COMS3261: Computer Science Theory

Fall 2013

Mihalis Yannakakis

Lecture 19, 11/13/13

Robustness of Turing machines

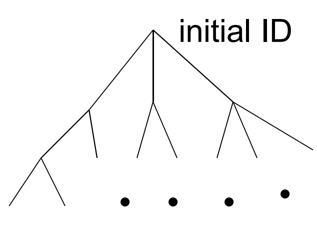
- Many other models, more general or less, equally powerful: all recognize the same languages & can compute the same functions
- Multitape TMs
- Nondeterministic TMs
- Multistack machines
- Counter machines
- “Real” computers

Nondeterministic Turing Machines (NTM)

- 1-tape or multitape: For each state q, and tape symbol X (or tuple $X_1,\dots,X_k$) can have many choices (0,1, or more)
- Acceptance: Input accepted if ∃ sequence of moves that leads to an accepting state.
- Note: possible that some computations run forever, while some others lead to accepting state (and halt): input is accepted in this case
- Multitape NTM simulated by 1-tape NTM (same as for DTM)
- 1-tape NTM → multitape DTM → 1-tape DTM

1-tape NTM → multitape DTM

- Idea: Generate systematically all the ID's reachable from initial state in n steps for n=1,2,3 …
- Breadth-First Search of tree of possible computations of NTM



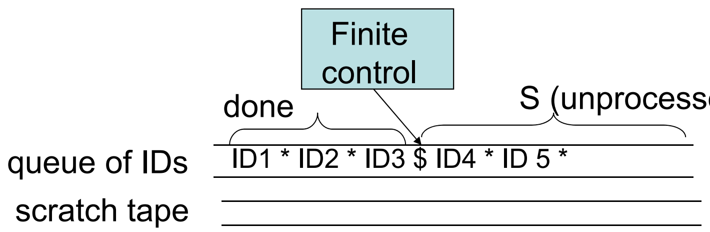
important that it is BFS and not DFS, because some computations may run forever

BFS: Maintain a queue of reached and unprocessed IDs

Simulation of 1-tape NTM by 2-tape DTM

- Queue S of ID's. Take out head ID, add all its successor ID's
- DTM: Tape 1 has queue S of ID's, 2nd tape =scratch tape

1. Copy next (head) ID from tape 1 to tape 2. If state of ID accepting, then accept and halt.
2. Determine in finite control the next moves of NTM, and go to end of tape 1.
3. For each of the next moves, copy the modified ID from tape 2 to end of tape 1.
4. Return to marked $ position, erase and move $ to next *



Time of Simulation

- If # choices in each step of NTM is b (≥2), then b possible IDs in step 1, b^2 in 2 steps, b^3 in 3 steps, …, b^i in i steps

$\Rightarrow$ in t steps of NTM on input of length n,

we have Σb^i IDs $\leq b^{t+1}$ IDs, each $\leq$ t+n long

Processing each ID takes time $\sim$ $(t+n)b^{t+1}+b(t+n)$ $\Rightarrow$

$\Rightarrow$ total time for t steps of NTM $\leq$ $O((t+n)^2 b^{2t})$

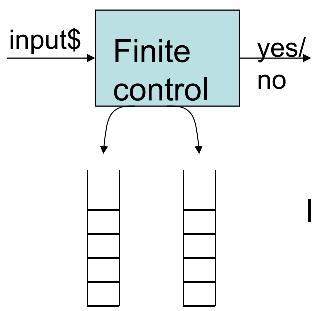
In general exponential blowup in time.

- Nondeterministic TMs can be simulated by Deterministic
- but time?

- Major Open problem of CS:
  Is the exponential blowup necessary?

Multistack Machines

- Assume input is w$, with endmarker $

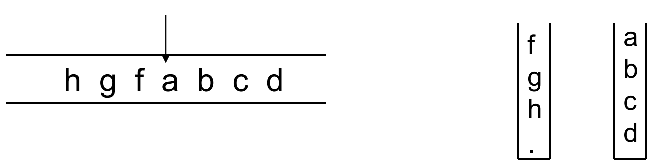


$\delta(q,a,X_1,X_2) = (p, \sigma_1,\sigma_2)$

$a \in \Sigma \cup \{\varepsilon\}$, $X_1,X_2 \in \Gamma$, $\sigma_1,\sigma_2 \in \Gamma^*$

Initially: Bottom symbol in each stack

1-tape DTM $\rightarrow$ 2 stack DPDA



- Simulation of TM.
- Initialization: Read input (to $) and put in stack 1. It is reversed , so transfer to stack 2. Now stack 2 has input with leftmost symbol on top.
- Loop (simulation of TM step): Update state, and modify and move top symbol from one stack to the other
- Example: change a to k, move right $\rightarrow$ push k on stack 1, pop a from stack 2

Counter Machines

- Counter: Holds a natural number (≥0).
- Operations: test for 0, increment (by 1), decrement (if >0)
  ⇔ stack with Z_0 at bottom and X's over it: counter i ⇔ X^iZ_0
- Move of k-counter machine depends on state and 0/non-0 status of counters

- 1 counter machine (1CM) = special case of DPDA
- Example: can recognize $\{0^n1^n\}$ with 1CM

- 2 counter machines ⇔ TMs

2 stacks → 3 counters

- 1 counter/stack + a scratch counter
- Suppose Γ of DPDA has r-1 symbols 1,…,r-1 ⇒ treat stack content = base-r number s, top = least significant digit
- top = s mod r: implemented by moving s from counter to scratch counter counting mod r, and then returning s
- push X ⇔ $s := r \cdot s + X$
- pop = [s/r] (integer part)

- Multiply/divide a counter by r: Use scratch counter, subtract r from one counter add 1 to the other or vice-versa

Multiply/divide counter by r

- Example: multiply counter 1 by r ($c1 := r \cdot c1$)

c3 =0
while (c1 $\neq$0)
 { c1 -- ; for j=1 to r c3++ }
   [now c3 = $r \cdot c1$]
while (c3 $\neq$0)
 { c3 -- ; c1++ }
 [now c1 = $r \cdot c1$, c3 =0]

3 counters $\rightarrow$ 2 counters

- Idea: counters i,j,k $\leftrightarrow$ integer $m = 2^i 3^j 5^k$
- One counter stores m (=all 3 counters), 2nd counter=scratch

- Test i (j,k) for 0: Transfer m to scratch counter counting mod 2 (3,5) in FC and then transfer back.
- Increment i (j,k): Multiply m by 2 (3,5) (using scratch counter)
- Decrement i (j,k): Divide m by 2 (3,5) (using scratch counter)

- Important that 2,3,5 are primes, so the mapping from the 3 counter values i,j,k in the 3CM to the 1st counter value m in the 2CM is 1-1.