CS 3313
Foundations of Computing:

Universal Turing Machine, Properties of RE Languages

http://gw-cs3313.github.io

1

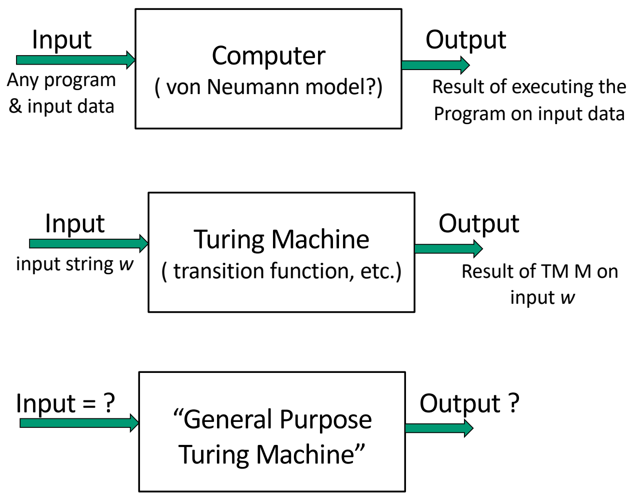
Recap…

- Turing Machine model
  - TM as an automaton/ acceptor of languages
  - TMs to compute functions
  - TM "programming" techniques
    - Storage in the state, Checking symbols, Subroutines…
- Modifications to the basic TM model
  - Multi-tape, Non-deterministic TMs
- Definition thus far: A turing machine computes one function
  - a special purpose computer ?
- Next…Turing machines that operate as General purpose "re-programmable" turing machines…**the Universal Turing Machine**

Important: You should be reading the textbook (or other textbooks)

2

General Purpose Computers & Turing Machines



3

"General purpose Turing machine"

- A Turing machine that can "execute" (simulate) any Turing machine sent as input
- but input to a TM is a string…..therefore

  need to *describe/encode a TM as a string*

  a TM is a "function" therefore we are encoding a program !
- General purpose TM known as Universal Turing Machine (UTM)

4

Universal Turing Machine (UTM)

- Input to a universal Turing machine is a description of a Turing machine *M* and the input *w*
  - UTM will simulate the behavior of *M* on input *w*
  - UTM accepts if and only if *M* accepts *w*
- Input to TMs (and other automata models) are strings over some alphabet…..*Therefore input to UTM is a string that describes another turing machine M!*
  - *First step: Design an unique encoding scheme to encode each TM M*
- Once we have a way to encode, a collection of Turing machines = set of strings…..i.e,. A language !
  - Machines and languages the same ????

5

Binary-Strings from TM's

- We restrict ourselves to TM's with input alphabet *{0, 1}*.
- Important result/Theorem: Given a Turing machine with tape alphabet Γ (consisting of *k* symboks) there is an equivalent turing machine with tape alphabet *{0,1,B}*
  - One simple proof: If we have *k* tape symbols, then use a binary encoding (using $m = log\ k$ bits) and use a *m*-track tape !!
- Every Turing machine has a finite number of states n
  - Without loss of generality, let these states be numbered $q_1, q_2, \ldots q_n$
  - wlog, let start state = q_1 and one final state q_2
    - If we have multiple final states then add transition to new final state q_2
- At each move the TM moves its tape head in direction D_1=L or D_2= R

6

6

Binary-Strings from TM's

- Assign positive integers to the three classes of elements involved in moves:
  1. States: q_1(start state), q_2 (final state), q_3, …q_i,..q_n
  2. Tape Symbols X_1 (0), X_2 (1), X_3 (blank), X_4, …
  3. Directions D_1 (L) and D_2 (R).
- Suppose $\delta(q_i, X_j) = (q_k, X_l, D_m)$
- Represent this transition rule by string $0^i10^j10^k10^l10^m$.
- Key point: since integers *i, j,* … are all > 0, there cannot be two consecutive 1's in these strings.

Ex.: $\delta(q_2, X_1) = (q_4, X_3, D_2)$ *encoded as:*

7

7

The Turing Machine M as encoded as a Binary String <M>

- A Turing Machine is defined as $M=(Q, \mathbf{\Sigma}, \Gamma, \delta, q_1, F)$
  - $F = \{ q_2 \}$ Tape movement $\{D_1=L, D_2=R\}$
  - $\delta(q_i, X_j) = (q_k, X_l, D_m)$ represented by string $0^i10^j10^k10^l10^m$.
- Set of all transitions is now a set of codes
- To represent the entire TM by one string, we need to look at *separators*: what strings can we use to separate the codes of each δ

Important: The textbook uses a different but equivalent encoding which switches the role of the 0's and 1's.

- $\delta(q_i, X_j) = (q_k, X_l, D_m)$ represented by string $1^i01^j01^k01^l01^m$

8

8

The Turing Machine M as encoded as a Binary String <M>

- A Turing Machine is defined as $M=(Q, \mathbf{\Sigma}, \Gamma, \delta, q_1, F)$
  - $F = \{ q_2 \}$ Tape movement $\{D_1=L, D_2=R\}$
  - $\delta(q_i, X_j) = (q_k, X_l, D_m)$ represented by string $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.
- Represent the TM by concatenating the codes for each of its moves, separated by **11**.
  - That is: $Code_1 11 Code_2 11 Code_3 11$ …
- Start and end of the TM specified by **111**
  - Ex: <M> = **111**0101000100010**110**^{i}10^{j}10^{k}10^{l}10^m**11**…….**111**

9

9

Important Property: Enumeration

- Recall our discussions (from Lab review) that we can enumerate binary strings
  - Ex: we can convert binary strings to integers by prepending a 1 and treating the resulting string as a base-2 integer.
  - Recall we can convert binary strings to integers by prepending a 1 and treating the resulting string as a base-2 integer.
- Therefore, given an encoding <*M*> of a Turing machine we can talk of the *i-th* Turing Machine
- Note: if *i* makes no sense as a TM, assume the *i-th* TM accepts nothing.

10

Exercise: Encoding of Turing Machines

- M= ({q_1, q_2, q_3}, {0,1}. {0,1,B},δ , q_1, {q_2})
- $\delta(q_1,1) = (q_3, 0, R)$
- $\delta(q_3,0) = (q_1, 1, R)$
- $\delta(q_3,1) = (q_2, 0, R)$
- $\delta(q_3, B) = (q_3, 1, L)$

- Question: Show the encoding of this Turing machine.

11

UTM: A Universal Turing Machine

- Input to the UTM is description of turing machine M and the input *w* to M.
- UTM simulates the behavior of M on input w

- Input: *<M,w>*
  - Encoding: Encode M and after the second set of 111, we place the input string *w*.

- What's happening here: Input to the UTM is a Program P (represented in binary) and the input to the program,

  We want the UTM to execute the program M

12

12

Designing the UTM

- Inputs are of the form:

  Code for 111 M 111 w

- Note: A valid TM code never has 111, so we can split M from w.
- The UTM must accept its input if and only if M is a valid TM code *and* TM M accepts w.

13

13

The UTM – (2)

- The UTM will have several tapes.
- Tape 1 holds the input *M,w*
- Tape 2 holds the tape of M
  - Simulates the tape of M on input *w*
- Tape 3 holds the state of M
  - Recall that each state is encoded in unary
  - If M is in state k, then Tape 3 contains 0^k

14

14

The UTM – (3)

- Step 1: The UTM checks that M is a valid code for a TM.
  - E.g., all moves have five components, no two moves have the same state/symbol as first two components.
- If M is not valid, its language is empty, so the UTM immediately halts without accepting.

15

The UTM – (4)

- Step 2: The UTM examines M to see how many of its own tape squares it needs to represent one symbol of M.
  - Note: this can be skipped if we use the property that the input TM M has a tape alphabet {0,1,B}
- Step 3: Initialize Tape 2 to represent the tape of M with input w, and initialize Tape 3 to hold the start state.

16

The UTM – (5)

- Step 4: Simulate M.
  - Look for a move on Tape 1 that matches the state on Tape 3 and the tape symbol under the head on Tape 2.
    - **How: key-value lookup !!!**
  - If found, change the symbol and move the head marker on Tape 2 and change the State on Tape 3.
  - If M accepts, the UTM also accepts.

17

17

The Universal Turing Machine

- The UTM is also a Turing machine…..
- So what happens if we input code of U to the UTM ?
  - *<U, w>*
  - Cool fact: this is an encoding of itself !!!

18

UTMs, Computing, and General Purpose Computers

- An algorithm is equivalent to a Turing machine
- A UTM is a reprogrammable Turing machine….equivalent to a General Purpose Computer that can execute any algorithm/program

- So how about the von Neumann model of computer architecture:
  - Control Unit
  - Instruction Set
  - Memory
  - A program is a set of Instructions in memory
  - Execution = instruction execution cycle

19

Random Access Machines (RAM) and TM

- RAM is the theoretical model of von Neumann machines
- Registers, Memory, Instructions
- Instructions are encoded in binary
  - Piece of cake after Architecture ☺
- Memory organized as a set of locations
  - Location has address, and each location has content
- Read/Write to Memory……??

- Program counter contains address of instruction to be executed…
  - Read content from this address
- Instruction processing cycle:
  - Decode the instruction
  - Execute instruction
  - Store result back to registers or memory

20

Random Access Machines (RAM) and TM

- Memory organized as a set of locations
  - Location has address, and each location has content
- How do we fetch contents from memory address x_i ?

  Name-Value (key-value) lookup !!!!
- How do we write content y_i to memory address x_i

  Insert into Name-Value stores !!!

Simulation of RAM on TM

- Write subroutine for each instruction
  - For LC3: Subroutine for Add, AND, NOT, Load (Read), Store(Write)
    - Conditional: easy ….check symbol and branch to a state !
- Use a multi-tape TM to simulate a RAM.
- Tape 1 holds the memory
- v_i is contents in binary of the *i-th* word in memory
  - At all times, there will be a finite number of words used by the machine
  -

$\# 0 * v_1 \# 1 * v_2 \# 2 * v_2 \# \ldots\ldots \# i * v_i \# \ldots$

Tape 1….memory!

Simulation of RAM on TM (2)

- A computer has a finite number of registers….
- We use one tape to hold each register's contents
- Instruction to be fetched is stored in a program counter (location counter)….use on tape to hold the program counter
  - This contains the number of the word from which the next instruction is taken
- We may need to read/write data to memory…address is in Memory Address Register (MAR)…

  Use one tape to hold the memory address register
- Fetch Instruction, Decode, Execute (go to subroutine), Store result!

23

Taking stock of where we are now….

- Turing machine model and modifications
  - Non-deterministic TM equal in computer power to Deterministic
    - But equivalence of time efficiency (P=NP) unknown
- Concept and design of a Universal Turing machine – a general purpose reprogrammable computer !
- Turing-Church Thesis: Any problem that can be solved on a computer can be solved on a Turing machine
  - This is not exactly how the thesis is stated !
- Question: Are there problems that are not solvable by a Turing machine
  - Are there problems for which no algorithms exist ?

24

Problems

- Informally, a (decision) "problem" is a yes/no question about an infinite set of possible *instances*.
- Example 1: "Does graph G have a *Hamilton cycle* (cycle that touches each node exactly once)?
  - Each undirected graph is an instance of the "Hamilton-cycle problem."
- Example 2: "Is graph G *k*-colorable ?
  - Each undirected graph, and value *k*, is an instance of the "graph coloring problem."

25

25

Problems – (2)

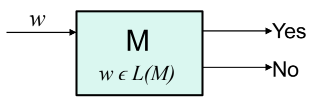
- Formally, a problem is a language.
- Each string encodes some instance.
- The string is in the language if and only if the answer to this instance of the problem is "yes."

26

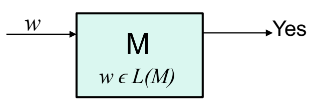
26

Recall Definitions

- Recursive Language: A language L is recursive if there is a Turing machine that accepts the language and halts on all inputs



- Recursively Enumerable Language: if there is a Turing machine that accepts the language by halting when the input string is in the language
  - The machine may or may not halt if the string is not in the language



27

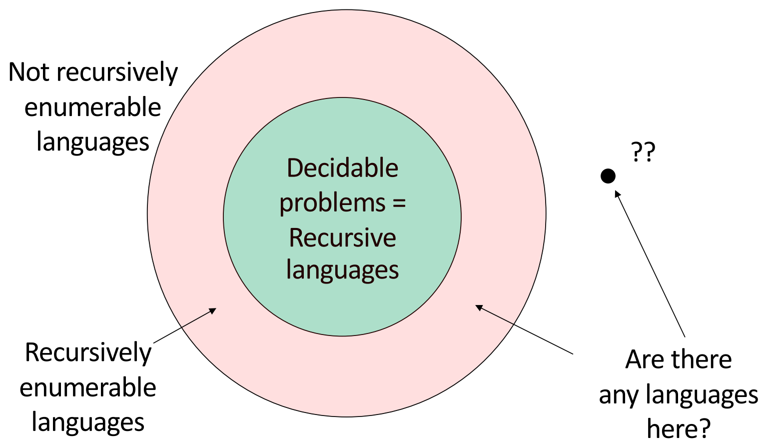
Decidable Problems

- A problem is *decidable* if there is an algorithm to answer it.
  - Recall: An "algorithm," formally, is a TM that halts on all inputs, accepted or not.
  - Put another way, *"decidable problem" = "recursive language."*
- Otherwise, the problem is *undecidable*.

28

28

Bullseye Picture



29

Closure Properties of Recursive and RE Languages

- Next topic is Decidability
  - Review Lab notes on Math review – diagonalization etc.
- First let’s look at closure properties of these classes of languages
- Both closed under union, concatenation, star, reversal, intersection, inverse homomorphism.
- Recursive closed under difference, complementation.
- RE closed under homomorphism.
- *Observe: To prove the closure properties we have to construct a Turing machine, i.e., an algorithm (!!!), to accept the language*
  - *Construction will be shown using a flowchart*
    - *Getting more and more like programming!*

30