

4/6/2019

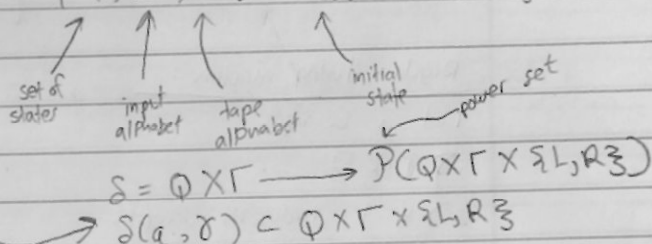
Homework due Friday! Discussion on HW this Wednesday.

Non-determinism: simultaneously branches in different ways.

Rather than "if-then", pursues various paths at same time.

NDTM is a 7-tuple, $M = \{Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}\}$

TM simultaneously computes along branches given by this subset.



Computing power is equivalent to TM.

Any non-deterministic Turing machine has an equivalent Turing machine.

PROOF: Given NDTM N , we want to design an equivalent TM, M , which accepts precisely the same set of strings as N .

IDEA: TM just iterates over all the possibilities given by transition function $\delta(q, \gamma)$.

DESIGN M : Three tapes.

a)

0	1	0	1	...	1	...
---	---	---	---	-----	---	-----

 ← INPUT TAPE. Static, never changes.

b)

0	1	0	1	...	1	...
---	---	---	---	-----	---	-----

 ← SIMULATOR

c)

--	--	--	--	--	--	--

 ← KEEPS TRACK OF BRANCHING.

Algorithm

1) Copy input onto tape 2.

2) Set $n=1$. For each string $a_{i_1} a_{i_2} a_{i_3} \dots a_{i_n}$ in $Q \times \Gamma \times L \times R^n$ of length n , simulate the branch of N given by $a_{i_1} \dots a_{i_n}$ on tape 2.

If $a_{i_j} \notin \delta(q, \gamma)$, skip.

3) Accept if NDTM N accepts.

4) $n=n+1$, repeat 3.

COR: A language $A \subseteq \Sigma^*$ is recognized by a NDTM
iff also recognized by a TM.

MEASURING COMPLEXITY OF ALGORITHMS

How can we measure complexity?

Big-O Notation:

EXAMPLE: NOTPRIME = $\left\{ \begin{array}{l} \text{integers } \geq 0 \\ \text{that aren't prime.} \end{array} \right.$

Regular Turing machine:

Let n be input

- 1) $i = 2$
- 2) if i divides n , accept.
- 3) otherwise, $i = i + 1$, repeat #2.

If $i = n$, reject.

$$l = \log_2(n) + 1$$

This is $O(n)$.

n steps $\approx 2^l$ steps where l = length of input in binary.

NDTM: 1) $i = 2$

- 2) Check simultaneously if $2, \dots, 2^i$ divide n .
- If yes, accept.

$$\# \text{ steps} = \log_2(n)$$

$$\approx l$$

Let $f(n)$ be a function, $\mathbb{N} \rightarrow \mathbb{N}$.

$g(n)$ is also a function.

$f(n)$ is $O(g(n))$ if \exists constant c $f(n) \leq c g(n) \forall n$

Example: $3n + 6$ is $O(n)$ $3n + 6 \leq 9n$

$7n^3 + 8n + 19$ is $O(n^3)$.

$2^n + n^{2019} = O(2^n)$, because exponential grows faster.

Defn: Let M be a TM.

The running time of M is $f(n) = \max(\# \text{ steps } M \text{ takes on input length } n)$.

Complexity classes

$P = \{ A \subset \{0,1\}^* \mid \exists \text{ T.M. which recognizes } A \text{ with polynomial runtime, i.e. } O(n^k) \}$

Runtime is bounded by a polynomial.

We define NP in the same way:

$NP = \{ A \subset \{0,1\}^* \mid \exists \text{ NDTM recognizes } A \text{ with } O(n^k) \}$

FACT: $P \subset NP$.

Question: Is $P = NP$?