

CS20a: Turing Machines (Oct 29, 2002)

- So far:
 - DFA = regular languages
 - PDA = context-free languages
- Today:
 - Computability



Handicapped machines

- DFA limitations
 - Tape head moves only one direction
 - 2-way DFA has equivalent power
 - Tape is read-only
 - Tape length is a constant
- PDA limitations
 - Tape head moves only one direction
 - 2-way PDA has a little more power (can accept $\{ ww \mid w \text{ in } \Sigma^+ \}$, which is not context free)
 - Tape is read-only, but stack is writable
 - Stack has only LIFO (last-in, first-out) access
 - Tape length is constant, but stack is not bounded
- What about:
 - Writable, 2-way tape?
 - Random-access "stack?"

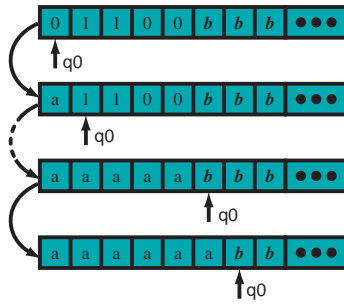


Computability

- Computible
 - Equivalence of DFAs (we gave an algorithm)
 - String membership in a regular language (gave an algorithm)
 - Determining valid executions of a PDA (there is an algorithm)
- Uncomputible
 - Equivalence of CFGs
 - Emptiness of the complement of a CFL (emptiness of CFL is computible)
 - Whether a C program ever terminates
 - **How to prove it?**



Nontermination execution



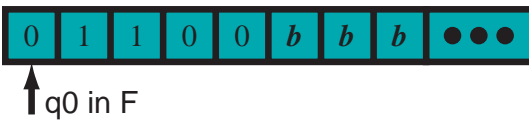
Termination

A *partial function* $U \rightarrow V$ is a function that may not be defined for every argument $u \in U$

- A TM that accepts Σ^*
- $Q = \{q_0\}$
- $\Gamma = \Sigma \cup \{\bar{b}\}$
- $\delta(q_0, c)$ is arbitrary (possibly not defined)
- $s = q_0$
- $F = \{q_0\}$



Termination execution



Instantaneous descriptions

- An *instantaneous description* $ID(\sigma)$ is $\alpha_1 q \alpha_2$, where
 - q is the current state of the TM,
 - $\alpha_1 \alpha_2 \in \Gamma^*$ if the contents of the tape (to the last non-blank symbol)
 - The current symbol is the first symbol of α_2



Transitions

- Define a *left transition*

$$a_1 \dots a_{i-1} \underline{q} a_i, \dots, a_n \rightarrow_L a_1, \dots, a_{i-2} \underline{p} c a_i \dots a_n$$
- Define a *right transition*

$$a_1 \dots a_{i-1} \underline{q} a_i, \dots, a_n \rightarrow_R a_1, \dots, a_{i-1} \underline{c} p a_{i+1} \dots a_n$$
- A transition \rightarrow is either \rightarrow_L or \rightarrow_R



Executions

- An execution is either:
 - A finite sequence $\sigma_1 \sigma_2 \dots \sigma_n$
 - An infinite sequence $\sigma_1 \sigma_2 \dots$
- where $\sigma_i \rightarrow \sigma_{i+1}$



Acceptance condition

- Let $x \in \Sigma^*$, and $M = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$
- M accepts x iff
 - $q_0 x \rightarrow^* \alpha_1 p \alpha_2$
 - $p \in F$
 - $\alpha_1 \alpha_2 \in \Gamma^*$

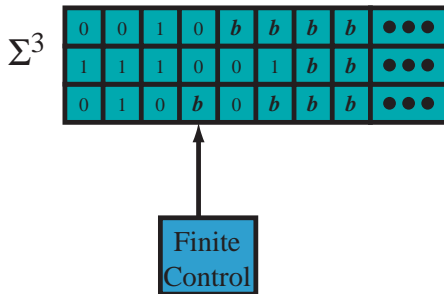


Definitions

- The languages accepted by TM are called *recursively enumerable* (r.e.)
 - Recursion is related to recursion in programs,
 - Enumerable means there is some TM that can enumerate the strings in the language (we'll see why)
- A language is *recursive* if it is accepted by some TM that halts on all inputs
- WLOG, we'll usually assume that TM halt if they enter a final state



Multi-tape machines



Making space

- To make space on the tape
 - Remember a symbol, and replace it with a \$
 - Move all the way to the right
 - Move left, copying symbols one position to the right
 - When reaching \$, write the original symbol to the right
 - Replace \$ with a blank



Unary addition

- To add $0^m 0^n \rightarrow 0^{n+m}$ \$ (in unary):
 - Find \$
 - Replace with 0, and move right to first blank
 - Back up one space, and write \$



Copying

- To copy $0^n \$ \rightarrow 0^n 0^n$
 - Move to first 0
 - Replace with X
 - Move to first blank and replace with 0
 - Move back to X, move right, and repeat
 - On reaching \$, convert all Xs to 0s

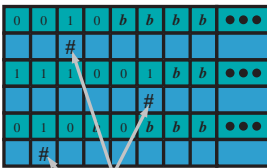


Unary multiplication

- To multiply $0^m 0^n \rightarrow 0^{mn}$
 - Place a \$ at right end of tape
 - Move to first 0, and replace with Y
 - Move past first \$
 - Invoke copy
 - Move back to Y and repeat



Multi-head, multi-tape machines



For each move:
 $\delta(\langle i, ci \rangle, \dots, \langle j, cj \rangle)$
 $= \langle i+D, ci' \rangle, \dots, \langle j+D, cj' \rangle$

Scan right:
 collect $\langle i, ci \rangle, \dots, \langle j, cj \rangle$

Scan left:
 write $\langle i+D, ci' \rangle, \dots, \langle j+D, cj' \rangle$



RAM machines (standard processors)

- A RAM has:
 - A finite number of random-access registers R_1, \dots, R_n
 - A program counter PC
 - A program I_1, \dots, I_n
 - Example program (R_1 factorial, R_2 gets result, R_0 is zero)
 - fact:
 - $CMP \quad R_1, R_0$
 - $JMP \quad EQ, out$
 - $MUL \quad R_2 \leftarrow R_2 * R_1$
 - $DEC \quad R_1$
 - $JMP \quad fact$
 - out:



Some properties

- The complement of a recursive set is recursive
 - Given M that halts on all inputs, construct M' that simulates M .
 - If M halts and accepts, M' halts and does not accept
 - If M halts and does not accept, M' halts and accepts
- The union/intersection of two recursive sets is recursive
 - Simulate machine M_1 , then simulate machine M_2
 - Union: accept if either accepts
 - Intersection: accept if both accept
- If L and $(\Sigma^* - L)$ are r.e., then L is recursive



Quiz



Writing down a Turing Machine

- Consider $M = (Q, \Sigma, \Gamma, \delta, s, b, F)$
- Write it down as a sequence of symbols
 - Kozen: write as $0^n \# 0^m \# 0^k \# 0^s \# 0^t \# 0^r \# 0^u \# 0^v \#$
 - Use integer representations for states, symbols, etc.
 - $n = |Q|$, $m = |\Gamma|$, $k = |\Sigma|$
 - s, t, r are start, accept, reject states
 - u and v are some translation of δ



Halting problem

- Problem: determine entries in the matrix
- Find a recursive machine K , given $M \# x$
 - K halts and accepts if M accepts x
 - K halts and rejects if M does not terminate
- Is there such a machine?
- Suppose so, then build a new machine N , that, given x , runs K on $M_x \# x$ and:
 - Accepts if K rejects
 - Loops forever if K accepts
- If K exists, then N does too; let $N = M_y$
 - $N \# y$ halts and accepts if $N \# y$ does not halt
 - $N \# y$ loops forever if $N \# y$ halts

