CS 3100, Models of Computation, Spring 20, Lec 22
April 1, 2020

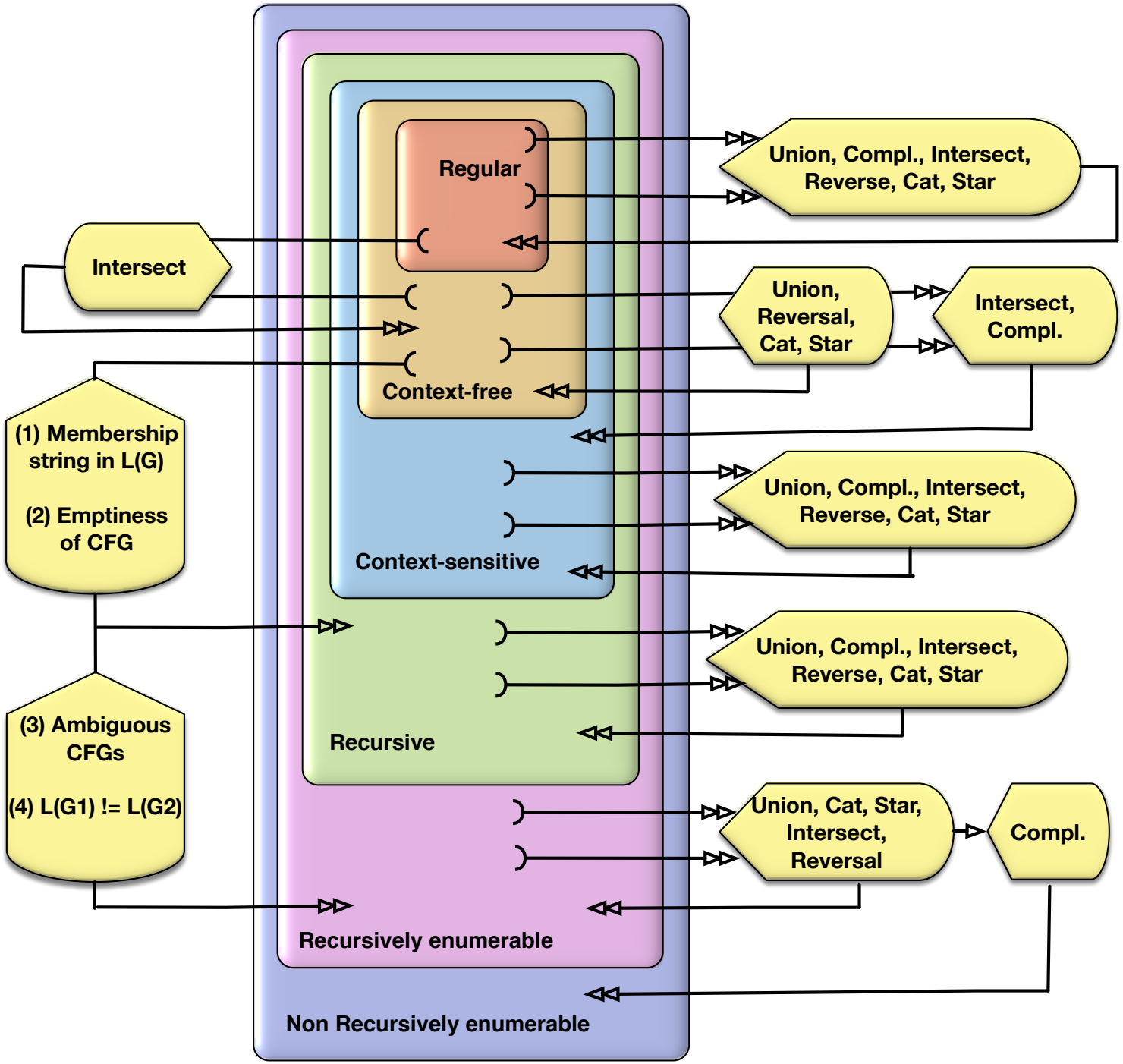
Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

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

Agenda for Wed Apr 1

- Review basic definitions
- Make sure everyone can do Asg-6
- Introduce Diagonalization and the A_TM problem

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



The notion of Recursively Enumerable Sets

* Regular Sets (Languages) <-> DFA or NFA

* Context-Free Sets (Languages) <-> PDA (not just DPDA)

* Context-Sensitive Sets (Languages) <-> LBA (open issue: NLBA versus DLBA)

* Recursively Enumerable Sets (Languages) <-> DTM or NDTM
  - Special case: Recursive Sets

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

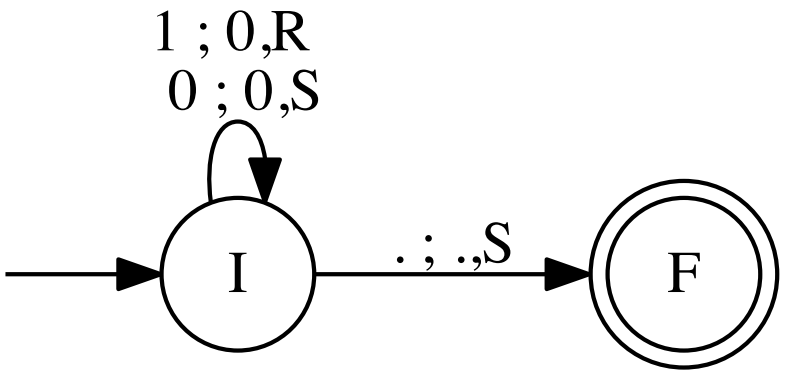
Key Features of Algorithms

- 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)

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. □

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

This is where we will study the “Halting Problem”

- This is colloquially stated as
  - We cannot tell if a program will halt on an input

- There are many connotations to this question
  - There is no way one can know whether a given problem is “solvable”
    - E.g. check the Ambiguity of a given CFG algorithmically!
    - E.g. Does this given PCP puzzle have a solution (or not)?
  - This also includes everyday problems
    - E.g. why is this program not executing this “print” statement?
    - Can I even show that this “print” will never get executed?

The Halting Problem is FOUNDATIONAL

- VIRTUALLY EVERYTHING we study in CS finally leads to the question of “what are the fundamental limits of computing”?

- We formulate and prove the undecidability of the Halting Problem using a proof that A_TM is undecidable
  - The A_TM problem really shows “acceptance is undecidable”
  - In Asg6, you will prove that H_TM is undecidable (“halting is undecidable”)

- Recall: ”Is undecidable” means “is not recursive”

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”
- Deci**d**able problems are desi**r**able also !!

How to argue that A_TM is not recursive

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

6: Diagonal(TM M) {
7:    accepts = DeciderA(M,M);
8:    if (!accepts)
9:      goto accept_Diagonal;
10:   else
11:     goto reject_Diagonal;
12:
13:   accept_Diagonal: print("I have accepted."); Exit;
14:   reject_Diagonal: print("I have rejected."); Exit;
15: }
```

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

- Your Asg-6

Why “diagonalization?” (See Appendix C, book)

- Arose from Cantor’s proof method called “diagonalization”
- Basics
- Nat is an infinite set
- Real is an infinite set
- There are “more Reals than Nat”
- There are Aleph_0 Nats (first infinite cardinality)
  - “Countably Infinite”
- There are Aleph_1 Reals (second infinite cardinality)
  - “Uncountably Infinite”

Why “diagonalization?” (See Appendix C, book)

- Arose from Cantor’s proof method called “diagonalization”
- Quick illustration: There are more languages than RE languages
- Each RE language goes with a TM
- There are an Aleph_0 number of TMs
  - How to show? Use the Schroder-Bernstein Theorem (SBT)
  - Also known as “mirror in front of a mirror” Theorem (I call it that, anyhow)
- Will present the SBT soon

Why “diagonalization?” (See Appendix C, book)

- Arose from Cantor’s proof method called “diagonalization”
- Quick illustration: There are more languages than RE languages

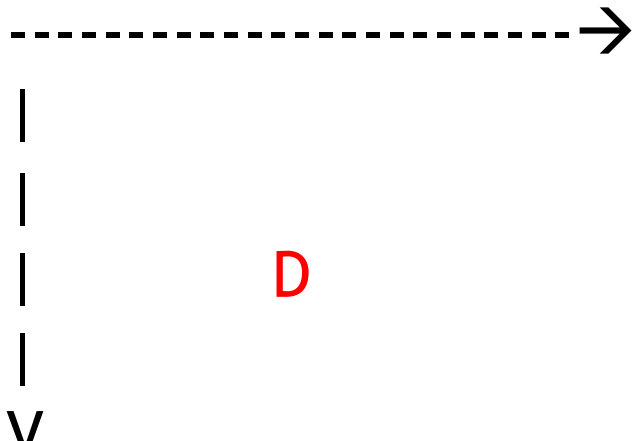
	‘’	0	1	00	01	10	11	000	001	
L0	0	1	0	0	1	1	0 …			→ language {0,01,10}
L1	0	0	0	….						-> Language { }
L2	1	1	1	1						-> Language Sigma*
L3	1	0	1	0	1	0	… ->			-> Alternate strings in num order

Diag language : Complement the diagonal (red bits) → is a language not in the listing!

L0 L1 L2 L3 … all go with TMs (Aleph_0). The diagonal language can’t be the language of a TM

In our proof of A_TM, we have diagonalization going on…

These are Machines



Being run on these machines

Machine “D” is on the diagonal, and D is being run on D
And that causes a contradiction! Hence “like diagonalization”

Why “diagonalization?” Two illustrations

- Arose from Cantor’s proof method called “diagonalization”
  - Cantor showed that no set and its powerset can have a bijective map
    - No 1-1 and onto map
  - For finite or infinite set
  - Example: Nat and Powerset of Nat don’t have a bijective map
  - Powerset of Nat and Reals have a bijective map
  - Thus Nat and Real don’t have a Bijective map
- Schroder-Bernstein Theorem
  - If A and B are two sets and there is a 1-1 map from A to B and vice-versa, then there is a bijective map between A and B

Illustration of the SBT

- There are are many C programs as Nat
- SBT requires two injections (1-1 maps)
  - One from Nat to C and the other from C to Nat
- Nat to C : 0 -> main(){} , 1 -> main(){;}, 2 -> main(){ ;; }, etc
  - All these trivial C programs are legal; they do compile and run!
- C to Nat : just take the ascii string and read it as a Nat ☺

Zoom quizzes

Concepts Covered

- 1. This is how I feed "equal a's and b's to a TM" - for example the string "aabb" (Multiple Choice)
  - Answer 1: By putting it as "aabb" and no other restriction
  - Answer 2: Put it as "aabb" with the TM head pointing to the left end
  - Answer 3: Put it as "aa.bb" on the tape (allow blanks in the input)

DTM versus NDTM

- Here are facts that apply to a DTM and an NDTM (Multiple Choice)
  - Answer 1: An NDTM can move without processing the current character under its tape head
  - Answer 2: An NDTM and DTM are one and the same syntactically
  - Answer 3: You would not even know whether the nondeterminism you coded in a TM will ever get exercised!
  - Answer 4: You can deliberately plan to be coding an NDTM (then choose the guess/check approach)
  - Answer 5: A DTM or an NDTM can leave the current character and tape-head position the same (it still REWRITES the character and executes the S move)

NDTM versus DTM

- It is possible to algorithmically determine whether the nondeterminism one coded in a TM will ever get exercised (Single Choice)
  - Answer 1: Yes
  - Answer 2: No

TM's alphabet, its makeup and size

- A TM's tape alphabet is (Multiple Choice)
  - Answer 1: Always larger than the "input alphabet" by exactly one
  - Answer 2: Could be larger than the input alphabet by at least one but any finite amount after that
  - Answer 3: Could be infinite

Turing machine and when it accepts its input

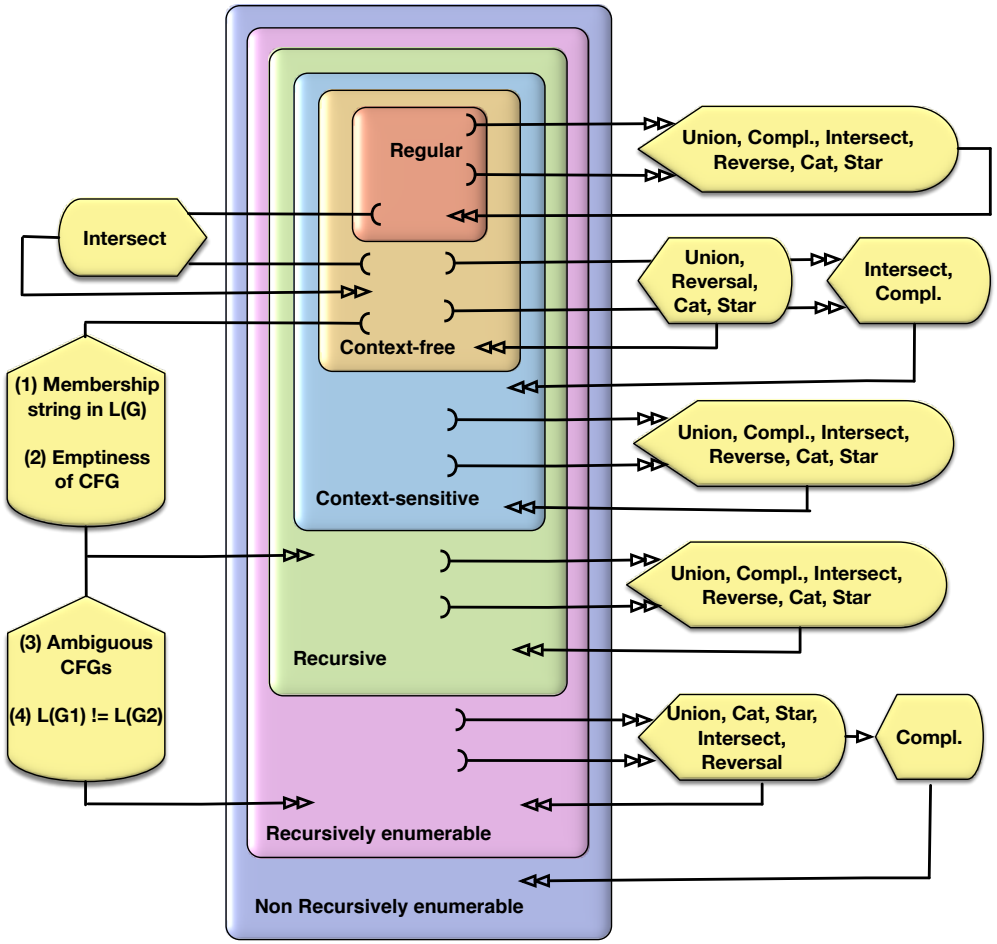
- A TM accepts its input under these conditions (Multiple Choice)
  - Answer 1: It has fully read its input exactly once (nothing more)
  - Answer 2: It has read its input more than once (nothing more)
  - Answer 3: It may read its input any number of times or even partially and be stuck in a state
  - Answer 4: It may read its input any number of times or even partially and be stuck in a FINAL state
  - Answer 5: It has to exactly read its input left-to-right and be stuck in a final state

The language of a TM

- The language of a TM is (Multiple Choice)
  - Answer 1: All the strings that lead it to get stuck in a final state
  - Answer 2: All the strings that lead it to get stuck in some state
  - Answer 3: For a given TM, its language can be regular
  - Answer 4: In general, the language of a TM is recursively enumerable (higher in the Chomsky Hierarchy which we will study)

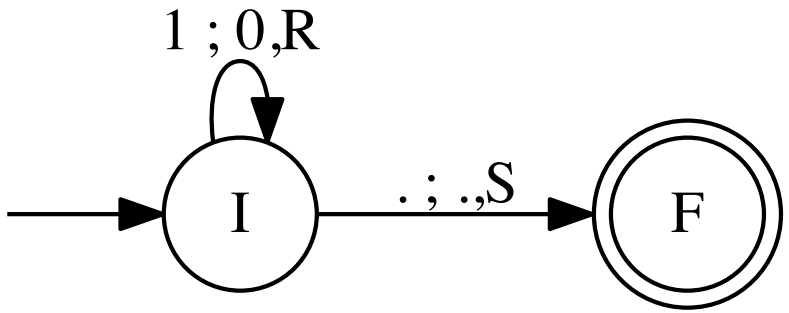
The Venn Diagram of languages (Fig 14.2)

- Answer 1: It shows closure and containment of formal languages
- Answer 2: It shows how you can fall outside of a class under some operations
- Answer 3: It shows that there are languages not even model-able by a TM (non-RE)



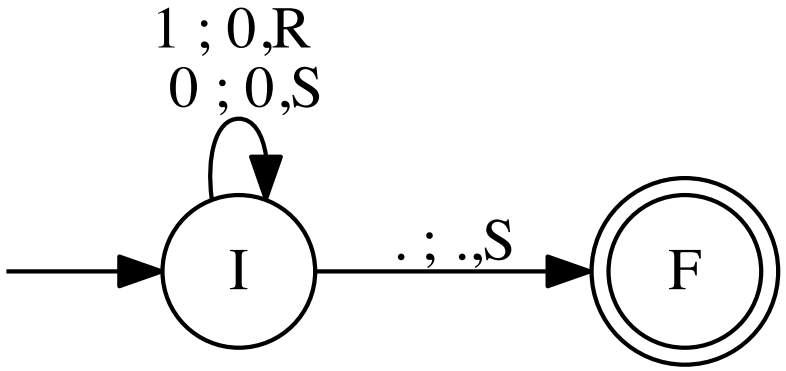
Can this TM (dtm1) loop? on what inputs? (Multiple Choice)

- Answer 1: It can loop
- Answer 2: It can't loop
- Answer 3: It can loop on THIS input (type in chat)



Can this TM (dtm2) loop? on what inputs? (Multiple Choice)

- Answer 1: It can loop
- Answer 2: It can't loop
- Answer 3: It can loop on THIS input (type in chat)

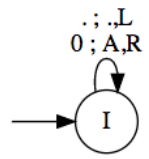


The three TMs

- Answer 1: first TM empty
- Answer 2: second TM empty
- Answer 3: second TM sigma*
- Answer 4: second TM unit language
- Answer 5: Third TM - no zero

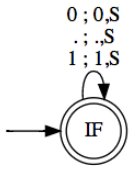
```
In [12]: RunAwayTM = md2mc('''TM
         I : . ; .,L | 0 ; A, R-> I
         ''')
         DORunAwayTM = dotObj_tm(RunAwayTM, FuseEdges=True)
         DORunAwayTM
```

Out[12]:



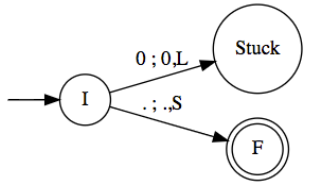

```
In [13]: YesManTM = md2mc('''TM
         IF : . ; .,S | 0 ; 0,S | 1 ; 1,S -> IF
         ''')
         DOYesManTM = dotObj_tm(YesManTM, FuseEdges=True)
         DOYesManTM
```

Out[13]:




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

Out[14]:



What is an RE Language?

- A recursively enumerable language L (Multiple Choice)
  - Answer 1: Is associated with one specific TM
  - Answer 2: Is associated with any one of an infinity of TMs that have language L
  - Answer 3: The language of a TM that must always halt regardless of whether it is run on a string in L or outside of L
  - Answer 4: The language of a TM that must halt on inputs in L (may loop on inputs outside of L)

Recursive Language?

- A recursive language L is (Multiple Choice)
  - Answer 1: The language of a single TM M
  - Answer 2: The language of any one of an infinity of TMs whose language is L
  - Answer 3: The language of a TM that halts on inputs inside and outside of L
  - Answer 4: A language L such that both L and L-bar (complement) are RE
  - Answer 5: Also known as a Decidable Language

RE Languages

- If a language L is finite then (Multiple Choice)
  - Answer 1: L is RE
  - Answer 2: L-bar (complement of L) is RE
  - Answer 3: L is recursive
  - Answer 4: It is possible that L is RE but not Recursive
  - Answer 5: It is possible that L is Recursive but not RE

Wof defining an RE language

- . An RE language L can be defined in many ways (Multiple Choice). Also many associated facts
  - Answer 1: There is a TM that enumerates L
  - Answer 2: There is a TM that recognizes L
  - Answer 3: There is a TM that enumerates L; then one can realize a TM that recognizes L also
  - Answer 4: There is a TM that recognizes L; then one can realize a TM that enumerates L also.