4/8/2019

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, $M = \{Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\}$

TM simultaneously computes along branches given by this subset.

Q: set of states; Σ: input alphabet; Γ: tape alphabet; q_0: initial state

$\delta = Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ (power set)

$\delta(q, \gamma) \subset Q \times \Gamma \times \{L, R\}$

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.

↓ step 2

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 $a_{i_1}, a_{i_2}, a_{i_3} \ldots a_{i_n}$ in $Q \times \Gamma \times \{L, R\}$ of length n, simulate the branch of N given by $a_{i_1} \ldots a_{i_n}$ 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 $A \subset \Sigma^*$ 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 = {integers ≥ 0 that aren't prime.

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.

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

This is $O(n)$.

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

NDTM: 1) $i = 2$

2) Check simultaneously if $2, \ldots, 2^\ell$ divide n.

If yes, accept.

steps $= \log_2(n)$

$\approx \ell$

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

$g(n)$ 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 $O(n)$ $\quad 3n+6 \leq 9n$

$7n^3 + 8n + 19$ 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 $P = NP$?