

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?”*

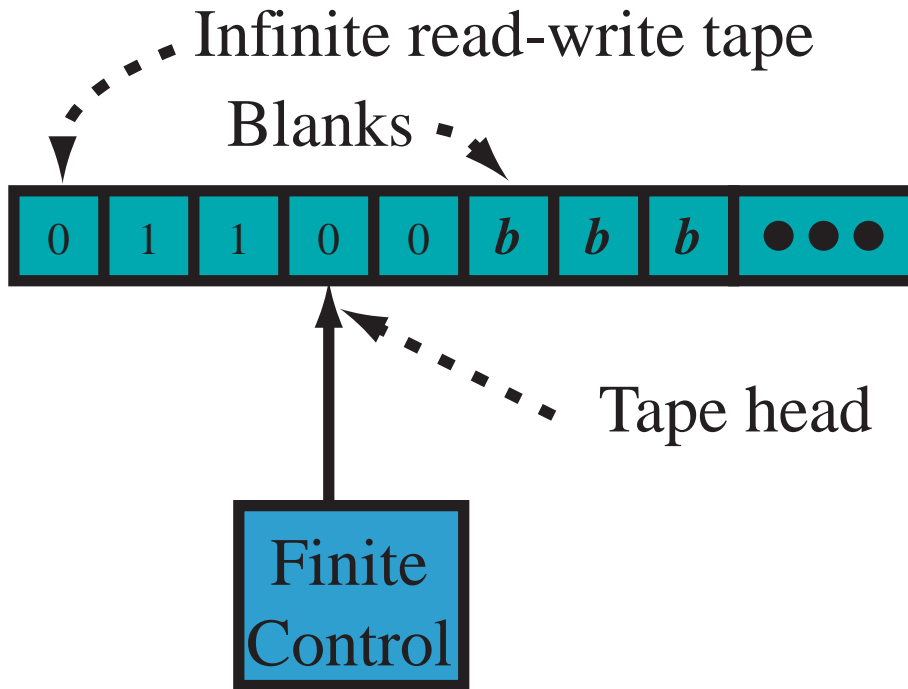


Computability

- **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?***



Turing machines



1. In state q
 - a. read a symbol c
 - b. print a new symbol c'
 - c. move the tape head one position left or right
 - d. goto state $\delta(q, c)$
2. Accept iff the TM ever enters a final state



Turing machine formal definition

A *Turing Machine* is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, s, b, F)$

- Q is a finite set of *states*,
- Γ is a finite set of *tape symbols*,
- $b \in \Gamma$ is the *blank* symbol,
- $\Sigma \subseteq \Gamma - \{b\}$ is the set of *input symbols*,
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a *partial transition function*,
- $s \in Q$ is the *start state*,
- $F \subseteq Q$ is the set of *final states*



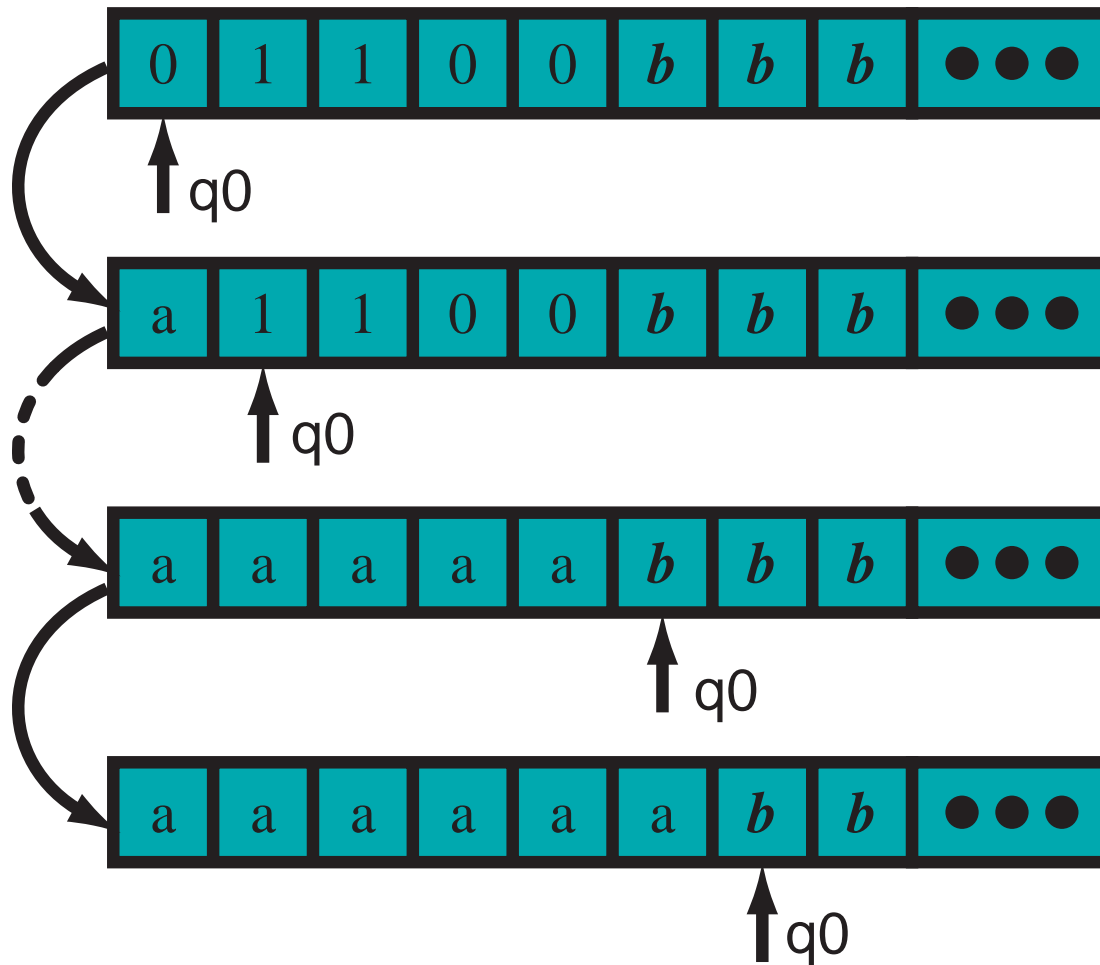
Nontermination

A TM that does not terminate $M = (Q, \Sigma, \Gamma, \delta, s, \bar{b}, F)$

- $Q = \{q_0, q_1\}$
- $\Sigma = \{a\}$
- $\Gamma = \{a, \bar{b}\}$
- $\delta(q_0, a) = (q_0, a, R)$
- $s = q_0$
- $F = \{\}$



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



↑ q_0 in F



Instantaneous descriptions

- An *instantaneous description* ID (σ) 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*

$$a_1 \dots a_{i-1} \underline{q} a_i, \dots, a_n \rightarrow_L a_1, \dots, a_{i-2} \underline{pc} a_i \dots a_n$$

- Define a *right transition*

$$a_1 \dots a_{i-1} \underline{q} a_i, \dots, a_n \rightarrow_R a_1, \dots, a_{i-1} \underline{cp} a_{i+1} \dots a_n$$

- A transition \rightarrow is either \rightarrow_L or \rightarrow_R



Executions

- An execution is either:
 - A finite sequence $\sigma_1 \sigma_2 \cdots \sigma_n$
 - An infinite sequence $\sigma_1 \sigma_2 \cdots$
- 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

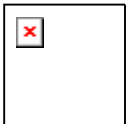


Multi-tape machines

Σ^3

0	0	1	0	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	● ● ●
1	1	1	0	0	1	<i>b</i>	<i>b</i>	● ● ●
0	1	0	<i>b</i>	0	<i>b</i>	<i>b</i>	<i>b</i>	● ● ●

Finite
Control



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 X s to 0s

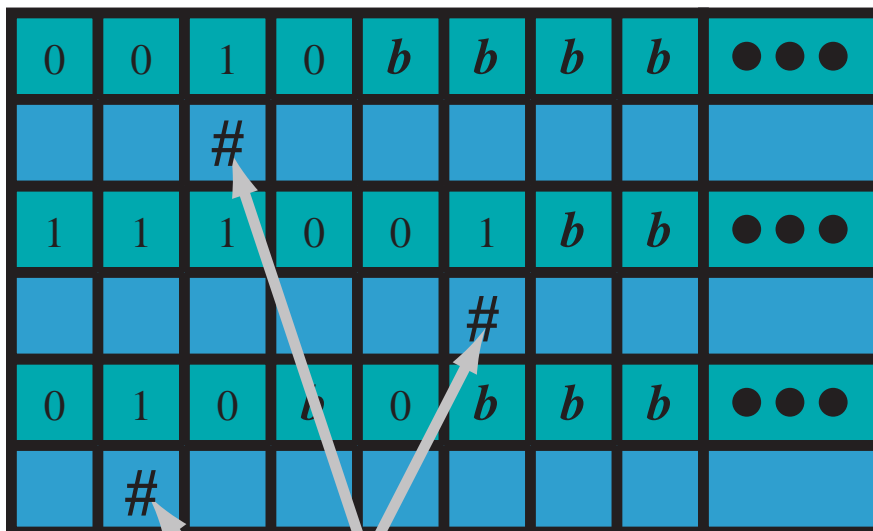


Unary multiplication

- To multiply $0^m \$ 0^n \longrightarrow 0^{mn}$
 - 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



Finite Control

For each move:

$$\begin{aligned} &\text{delta}(\langle i, c_i \rangle, \dots, \langle j, c_j \rangle) \\ &= \langle i+D, c_i' \rangle, \dots, \langle j+D, c_j' \rangle \end{aligned}$$

Scan right:

collect $\langle i, c_i \rangle, \dots, \langle j, c_j \rangle$

Scan left:

write $\langle i+D, c_i' \rangle, \dots, \langle j+D, c_j' \rangle$



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 R_1, R_0
 - JMP EQ, out
 - MUL $R_2 \leftarrow R_2 * R_1$
 - DEC R_1
 - JMP $fact$
 - *out*:



A RAM

#0\$R1 #1\$R2 #01\$R3 #10\$R4...

PC

```
#0 $CMP R1,R0
#01 $JMP EQ,110
#10 $MUL R2<-R2,R1
#11 $DEC R1
#101$JMP 0
```

Each move:

1. Use PC to find instruction
For example: ADD Ri<-Rj,Rk
2. Find Ri,Rj,Rk
3. Perform arithmetic
4. Increment PC
5. Continue



Church's thesis

- “The computable functions are the same as the partial recursive functions”
 - *What is a “computable function?”*
 - *Lots of models: λ -calculus, mechanical devices, fluids and valves, DNA, quantum devices...*
 - *All of these seem to define the same set of functions (but some have better performance)*



Definitions

- A problem is a yes-or-no question
 - *Are two CFGs equivalent?*
 - *Does a TM halt on blank tape?*
- An instance of a problem has specific arguments
 - *Does this TM halt on blank tape?*
- An algorithm is a program that always halts (with the correct answer), so it is recursive
- A problem is decidable if it is recursive
- If not, it is undecidable
- Classically, every instance of a problem is decidable
 - *Not true constructively...*



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 $M1$, then simulate machine $M2$*
 - *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



Universal Turing Machines

Next, build a *universal Turing machine* U that accepts

$$L(U) \equiv \{M\#x \mid x \in L(M)\}$$

- U checks that $M\#x$ is a valid description
- Copy M to one tape, and x to another tape
- Use a scratch tape to represents current state, head position, etc.
- Simulate M : for each move:
 - Figure out δ function from M
 - Move the head,
 - **modify the tape x**



Reals are uncountable

- Cantor's diagonalization argument



0	0.	7	3	5	9	2...
1	0.	3	7	7	1	8...
2	0.	0	0	1	0	0...
3	0.	9	7	9	8	4...
4	0.	8	1	5	2	5...
5	0.	6	6	6	0	0...

0. 8 8 2 0 6...



Another diagonalization argument

Input string
Machine

	ϵ	0	1	10	01	11	...
$M\epsilon$	X		X			X	
$M0$		X		X	X		
$M1$		X		X		X	
$M01$	X	X	X				
$M10$		X		X	X		
...							

Place an x iff M halts on x



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

