

CS 3100, Models of Computation, Spring 20, Lec 20

March 25, 2020

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

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



Agenda for Wed March 25

- Walk through First_Jove_Tutorial/CH13/CH13.ipynb in full detail
- Live-code a TM for $w\#w$ where w in $\{0\}^*$
 - At this point, expect students to read and try out the other examples suggested in these notebooks and successfully code-up a DTM
 - Also expect them to do Asg-6 Problem-1
 - START EARLY! This is potentially tricky!
- Live-code an NDTM to look for “11” in a string
 - At this point, assume the students can do Asg-6 Problem-2
- Go over PCP once more
 - At this point, assume the students can do Asg-6 Problem-3
- Devote the rest of the lecture to Quiz-7 and Asg-6 Qn 4,5
 - 2-stack simulation of a TM
 - RE and Recursive Sets
- But we will go thru all the basic material quickly, do quizzes etc

Take a look at Asg-6, Question 3

- The topic is the Post Correspondence Problem (Chapter 15)
- Let us examine its significance
 - A hugely important problem in CS --- yet, easily explained even to a 7-year old
 - Can be used as the starting point to show that
 - CFG Ambiguity is **undecidable**
 - First-order Validity is **undecidable**
 - Checking the reachability of communicating finite-state machines is **undecidable**

Elements of a TM, one by one

- Turing machines are structures $(Q, \Sigma, \Gamma, \Delta, q_0, B, F)$
- Has finite-state control (as with a DFA, NFA, or PDA)
- Has an input alphabet Σ
 - Strings over this input alphabet can be submitted as “input” to the TM
 - The input w in Σ^*
- Has a tape that is doubly infinite
 - The read/write head faces one cell initially
 - That cell
- Transitions mention what character is being looked at, what it must be changed to, and which way the head must move
 - Head motions are L,R,S for “move left”, “move right”, “stay at the same place”

General TM conventions

- Initialize input on tape
 - don't put blanks in the middle of your input
- End input with blanks
 - To present “epsilon”, leave a complete blank tape
 - Blanks denoted by “.” in our Jove input
 - “readable blank”
- A TM may read an input multiple times
 - It can move its head back to an already read input
 - It can modify a tape cell and read that modified value also
- A TM halts when it gets “stuck”
 - It may not have read any of its input
 - Still, if it gets stuck in the accept state, that input which was laid out on the tape is deemed to have been accepted
 - All other states are “reject” states – being stuck in a non-accept state is tantamount to rejecting the input
 - A TM may never get stuck (in an accept or a non-accept state)
 - It is then said to loop
 - It may zig-zag away .. Never repeating any looping pattern

“Care and feeding” of a TM

- To feed a TM epsilon
 - Leave its tape fully blank
- To feed a TM something else other than Epsilon
 - Make sure that we don't put anything outside of its Sigma on the tape
 - The TM may in fact put things outside of Sigma on its tape
 - But it can do so as special “markers”
 - This was allowed even for a PDA, so a TM is also allowed
 - But when a string from Sigma* is to be presented,
 - DO NOT insert any blanks in the middle!
 - A TM “thinks” that a blank in the middle is the end of your string.

Language of a TM

- The language of a TM is the set of inputs that lead to the TM eventually being stuck in an accepting state
 - A TM may **halt** or **loop**
 - "Loop" means "not stuck yet" (I'm still a journey-man)
 - **Halt** means found stuck in an "F" state
 - We will make sure that **all useful TMs** (from which we want an accept/reject decision)
 - Go to an "F" state from which there are no outlets (no moves possible) for announcing "success"
 - It is like a program that goes to the "success label"
 - Go to a "non-F" state to announce rejection (no outlets from there either)
 - It is like a program that goes to a label "unsuccessful"
- Till there is positive evidence (of being stuck in an F state), we can't say what is in the language of a TM
- A TM may never have read the input or only partially read the input or read the input multiple times....
 - Just the mere act of placing "w" on the tape and later finding the Turing machine stuck at "F" is enough to declare "w is accepted"

Run this!

[First_Jove_Tutorial/Start_with_These_Animations.ipynb](#)

Study the basics of DTM and NDTM behavior from there

Consult the book for larger TMs

- To solve Problems 1 and 2, we need to learn about larger TMs
- Chapters 13.1 through 13.6 cover these (take a look)

Run this!

[First_Jove_Tutorial/CH13/CH13.ipynb](#)

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

Run this!

First_Jove_Tutorial/CH15/CH15.ipynb

Get to use a PCP-solver

The only one out there that solves hard problems (as far as I know)

Authored by Ling Zhao of Univ of Alberta in early 2000's
(posted by some kind person at <https://github.com/chrozz/PCPSolver>)

Where is all this material going?

- We will study the Chomsky Hierarchy
- We will understand the full picture of formal language results

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 $ LHS \leq RHS $, ignoring ϵ productions.
DTM/NDTM	Recursively Enumerable	General grammars ($ LHS \geq RHS $ allowed).

Studying
This
Now



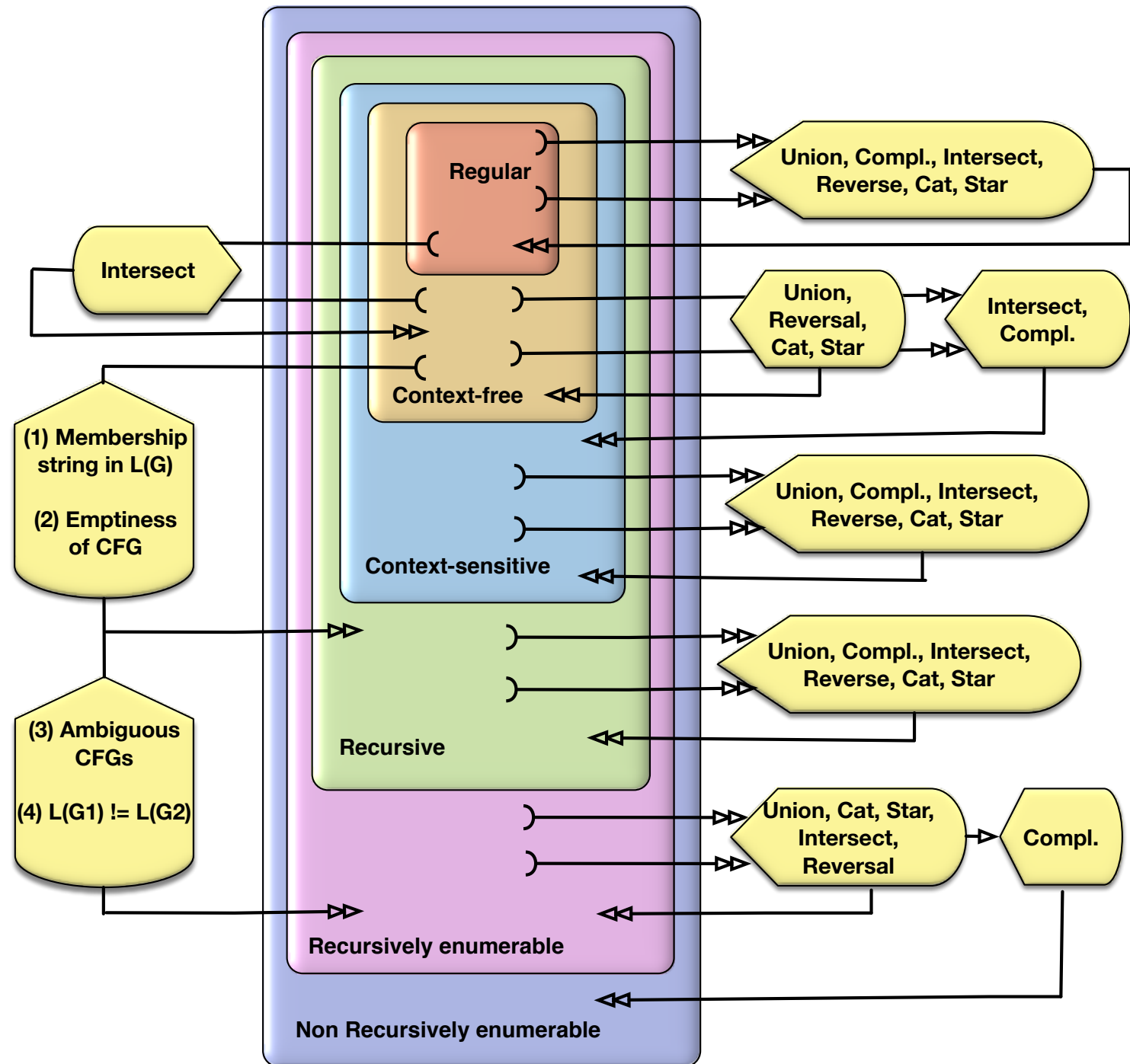
Chomsky in 2017

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

Figure 13.16: Situation of TMs in the Chomsky Hierarchy.

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

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



TM in Turing's Own Words... (Hodge's biography)

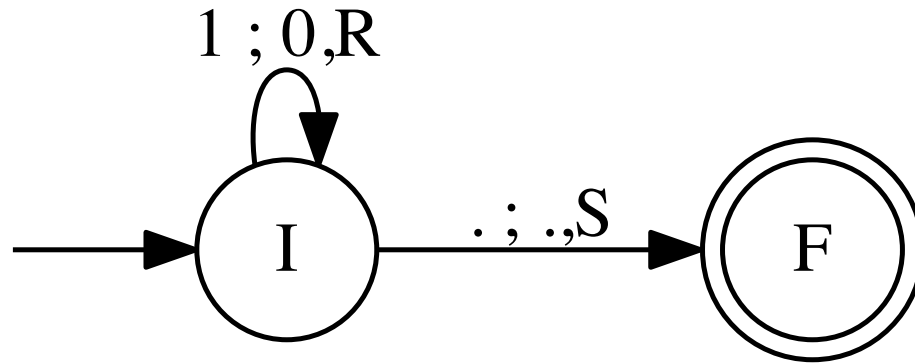
Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed 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.



An in-depth study of TMs and Computability

- We will now go through the details of TMs
- This will be followed by results from Computability
- We will need to carefully understand the notions of recursively enumerable and recursive sets (Asg-6, Question-4's parts)
- The goal of Wed's lecture (3/25) will be to cover these topics

Example TM dtm1



Task for you:

Is this a DTM or NDTM?

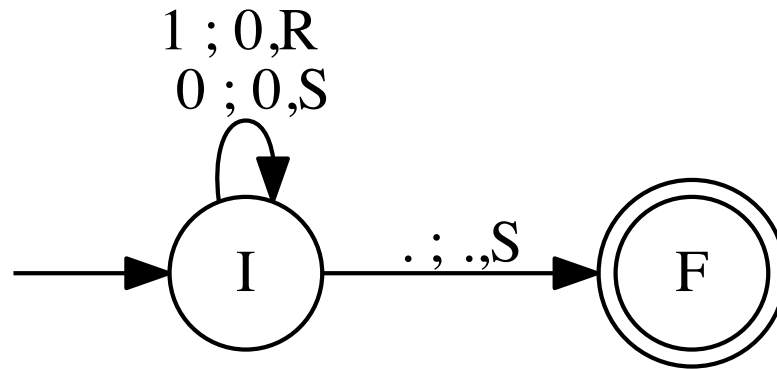
What is the language of this TM?

Answer in set-comprehension form.

Can this TM loop? If so, on what input(s)?

How do you feed an “” to any TM? This TM?

Example TM dtm2



Task for you:

Is this a DTM or NDTM?

What is the language of this TM?

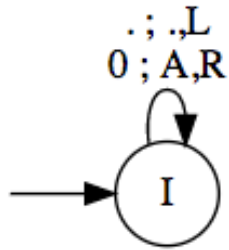
Answer in set-comprehension form.

Can this TM loop? If so, on what input(s)?

Languages of these TM?

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

Out[12]:



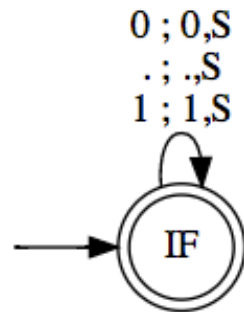
Languages of these TM?

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

Out[12]: . I

```
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
```

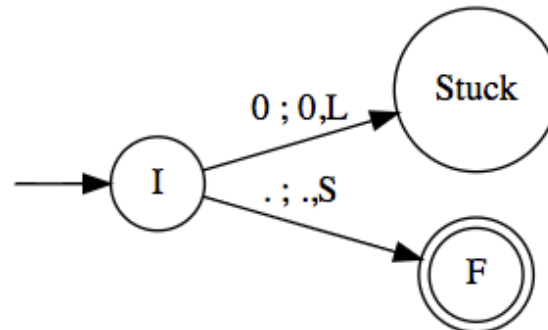
Out[12]: . I

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

Out[13]:

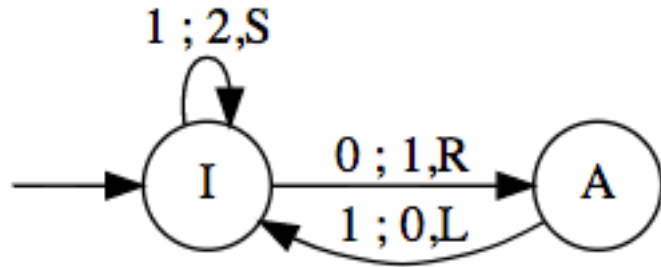
```
In [14]: ZeroPhobeTM = md2mc('''TM
I : . ; .,S -> F
I : 0 ; 0,L -> Stuck
''')
DOZeroPhobeTM = dotObj_tm(ZeroPhobeTM, FuseEdges=True)
DOZeroPhobeTM
```

Out[14]:



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 $[\dots)$ mean the left stack

Let $(\dots]$ mean the right stack

$[\dots a)$ means “a” is on top of the left stack

$(b \dots]$ means “b” is on top of the right stack

$[\dots a) (b \dots]$ means that we have L-stack and R-stack

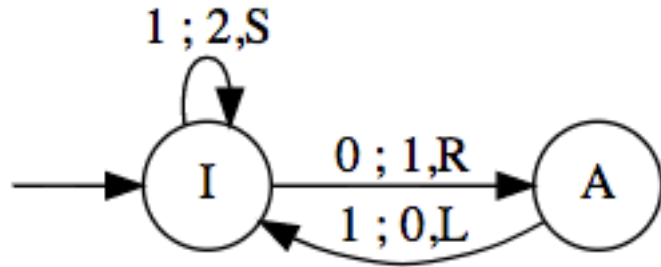
This is how the “TM” tape is modeled:

$[\dots x a) (b y \dots]$... 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 $[\dots ab) (qp \dots] \rightarrow [\dots abx) (p \dots]$

i.e.

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

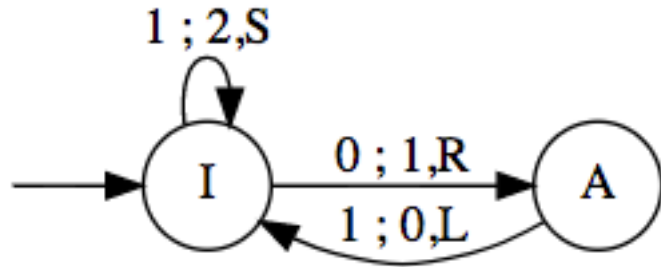
Example: the transition from I to A is 0; 1, R

This will be captured as

$[\dots ab) (0p \dots] \rightarrow [\dots ab1) (p \dots]$

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 $[\dots ab) (qp \dots] \rightarrow [\dots a) (bxp \dots]$

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

$[\dots ab) (1p \dots] \rightarrow [\dots a) (b0p \dots]$

Checklist to do Quiz-7

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

Procedure versus Algorithm

- When a program is known to halt on all inputs, it is said to realize an **algorithm**
- 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**

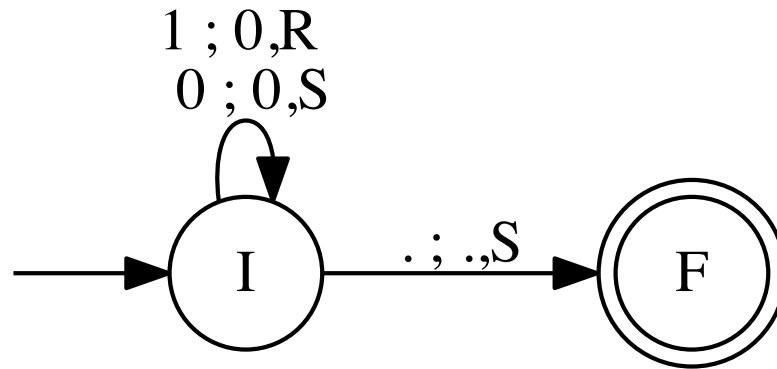
Key Features of Algorithms

- 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

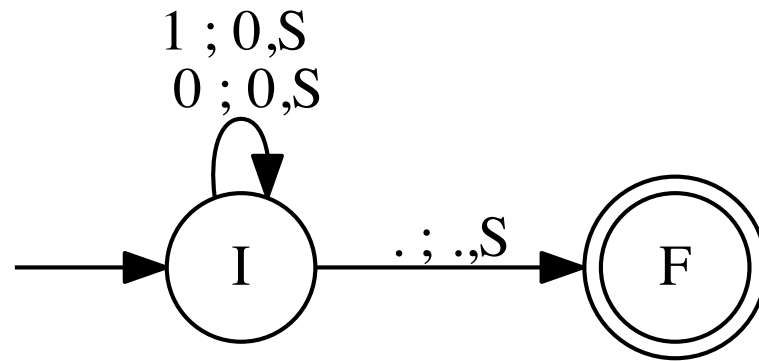


Task for you:

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)

Example DTM dtm3



Task for you:

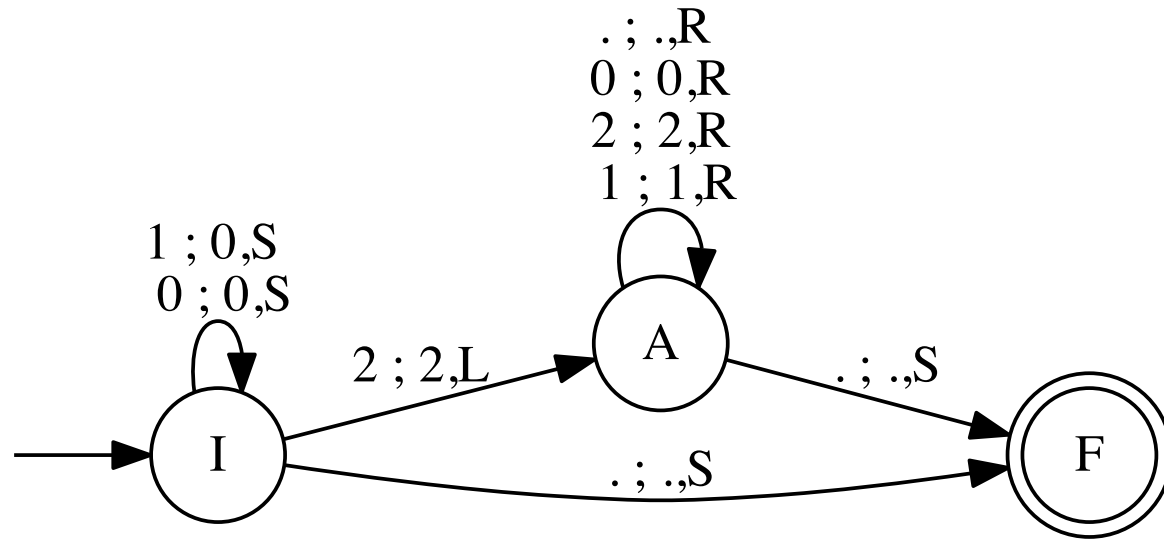
Is this a DTM or NDTM?

What is the language of this TM?

Answer in set-comprehension form.

Can this TM loop? If so, on what input(s)?

Example DTM dtm4



Task for you:

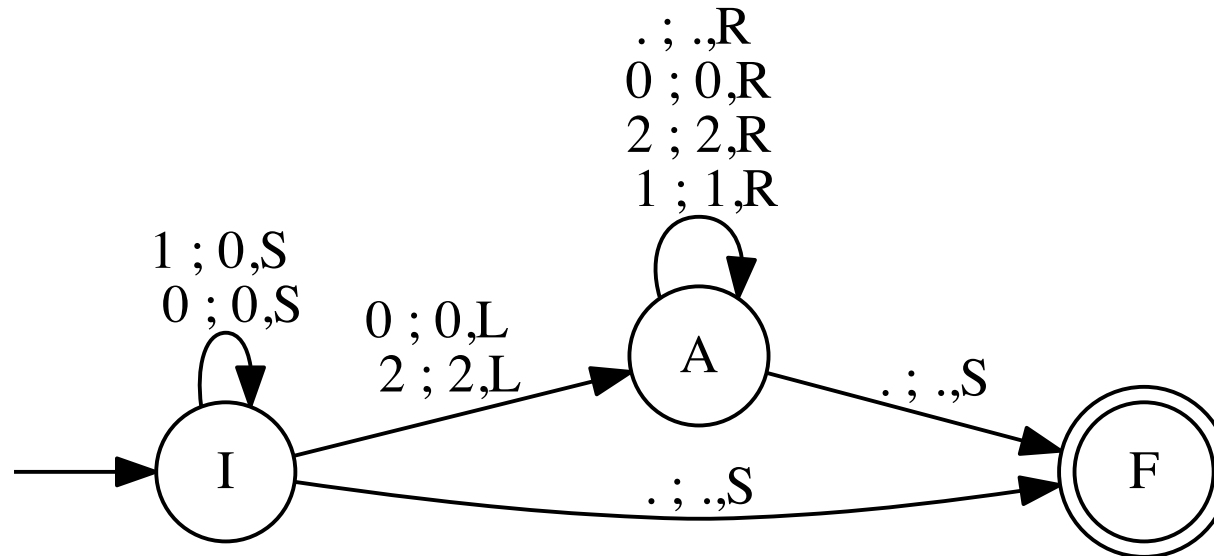
Is this a DTM or NDTM?

What is the language of this TM?

Answer in set-comprehension form.

Can this TM loop? If so, on what input(s)?

Example DTM dtm5



Task for you:

Is this a DTM or NDTM?

What is the language of this TM?

Answer in set-comprehension form.

Can this TM loop? If so, on what input(s)?

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 $ LHS \leq RHS $, ignoring ϵ productions.
DTM/NDTM	Recursively Enumerable	General grammars ($ LHS \geq RHS $ allowed).

Figure 13.16: Situation of TMs in the Chomsky Hierarchy.

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

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 $TM_{\bar{L}}$, that can enumerate \bar{L} (complement of L) also

Here are some sets: some RE and some Rec

- Set of DFA descriptions whose language is empty
 - $\{ \langle D \rangle : D \text{ is a DFA with an empty language} \}$
 - RE? (if so TM? enum procedure?) Rec? (if so algo? Method to list $L\text{-bar}$?)
- Set of DFA descriptions whose language is non-empty
 - RE? (if so TM? enum procedure?) Rec? (if so algo? Method to list $L\text{-bar}$?)
- Set of PDA descriptions whose language is non-empty
 - RE? (if so TM? enum procedure?) REC? (If so, algo? Method to list $L\text{-bar}$?)

Let us solve the first example: L-emptyD

- Set of DFA descriptions whose language is empty
 - $L\text{-emptyD} = \{ \langle D \rangle : D \text{ is a DFA with an empty language} \}$
 - RE?
 - YES!
 - TM?
 - Build a TM that examines D, runs a DFA emptiness checking algo on D
 - Method to list L-emptyD ?
 - Keep listing various candidate D's on a tape (some "scratch area of the tape")
 - Check D's language to be empty
 - If so, print that D on an output tape

Let us solve the first example: L-emptyD

- Set of DFA descriptions whose language is empty
 - $L\text{-emptyD} = \{ \langle D \rangle : D \text{ is a DFA with an empty language} \}$
 - Rec?
 - YES!
 - ALGO?
 - Accept D and check if its language is empty. Accept if so, else reject
 - Method to list $L\text{-emptyD-Bar}$?
 - Keep listing various candidate D's on a tape (some "scratch area of the tape")
 - Check D's language to be empty
 - If so, **don't print** that D on an output tape, **else print it !!**

A general proof of a set being RE (14.3.3)

Theorem 14.3.3: A_{TM} is RE.

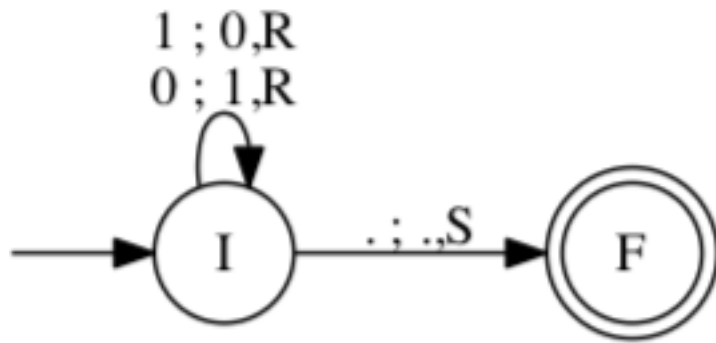
Alternate Proof:

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. □

Example-1: Flip bits given on input tape

- Initialize input on tape
- Fire-up the machine
- See how it gets stuck (with input-tape flipped)



Jove markdown description:

I : 0 ; 1,R -> I

I : 1 ; 0,R -> I

I : . ; .,S -> F

How to run it:

explore_tm(TM, input, fuel)

“fuel” = number of allowed time-steps

- Prevents “run-away” looping (no ^C)
- Measures time-complexity (number of TM-steps executed)

Allocating 8 tape cells to the RIGHT!

Detailing the halted configs now.

Accepted at

('F', 6, '101100.....', 93)

via ..

->('I', 0, '010011', 100)

->('I', 1, '110011', 99)

->('I', 2, '100011', 98)

->('I', 3, '101011', 97)

->('I', 4, '101111', 96)

->('I', 5, '101101', 95)

->('I', 6, '101100', 94)

->('F', 6, '101100...', 93)

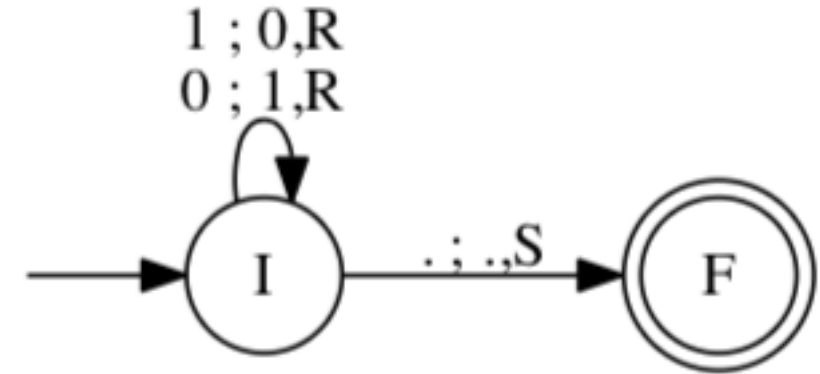
Example-2: Look for 0; turn into a 1; accept

```
In [25]: explore_tm(TM0tol, "101", 10)

Detailing the halted configs now.
Accepted at ('F', 1, '111', 8)
via ..
->('I', 0, '101', 10)
->('I', 1, '101', 9)
->('F', 1, '111', 8)
```

```
In [26]: explore_tm(TM0tol, "111", 10)

Allocating 8 tape cells to the RIGHT!
Detailing the halted configs now.
Rejected at ('I', 3, '111', 7)
via ..
->('I', 0, '111', 10)
->('I', 1, '111', 9)
->('I', 2, '111', 8)
->('I', 3, '111', 7)
```



Example-3: Look for 01 or 10 using nondet.

```
explore_tm(TM01OR10, "1111",10)
```

Allocating 8 tape cells to the RIGHT!
Detailing the halted configs now.

Rejected at ('Seek01', 4, '1111', 5)

via ..

->('I', 0, '1111', 10)

->('Seek01', 0, '1111', 9)

->('Seek01', 1, '1111', 8)

->('Seek01', 2, '1111', 7)

->('Seek01', 3, '1111', 6)

->('Seek01', 4, '1111', 5)

Rejected at ('Got1_Nxt0', 1, '1111', 8)

via ..

->('I', 0, '1111', 10)

->('Seek10', 0, '1111', 9)

->('Got1_Nxt0', 1, '1111', 8)

```
explore_tm(TM01OR10, "0010", 10)
```

Detailing the halted configs now.

Rejected at ('Got0_Nxt1', 1, '0010', 8)

via ..

->('I', 0, '0010', 10)

->('Seek01', 0, '0010', 9)

->('Got0_Nxt1', 1, '0010', 8)

Accepted at ('Found10', 3, '0010', 5)

via ..

->('I', 0, '0010', 10)

->('Seek10', 0, '0010', 9)

->('Seek10', 1, '0010', 8)

->('Seek10', 2, '0010', 7)

->('Got1_Nxt0', 3, '0010', 6)

->('Found10', 3, '0010', 5)

```
#
TM01OR10 = md2mc(''TM
I : 0; 0,S | 1; 1,S -> Seek10
I : 0; 0,S | 1; 1,S -> Seek01

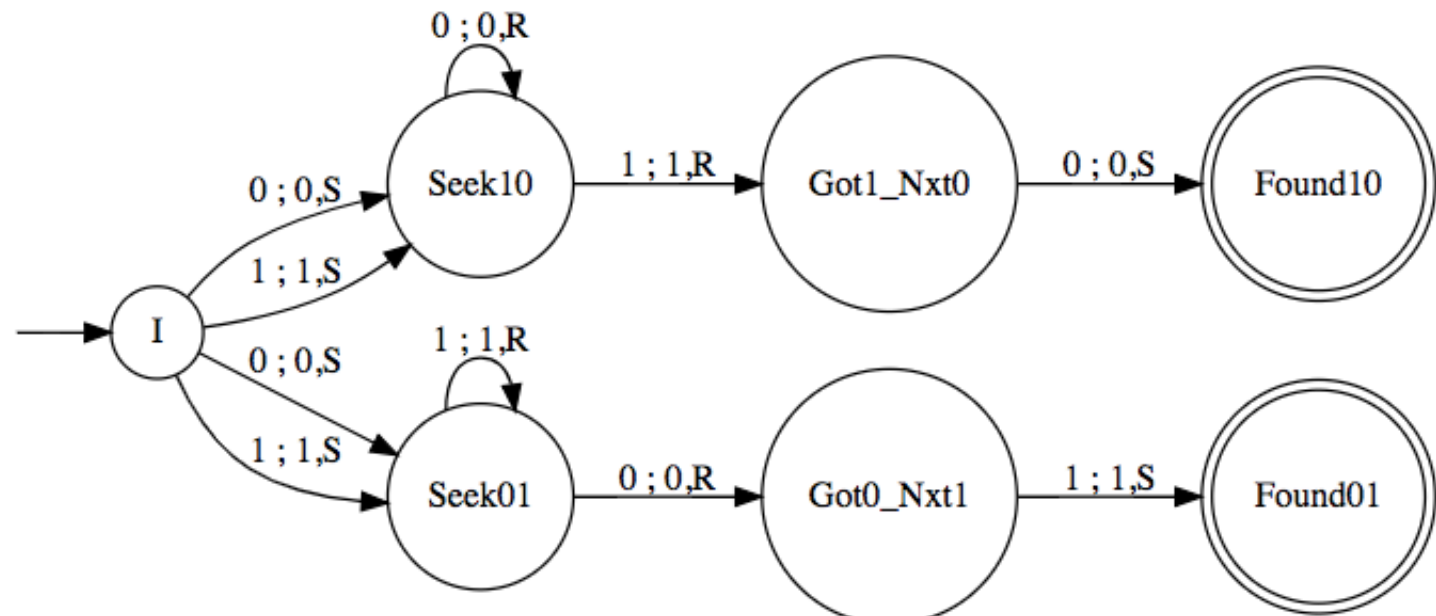
Seek10 : 0;0, R -> Seek10
Seek10 : 1;1, R -> Got1_Nxt0
Got1_Nxt0 : 0;0, S -> Found10

Seek01 : 1;1, R -> Seek01
Seek01 : 0;0, R -> Got0_Nxt1
Got0_Nxt1 : 1;1, S -> Found01

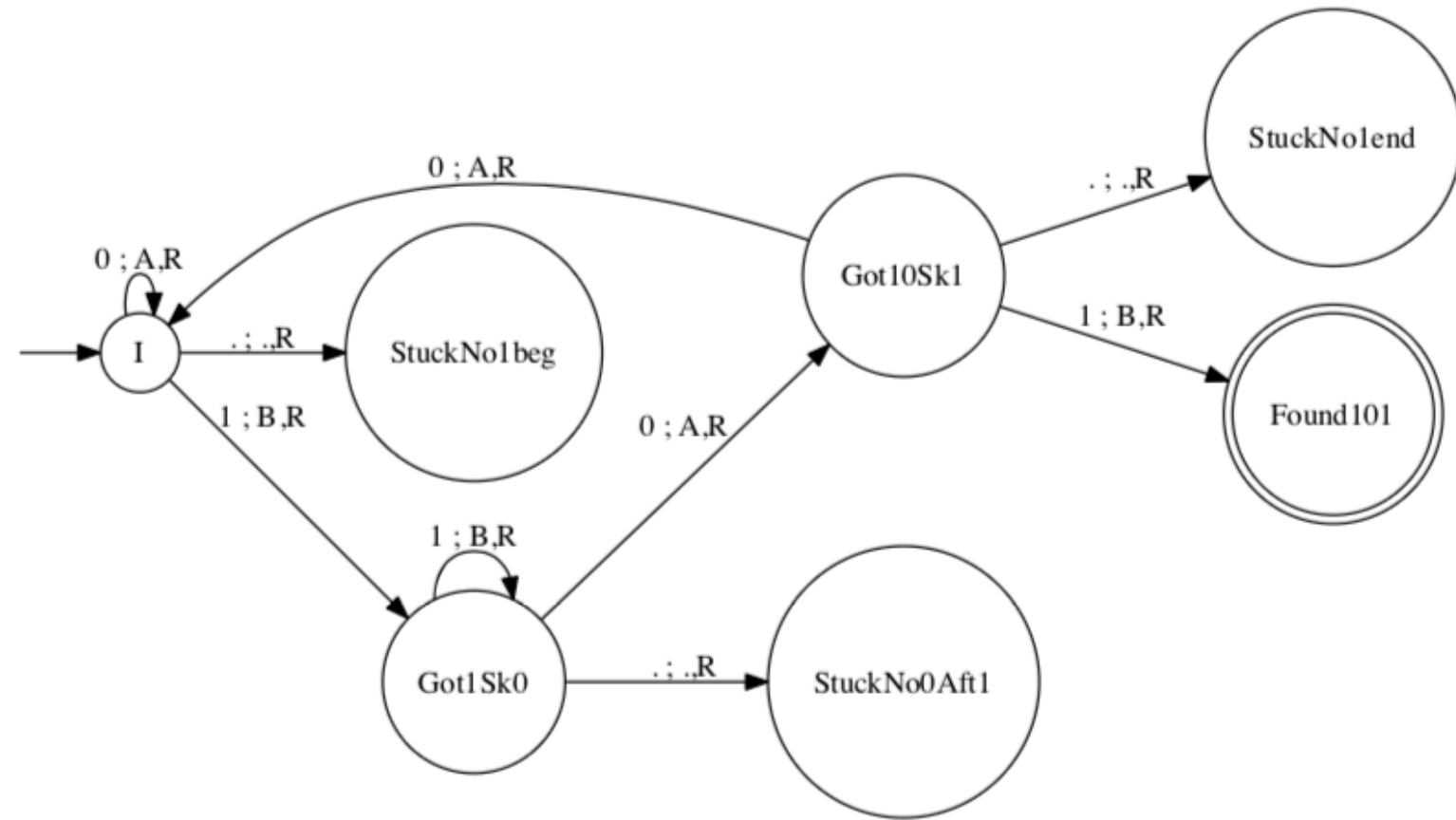
'')
```

```
DO_TM01OR10 = dotObj_tm(TM01OR10)
DO_TM01OR10.render('TM01OR10')
DO_TM01OR10
```

Generating LALR tables



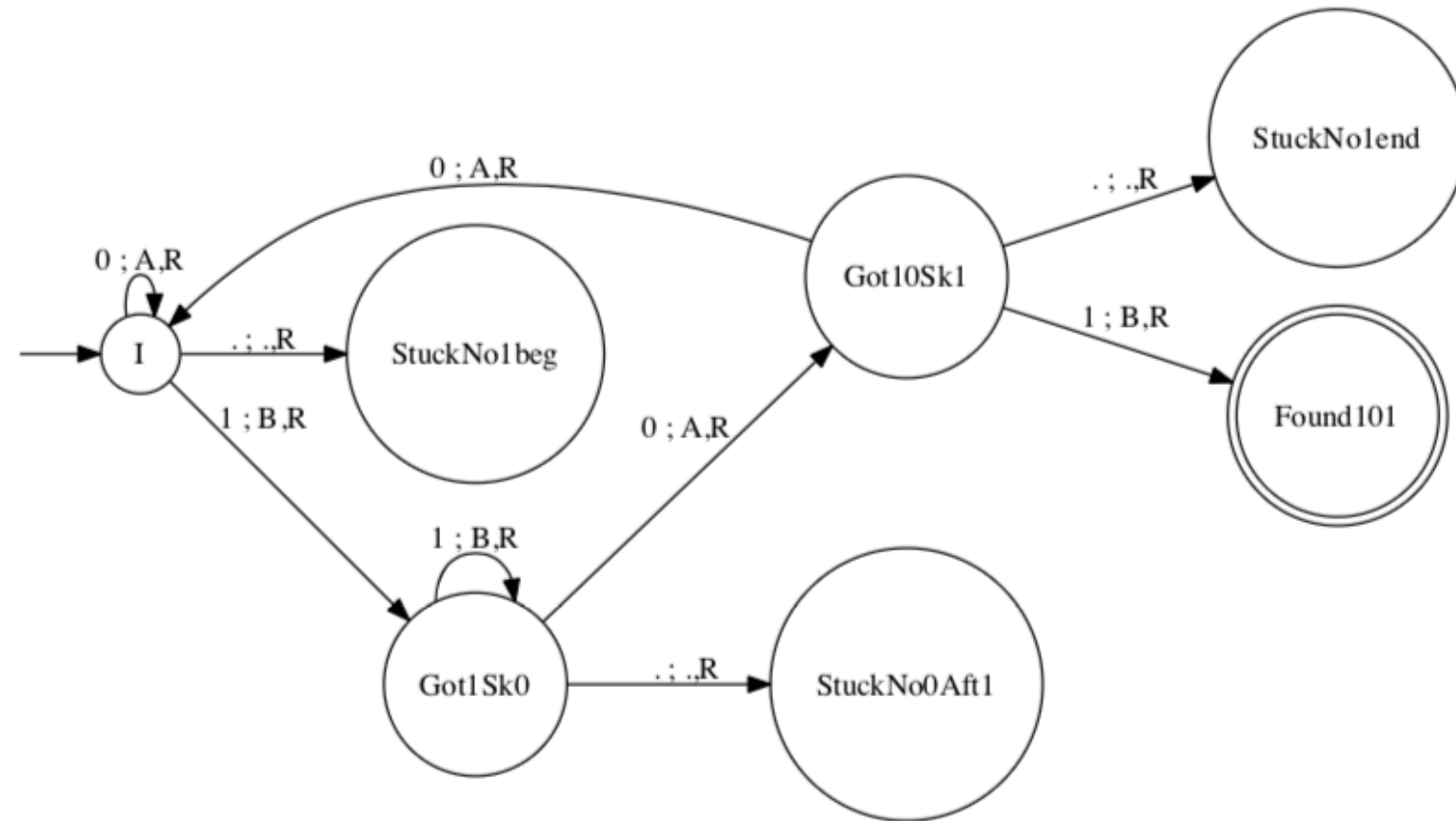
Example-4: DTM to accept inputs with “101” in it



This is what we write against each “arrow” (transition) of a TM :

oneInChrOrBlnk ; oneOutChrOrBlnk , headMoveSpec

Example-4: DTM to accept inputs with “101” in it



Task for you:

Language of this TM?

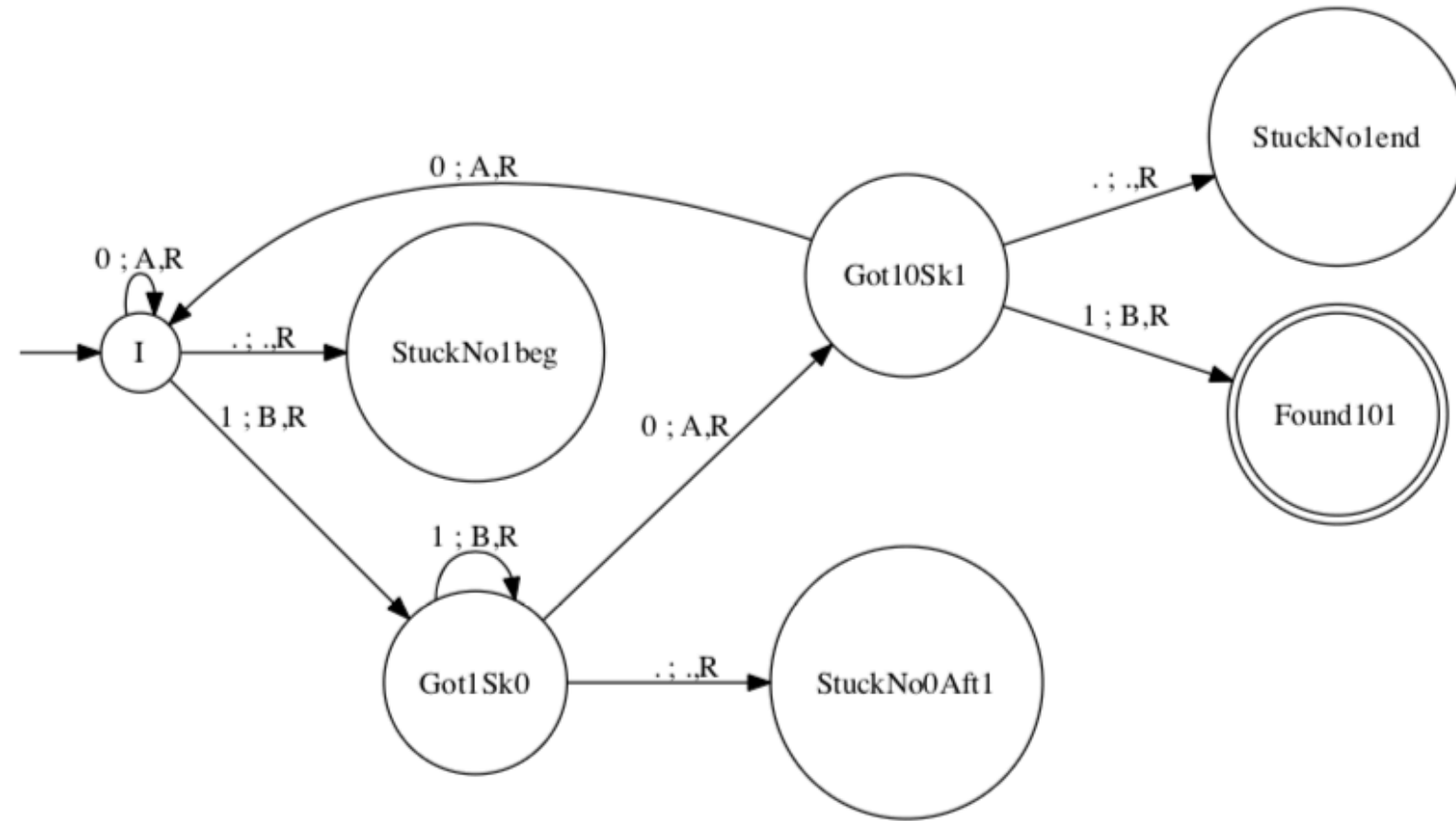
Realizes a procedure or an algorithm?

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

This is what we write against each “arrow” (transition) of a TM :

oneInChrOrBlk ; oneOutChrOrBlk , headMoveSpec

Example-4: DTM to accept inputs with “101” in it



$$\Delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

Task for you:

Specify the mapping for
(Got10Sk1, 0) that yields
(Q, Gamma, {L,R,S})

Answer:

```

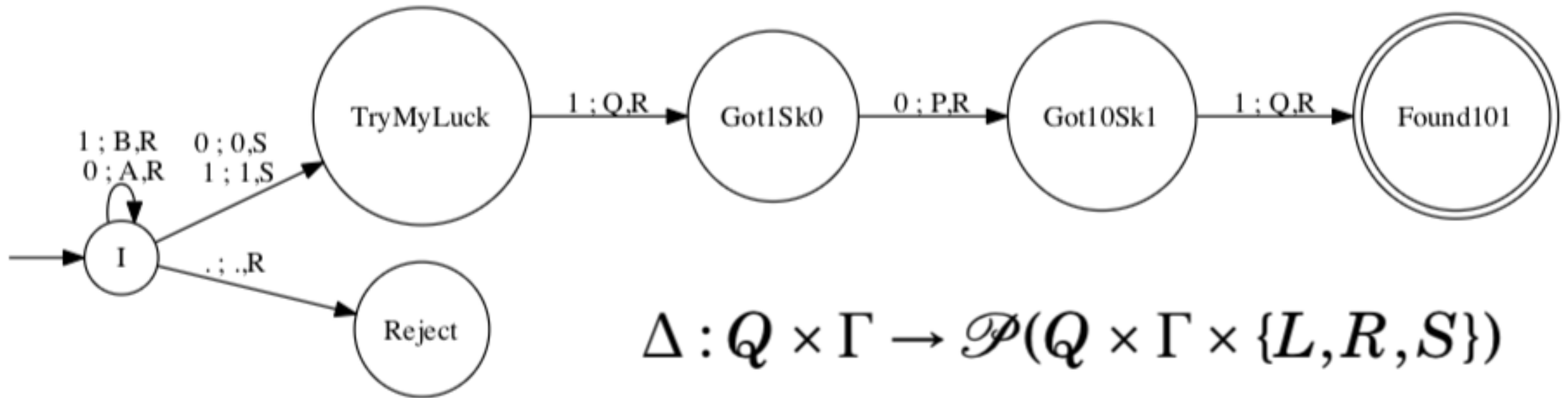
!! Sweep, starting at the beginning, looking for 101
whas101DTM = md2mc(''TM

I : 1; B, R -> Got1Seek0  !! Partial success; climb to next stage
I : 0; A, R -> I          !! Continue hunting for 1
I : .; ., R -> StuckNo1beg !! Ran off end; REJECT

Got1Seek0 : 0; A, R -> Got10Seek1 !! More success; climb onto next state
Got1Seek0 : 1; B, R -> Got1Seek0  !! Didn't find 0, but starts with 1; so Seek0
Got1Seek0 : .; ., R -> StuckNo0Aft1 !! Ran off end w/o finding 0; so REJECT

Got10Seek1 : 1; B, R -> Found101  !! Successfully found 101. ACCEPT!
Got10Seek1 : 0; A, R -> I          !! Failure finding 1; start over
Got10Seek1 : .; ., R -> StuckNo1end !! Ran off end; so REJECT
''')
    
```

Example-5: NDTM to accept inputs with “101” in it



Task for you:

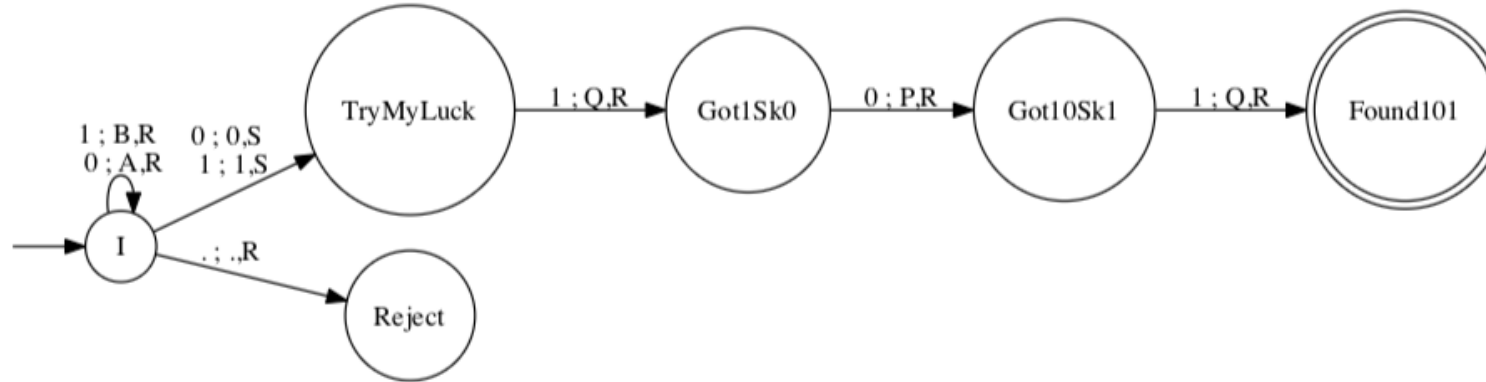
Locate nondeterminism
in the Delta function above.
Express your answer as a set of
(Q, Gamma, {L,R,S})

Answer:

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

Which Machines have ND “more powerful” ?

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 $ LHS \leq RHS $, ignoring ϵ productions.
DTM/NDTM	Recursively Enumerable	General grammars ($ LHS \geq RHS $ allowed).

This one is known to be so (ND more powerful)

This one is open (no one knows if NLBA and DLBA are equivalent)

Figure 13.16: Situation of TMs in the Chomsky Hierarchy.

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

Explain This Halting Configuration

```
In [38]: 1 from jove.AnimateTM import *  
2 AnimateTM(TM1, FuseEdges=True)
```

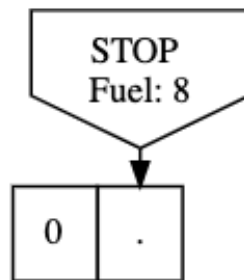
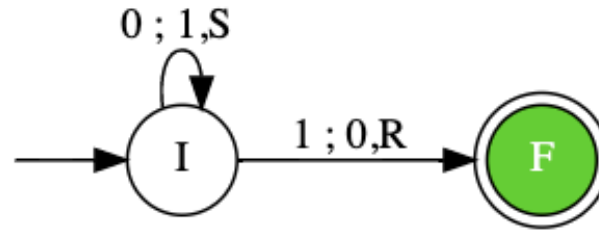
Input:

0

Fuel:

10

Change Input



Path 1: Accepted



Speed:



1

```
Out[38]: <jove.AnimateTM.AnimateTM at 0x111fa4160>
```

```
In [ ]:
```

1

What is a TM (formal version)?

Turing Machines are structures $(Q, \Sigma, \Gamma, \Delta, q_0, B, F)$ where

- Q is a finite non-empty set of states (“program locations or labels”).
- Σ is a finite non-empty **input alphabet**.
- Γ is a finite non-empty **tape alphabet**. Γ is a proper superset of Σ , as we allow the blank tape cell in $\Gamma - \Sigma$, helping to model unwritten parts of the tape.
- $q_0 \in Q$ is the (unique) start state of the Turing machine.
- Generically, “ B ” represents the ‘blank’ tape cell. **In Jove**, we use ‘.’ (period) for blanks, so that they stand out in Jove simulations.
- $F \subseteq Q$ is the set of final (accepting) states.
- Computations are set up by writing the user-given input on the tape.
To feed ε as input to a TM, leave the tape entirely blank.

Transition Functions for DTM and NDTM

DTM: $\Delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$

NDTM: $\Delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$

Feeding inputs to a TM, etc.

We always feed a finitary input to a TM

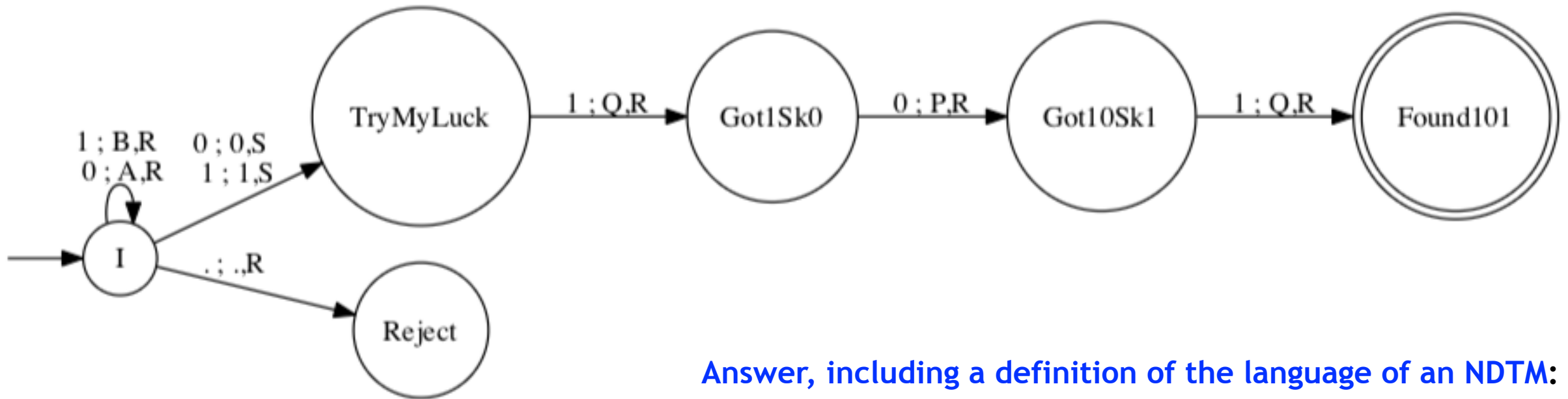
That is, eventually there will be a blank to the RHS

The left-side of the head always has an infinite number of blanks

Key Features of NDTMs

- NDTMs can “take a guess”
- If one of the guesses succeeds, the TM accepts
 - This makes the construction of NDTMs often “easier”

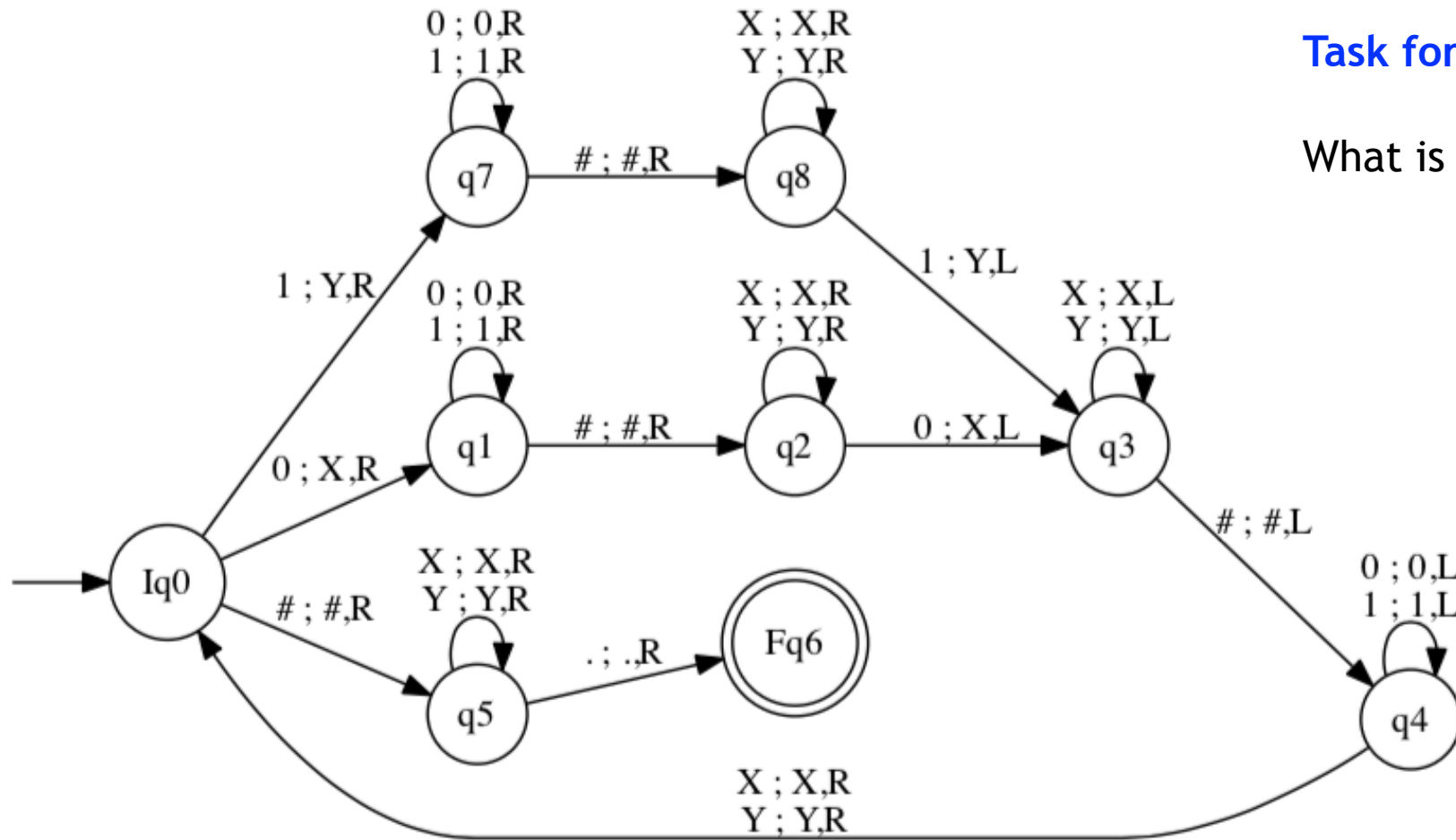
Example-5: NDTM to accept inputs with “101” in it



Answer, including a definition of the language of an NDTM:

Task for you: What is the language of this NDTM ?

Example-6: DTM to accept $w\#w$



Task for you:

What is the language of this TM?

Answer:

The set of strings that
Make the TM “stuck at Fq6”

Which is

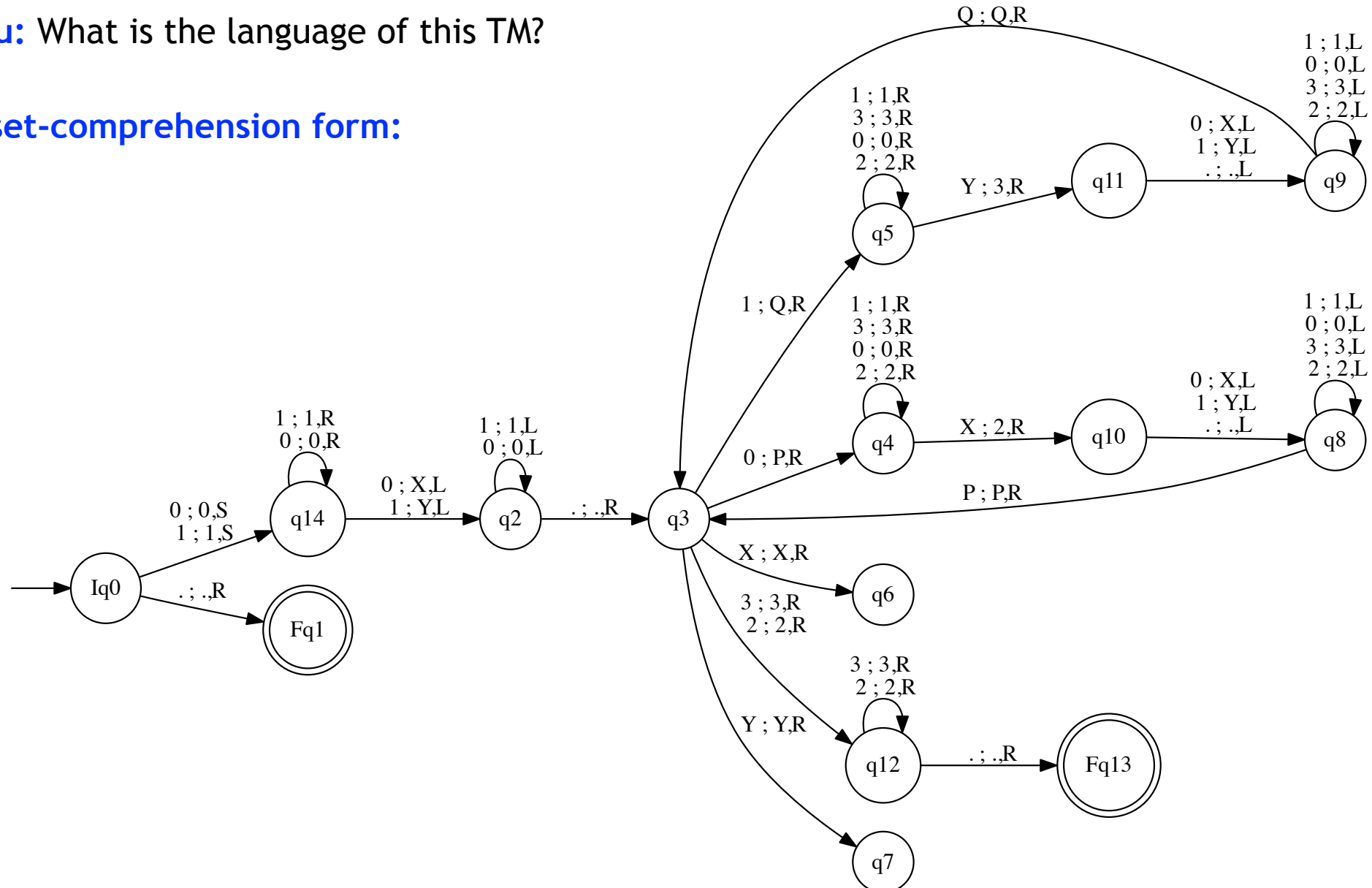
{ ... , ... , ... }

OR in set comprehension:

Example-7: NDTM to accept ww

Task for you: What is the language of this TM?

Answer in set-comprehension form:



Most Important Proof in Chapter 14

- **There is no algorithm to tell whether a given TM realizes a procedure or an algorithm**

Most Important Proof in Chapter 14

- There is no algorithm to tell whether a given TM realizes a procedure or an algorithm
- There are of course procedures that tell whether a given TM realizes a procedure or an algorithm
 - But.... It..... May..... Looooooooooooooooooooooooooooo...

Most Important Proof in Chapter 14

- **There is no algorithm to tell whether a given TM realizes a procedure or an algorithm**

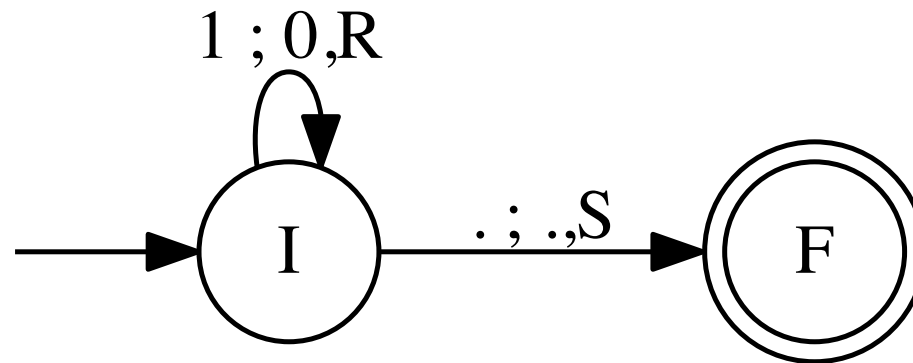
In other words, the “halting problem” is undecidable.

IMPORTANT: Recursive Sets

- The language of a TM that always halts (i.e. the language of a TM realizing an algorithm) is a RECURSIVE SET
- This term is historic in importance

IMPORTANT: Recursive Sets

- The language of this TM is a recursive set

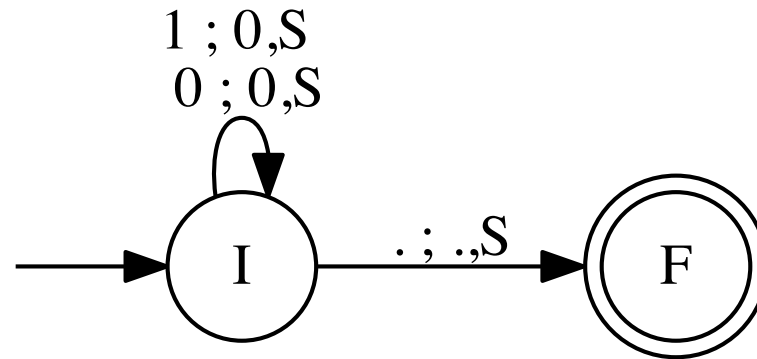


IMPORTANT: Recursively Enumerable Sets

- The language of a TM that may loop (i.e. the language of a TM realizing a PROCEDURE) is a RECURSIVELY ENUMERABLE set
- This term is historic in importance

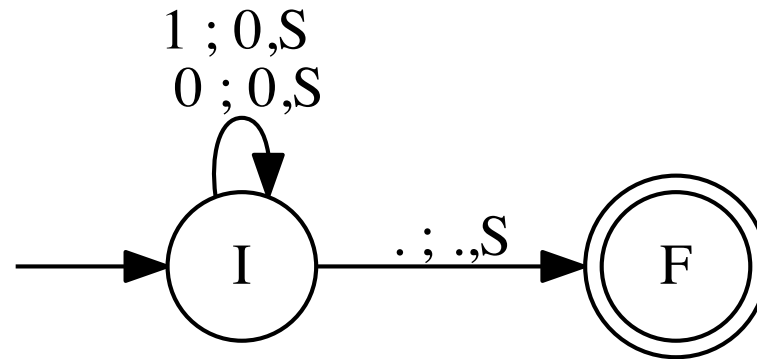
IMPORTANT: RE sets (not 'regexp' but rec. enum)

- The language of this TM is an RE set



IMPORTANT: RE sets (not 'regexp' but rec. enum)

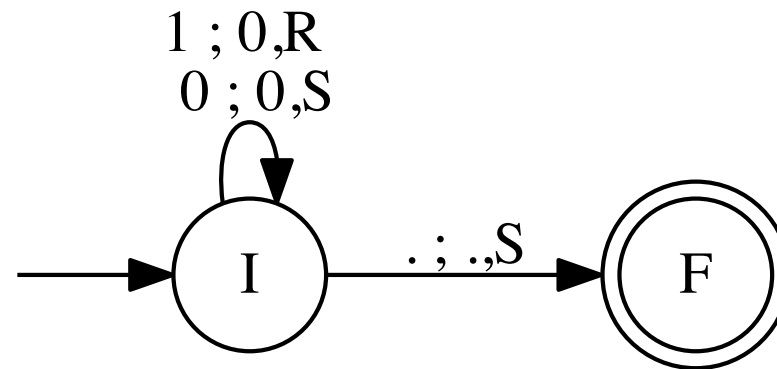
- The language of this TM is an RE set



Question: What is that recursive set?

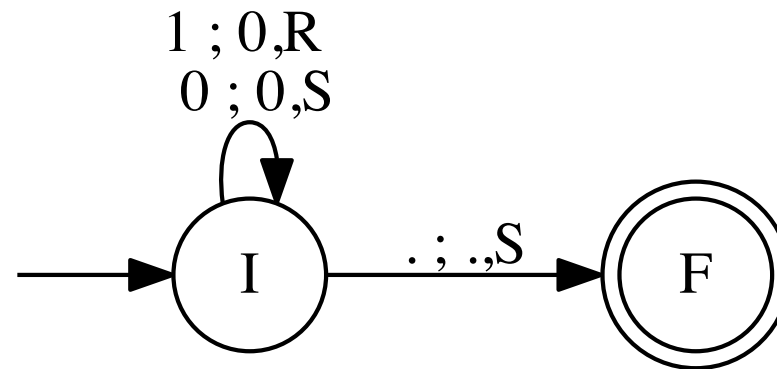
IMPORTANT: RE sets (not 'regexp' but rec. enum)

- The language of this TM is an RE set



IMPORTANT: RE sets (not 'regexp' but rec. enum)

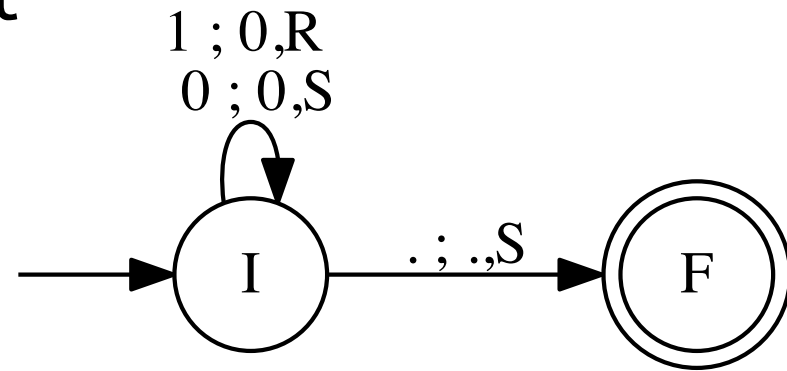
- The language of this TM is an RE set



Question: What is that RE set?

IMPORTANT: RE sets (not 'regexp' but rec. enum)

- The language of this TM is an RE set



Reason for calling them Recursively Enumerable:

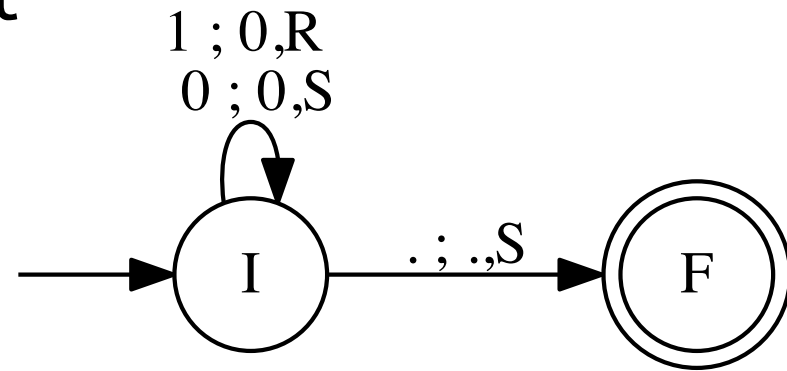
We can enumerate these sets by going by the numeric order over Σ^*

Run the TM on each such string under an increasing amount of fuel

And record in an output set IF and WHEN an input string causes a TM to halt in the accepting state

IMPORTANT: RE sets (not 'regexp' but rec. enum)

- The language of this TM is an RE set



Reason for calling them Recursively Enumerable:

If we do it any other way, we may not discover a string in the RE set

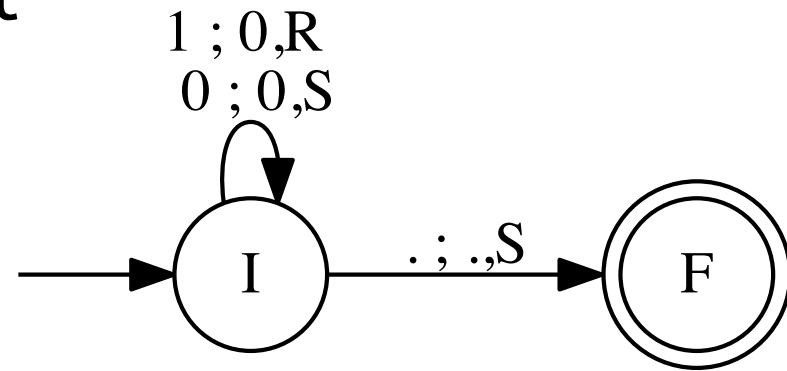
Example: if you run this TM on input 0, the TM loooooooooops

And you don't get to run "1"

Hence you cannot discover that "1" is in the set

IMPORTANT: RE sets (not 'regexp' but rec. enum)

- The language of this TM is an RE set



Reason for calling them Recursively Enumerable:

If we do it any other way, we may not discover a string in the RE set

**Example: if you run this TM on input 0, the TM loooooooops
And you don't get to run "1"
Hence you cannot discover that "1" is in the set**

Instantaneous Description

The ID (instantaneous description, “snapshot”) of a TM is a full description of its state

Knowing the ID of a TM, we can predict its next ID (for a DTM)

Knowing the ID of a TM, we can predict the set of next possible IDs (for an NDTM)

The Instantaneous Description (ID) of a TM

Jove-style

(control state, head-index
That starts at 0, Full tape,
"remaining fuel")

('I', 0, '010011', 100)

('I', 1, '110011', 99)


...

('I', 6, '101100', 94)

('F', 6, '101100...', 93)


Mathematical-style

(shows control state + what
the head is looking at).
Suppresses "remaining fuel"

 I 0 1 0 0 1 1

 1 I 1 0 0 1 1

 1 0 1 1 0 I 0

 1 0 1 1 0 0 F

The Instantaneous Description (ID) of a TM

Jove-style

(control state, head-index
That starts at 0, Full tape,
"remaining fuel")

('I', 0, '010011', 100)

('I', 1, '110011', 99)


...

('I', 6, '101100', 94)


('F', 6, '101100...', 93)


Mathematical-style

(shows control state + what
the head is looking at).
Suppresses "remaining fuel"

I010011

1I10011

10110I0

101100F

The arrow points to
the symbol that the
"TM head" is looking at.

The origin of the arrow
signifies the current
control state
of the TM

The Instantaneous Description (ID) of a TM

Jove-style

(control state, head-index
That starts at 0, Full tape,
"remaining fuel")

('I', 0, '010011', 100)

('I', 1, '110011', 99)


...

('I', 6, '101100', 94)


('F', 6, '101100...', 93)


Mathematical-style

(shows control state + what
the head is looking at).
Suppresses "remaining fuel"

 *I*010011

 1*I*10011

 10110*I*0

 101100*F*

What is the "head"
looking at now?

Answer:

The Instantaneous Description (ID) of a TM

Jove-style

(control state, head-index
That starts at 0, Full tape,
"remaining fuel")

('I', 0, '010011', 100)

('I', 1, '110011', 99)


...

('I', 6, '101100', 94)


('F', 6, '101100...', 93)

Mathematical-style

(shows control state + what
the head is looking at).
Suppresses "remaining fuel"


*I*010011


1*I*10011


10110*I*0

101100*F*

We shall use
the mathematical
kind when doing
Proofs.

I'll remind you of the
Jove kind when necessary.

The Language of a TM: Mathematical IDs

The language of a TM (DTM or NDTM) M is the set of strings w such that the ID

$q_0 w$

evolves into this ID

$w' f w''$

The Language of a TM: Jove IDs

The language of a TM (DTM or NDTM) M is the set of strings w such that

When M is started from Jove-style ID

$(q_0, 0, w, \text{Fuel})$ where q_0 is the starting state

The TM runs for a while, and is found "stuck" (halted) at some ID of the form

(f, h, w', Fuel') for some f in F