CS 3100, Models of Computation, Spring 20, Lec 20
March 30, 2020

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

URL: **https://bit.ly/3100s20Syllabus**

SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

Agenda for Wed March 30

- Help students answer the assignment questions
- Launch into the topics of RE and Recursive sets
- Present the Halting problem (start it today; finish April 1^{st})

- Asg drop policy:
    - Drop a single 200-pt asg OR two 100-pt asgs (lowest 200 pts)
    - Grade the remaining out of 700

- Run this to gain some footing wrt DTM and NDTM

  First_Jove_Tutorial/Start_with_These_Animations.ipynb

  Study the basics of DTM and NDTM behavior from there

  Also helps debug file include issues

- Thereafter, run this!

  First_Jove_Tutorial/CH13/CH13.ipynb

    See some serious TMs from here, and get ideas for Asg-6 from here

    Includes w#w DTM runs

    Includes ww NDTM runs

    This last one gives you good clues for Asg-6

    You can also see how to write binary addition – a large TM you can run!

Your approach to “nail” Asg-6

- Build + test your DTM (study examples as much as you want)
- NDTM and PCP then
- Then the theoretical material (starts today)

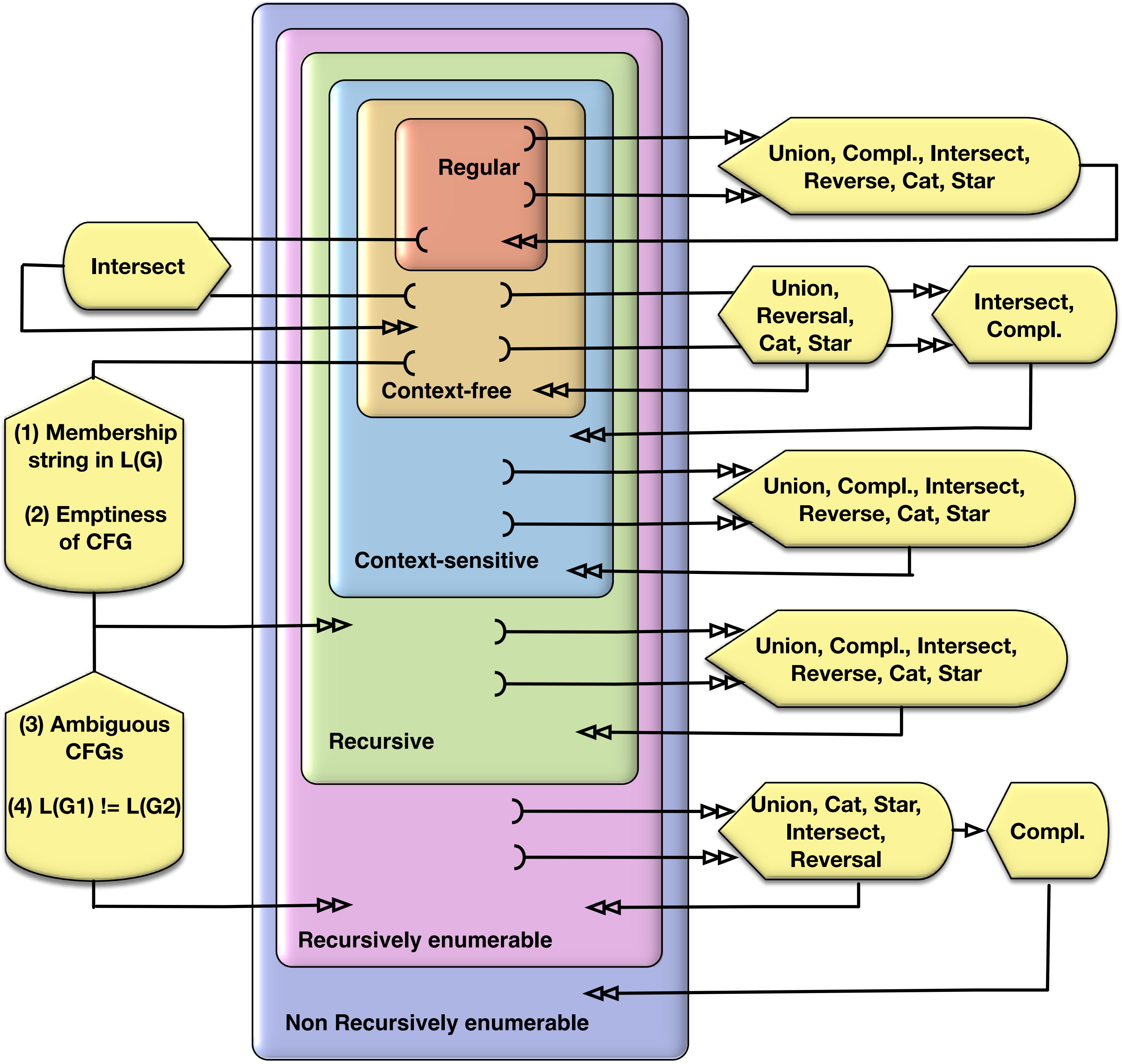
We now begin studying TMs with these views

- To convince ourself that anything that a “real” computer can do can be done on a TM (takes much longer; but feasible)
  - See a mechanical TM working here : https://youtu.be/E3keLeMwfHY
- We will then model problems using TM’s language
  - ”Solve a problem” turns into “is x a member of this TM’s language?”
- We will then use TM-based arguments as a way to “settle” many open questions out there
  - Which problems can be solved by a computer?
    - “Solved” means they have full algorithms
  - Which problems can be “semi-solved”?
    - “Semi-solved” means a “half algorithm” or “semi-algorithm”
      - You get answers when a certain language membership is true
  - Which problems cannot be solved?
    - Even a “half solution” is unavailable”

We now begin studying TMs with these views

- We will then use TM-based arguments as a way to “settle” many open questions out there
  - Which problems can be solved by a computer?
    - “Solved” means they have full algorithms
    - **The language in question is RECURSIVE (recursive implies recursively enumerable)**
  - Which problems can be “semi-solved”?
    - “Semi-solved” means a “half algorithm” or “semi-algorithm”
      - You get answers when a certain language membership is true
    - **The language in question is NOT RECURSIVE but RECURSIVELY ENUMERABLE**
  - Which problems cannot be solved?
    - Even a “half solution” is unavailable”
    - **The language in question is NOT EVEN RECURSIVELY ENUMERABLE**

Full picture of Formal Language Results (Ch 14, Fig 14.2)



TM in Turing’s Own Words… (Hodge’s biography)

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child’s arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e., *on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite … The behavior of the [human] computer at any moment is determined by the symbols which he is observing, and his state of mind at that moment.*

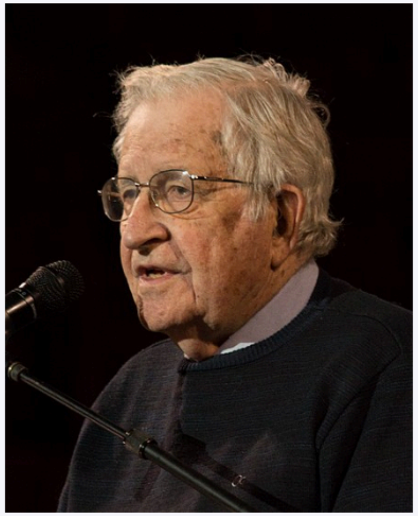
Checklist to do Quiz-7

- TM
  - Alphabets, looping, language
  - NDTM versus DTM (go over more)
  - Two-stack simulation (TODAY)
  - RE versus Recursive languages (TODAY)

The Chomsky Hierarchy of Machines/Languages

Machines	Languages	Nature of Grammar
DFA/NFA	Regular	Purely left-/right- linear productions
DPDA	Deterministic CFL	Each LHS has one nonterminal. The productions are deterministic.
NPDA (or “PDA”)	CFL	Each LHS has only one nonterminal.
LBA	Context Sensitive Languages	LHS may have length > 1, but $\lvert LHS \rvert \le \lvert RHS \rvert$, ignoring ε productions.
DTM/NDTM	Recursively Enumerable	General grammars ($\lvert LHS \rvert \ge \lvert RHS \rvert$ allowed).

Studying This Now



Chomsky in 2017

Born Avram Noam Chomsky
December 7, 1928 (age 90)
Philadelphia, Pennsylvania, U.S.

Figure 13.16: Situation of TMs in the Chomsky Hierarchy.

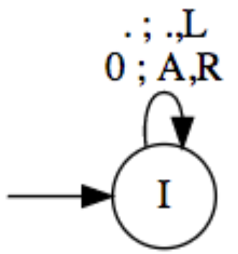
w define a crucially important notion called the **Chomsky**

The notion of the Language of a TM

Languages of these TM?

In [12]:
```
RunAwayTM = md2mc('''TM
I : . ; .,L | 0 ; A, R-> I
''')
DORunAwayTM = dotObj_tm(RunAwayTM, FuseEdges=True)
DORunAwayTM
```

Out[12]:



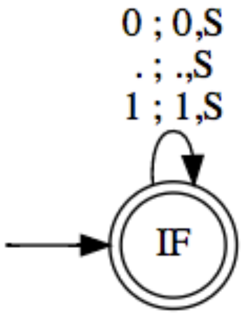
Languages of these TM?

In [12]:
```
RunAwayTM = md2mc('''TM
I : . ; .,L | 0 ; A, R-> I
''')
DORunAwayTM = dotObj_tm(RunAwayTM, FuseEdges=True)
DORunAwayTM
```

Out[12]:

In [13]:
```
YesManTM = md2mc('''TM
IF : . ; .,S | 0 ; 0,S | 1 ; 1,S -> IF
''')
DOYesManTM = dotObj_tm(YesManTM, FuseEdges=True)
DOYesManTM
```

Out[13]:



Languages of these TM?

In [12]:
```
RunAwayTM = md2mc('''TM
I : . ; .,L | 0 ; A, R-> I
''')
DORunAwayTM = dotObj_tm(RunAwayTM, FuseEdges=True)
DORunAwayTM
```

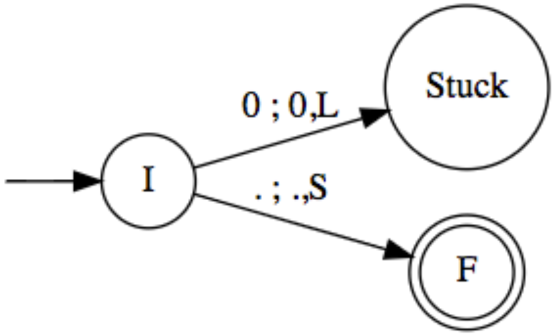
Out[12]:

In [13]:
```
YesManTM = md2mc('''TM
IF : . ; .,S | 0 ; 0,S | 1 ; 1,S -> IF
''')
DOYesManTM = dotObj_tm(YesManTM, FuseEdges=True)
```

Out[13]:

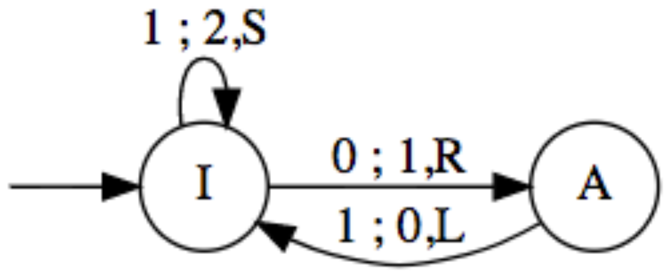
In [14]:
```
ZeroPhobeTM = md2mc('''TM
I : . ; .,S -> F
I : 0 ; 0,L -> Stuck
''')
DOZeroPhobeTM = dotObj_tm(ZeroPhobeTM, FuseEdges=True)
DOZeroPhobeTM
```

Out[14]:



Simulating TMs using “PDA with 2 stacks”

If you can operate on 2 stacks in an extended PDA, you get a TM



Simulate the various moves as follows

Let [….) mean the left stack

Let (….] mean the right stack

[….a) means “a” is on top of the left stack

(b…..] means “b” is on top of the right stack

[…..a) (b….] means that we have L-stack and R-stack

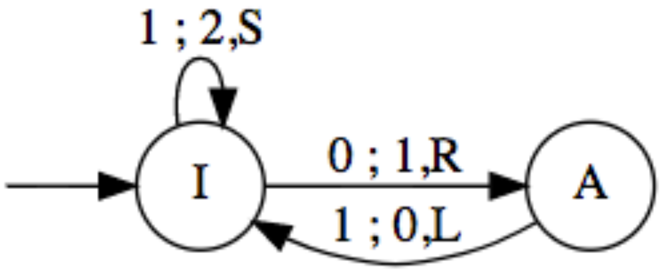
This is how the “TM” tape is modeled:

[…xa) (by….] … i.e. we choose to show what’s under the T.O.S. also!

We are always looking at the top of the right-hand side stack – arrange things to be so!

Simulating TMs using "PDA with 2 stacks"

If you can operate on 2 stacks in an extended PDA, you get a TM



Simulate the move
"If I'm looking at q under my TM head, I want to change q to an x,
and then move right"

I'll write […ab) (qp…] → [….abx) (p…]
i.e.

- Pop the right stack
- Push x on the left stack

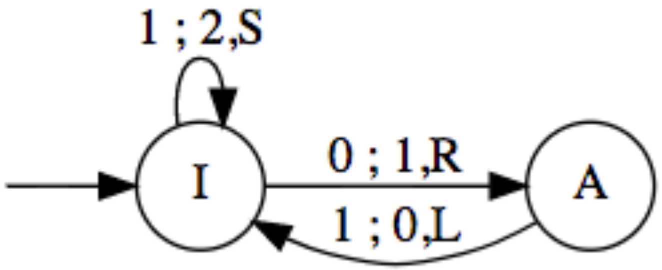
Example: the transition from I to A is 0; 1,R

This will be captured as

[..ab) (0p …] → [..ab1) (p…]

Simulating TMs using "PDA with 2 stacks"

If you can operate on 2 stacks in an extended PDA, you get a TM



Simulate the move
"If I'm looking at q under my TM head, I want to change q to an x,
and then move left"

I'll write […ab) (qp…] → [….a) (bxp…]
i.e.

- Capture the top of the left stack (call it b)
- Pop it (left stack)
- Push x on the left stack and then push b on the right stack

Example: the transition from A to I is 1 ; 0, L

This will be captured as

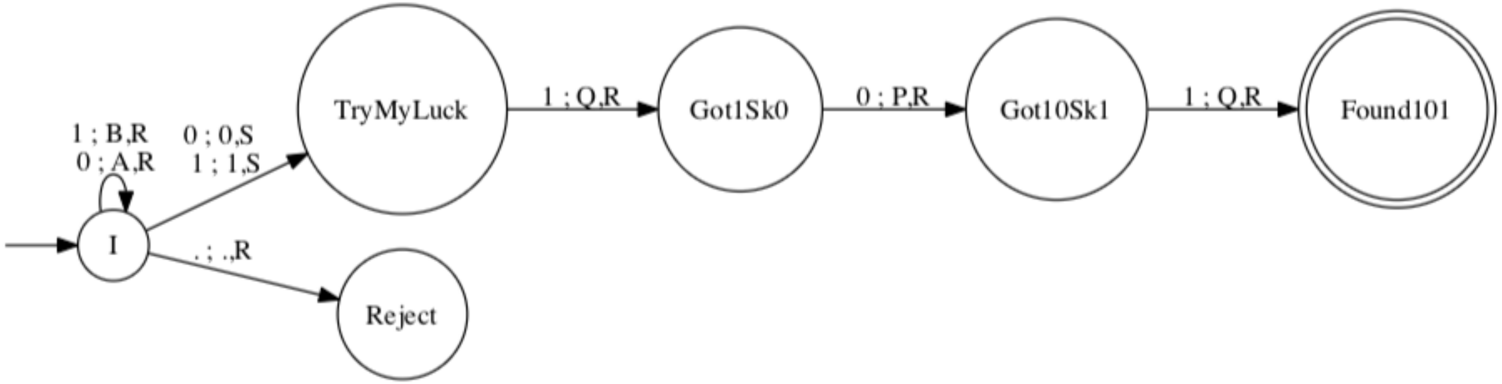
[..ab) (1p …] → [..a) (b0p…]

For every NDTM, there is an equivalent DTM

DTMs and NDTMs are Equivalent in Power

- Given any DTM, there is a language-equivalent NDTM
  - Proof: Direct, because any DTM is also an NDTM
- Given any NDTM, there is a language-equivalent DTM
  - Proof Sketch: One can build a DTM that can simulate each non-deterministic option taken along the computational tree
  - The simulation may increase the runtime exponentially
  - But it still ensures halting!

How to “Determinize” this NDTM



Determinizing any NDTM (the way it is usually done):

- Show that a “multi-tape TM” is equivalent to a single-tape TM
- Keep the ND choices on one tape
- Search through them one by one

- In this example, this conversion would remember that at “I”, one could have executed an ND step, and builds a tree of choices to explore

Therefore, we can study procedures/algorithms using DTM

Procedure versus Algorithm

- When a program is known to halt on all inputs, it is said to realize an algorithm
- “Algorithm” goes with “Recursive Sets”
- When a program may loop on some of its inputs (we don’t know whether it would halt on all inputs), we say that the program realizes a procedure
- ”Procedure” goes with “Recursively Enumerable” sets

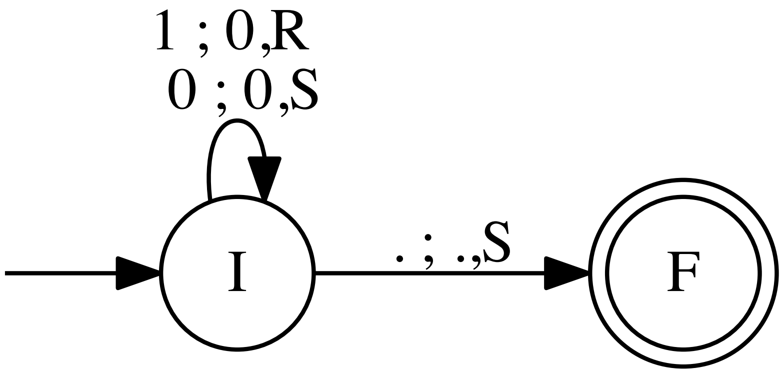
Key Features of Algorithms

- Algorithms are special cases of procedures (”always halt”)
- It is only for algorithms that we meaningfully specify the runtime using the Big-O notation
- For a procedure, the Big-O runtime is INFINITY!
- **REASON ?**

Key Features of Algorithms

- **REASON:**
- **Big-O tracks the worst-case runtime of a program.**
- **If a program can loop, the worst-case is infinity.**

Example TM dtm2



Task for you:
Does this TM realize a procedure or an algorithm?

If an algorithm, what time complexity (# of steps taken by the TM as a function of the input length)

See Rec Enum sets (bottom); Rec sets are a special case

Machines	Languages	Nature of Grammar
DFA/NFA	Regular	Purely left-/right- linear productions
DPDA	Deterministic CFL	Each LHS has one nonterminal. The productions are deterministic.
NPDA (or “PDA”)	CFL	Each LHS has only one nonterminal.
LBA	Context Sensitive Languages	LHS may have length > 1, but $\mid$ LHS$\mid$ $\leq$ $\mid$RHS$\mid$, ignoring ε productions.
DTM/NDTM	Recursively Enumerable	General grammars ($\mid$LHS$\mid$ $\geq$ $\mid$RHS$\mid$ allowed).

Figure 13.16: Situation of TMs in the Chomsky Hierarchy.

w define a crucially important notion called the **Chomsky**

The notion of Recursively Enumerable Sets

* Regular Sets (Languages) <-> DFA

* Context-Free Sets (Languages) <-> PDA

* Recursively Enumerable Sets (Languages) <-> DTM

Recursively Enumerable Language L

- L is a recursively enumerable (RE) language if there is a TM (call it TM_L) whose language L is
- EQUIVALENTLY
- L is a recursively enumerable language (RE) if the contents of L can be listed systematically (e.g. in numeric order) by a single TM, say TM_L

Recursive Language L

- L is a recursive (Rec) language if there is a TM (call it TM_L) whose language L is, and furthermore given something, say "x" not in L, TM_L can examine "x", reject it, and halt [DECIDER for L or ALGORITHM TO CHECK MEMBERSHIP IN L)

- EQUIVALENTLY

- L is a recursive language (Rec) if the contents of L can be listed systematically (e.g. in numeric order) by a single TM, say TM_L, and furthermore there is also a TM, say TM_Lbar, that can enumerate L-bar (complement of L) also

Examples of RE and Recursive Languages

- Boring/uninteresting ones
  - {} is Recursive (hence also RE)
  - {1} is Recursive
  - {1,2,3,44} is Recursive
  - {“hello”, “there”} is Recursive
  - {1,2,3,4,…. To infinity} is Recursive (set of Nat)
  - Primes are Recursive
  - Sets of all Checkmate positions in Chess boards: Recursive
  - All { <In,Out> … } where In are arrays to be sorted and Out are sorted arrays
    - Again Recursive
  - These are boring / uninteresting because we KNOW that there are algorithms to check membership
- Really interesting ones: that study OTHER MACHINE’s BEHAVIORS!!

Examples of RE and Recursive Languages

- Really interesting ones: that study OTHER MACHINE’s BEHAVIORS!!
  - { <G> : G is a CFG } is Recursive
  - { <P> : P is a legal Java Program } is Recursive
  - { <D> : Language(DFA D) is empty } is Recursive
  - { <G> : Language(CFG G) is empty } is Recursive

- We will learn how to argue that the above are true

Examples of RE and Recursive Languages

- Really interesting ones: that study OTHER MACHINE's BEHAVIORS!!
  - { <G,IN> : G is a CFG and IN is an input and Parser(G) accepts IN } is Recursive
  - { <G,IN> : G is a CFG and IN is an input and Parser(G) doesn't accept IN } is Recursive

  - { <M , w> : M is a legal TM and w is its input and M accepts w } : RE not Rec!
  - { <M, w> : M is a legal TM and w is an input and M does not accept w } : not even RE !!

  - { <P, in> : P is a legal Java Program and in is any input submitted to P and P when run on in halts } is Recursively Enumerable but not Recursive !!
    - Same behavior as <M,w> because Java programs and TMs are similar !!

  - We will learn how to argue that the above are true

Example of how to show “Recursive”

- Set of DFA descriptions whose language is empty
  - L = { < D > : D is a DFA with an empty language }
  - Is this an RE language?
    - If so which TM shows it?
    - What enum procedure? Shows it?
  - Is this a Recursive language?
    - If so which algo would you propose for membership?
    - Can you now tell me how to enumerate L-bar, the complement of L ?

Example of how to show "Recursive"

- We just studied this in the previous slide:
  - Set of DFA descriptions whose language is empty
    - { < D > : D is a DFA with an empty language }
    - RE? (if so TM? enum procedure?) Rec? (if so algo? Method to list L-bar?)
- Now study the same situations on the following languages
  - Set of DFA descriptions whose language is non-empty
    - RE? (if so TM? enum procedure?) Rec? (if so algo? Method to list L-bar?)
  - Set of PDA descriptions whose language is non-empty
    - RE? (if so TM? enum procedure?) REC? (If so, algo? Method to list L-bar?)

Two centrally important languages

- A_TM = { <M,w> : M is a TM with input alphabet Sigma,
  and w is a string in Sigma* and M accepts w }

- H_TM = { <M,w> : M is a TM with input alphabet Sigma,
  and w is a string in Sigma* and M halts on w }

We will study these closely related languages mainly to understand the various concepts we need to deeply understand

A general proof of a set being RE (14.3.3)

Theorem 14.3.3: A_{TM} is RE.

Alternate Proof:

Approach: By building this enumerator for A_{TM}:

- Keep listing pairs $\langle A, B\rangle$ of strings from Σ^* on an "internal tape."
- Keep checking whether A is a Turing machine description (e.g., our markdown language for the TM has a parser; one can run this parser and see if it accepts A). If so, A happens to be a Turing machine description.
- Run Turing machine A on B, treating B as its input. Again, do not run to completion; instead, *engage in a dovetailed execution with all other TMs and inputs meanwhile being enumerated internally.*
- When the dovetailed simulation finds an $\langle A, B\rangle$ pair such that A accepts B, it lists the $\langle A, B\rangle$ pair on the output tape.
- This listing will produce every $\langle M, w\rangle$ such that M accepts w.
- The existence of this enumerator means that A_{TM} is RE. $\square$

How to argue that A_TM is RE

- A_TM = { <M,w> : M is a TM with input alphabet Sigma, and w is a string in Sigma* and M accepts w }

Now, study Asg-6's remaining problems

- Go through Problem 4, except for Part (e)

How to argue that A_TM is **not** recursive

- Also can be stated as "A_TM is undecidable"
  - Undecidable means the same as "not recursive"
  - Decidable means "Recursive"
- Decidable problems are desirable also !!

How to argue that A_TM is **not** recursive

```
/* Let there be a LIBRARY FUNCTION DeciderA(TM M, input x)
/* Property: DeciderA always returns with a True/False
/* True if M accepts x; False if not
/* We want to show DeciderA does not exist.
/* To achieve this proof, we are going to define function D

Diagonal(TM M) {
  accepts = DeciderA(M,M);
  if (!accepts)
    goto accept_Diagonal;
  else
     goto reject_Diagonal;

  accept_Diagonal: print("I have accepted."); Exit;
  reject_Diagonal: print("I have rejected."); Exit;
}
```

How to argue that H_TM is **not** recursive

- Your Asg-6