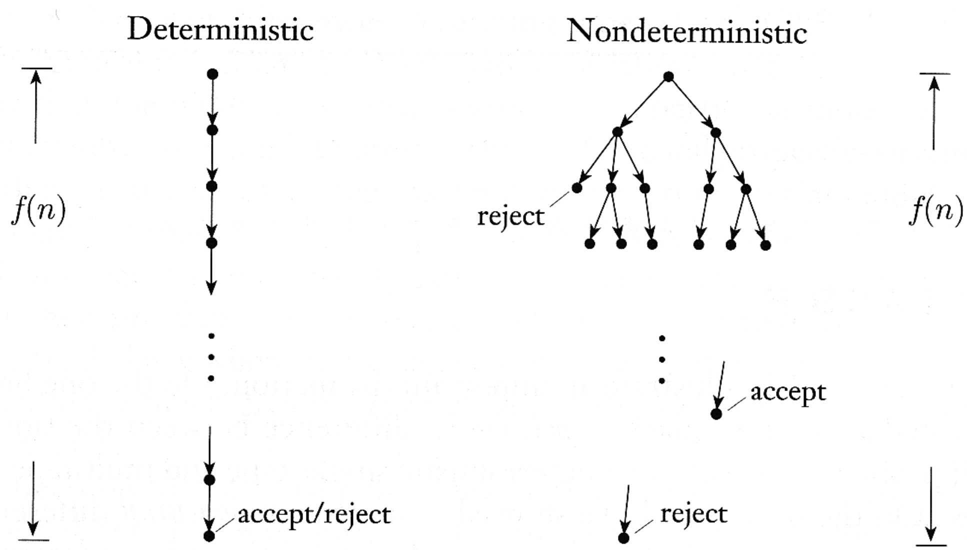
Nondeterministic Time Complexity

Definition: Let N be a nondeterministic Turing machine that is a decider. The ***running time*** of N is the function $f: \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on *any branch of its computation* on any input of length n.



Some Theorems

Theorem: Let $t(n)$ be a function, where $t(n) \geq n$, then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single tape Turing machine.

Proof Sketch: It is possible to show that simulating each of the $t(n)$ computation steps of the multitape machine on a single tape machine takes at most $O(t(n))$ steps. Therefore, to simulate the complete multitape computation on a single tape machine will take $t(n) \times O(t(n)) = O(t^2(n))$ steps.

Observation: Moving a computation from a multi-tape machine to a single-tape machine incurs an polynomial runtime penalty.

Some Theorems

Theorem: Let $t(n)$ be a function where $t(n) \geq 0$, then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Proof Sketch: Recall that simulating a nondeterministic Turing machine with a deterministic Turing machine can be viewed as searching the tree of nondeterministic computations for accepting states. Since the nondeterministic TM is a $O(t(n))$ time machine, the path from the root to a leaf node is bounded by $O(t(n))$ steps. There are at most $b^{O(t(n))}$ leaf nodes in the tree. Therefore we need to search

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

positions. Now, if we use a multi-tape deterministic TM, then we incur a polynomial penalty,

$$(2^{O(t(n))})^2 = 2^{O(t(n))+O(t(n))} = 2^{2O(t(n))} = 2^{O(t(n))}.$$

Searching the tree for an accepting state is an exponential operation bounded by $2^{O(t(n))}$ steps.

Observation: Moving a computation from a nondeterministic machine to a deterministic machine incurs an exponential runtime penalty.

Time Complexity Classes

Definition: Let $t: \mathbb{N} \rightarrow \mathbb{R}^+$ be a function. Define the ***time complexity class***, $TIME(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine, formally,

$$TIME(t(n)) = \{L | L \text{ is a language decided by an } O(t(n)) \text{ time TM}\}.$$

Example: Consider $A = \{0^k 1^k | k \geq 0\}$. We have shown that $A \in TIME(n^2)$ and also that $A \in TIME(n)$.

Observation: Notice that the same language can be a member of many time complexity classes depending on how clever we are with devising algorithms.

Observation: Perhaps this classification scheme is too fine grained, it does not capture the fundamental complexity differences between languages.

The Class P

Note that the difference between the algorithms of deciding the language A are polynomial differences, that is, $O(n^2)$ versus $O(n)$.

In general we can say that all reasonable deterministic computational models are ***polynomially equivalent*** – that is, any one of them can simulate another with only a polynomial increase in running time.

Compare this to algorithms that run efficiently on nondeterministic machines; simulating these algorithms on deterministic machines would incur an exponential increase in running time.

This motivates the polynomial time class P.

Definition: P is the class of languages that are decidable in polynomial time on a deterministic Turing machine,

$$P = \bigcup_k TIME(n^k), \text{ for } k \geq 0.$$

The Class P

The class P is interesting because:

- P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing machine.
- P roughly corresponds to the class of problems that are realistically solvable on a computer (i.e., the computer is a reasonable deterministic computational model).

Example: Note that with this definition our language $A = \{0^k 1^k | k \geq 0\}$ is clearly a member of P regardless of which exact algorithm we use to decide it.

$PATH \in P$

Theorem: $PATH \in P$, where $PATH = \{\langle G, s, t\rangle |\ G$ is a directed graph that has a directed path from node s to node t $\}$.

Proof: A brute force search for the path does not work, since such an algorithm will run in exponential time (compare to the tree searching when simulating a nondeterministic TM).

However, we can be clever and implement an incremental search.

M = "On input $\langle G, s, t\rangle$:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
   3. Scan all edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*; otherwise, *reject*."

Analysis. It is clear that stages 1 and 4 run in constant time (or $O(n)$ where $n = |\langle G, s, t\rangle|$). Stage 3 runs at most m times where m is the number of nodes in the graph. Considering that we have to scan the representation of G every time through the loop we obtain $O(m \times n)$ computation steps. Now considering that $m = \frac{1}{k} \times n$ with $k = 1, 2, \ldots$, that is, m is a fraction of the total representation. This gives us an overall complexity of the algorithm of

$$O(m \times n) = O(\frac{1}{k} \times n \times n) = O(n^2).$$

Thus $PATH \in P$. $\square$

$CFL \in P$

Theorem: Every context-free language is in P.

Proof: The brute force method does not work, have to search the full derivation tree for all possible derivation, we can use the Chomsky normal form for this - but this is an exponential time problem.

However, we can use dynamic programming where we store previously computed partial solutions. This is how real parsing algorithms work. Time complexity $O(n^3)$.$\square$

Class NP

This is the class of problems that either have as of yet unknown polynomial solutions or are intrinsically difficult (simply do not have a polynomial solution).

Definition: We define the ***nondeterministic time complexity class*** NTIME as follows, $NTIME(t(n)) = \{L | L$ is a language decided by an $O(t(n))$ time nondeterministic TM$\}$.

Definition:

$$NP = \bigcup_k NTIME(n^k), \text{ for } k \geq 0.$$