

# CS 39

## Theory of Computing

Nondeterministic Turing Machines  
Church-Turing Thesis  
Amit Chakrabarti

## A closure property

- If  $L_1$  and  $L_2$  are decidable, so is  $L_1L_2$ .
- **Proof:** Let  $M_i$  be a decider for  $L_i$  ( $i = 1,2$ ).  
Let  $M$  be the following 3-tape TM:  $M =$  "On input  $x$ ,
  1. Insert a '#' symbol to the left of  $x$ .
  2. While char to right of '#' is non-blank:
    - 2.1. Copy string to left of '#' to tape 2.
    - 2.2. Copy string to right of '#' to tape 3.
    - 2.3. Simulate  $M_1$  using tape 2 as input tape.
    - 2.4. If  $M_1$  accepts, then:
      - 2.4.1. Simulate  $M_2$  using tape 3 as input tape.
      - 2.4.2. If  $M_2$  accepts, then ACCEPT.
    - 2.5. Shift the '#' symbol on tape 1 one space to the right.
  3. REJECT.

## More closure properties

- If  $L_1$  and  $L_2$  (over  $\Sigma$ ) are decidable, so is
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $\Sigma^* \cdot L_1$
  - $L_1L_2$
  - $L_1^*$
- Think how you might prove these.

## Nondeterministic Turing Machines

- Abbreviated NDTM
- TM's by default deterministic
  - *Always say NDTM if you want nondeterminism*
- 7-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ , just like a TM
  - Except,  $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$
- This means, a configuration can now yield several different new configurations.

## NDTM computation formalized

- Consider NDTM  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$
- We say  $M$  accepts  $x \in \Sigma^*$  if
  - $\exists$  sequence  $C_0, C_1, \dots, C_t$  of configurations of  $M$  s.t.
  - $C_0 = q_0x$
  - $C_{i-1}$  yields  $C_i$  (for all  $i, 1 \leq i \leq t$ )
  - $C_t$  is an accepting configuration
- Want NDTM  $M$  for language  $L$ . What if  $x \notin L$ ?
  - Happy if  $M$  never enters accept state (recognizer)
  - Require  $M$  to always enter reject state (decider)

## Uses of nondeterminism

- Simpler solutions for  $\{ww : w \in \Sigma^*\}$ 
  - Use nondeterminism to guess midpoint; insert a '#' symbol there. Then proceed as for  $\{w#w : w \in \Sigma^*\}$ .
  - Better yet, use two tapes *and* nondeterminism.
- Simpler proofs of closure properties
  - For  $L_1L_2$  and  $L_1^*$ , just use nondeterminism to decide how to “break up” the input.
- But, to prove decidability, we want TMs, not NDTMs and not  $k$ -tape NDTMs!

## Digression: Lexicographic ordering

- Let  $\Sigma$  be an alphabet. Order the elements of  $\Sigma$  arbitrarily, as though creating a lexicon. E.g.,
  - English alphabet is usually ordered  $a, b, c, \dots, y, z$
  - ASCII characters are ordered in a specific way
- This ordering on  $\Sigma$  induces an ordering on the strings in  $\Sigma^*$  as follows:
  - A shorter string comes before a longer one
  - Strings of the same length are ordered as in a lexicon (a.k.a. dictionary). Thus,  $aabfcda < aadaaba$ .

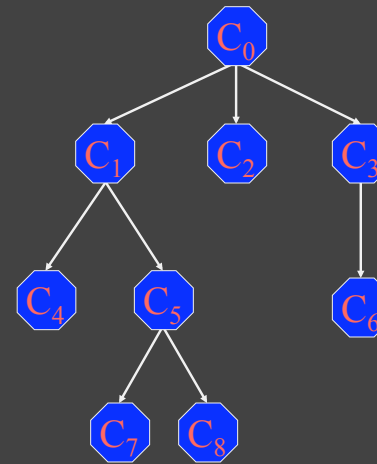
## Digression: Lexicographic ordering

- A TM can replace a string on its tape with the lexicographically next one.
- Method best illustrated by example:  $\Sigma = \{1,2,3\}$ 
  - Lexicographic order is:  $\epsilon, 1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, 112, \dots, 1321233, 1321311, \dots$
  - Move head to rightmost symbol. If possible, increment symbol under head and stop. Else, set this symbol to 1, move head left and repeat. If head moves all the way to the left, append a 1 to the string.

## Turning a NDTM into a TM

- Let  $N$  be a NDTM; design TM  $M$  to simulate  $N$ .
- $M$  will try out every possible **branch** of  $N$ 's computation. A branch is a particular sequence of nondeterministic choices.
- $M$  will have three tapes
  - tape 1: the input to  $N$ , never overwritten
  - tape 2: the tape of  $N$  in the current branch
  - tape 3: string of integers describing current branch

## Representing the branches as strings



Config  $C_6 \rightarrow$  string "3 1"

Config  $C_7 \rightarrow$  string "1 2 1"

Branching factor of tree  
 = max # nondeterministic  
 choices at a configuration  
 =  $\max \{ |\delta(q,a)| : q \in Q, a \in \Gamma \}$

## Sample configs of $M$

Suppose

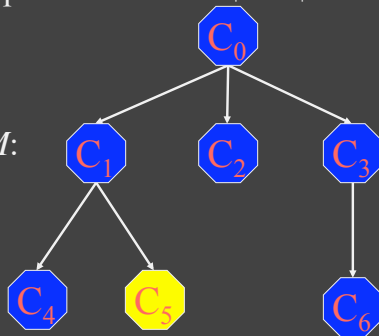
- input to  $N$  is 11010
- in configuration  $C_5$  the tape of  $N$  reads 10\$a0b\$1

Then, possible config for  $M$ :

Tape 1: 1 1 0 1 0

Tape 2: 1 0 \$ a 0 b \$ 1

Tape 3: 1 2



## Sample configs of $M$

Suppose

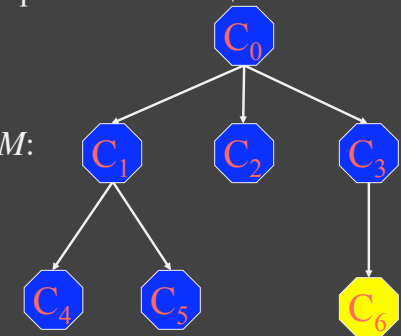
- input to  $N$  is 11010
- in configuration  $C_6$  the tape of  $N$  reads a\$01

Then, possible config for  $M$ :

Tape 1: 1 1 0 1 0

Tape 2: a \$ 0 1

Tape 3: 3 1



## Simulating the NDTM

- $N$  accepts if somewhere in its computation tree there is an accepting configuration.
- $M$  will visit the entire tree looking for such a configuration.
- Pitfall: some of the branches in  $N$ 's computation could be infinitely long (if  $N$  loops). This may cause  $M$  to loop and not “find” an accepting configuration elsewhere in the tree.
- Solution: **traverse the tree breadth-first!**

## Simulation, continued

- Breadth-first traversal == lexicographic ordering
- $M =$  “On input  $x$ :
  1. Write “1” on tape 3.
  2. Copy  $x$  from tape 1 to tape 2.
  3. Simulate  $N$  using tape 2. Whenever  $N$  has a nondeterministic choice, follow the  $j$ th branch, where  $j$  is the next symbol on tape 3. If there are fewer than  $j$  branches at this stage, or there are no more symbols on tape 3, or if  $N$  is in a rejecting config, GOTO 4. If  $N$  is in an accepting config, then ACCEPT.
  4. Replace the string on tape 3 with the lexicographically next string and GOTO 2.”

## Random Access Machine (RAM)

- A formal model of a typical computer:
- Infinite number of **memory words** numbered  $0, 1, 2, \dots$
- Finite number of **arithmetic registers**, including a **program counter** that holds location of next instruction.
- Each word and register can hold any integer. Each initially holds 0 until overwritten.
- Integers decoded into usual computer instructions: LOAD, STORE, ADD, MULT, JUMP, JNZ, etc.
- A Turing Machine is as powerful as a RAM.

## Simulation of a RAM by a TM

- Use 4 tapes:
  - Tape 1: All overwritten memory words of the RAM. Format:  
# 0 \$  $v_0$  # 1 \$  $v_1$  # 10 \$  $v_2$  # 101 \$  $v_5$  ..... # 11010 \$  $v_{26}$  # ...
  - Tape 2: Program counter
  - Tape 3: Contents of all other registers
  - Tape 4: Memory address tape (= simulation's scratch work space)
- Main simulation loop:
  - Fetch next instruction from location given on tape 2. Decode it using state transitions. If instruction involves read/write at a memory address, copy address to tape 4 and look for contents on tape 1. Increment tape 2, or modify appropriately if this is a JUMP instruction.

## High-Level Descriptions

- Now that we can do on a TM anything a computer can...
  - Feel free to use IF-ELSE, FOR, WHILE, etc.
  - Feel free to use pseudocode
- Three levels of descriptions of a TM:
  - Formal description: spell out all the states and the transition function.
  - Implementation description: in English prose, describe how the TM moves its heads, what it writes, etc.; leave out states.
  - High-level description: in English prose, describe an algorithm, ignoring implementation details.

## Church-Turing Thesis

- Every known reasonable model of computing turns out to be equivalent to a TM. So, we **suspect** that TMs correctly capture our intuitive notion of “algorithm.”
- *“Intuitive notion of algorithms = TM algorithms”*
  - Alonzo Church
  - Alan M. Turing
- Can’t prove: what is a “reasonable” model?