# Turing Machines

And the languages they handle

## Homework Swap

- Homework #5
  - Returned today
- Homework #6
  - Due today
- Homework #7 (Due 11/5)
  - 6.39d
  - 8.1a,c
  - 9.1
  - 9.2a,b,c
  - 9.6a,b

## Reminder

- Final exam
  - The date for the Final has been decided:
  - Saturday, November 16th
  - 12:30pm – 2:30pm
  - 07 – 1420

## Announcement

- October 31st is halloween
  - Dress up and win big prizes!
  - 4pm…Refreshments at CS Offices

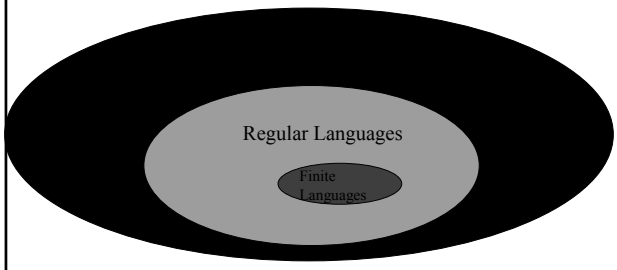
## Before We Start

- Any questions?

## Languages

- Is that your final answer?
  - What is a language?
  - What is a class of languages?

## Now our picture looks like

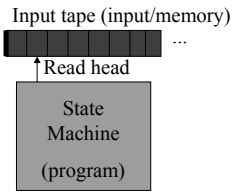



We're going to start to look at languages out here

## Turing Machine

- A Machine consists of:
  - A state machine
  - An input tape
  - A movable r/w tape head
- A move of a Turning Machine
  - Read the character on the tape at the current position of the tape head
  - Change the character on the tape at the current position of the tape head
  - Move the tape head
  - Change the state of the machine based on current state and character read

## Turing Machine




- Tape that holds character string
- Movable tape head that reads and writes character
- Machine that changes state based on what is read in

## Turing Machine

- In the original Turing Machine
  - The read tape was infinite in both directions
  - The text describes a semi-infinite TM where
    - The tape is bounded on the left and infinite on the right
    - It can be shown that the semi-infinite TM is equivalent to the basic TM.
  - As not to confuse, I'll follow the conventions in the book…(as much as that bothers me!)

## Turing Machines

- About the input tape
  - Bounded to the left, infinite to the right
  - All cells on the tape are originally filled with a special "blank" character Δ
  - Tape head is read/write
  - Tape head can not move to the left of the start of the tape
    - If it tries, the machine "crashes"

## Turing Machines

- About the machine states
  - A Turing Machine does not have a set of accepting states
  - Instead, each TM has a special halting state, h.
    - Once in the halting state, the machine halts.
  - Unlike PDAs, the basic TM is deterministic!

## Turing Machines

- Let's formalize this
  - A Turing Machine M is a 5-tuple:
  - M = $(Q, \Sigma, \Gamma, q_0, \delta)$ where
    - Q = a finite set of states (assumed not to contain the halting state h)
    - $\Sigma$ = input alphabet (strings to be used as input)
    - $\Gamma$ = tape alphabet (chars that can be written onto the tape. Includes symbols from $\Sigma$)
    - Both $\Sigma$ and $\Gamma$ are assumed not to contain the "blank" symbol
    - $\delta$ = transition function

## Turing Machines

- Transition function:
  - $\delta$: $Q \times (\Gamma \cup \{ \Delta \}) \rightarrow (Q \cup \{h\}) \times (\Gamma \cup \{ \Delta \}) \times \{R, L, S\}$
  - Input:
    - Current state
    - Tape symbol read at current position of tape head
  - Output:
    - State in which to move the machine (can be the halting state)
    - Tape symbol to write at the current position of the tape head (can be the "blank" symbol)
    - Direction in which to move the tape head (R = right, L = left, S= stationary)

## Turing Machine

- Accepting a string
  - A string x is accepted by a TM, if
    - Starting in the initial configuration
    - With x on the input tape
    - The machine eventually ends up in the halting state.
  - I.e.
    - $(q_0, \underline{\Delta}x) \mapsto^* (h, x\underline{a}y)$
      - for $x,y \in (\Gamma \cup \{\Delta\})^*$, $a \in (\Gamma \cup \{\Delta\})$

## Turing Machine

- Running a Turing Machine
  - The execution of a TM can result in 4 possible cases:
    - The machine "halts" (ACCEPT)
    - The machine has nowhere to go (REJECT)
    - The machine "crashes" (REJECT)
    - The machine goes into an "infinite loop" (REJECT but keeps us guessing!)

## Turing Machine

- Language accepted by a TM
  - The language accepted by a TM is the set of all input strings x on which the machine halts.
- Questions?

## Plan for today

- TM Variants
- Computing a function using TMs
- Languages, Languages, Languages

## Turing Machine Variants

- Some variants of the TM
  - Multitaped TMs
    - Have multiple tapes
    - With a single tape head that is read/writing the same position on each tape at any one given configuration

## Turing Machine Variants

- Some variants of the TM
  - Non-deterministic TMs
    - More than 1 move is possible from any given configuration.

## Turing Machine Variants

- Some variants of the TM
  - Semi-infinite TM
    - What our book calls the "basic" TM.
    - The real "basic" TM has an infinite tape in both directions.

## Turing Machine Variants

- Thankfully, all these variants can be shown to be equivalent to the "basic" TM.
  - Specifically, non-determinism does not add extra computing power to a TM
    - Much like FAs
    - But unlike PDAs

## Computation with Turing Machines

- Computing functions with TMs
  - The result of the function applied to an input string x, will be left on the tape when the machine halts.
  - Functions with multiple arguments can be placed on the input tape with arguments separated by blanks.

## Computation with Turing Machines

- Computing functions with TMs
  - Formally,
    - Let $T = (Q, \Sigma, \Gamma, q_0, \delta)$ be a TM and let f be a partial function on $\Sigma^*$. We say T computes f if for every $x \in \Sigma^*$ where f is defined:
      - $(q_0, \underline{\Delta}x) \mapsto^* (h, \underline{\Delta}f(x))$
    - And for every other x, T fails to halt on input x.

## Computation with Turing Machines

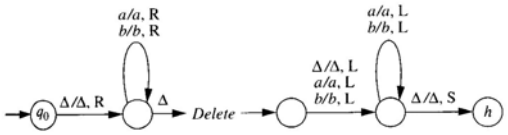
- Computing functions with TMs
  - Formally (with multiple arguments)
    - Let $T = (Q, \Sigma, \Gamma, q_0, \delta)$ be a TM and let f be a partial function on $(\Sigma^*)^k$. We say T computes f if for every $(x_1, x_2. \ldots, x_k)$ where f is defined:
      - $(q_0, \underline{\Delta}x_1 \Delta x_2 \ldots \Delta x_k) \mapsto^* (h, \underline{\Delta}f(x_1, x_2. \ldots, x_k))$
    - And for every other k-tuple, T fails to halt on input x.

## Computation with Turing Machines

- Example:
  - A TM that computes the concatenation of 2 strings x and y
    - $(q_0, \underline{\Delta}x\ \Delta y) \mapsto^* (h, \underline{\Delta}x\ y)$
    - Basic idea:
      - Use the Delete "subroutine" to delete the blank between the characters
      - Move the tape head back to the start of the tape

## Computation with Turing Machines

- Concatenation TM




## Computation with Turing Machines

- Characteristic function of a set:
  - For any language L, the characteristic function is defined as:
    - 1 if $x \in L$
    - 0 otherwise
  - If a TM computes the characteristic function of a language, it will <u>always</u> halt with either a 1 or 0 left on the tape
    - This is stronger than saying that x is accepted by the TM since if $x \notin L$, for a TM that accepts L, there is no guarantee that the the machine will halt.

## TMs and languages

- Two classes of languages that involve TMs:
  - Languages, L <u>accepted</u> by TM
    - Halts on $x \in L$, "rejects" on $x \notin L$
  - Languages, L <u>recognized</u> by a TM
    - TM computes the characteristic function of L
    - Halts on all x
    - Give yes/no answer as to whether $x \in L$

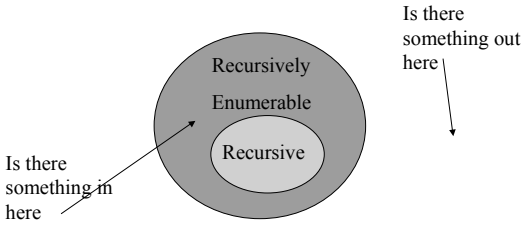
## TMs and languages

- Two classes of languages that involve TMs:
  - A language is <u>recursively enumerable</u> if there is a TM that <u>accepts</u> it
  - A language is <u>recursive</u> if there is a TM that <u>recognizes</u> it.

## TMs and languages

- First observation:
  - Every recursive language is also recursively enumerable
    - Modify the TM that recognizes the language so that it goes into a "nowhere state" just before placing the 0 on the tape indicating that a string is not in the language.

## TMs and languages




## TMs and languages

- Game plan
  1. Show that there exists a language that is not recursively enumerable
  2. Show that there exists a language that is recursively enumerable but not recursive.

A language that is not recursively enumerable

- Sorry, no pumping Lemma for TMs!
- Instead we introduce the notion of a Universal Turning Machine

## Universal Turing Machine

- Universal Turing Machine
  - A TM $T_u$ that takes as input, an encoded version of another TM M' and an input to M'.
  - M will simulate the processing of M'
    - If M' halts on input x, $T_u$ will halt
    - If M' rejects on input x,$T_u$ will reject.
  - $T_u$ is a general purpose TM
    - M' is a program run on the general purpose TM

## Universal Turing Machine

- Encoding scheme
  - We need a way to encode a TM so that it can be provided as input to $T_u$
  - Define sets:
    - $\boldsymbol{Q} = \{q_1, q_2, \ldots\}$ = set of all possible states that may appear in any TM
    - $\boldsymbol{S} = \{a_1, a_2, a_3, \ldots\}$ = set of all possible tapes symbols that can be written on any TM.
  - So for any TM
    - $Q \subseteq \boldsymbol{Q}$ and $\Gamma \subseteq \boldsymbol{S}$

## Universal Turing Machine

- Encoding scheme
  - Associate states, symbols, and directions with binary numbers:
    - States:
      - $s(h) \rightarrow 0$
      - $s(q_i) \rightarrow 0^{i+1}$
    - Symbols
      - $s(\Delta) \rightarrow 0$
      - $s(a_i) \rightarrow 0^{i+1}$

## Universal Turing Machine

- Encoding scheme
  - Associate states, symbols, and directions with binary numbers:
    - Directions:
      - $s(S) \rightarrow 0$
      - $s(L) \rightarrow 00$
      - $s(R) \rightarrow 000$

## Universal Turing Machine

- Encoding scheme
  - Encoding transitions:
    - Encode each state, symbol, direction
    - Separate by 1's
  - The transition $\delta(p,a) = (q, b, D)$ can be encoded as:
    - s(p)1s(a)1s(q)1s(b)1s(D)

## Universal Turing Machine

- Encoding
  - Encoding for a TM
    - Encode the start state
    - Encode each of it's moves
    - Separate by 1's
  - $e(T) = s(q)1e(m_1)1e(m_2)1\ldots1e(m_k)1$

## Universal Turing Machine

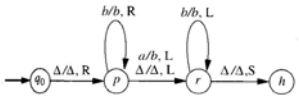
- The input to $T_u$ will be
  - an encoded TM and
  - a string x to run on that TM
  - We encode x by individually encoding the characters of x (separated by 1's)
    - $x = z_1z_2\ldots z_n$
    - $e(x) = s(z_1)1s(z_2)1\ldots1s(z_n)1$

## Universal Turing Machine

- Encoding Scheme
  - Finally, the description of the TM and the encoding of the string to run on the TM will be separated by 2 1's
    - Input to $T_u$
      - e(M)11e(x)

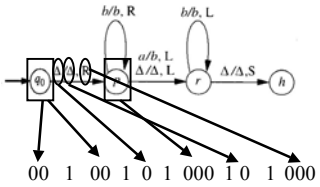
# Universal Turing Machine

- Encoding example
  - States
    - $q_0$ – state 1
    - p – state 2
    - s – state 3
  - Symbols
    - a – symbol 1
    - b – symbol 2




# Universal Turing Machine

- Encoding example
  - States
    - s (h) = 0
    - $s(q_0)$ = 00
    - s(p) = 000
    - s(r) = 0000
  - Symbols
    - s(Δ) = 0
    - s(a) = 00
    - s(b) = 000
  - Directions
    - s(S) → 0
    - s(L) → 00
    - s(R) → 000




# Universal Turing Machine

- Encoding scheme
  - The actual scheme isn't really important (in fact, there are many)
  - What is important is that we can encode a TM and it's input as a binary string.

# Universal Turing Machine

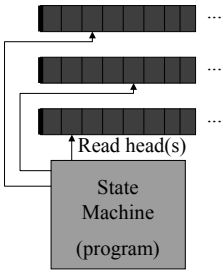
- For the mathematically inclined…
  - The set of all TMs is countably infinite
    - We can define a one-to-one and onto function between the set of TMs and the set of natural numbers { 0, 1, 2, …}

# Universal Turing Machine

- How it works
  - 3 tape TM
    - Tape 1 – Input and output tape
    - Tape 2 – Working tape
    - Tape 3 – encoded form of the state that machine being simulated is in
  - Note that a 3 tape TM is equivalent to a "basic" TM.

# Universal Turing Machine

- How it works




- On a single move,
  - The symbol on each tape is read
  - The symbol on each tape is written
  - The tape head for each tape is moved independently

## Universal Turing Machine

- How it works
  - When we start $\Delta e(T)11e(x)$ is on tape 1
  - Step 1: Move e(x) from tape 1 to tape 2 and delete from tape 1
    - Tape 2: $\Delta 01e(x)$
  - Step 2: Copy encoded initial state of T to tape 3 and delete from tape 1
    - Tape 3: $s(q_0)$

## Universal Turing Machine

- How it works
  - Step 3: Now simulation begins
    - "read" and replace character on tape 2
    - "read" state on tape 3
    - "find" a transition in the encoded machine on tape 1.
    - Replace destination state onto tape 3
    - Replace character on tape 2.
    - Move head on tape 2 appropriately

## Universal Turing Machine

- How will the machine finish?
  - The TM being simulated has nowhere to go.
    - $T_u$ will never find a suitable transition
  - The TM being simulated "crashes"
    - Can arrange for $T_u$ to crash
  - The TM being simulated goes into an infinite loop
    - $T_u$ will continually simulate these moves
  - The TM being simulated halts
    - $T_u$ will halt with 0 on it's 3rd tape
    - $T_u$ will copy the contents of Tape 2 to Tape 1

## Universal Turing Machine

- Reality check: What have we done?
  - Defined:
    - A means to encode a TM as a binary string
    - A Universal TM ($T_u$) that takes as input:
      - An encoded TM, M
      - An encoded input string, x
      - That will simulate the running of x on M
  - Time for a break

## Back to our problem

- Show that there exists a language that is not recursively enumerable
  - The encoding provides each TM with a unique "serial" number.
    - That "serial number" can be considered the decimal equivalent of the binary encoding of the TM
    - $M_i$ = the TM whose code is $w_i$ , i represented in binary
    - $L(M_i)$ is the language accepted by $M_i$
    - Note: if $w_i$ is not a valid TM encoding, we define $M_i$ to be the TM with 1 state and no transitions. In this case $L(M_i) = \varnothing$.

## Back to our problem

- So far…
  - For i=0, 1, 2, …
    - $w_i$ is the binary representation of i
    - $M_i$ is the TM that has $w_i$ as it's encoding
    - $L(M_i)$ is the language accepted by $M_i$

## Diagonalization Language

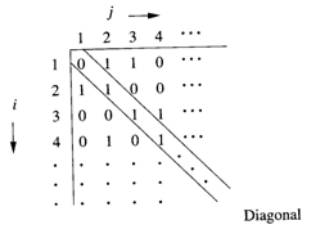
- Diagonalization Language ($L_d$)
  - Set of all strings $w_i$ such that $w_i \notin L(M_i)$
  - All strings w that are not accepted by the TM with w as it's encoding.
  - All encodings for TMs that don't accept their own encoding when input

## Diagonalization Language

- Why is it called diagonalization?
  - Consider all pairs (i,j).
    - i represents TM "serial numbers"
    - j represents decimal representations of binary string
    - (i, j) represents if j is accepted by the TM with "serial number" i.

## Diagonalization Language

- $L_d$ is the complement of the diagonal




## The punch line

- $L_d$ is not recursively enumerable
  - Proof by contradiction
    - Assume $L_d$ was recursively enumerable then there exists a TM, M that accepts $L_d$. Let M have "serial number" i, for some i.
      - $L_d = L(M_i)$
    - Now ask: is $w_i$ in $L_d$?
      - If yes, then $M_i$ will accept $w_i$. This contradicts the definition of $L_d$.
      - If no, then $M_i$ will not accept $w_i$. In this case, by the definition of $L_d$, $w \in L_d$. Can't both be in $L_d$ and not in $L_d$.
      - Contradiction!
    - $L_d$ must not be recursively enumerable.

A language that is not recursively enumerable

- Questions?

## For the mathematically inclined…

- A language that is not recursively enumerable
  - It can be shown that the power set $2^S$ of a set (I.e. the set of all subsets of S) is infinitely uncountable.
  - More specifically, the power set of the set of strings composed of symbols from an alphabet $\Sigma$ (what is a language again?) is infinitely uncountable.
  - This power set is merely the set of all languages over $\Sigma$.

## For the mathematically inclined…

- A language that is not recursively enumerable
  - Countably infinite sets are smaller than uncountably infinite sets.
  - The set of TMs is countable
  - The set of languages is uncountable.
  - There are just not enough TMs to go around!
  - There are more languages than TMs, thus there must be some languages that have no TMs that accept them.

## RE but not recursive

- 2. Show that there exists a language that is recursively enumerable but not recursive.
- Helps to show a nice fact about the complements of recursive languages.

## Complements of Recursive Languages

- If L is recursive, then L', its complement is also recursive
  - If L is recursive, there exists a TM, M, that, on a given input x
    - Always halts
    - Will leave 1 on the tape if $x \in L$
    - Will leave 0 on the tape if $x \notin L$.
  - Build a new TM, M' that simulates M except
    - Will leave 0 on the tape if $x \in L$
    - Will leave 1 on the tape if $x \notin L$.
  - M' accepts L'

## Universal Language

- Universal Language ($L_u$)
  - Set of all strings $w_i$ such that $w_i \in L(M_i)$
  - All strings w that <u>are</u> accepted by the TM with w as it's encoding.
  - All encodings for TMs that <u>do</u> accept their encoding when input
- Is $L_u$ recursively enumerable?

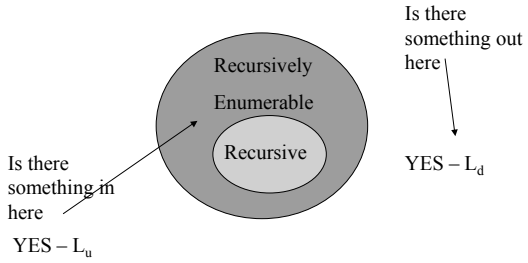
## Universal Language

- Is $L_u$ recursively enumerable?
  - Is there a TM that will always halt for $w_i \in L(M_i)$
  - Yes! The Universal TM, $T_u$
  - Running $T_u$ on $w_i 11 w_i$
    - Will halt if $w_i$ is accepted by the TM with encoding $w_i$
    - Will reject otherwise.
  - $L_u$ is recursively enumerable.

## Universal Language

- Is $L_u$ recursive?
- Assume that it is:
  - Then $L_u$' is also recursive
  - $L_u$ = Set of all strings $w_i$ such that $w_i \in L(M_i)$
  - $L_u$' = Set of all strings $w_i$ such that $w_i \notin L(M_i)$
  - $= L_d$
  - We just showed $L_d$ not to be recursively enumerable.
  - Since all recursive languages are also recursively enumerable, then $L_d$ is certainly not recursive.
  - Contradiction!
  - $L_u$ must not be recursive!

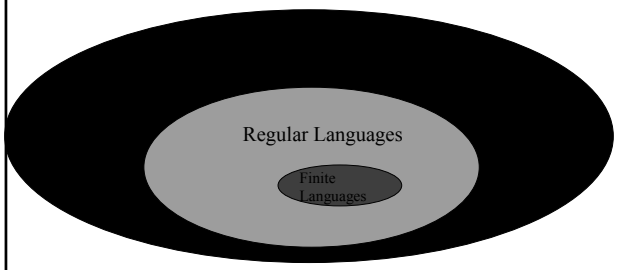
## TMs and languages




## Summary

- Recursive vs. Recursively Enumerable
- Universal TM
- Language that is not RE
  - Diagonalization Language
- Language that is RE but not recursive
  - Universal language.

## But what about this?




How does this relate to Recursive/RE Languages?

## But what about this?

- Save for next time.