

CS20a: Turing Machines (Oct 29, 2002)

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



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



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

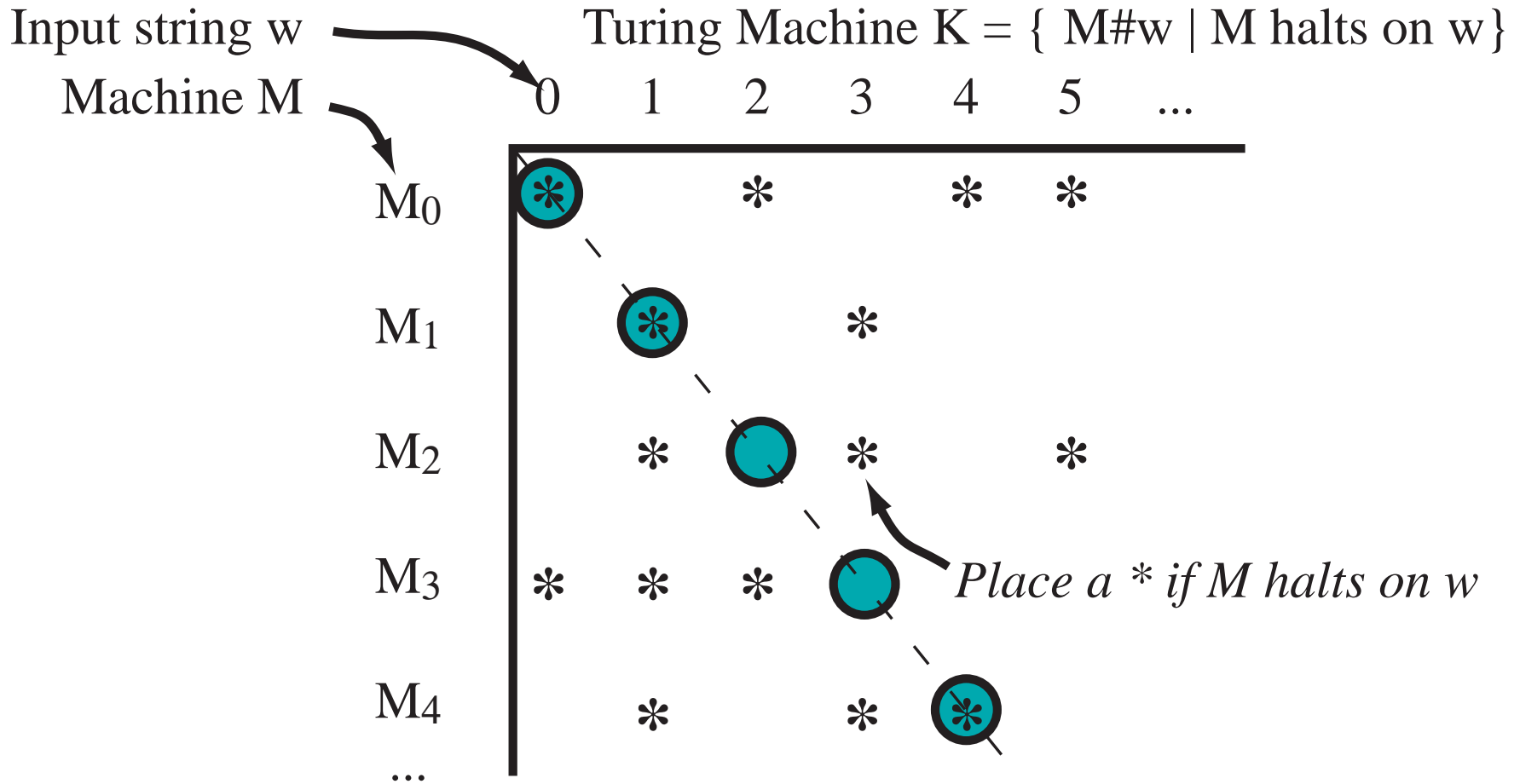


0	0.	7	3	5	9	2...
1	0.	3	7	7	1	8...
2	0.	0	0	1	0	0...
3	0.	9	7	9	8	4...
4	0.	8	1	5	2	5...
5	0.	6	6	6	0	0...

0. 8 8 2 0 6...



Another diagonalization argument



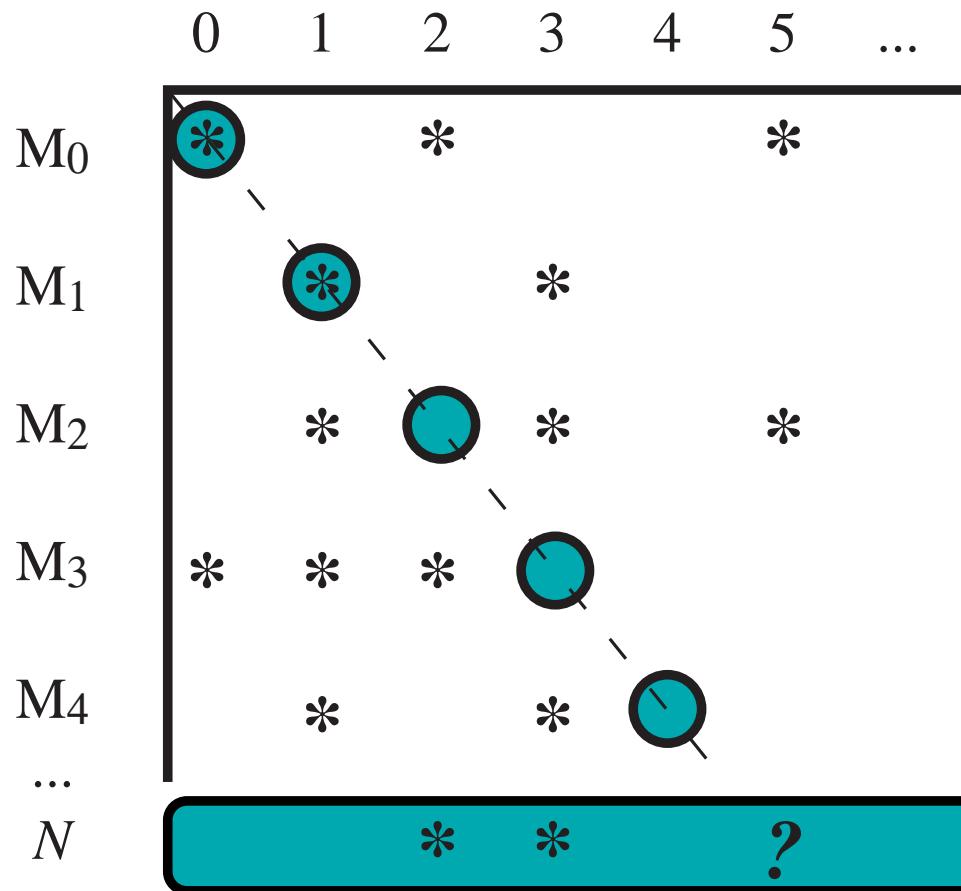
Halting problem

- 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



Halting problem

Turing Machine $K = \{ M\#w \mid M \text{ halts on } w \}$



$N(x) = \text{if } K(M_w\#w) \text{ accepts then loop else halt}$

Computation, Computers, and Programs
<http://www.cs.caltech.edu/courses/cs20/a/>

Course Introduction
October 30, 2002



Some undecidable problems

- Determining if M halts on blank tape
 - *To test if M' halts on string x , build a machine that writes x onto the tape, then simulates M*
- Determining whether $L(M) = \{\}$
 - *To test if M' halts on string x , build a machine that erases the input, writes x onto the tape, then simulates M (accept iff M terminates)*
- Determining whether $L(M) = L(M')$
 - *Let M' be the machine that always halts and rejects*



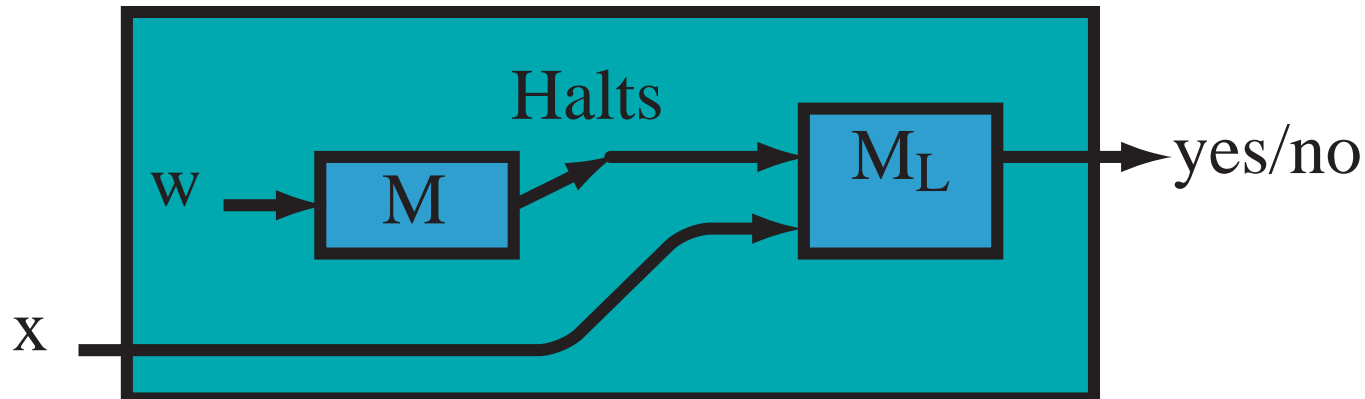
Rice's theorem

- A property P is a set of r.e. languages (so it is a set containing sets of strings)
- For any r.e. language L , we say $P(L)$ is true iff L is a member of P
- Theorem: Any nontrivial property of the r.e. languages is undecidable
 - *Nontrivial means: the property is neither always true nor always false*



Proof of Rice's theorem

- Consider a property P
 - Assume $P(\{\}) = \text{false}$, otherwise invert P
- Since P is nontrivial, there is a language L in P
 - Let M_L be a TM accepting L
- To determine if M halts on input w , build M'
 - Ignore the input, and simulate M on w
 - If M halts, then start M_L on the input string
 - So $P(M')$ iff M halts on w



Properties of TMs (not r.e. languages)

- Example 1: it is decidable if a TM has more than 10 states
 - *Count them*



Properties of TMs (not r.e. languages)

- Example 2: it is undecidable if a TM ever prints 3 consecutive 0s
 - For any M , construct a new machine M' that represents 0 by 01, and 1 by 01
 - Use M' to simulate M (M' never has 3 consecutive 0s on the tape)
 - Except: if M halts, then M' prints 3 zeros



Properties of TMs (not r.e. languages)

- Example 3: it is decidable if a TM ever scans a tape cell 4 or more times
 - *If the TM never scan a cell 4 or more times, then each crossing sequence (where tape head move past the boundary between cells) happens at most 3 times*
 - *But there is a finite number of crossing sequences of length 3 or less*
 - *So the TM either stays within a fixed number of cells, or some crossing sequence repeats*
 - *If some crossing sequence repeats, the tape head move right with a detectable pattern, and the problem is decidable*



TM executions

- Let's revisit the past
 - *This time with readable symbols*



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 $\sigma_i \rightarrow \sigma_{i+1}$



Valid computations (VALCOMPs)

A VALCOMP for $M = (Q, \Sigma, \Gamma, \delta, s, \bar{b}, F)$ is a string

$$w_1 \# w_2^R \# w_3 \# w_4^R \# \dots \#$$

- Each w_i is an ID $\alpha_1 q \alpha_2$ of M
- w_1 is the initial ID sx
- w_n is a *final* ID, $\alpha_1 q_f \alpha_2$ for $q_f \in F$
- $w_i \rightarrow w_{i+1}$



Context-free/VALCOMP

Theorem The set of valid computations of a TM M is the intersection of two CFLs L_1 and L_2 .

Proof

- L_1 enforces $w_i \rightarrow w_{i+1}$ for odd i
- L_2 enforces $w_i \rightarrow w_{i+1}$ for even i
- To accept $w_i \# w_{i+1}^R$
 - Push w_i on the stack
 - Read off w_{i+1}^R comparing with stack
 - When the state/tape head is reached, check the transition function (remember if the state is final)
 - Compare the rest



CFG intersection

Theorem It is undecidable for two CFGs G_1, G_2 whether $L(G_1) \cap L(G_2)$ is empty.

Proof If it were decidable, then we can effectively determine if $\text{VALCOMP}(M)$ is empty.



Invalid computations

Theorem The set of *invalid computations* is a CFL.

Proof If w is not a valid computation then one of:

1. w does not have the form $x_1\#x_2\#\dots\#$
2. x_1 is not initial
3. x_n is not final
4. $x_i \rightarrow x_{i+1}$ is false for some i



Invalid computations

- (1), (2), (3) are regular
- (4) is context-free
- So the invalid computations are a union of regular language and a context-free language.
- So the language is context-free.



Theorem It is undecidable whether $L(G) = \Sigma^*$

Proof

- Given a TM M , construct a CFG G such that $L(G) = \Sigma^*$ iff $L(M) = \{\}$
- Construct G to generate the inVALCOMPs of M
- If $L(G) = \Sigma^*$ where decidable, then so is $L(M) = \{\}$



Turing Machines

- TM are basically 2-way FA, with a writable tape
 - *The TM accept the r.e. languages*
 - *TM that always halt accept the recursive languages*
- Every known computational model can be mapped to TM (~Church's thesis)
- Nearly everything about TM is undecidable
 - *Wonderful!*

