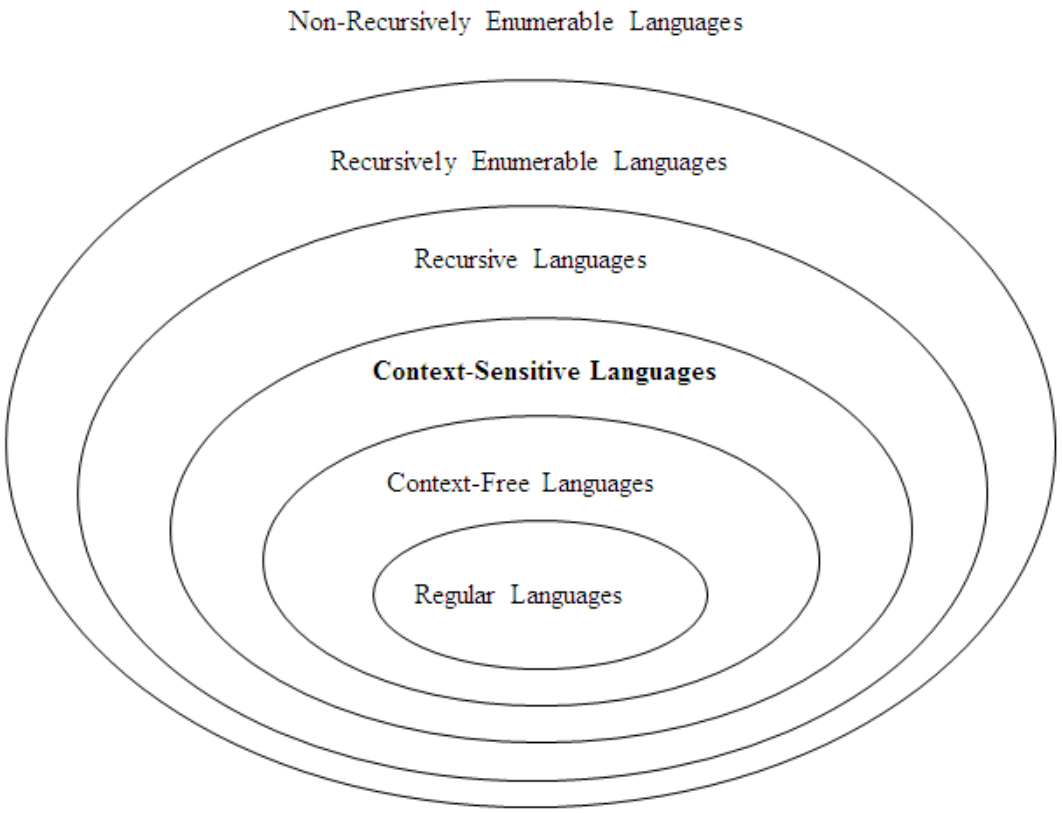
Turing Machines

Recursively Enumerable and Recursive Languages

- **The Hierarchy of Languages:**



- Recursively enumerable languages are also known as *type 0* languages.
- Context-sensitive languages are also known as *type 1* languages.
- Context-free languages are also known as *type 2* languages.
- Regular languages are also known as *type 3* languages.

- TMs model the computing capability of a general purpose computer, which informally can be described as:
  - Effective procedure
    - Finitely describable
    - Well defined, discrete, “mechanical” steps
    - Always terminates
  - Computable function
    - A function computable by an effective procedure

- TMs model the computing capability of a general purpose computer, which informally can be described as:
  - Effective procedure
    - Finitely describable
    - Well defined, discrete, “mechanical” steps
    - Always terminates
  - Computable function
    - A function computable by an effective procedure

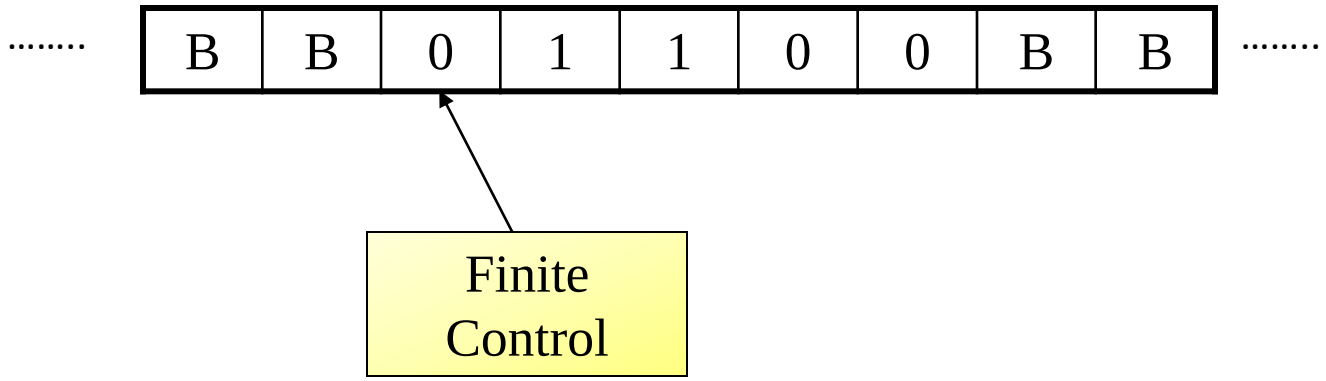
- TMs formalize the above notion.

- **Church-Turing Thesis:** There is an effective procedure for solving a problem if and only if there is a TM that halts for all inputs and solves the problem.
  - There are many other computing models, but all are equivalent to or subsumed by TMs. *There is no more powerful machine* (Technically cannot be proved).

- DFAs and PDAs do not model all effective procedures or computable functions, but only a subset.

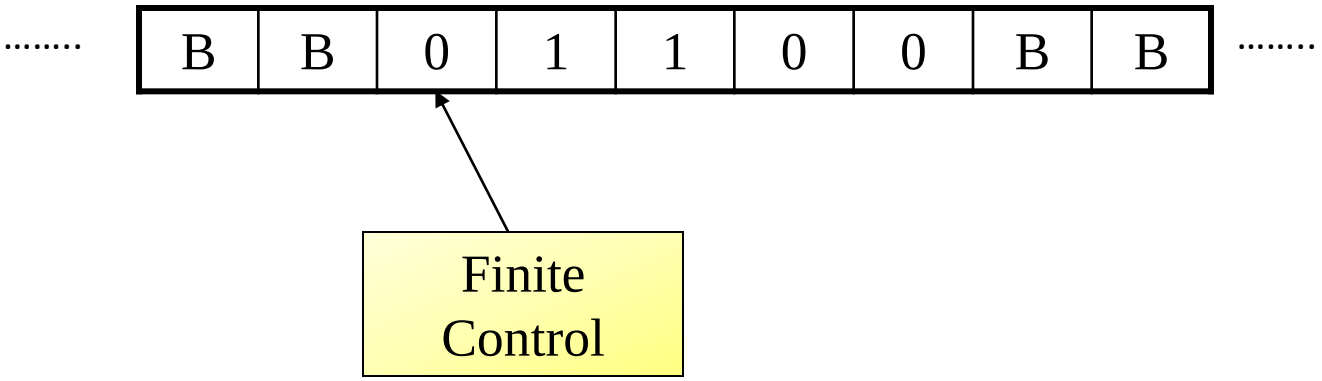

Universal
Turing Machine
I fear no man.
But that thing...
Halting
Problem
it scares me.

Deterministic Turing Machine (DTM)



- Two-way, infinite tape, broken into cells, each containing one symbol.
- Two-way, read/write tape head.
- An input string is placed on the tape, padded to the left and right infinitely with blanks, read/write head is positioned at the left most input character.
- Finite control (read/write head is part of this control), knows current symbol being scanned, and its current state.

Deterministic Turing Machine (DTM)



- In one move, depending on the current state and the current symbol being scanned, the TM does: (1) changes state, (2) prints a symbol over the cell being scanned, and (3) moves its' tape head one cell left or right.
- Many modifications possible, but Church-Turing declares equivalence of all.

Formal Definition of a DTM

- **A DTM is a seven-tuple:**

  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

  Q A finite set of states
  Σ A finite input alphabet, which is a subset of Γ– {B}
  Γ A finite tape alphabet, which is a strict superset of Σ
  B A distinguished blank symbol, which is in Γ
  q_0 The initial/starting state, q_0 is in Q
  F A set of final/accepting states, which is a subset of Q
  δ A next-move function, which is a *mapping* (i.e., may be undefined)
  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

  Intuitively, $\delta(q,s)$ specifies the next state, symbol to be written, and the direction of tape head movement by M after reading symbol s while in state q.

- **Example #1:** {w | w is in {0,1}* and w ends with a 0}

  = {0, 00, 10, 10110, …}

  Note: ε is not in the language

- **Example #1:** {w | w is in {0,1}* and w ends with a 0}

= {0, 00, 10, 10110, …}

Note: ε is not in the language

Q = $\{q_0, q_1, q_2\}$

Γ = {0, 1, B}

Σ = {0, 1}

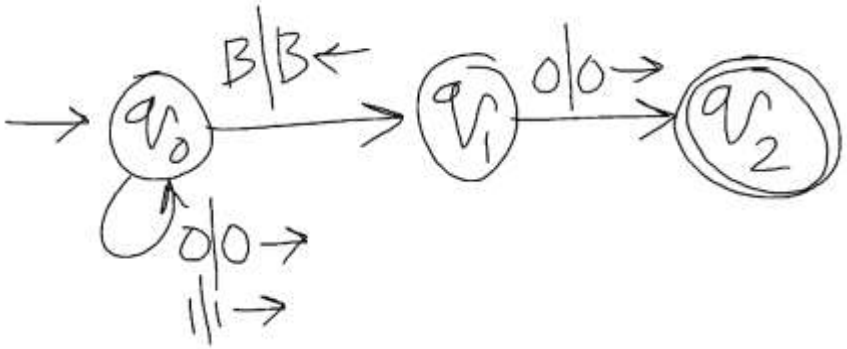
F = $\{q_2\}$

δ:

	0	1	B
->q_0	**(q_0, 0, R)**	**(q_0, 1, R)**	**(q_1, B, L)**
q_1	**(q_2, 0, R)**	**-**	**-**
q_2*	**-**	**-**	**-**

- **q_0 is the start state and the "scan right" state, until hits B**
- **q_1 state is to begin the verification – last character is 0 or not**
- **q_2 is the final state**

- **Example #1:** {w | w is in {0,1}* and w ends with a 0}



Q = {q_0, q_1, q_2}

Γ = {0, 1, B}

Σ = {0, 1}

F = {q_2}

δ:

	0	1	B
->q_0	(q_0, 0, R)	(q_0, 1, R)	(q_1, B, L)
q_1	(q_2, 0, R)	-	-
q_2*	-	-	-

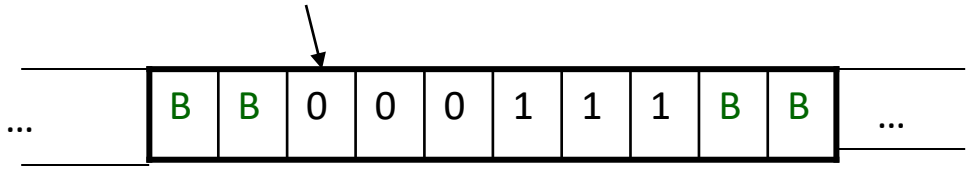
- **q_0 is the start state and the "scan right" state, until hits B**
- **q_1 state is to begin the verification – last character is 0 or not**
- **q_2 is the final state**

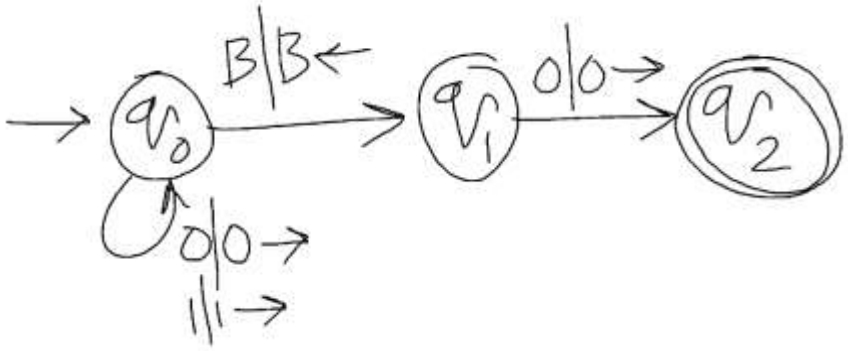
ID

- To describe the steps we use Instantaneous Descriptions.
- An ID is $\alpha q \beta$
- $\alpha, \beta \in \Gamma^*$
- $\alpha\beta$ is on the tape. Left and right of this are blanks.
- Head is on the first character of β
- The TM is in state q

Step

- $id_1 \vdash id_2$ describes one step
- $\vdash^*$ is reflexive and transitive closure of $\vdash$





- For input 1010, computation steps…

$q_0 1010 \vdash 1 q_0 010 \vdash 10 q_0 10$

$\vdash 101 q_0 0 \vdash 1010 q_0 B$

$\vdash 101 q_1 0 \vdash 1010 q_2$

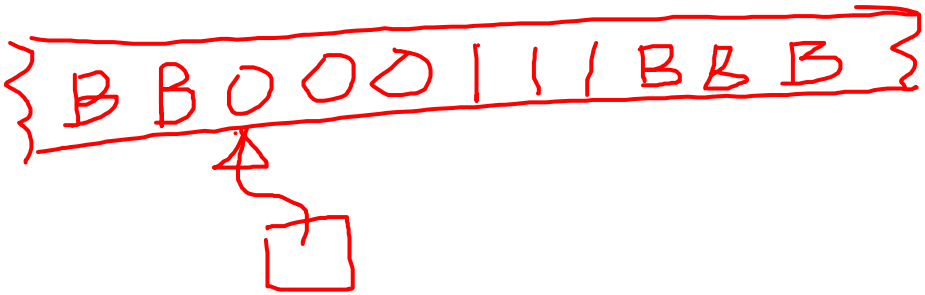
Have you noticed

- Recognition is immediate.
  - TM, once hits one of the final states, it accepts and halts.
- No need for the input to be exhausted !!

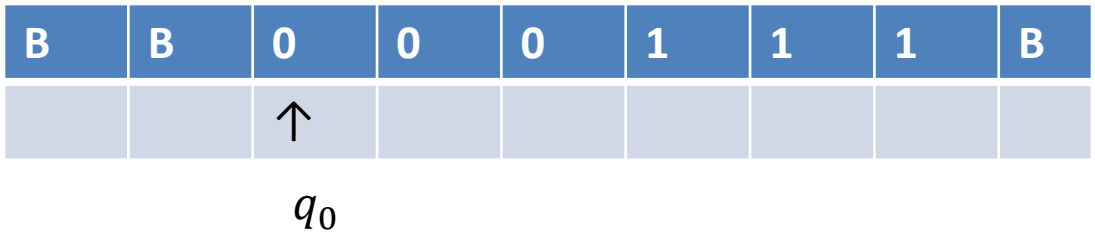
- How TM rejects?
  - It gets stuck.
  - It goes on infinitely without ever hitting a final state.

- $\{0^n 1^n | n \geq 1\}$
- Idea??

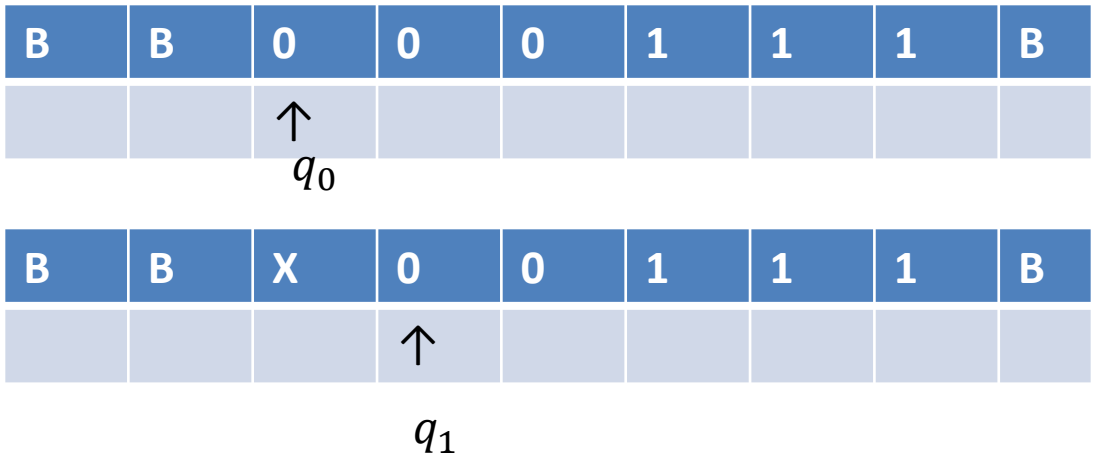
- $\{0^n 1^n | n \geq 1\}$
- Idea??



- $\{0^n 1^n | n \geq 1\}$
- Idea??



- $\{0^n 1^n | n \geq 1\}$
- Idea??

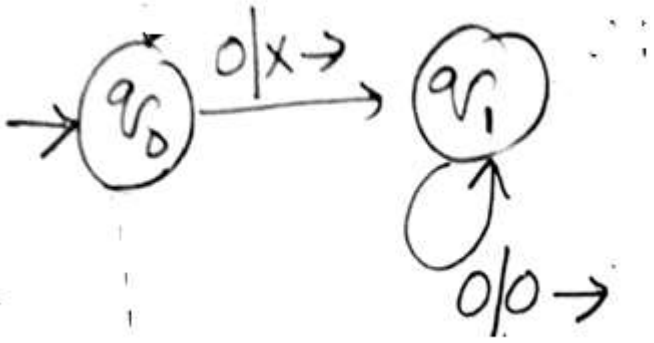


- We turn

0 in to X and

probe for 1.

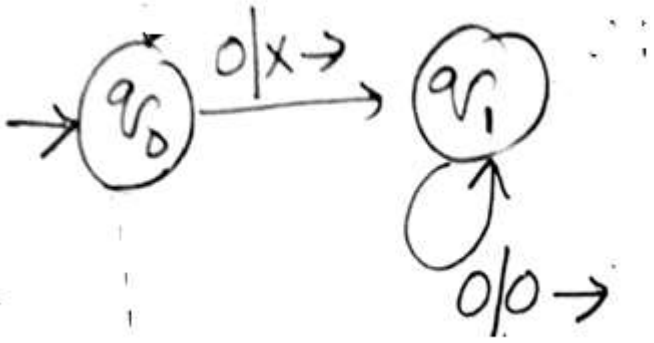
We turn 1 to Y.



B	B	0	0	0	1	1	1	B
		↑						
		q_0						

B	B	X	0	0	1	1	1	B
			↑					
			q_1					

B	B	X	0	0	1	1	1	B
					↑			
					q_1			



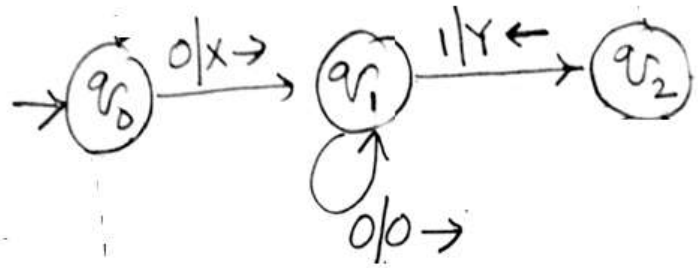
B	B	0	0	0	1	1	1	B
		↑						
		q_0						

B	B	X	0	0	1	1	1	B
			↑					
			q_1					

⋮

B	B	X	0	0	1	1	1	B
					↑			
					q_1			

B	B	X	0	0	Y	1	1	B
				↑				
				q_2				



B	B	0	0	0	1	1	1	B
		↑						
		q_0						

B	B	X	0	0	1	1	1	B
			↑					
			q_1					

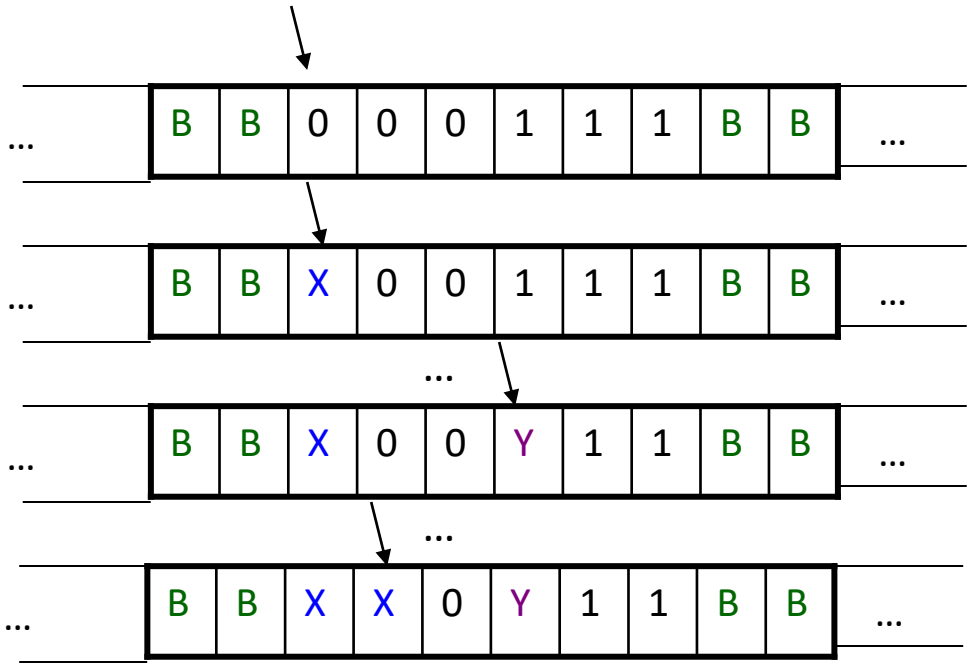
⋮

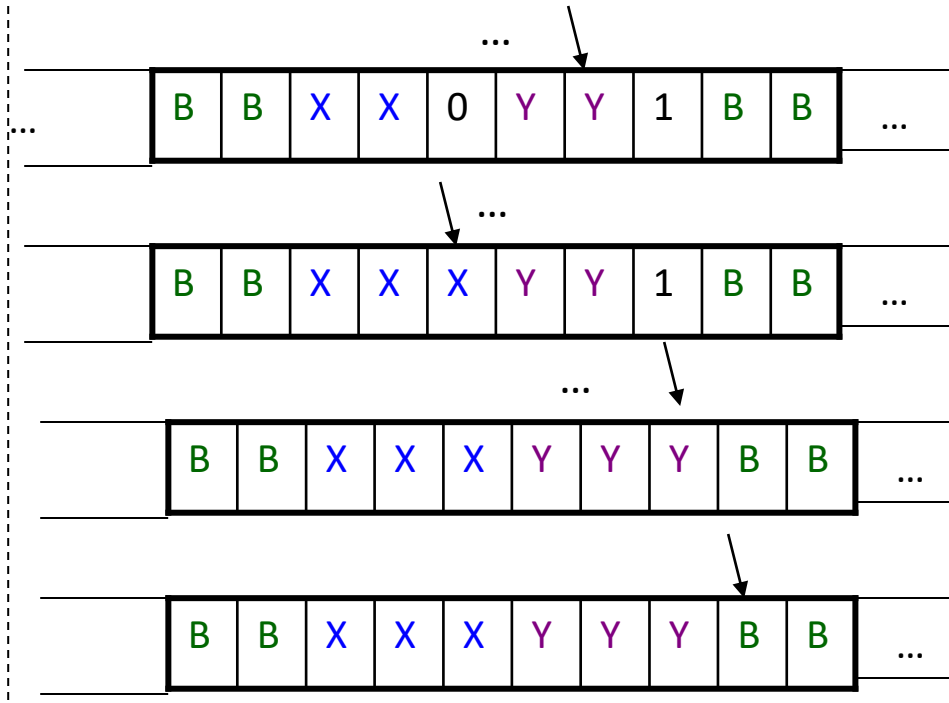
B	B	X	0	0	1	1	1	B
					↑			
					q_1			

B	B	X	0	0	Y	1	1	B
				↑				
				q_2				

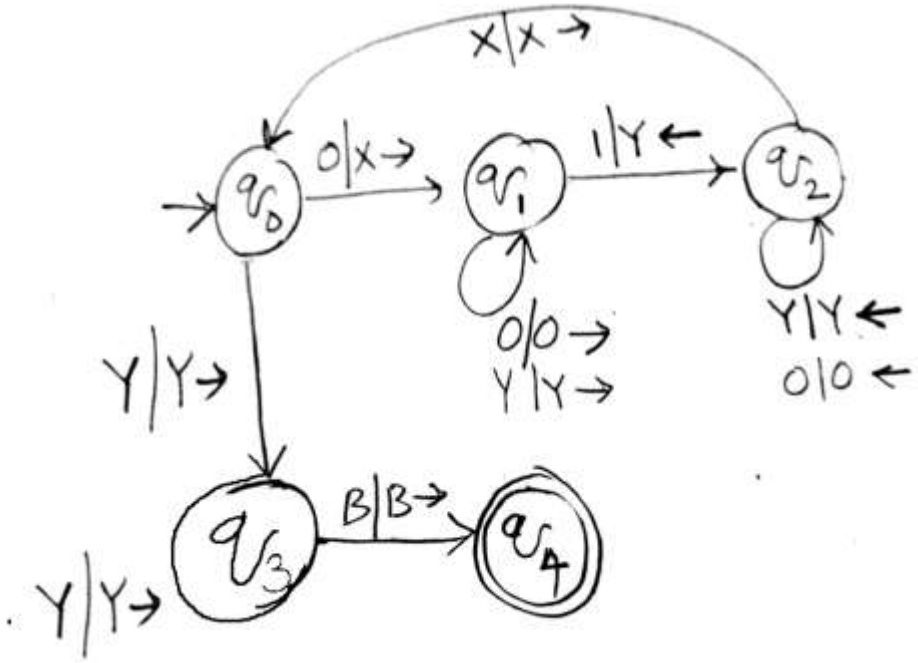
Example: L = {0^n1^n | n≥1}

- Strategy: w = 000111





Accept

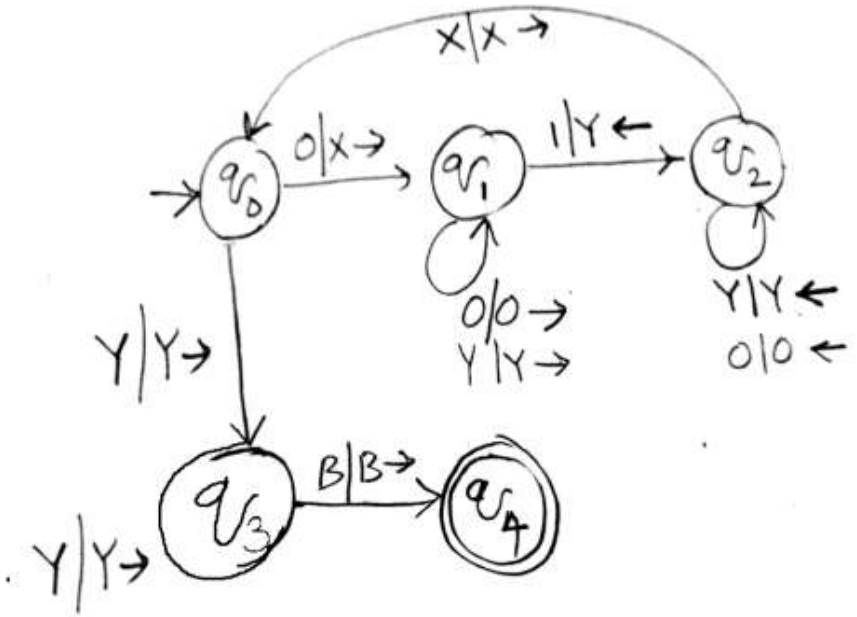

X|X →
0|X →
1|Y ←
q_0
q_1
q_2
0|0 →
Y|Y →
Y|Y ←
0|0 ←
Y|Y →
B|B →
q_3
q_4
Y|Y →

- **Example #2:** $\{0^n1^n \mid n \geq 1\}$

	0	1	X	Y	B
->q_0	(q_1, X, R)	-	-	(q_3, Y, R)*0's finished*	-
q_1	(q_1, 0, R)*ignore1*	(q_2, Y, L)	-	(q_1, Y, R) *ignore2*	- [more 0's]
q_2	(q_2, 0, L) *ignore2*	-	(q_0, X, R)	(q_2, Y, L) *ignore1*	-
q_3	-	- [more 1's]	-	(q_3, Y, R) *ignore*	(q_4, B, R)
q_4*	-	-	-	-	-

- **Sample Computation:** (on input: 0011),

$q_0$0011BB.. |— X$q_1$011
|— X0$q_1$11
|— X$q_2$0Y1
|— q_2X0Y1
|— X$q_0$0Y1
|— XXq_1Y1
|— XXY$q_1$1
|— XXq_2YY
|— Xq_2XYY
|— XXq_0YY
|— XXYq_3Y B…
|— XXYYq_3 BB…
|— XXYYBq_4



TMs for calculations

- TMs can also be used for calculating values
  - Like arithmetic computations
  - Eg., addition, subtraction, multiplication, etc.

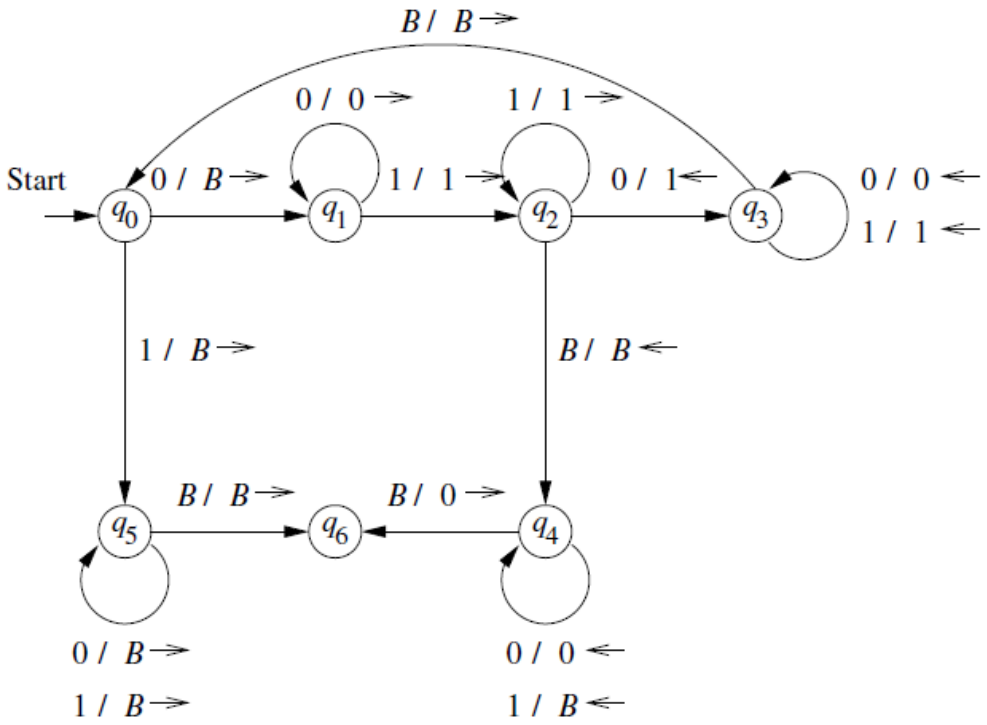
Example 2: monus subtraction

"m -- n" = max{m-n,0}

0^m10^n ➔ ...B 0^{m-n} B.. (*if m>n*)
...BB...B.. (*otherwise*)

1. For every 0 on the left (mark B), mark off a 0 on the right (mark 1)
2. Repeat process, until one of the following happens:
    1. // No more 0s remaining on the left of 1
       Answer is 0, so flip all excess 0s on the right of 1 to Bs (and the 1 itself) and halt
    2. //No more 0s remaining on the right of 1
       Answer is m-n, so simply halt after making 1 to B

Give state diagram


B / B →
0 / 0 →
1 / 1 →
Start
0 / B →
1 / 1 →
0 / 1 ←
0 / 0 ←
1 / 1 ←
q0
q1
q2
q3
1 / B →
B / B ←
B / B →
B / 0 →
q5
q6
q4
0 / B →
1 / B →
0 / 0 ←
1 / B ←

Turing Machines--
Machines entering into infinite loop

Supplement Slides

Recursively enumerable

- If $w \in L(M)$, then M accepts/recognizes the string w.
- If, $w \notin L(M)$, then M may or may not halt.
  - No transition means M halts and rejects.
- We say $L(M)$ for a given TM M is recursively enumerable (RE).

Recursive languages

- Recursive languages (R) is a subset of RE.
- We say $L(M)$ for a TM M is recursive, if for any given input string w, M halts.
- That is, if $w \in L(M)$, M halts in an accepting (final) state.
- Else, M halts in a non-final state.
  - i.e., It gets stuck in a non-final state.
- M never goes in to an infinite loop.

RE Vs. R

RE

- A TM M recognizes.

R

- A TM M decides.

RE Vs. R

RE

- A TM M recognizes.
- If $L \in RE - R$, then complement of L, that is $\bar{L} \notin RE$.

R

- A TM M decides.
- $L \in R \Leftrightarrow \bar{L} \in R$

RE Vs. R

RE

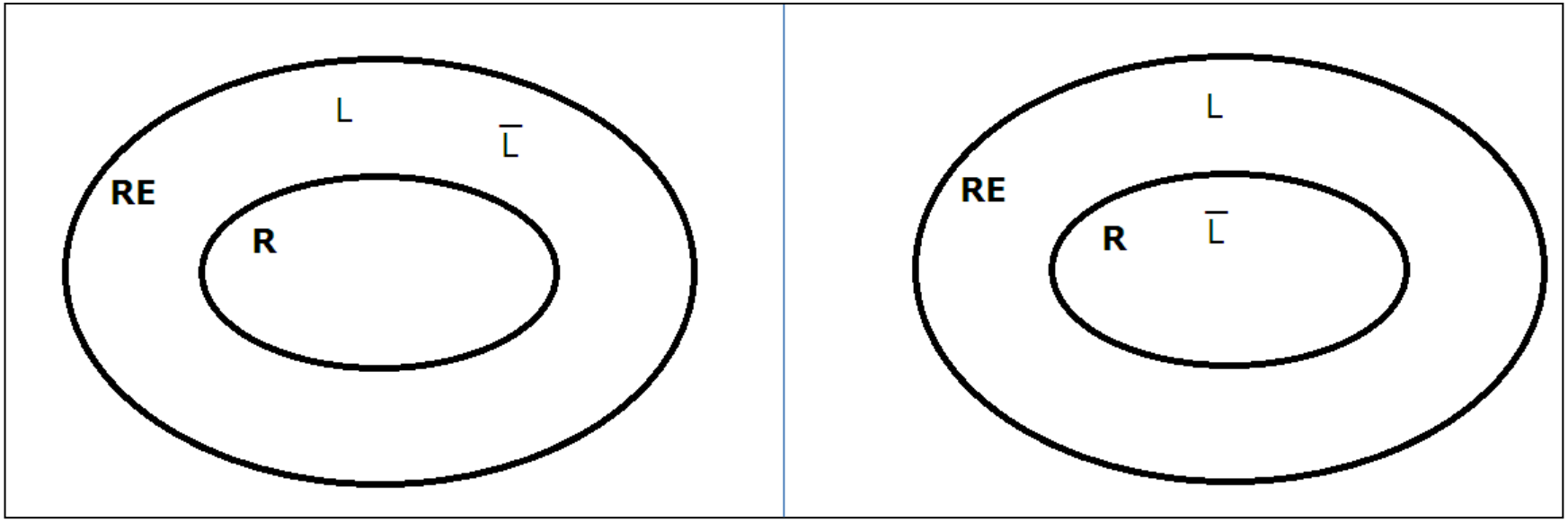
- A TM M recognizes.
- If $L \in RE - R$, then complement of L, that is $\bar{L} \notin RE$.

R

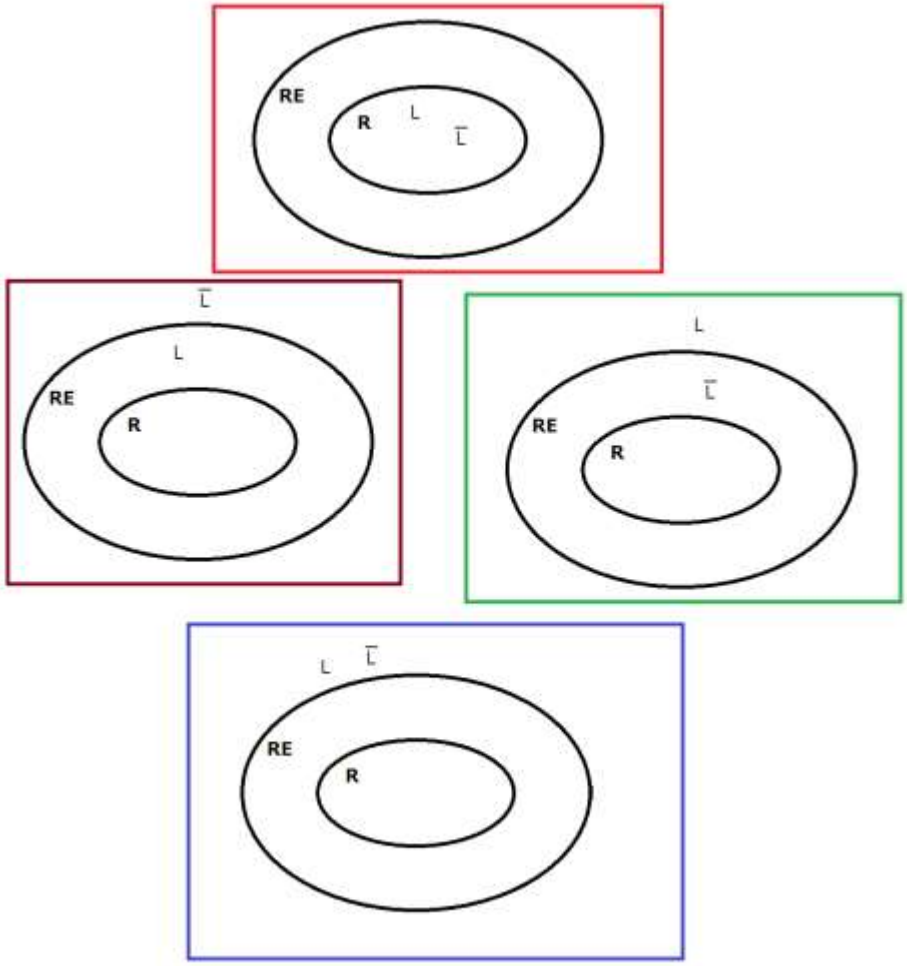
- A TM M decides.
- $L \in R \Leftrightarrow \bar{L} \in R$

$$L \in RE \text{ and } \bar{L} \in RE \quad \Rightarrow \quad L \in R$$

Not Possible



Possible



Recognizing or accepting a language by a machine (automaton)

- Let us define ID for a DFA/NFA as
  - (q, x) where q is the current state and x is the remaining input.
- We say L is recognized/accepted by a machine M (may be a DFA/NFA/PDA/TM)

$$L = \{w \in \Sigma^* | id_0 \vdash^* id_f \text{ where } id_0 \text{ and } id_f \text{ are initial and accepting ids, respectively}\}$$

DFA/NFA

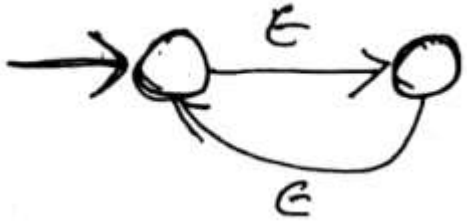
- $L = \{w \in \Sigma^* | \, (\, q_0, w \,) \, \vdash^* \left(q_f, \epsilon\right) \text{where } q_f \text{ is a final state}\}$
- Input has to be exhausted. { The remaining string should become ϵ }
- This is the same criterion even for NFAs.

DFA

- A DFA **never** enters in to an infinite loop.
  - Since there are no ϵ transitions, and
  - There is exactly one choice at every stage of the computation.
  - The input has to exhaust progressively (one character at each step) and should become ϵ
    - At this stage if the state is one of final, the input string is accepted, else rejected.

NFA

- A NFA can enter in to an infinite loop.



- For any given input this NFA enters in to an infinite loop.
- The language recognized by this machine is ϕ
- The language ϕ is regular, because there is a NFA to recognize this.

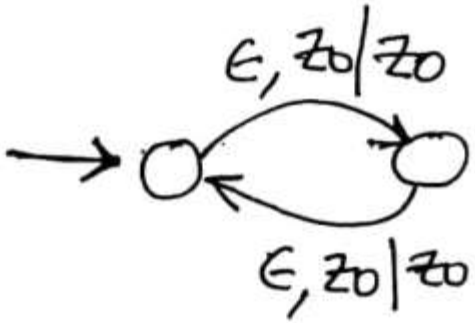
For this language we can find NFA and DFA that always halts.

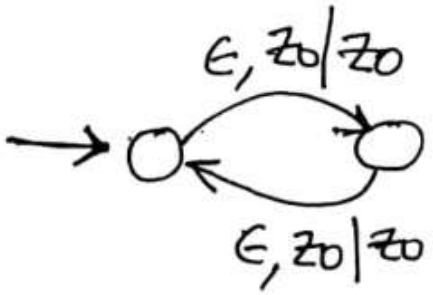
- NFA

- There is a DFA (existence of either NFA or DFA is enough) also to accept ϕ. {DFA always halts}.

PDA by final state and empty stack

- For a PDA to recognize a language by final state, $L = \{w \in \Sigma^* | \, (\, q_0, w, Z_0 \,) \, \vdash^* \, (q_f, \epsilon, \alpha)$ where q_f is a final state and $\alpha \in \Gamma^*\}$
- A PDA can also enter in to an infinite loop





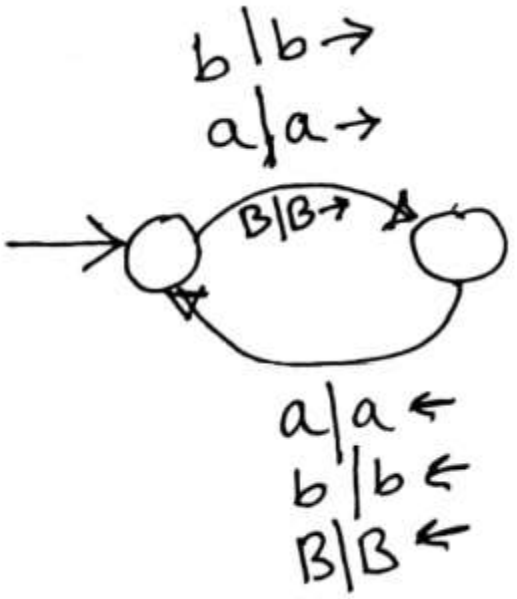
- The language accepted by this PDA is also ϕ.
- we can find a PDA which recognizes the language ϕ without entering in to an infinite loop.

- Have you noted, both these PDAs are indeed DPDAs.

- For PDA by empty stack also similar arguments can be given.

TM

- For the given DTM



also the language

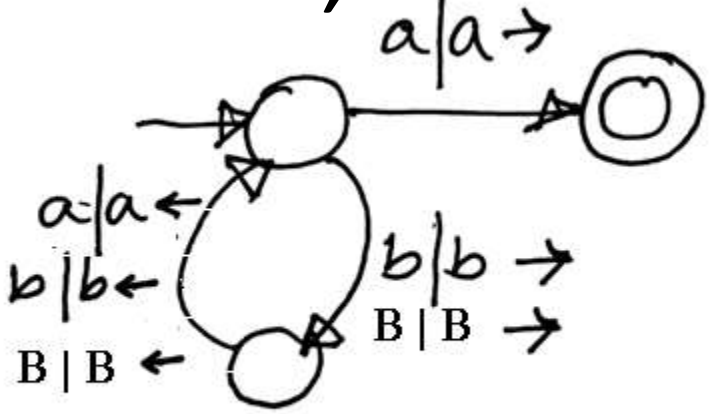
recognized is ϕ.

- Again there is a DTM that

Recognizes this language without entering in to infinite loop, which is

- So, $\phi \in RE$
- In fact $\phi \in R$ also.

To elucidate further,



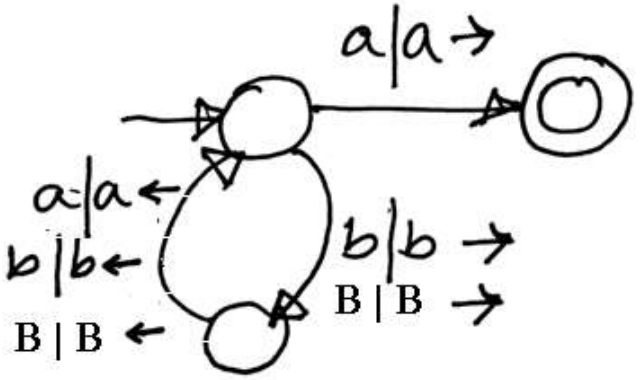
- The given DTM

recognizes the language

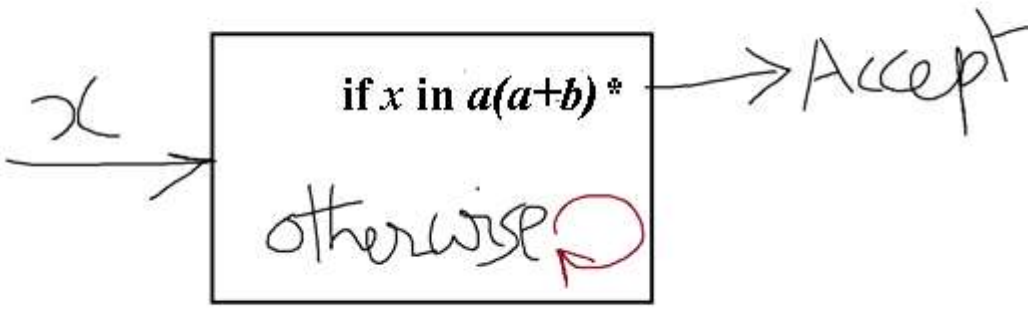
$a(a+b)^*$, but enters in to an

infinite loop for all other inputs.

- So, the language recognized by this DTM is $a(a+b)^*$
- This is in RE.



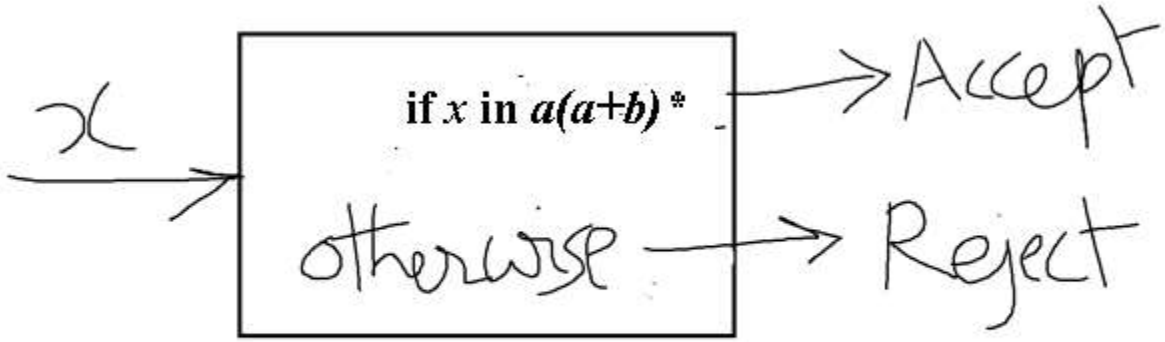
- Behaviour of this machine is



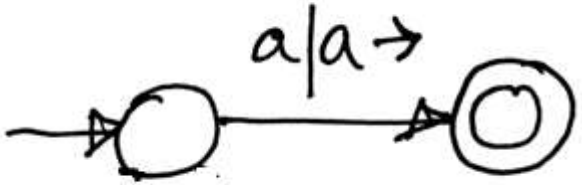
- so, the language $a(a+b)^* \in RE$

Can we say $a(a+b)^*$ is in R ?

- Yes.



- We need a TM which behaves like above



- Note, existence of one such TM is enough.

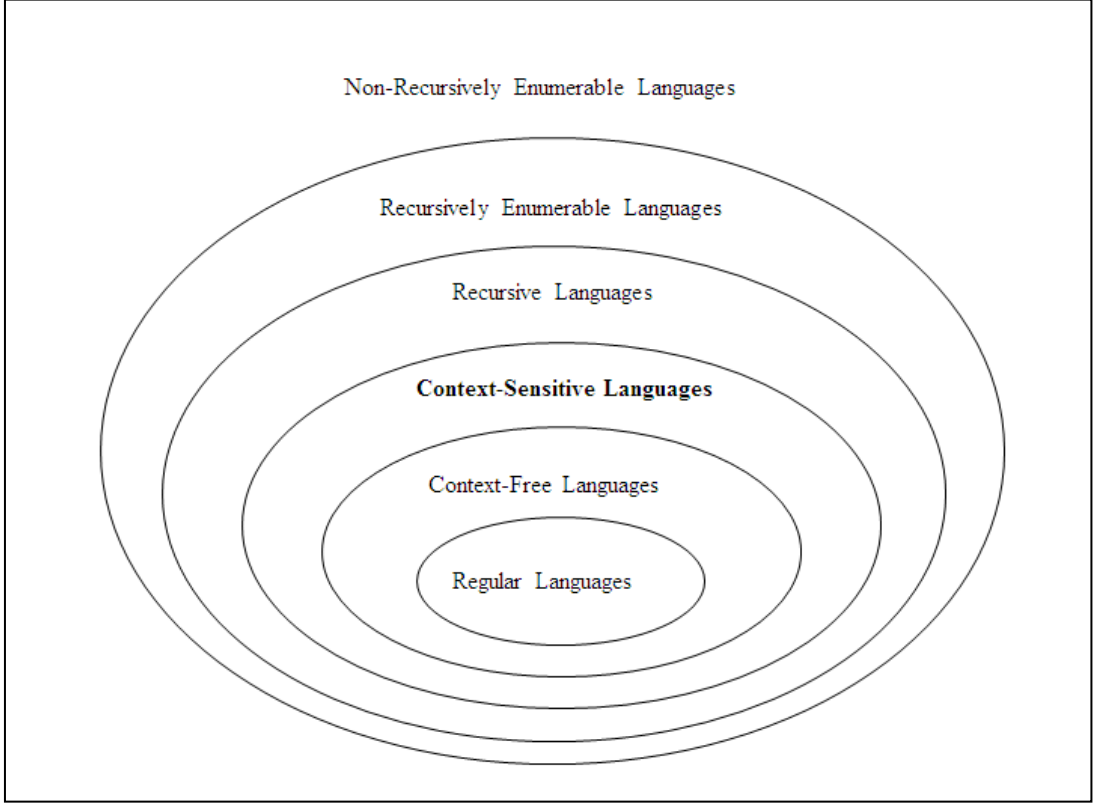
- Are there languages which are in RE but not in R ?

- Yes.
- Finding an $L \in RE - R$ is a challenging task.

Finding an $L \in RE - R$ is a challenging task

- That is, for any string in L, the machine should halt in a final state.
- There is a string which is not in L, for which every possible TM (which accepts L) enters in to an infinite loop.

- There exists such languages !!
- This one important aspect of the theory of computation.
- This, indeed points at limitations of computing machines.



- This diagram is saying that regular, context free and some other languages are recursive.
- That is, all these languages are Turing Decidable.

- We can say, for a regular language there exists a DFA (and also a NFA) that decides the language.
- Similarly for a CFL, a PDA that decides the language always exists.

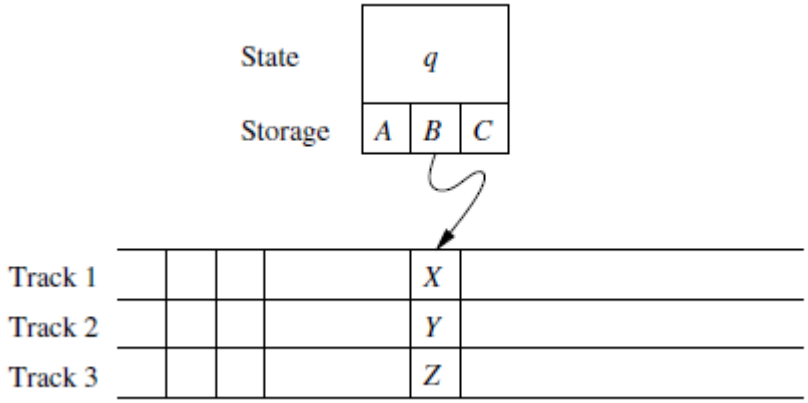
- So, we do not need powerful TMs to decide regular and CFLs.

Variants of TM

Multi-track, multi-tape, NTM

https://www.andrew.cmu.edu/user/ko/pdfs/lecture-14.pdf

State storage & Multi-track



- In this example, the tape symbol is the triplet (X, Y, Z) and we can see the tape as a single-track one.

Example We shall design a TM

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], B, \{[q_1, B]\})$$

that remembers in its finite control the first symbol (0 or 1) that it sees, and checks that it does not appear elsewhere on its input. Thus, M accepts the language $\mathbf{01}^* + \mathbf{10}^*$.

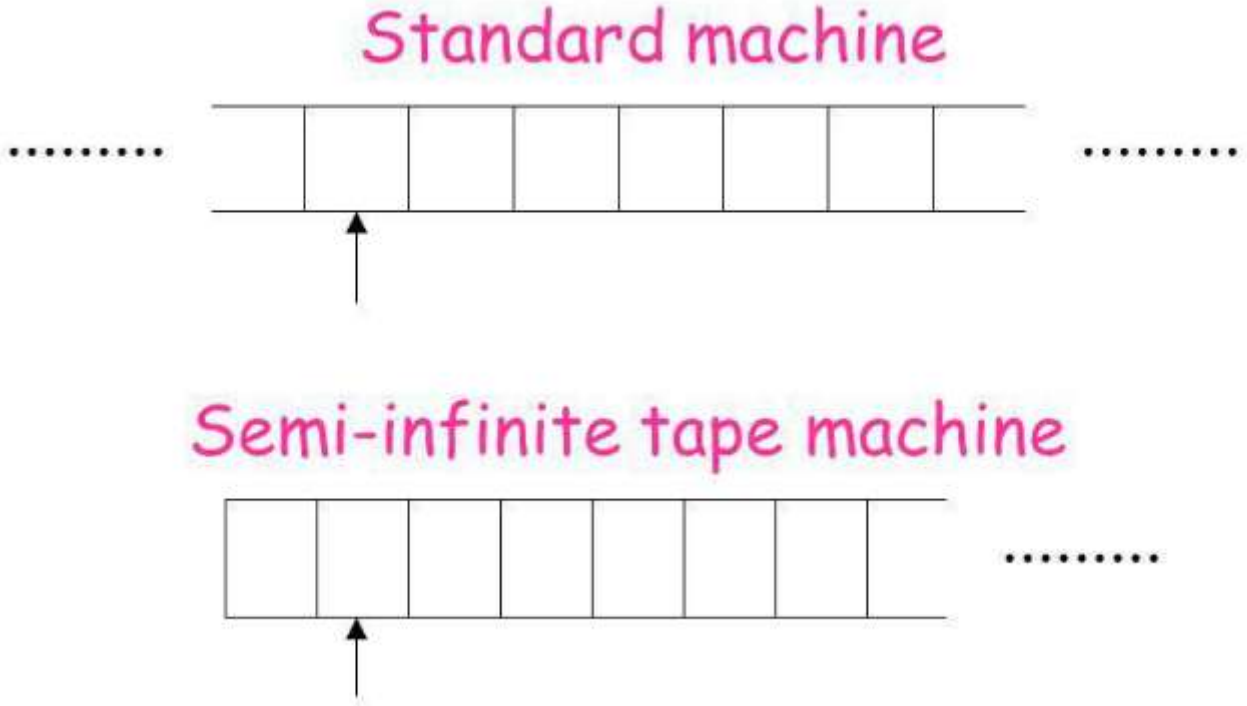
The transition function δ of M is as follows:

1. $\delta([q_0, B], a) = ([q_1, a], a, R)$ for $a = 0$ or $a = 1$. Initially, q_0 is the control state, and the data portion of the state is B. The symbol scanned is copied into the second component of the state, and M moves right, entering control state q_1 as it does so.

2. $\delta([q_1, a], \overline{a}) = ([q_1, a], \overline{a}, R)$ where $\overline{a}$ is the "complement" of a, that is, 0 if $a = 1$ and 1 if $a = 0$. In state q_1, M skips over each symbol 0 or 1 that is different from the one it has stored in its state, and continues moving right.

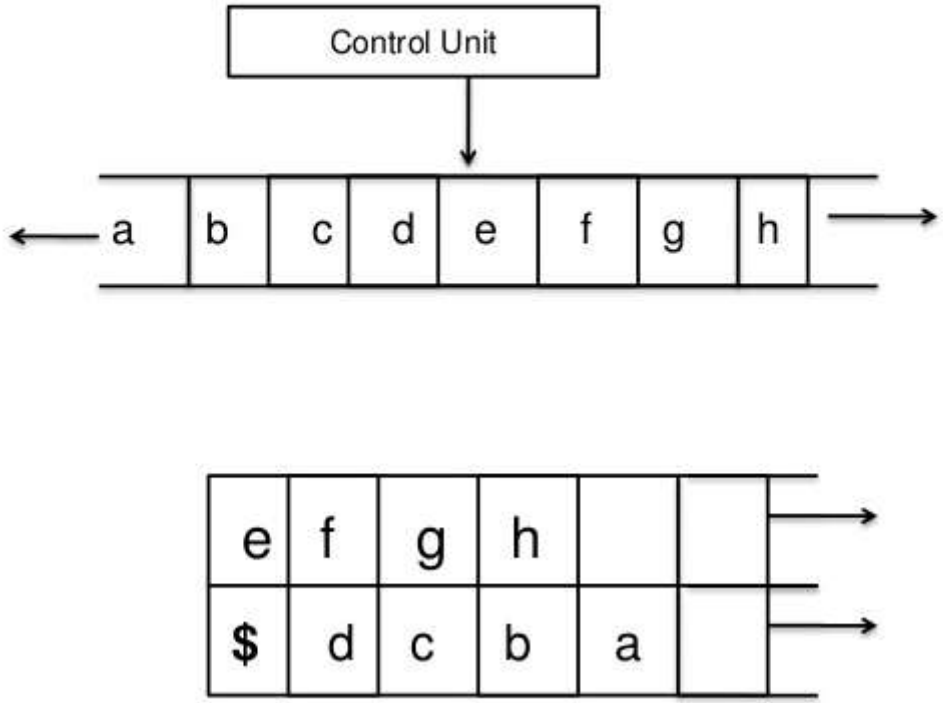
3. $\delta([q_1, a], B) = ([q_1, B], B, R)$ for $a = 0$ or $a = 1$. If M reaches the first blank, it enters the accepting state $[q_1, B]$.

Semi-infinite tape



Simulation of two way infinite by semi-infinite tape

- Two way infinite tape simulated by semi -infinite tape



Multi-tape

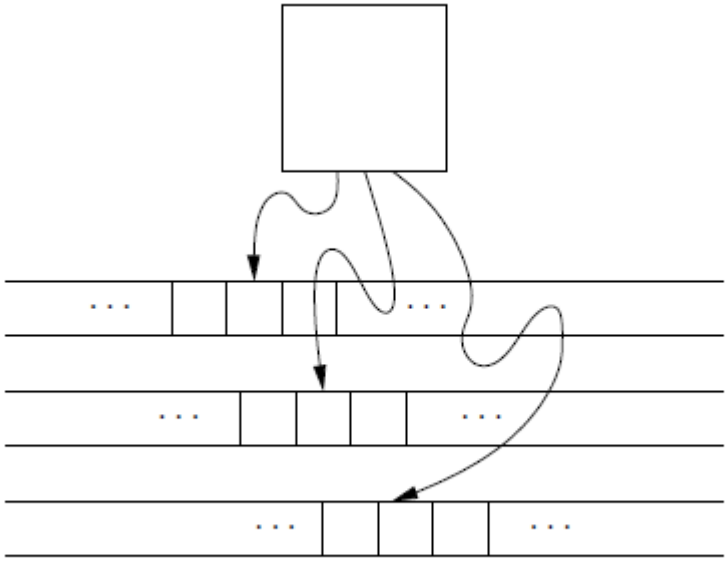


Figure 8.16: A multitape Turing machine

Simulation of multi-tape by one-tape

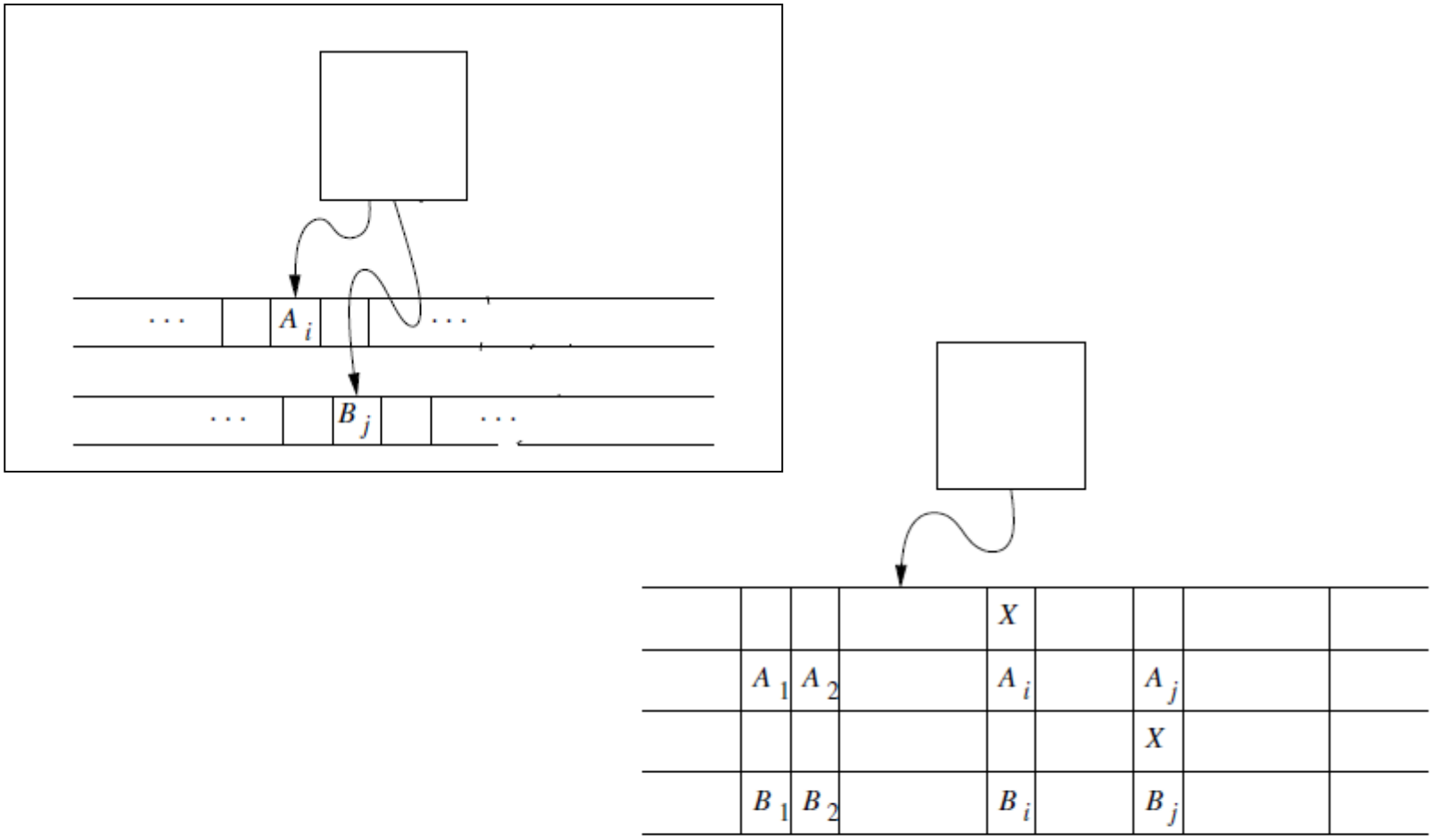


Figure 8.17: Simulation of a two-tape Turing machine by a one-tape Turing machine

NTM

NONDETERMINISTIC TM

- There is a choice in the next move.

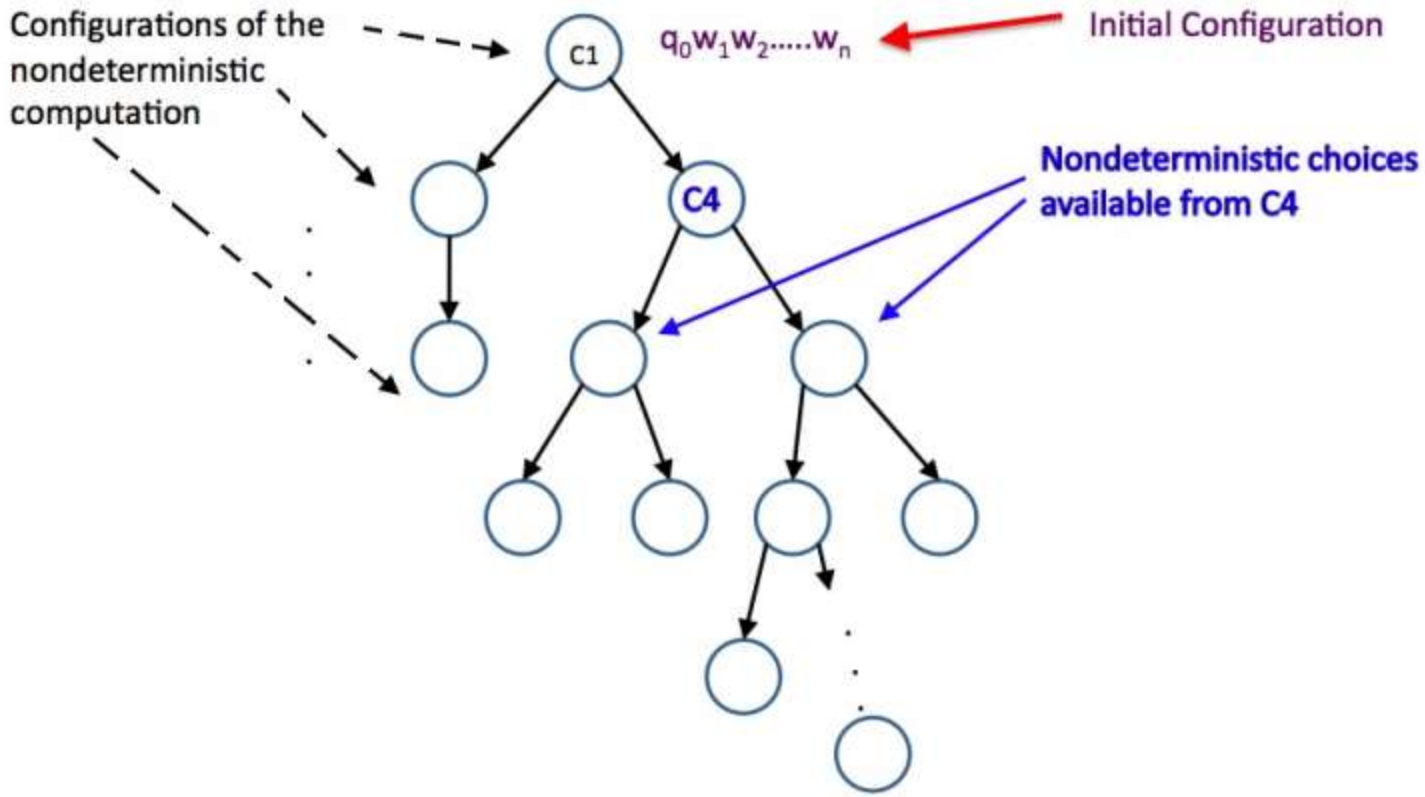
$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

- Here, Y_i is a tape symbol, and D_i is one from $\{L, R\}$, the direction of movement of the head.

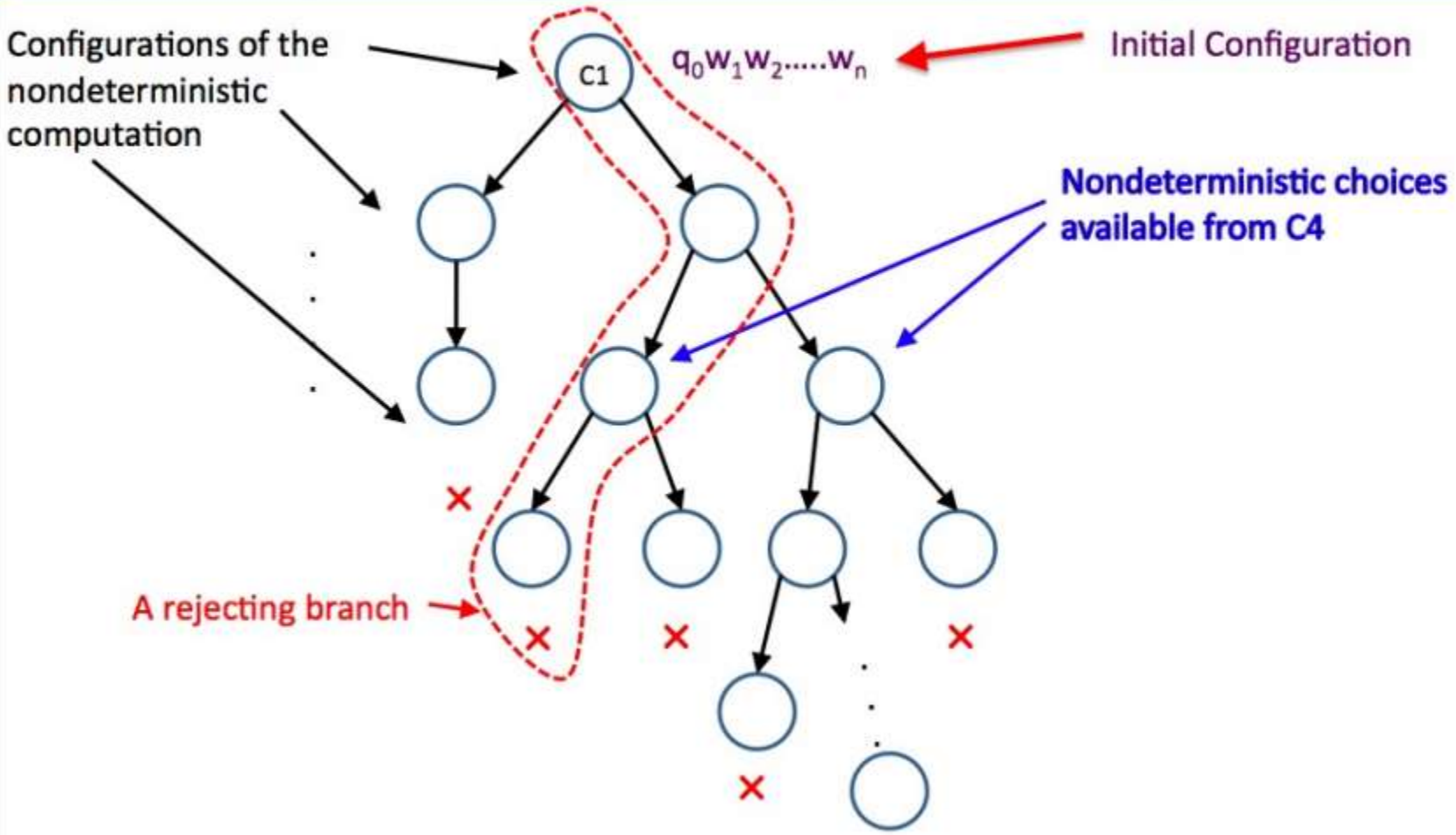
The transition function for a nondeterministic Turing machine has the form

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}).$$

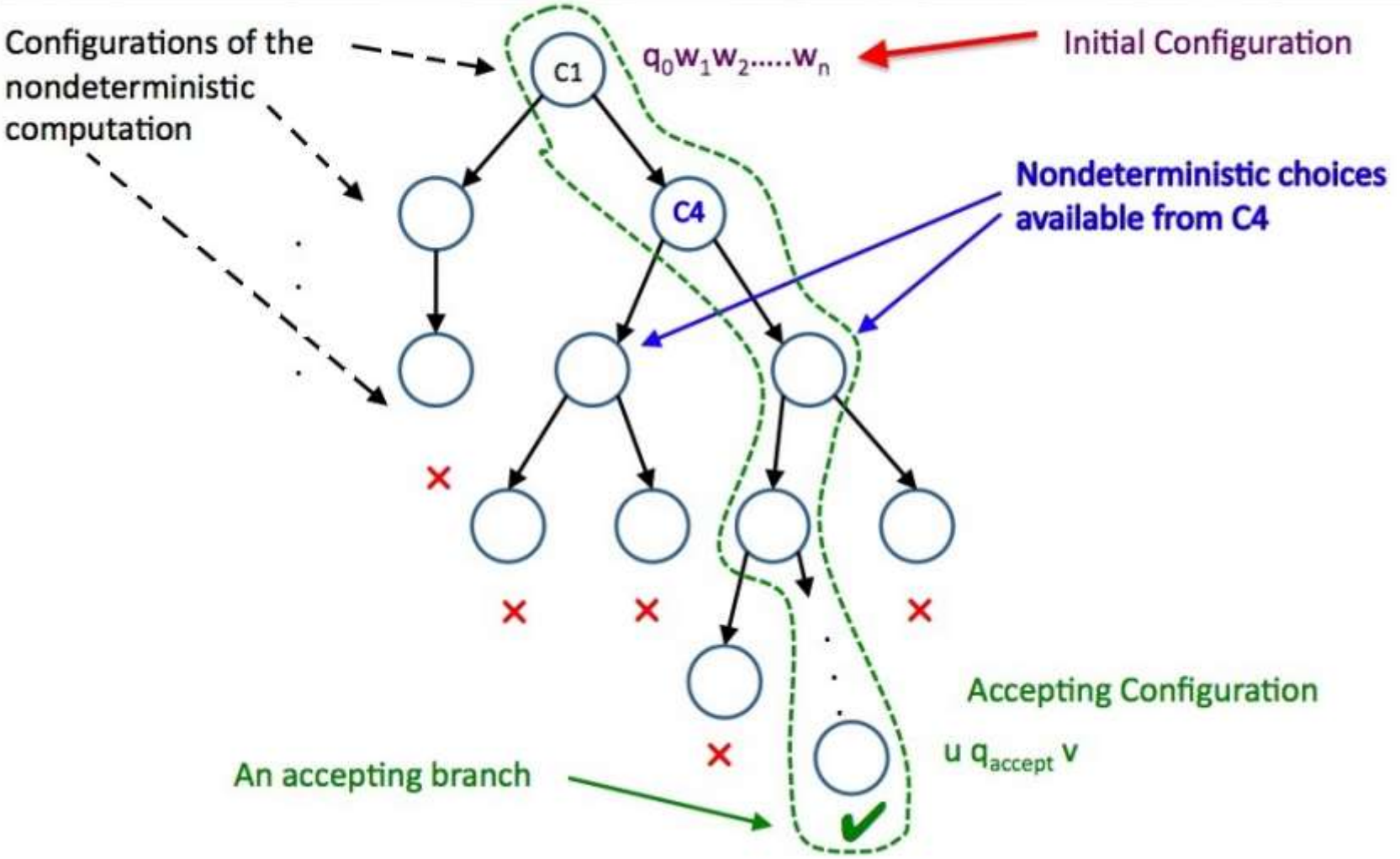
NONDETERMINISTIC COMPUTATION



NONDETERMINISTIC COMPUTATION

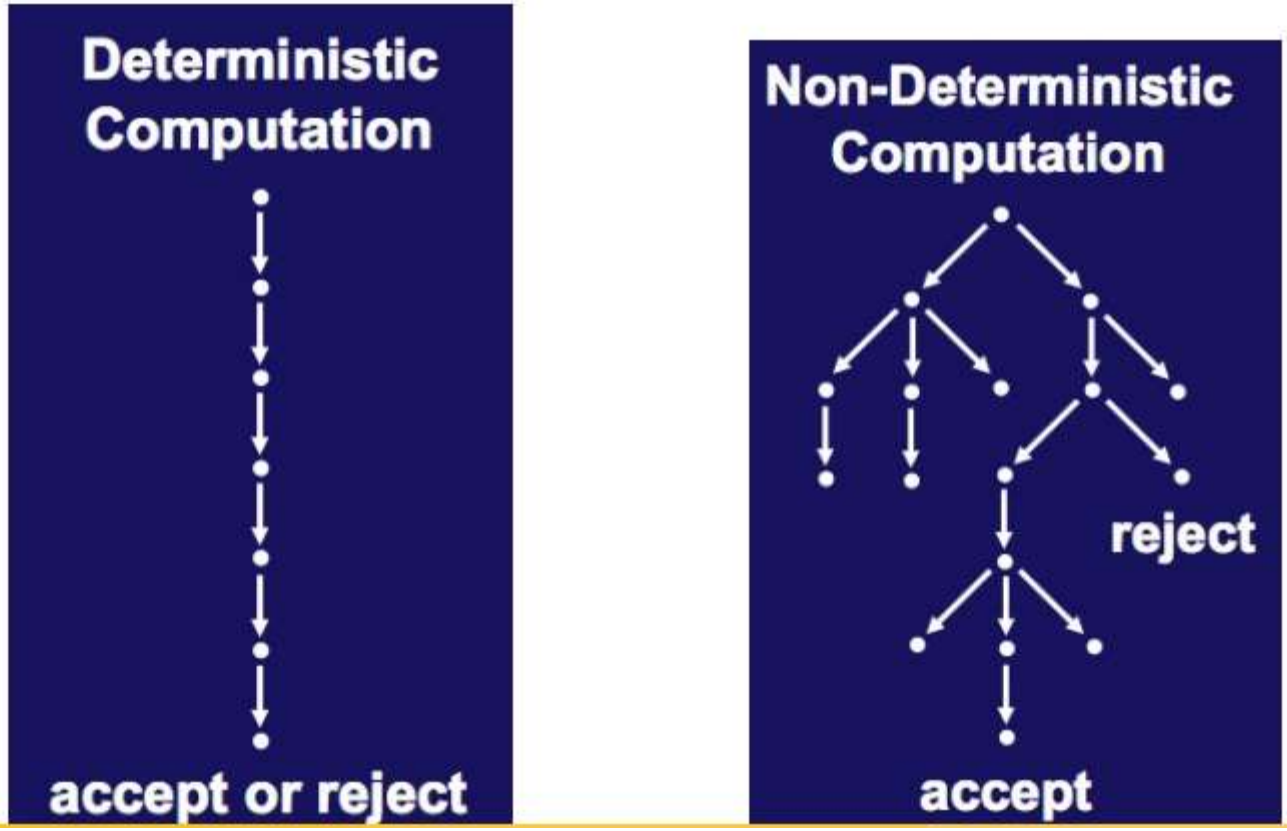


NONDETERMINISTIC COMPUTATION



NONDETERMINISTIC TURING MACHINES

- A computation of a Nondeterministic TM is a tree, where each branch of the tree is looks like a computation of an ordinary TM.



NONDETERMINISTIC TURING MACHINES

- If a single branch reaches the accepting state, the Nondeterministic TM accepts, even if other branches reach the rejecting state.
- What is the power of Nondeterministic TMs?
  - Is there a language that a Nondeterministic TM can accept but no deterministic TM can accept?

- Note, NTM rejects means
  - Each of the branch either explicitly rejects (by getting stuck in a non-final state), or goes in to an infinite loop.

Nondeterministic Turing Machines

Theorem

Every nondeterministic Turing machine has an equivalent deterministic Turing Machine.

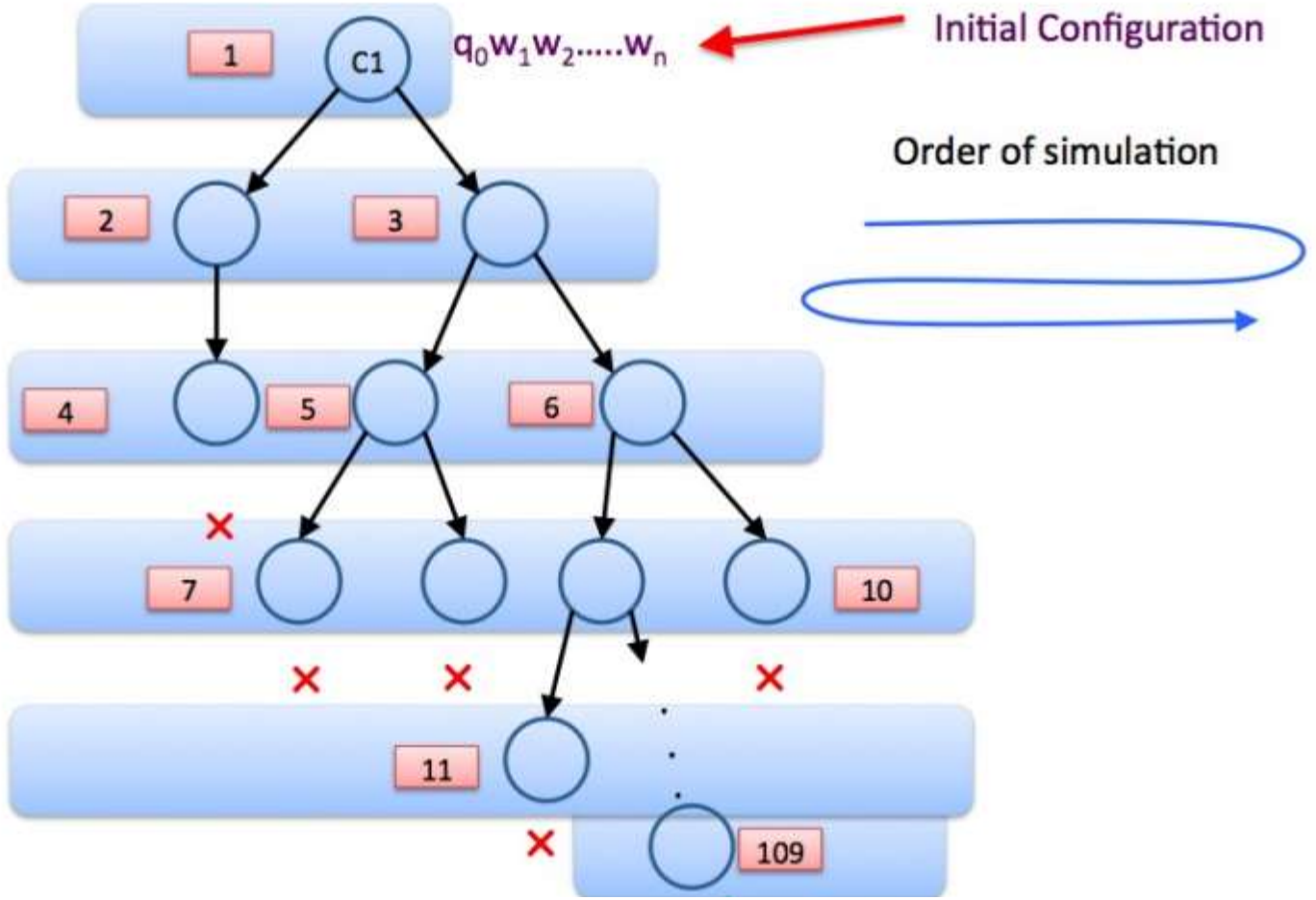
Proof Idea

- Timeshare a deterministic TM to different branches of the nondeterministic computation!
- Try out all branches of the nondeterministic computation until an accepting configuration is reached on one branch.
- Otherwise the TM goes on forever.

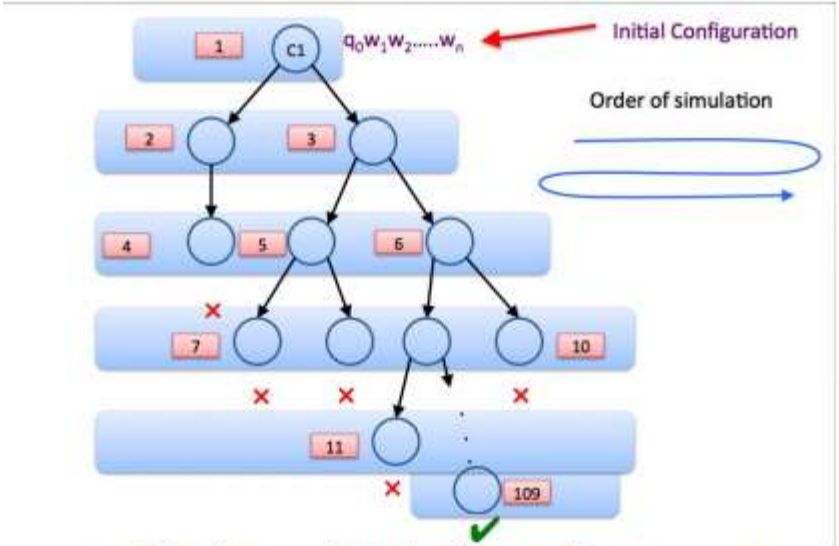
NONDETERMINISTIC TURING MACHINES

- Deterministic TM *D* simulates the Nondeterministic TM *N*.
- Some of branches of the *N*'s computations may be infinite, hence its computation tree has some infinite branches.
- If *D* starts its simulation by following an infinite branch, *D* may loop forever even though *N*'s computation may have a different branch on which it accepts.
- This is a very similar problem to processor scheduling in operating systems.
  - If you give the CPU to a (buggy) process in an infinite loop, other processes "starve".
- In order to avoid this unwanted situation, we want *D* to execute all of *N*'s computations concurrently.

Simulating Nondeterministic Computation



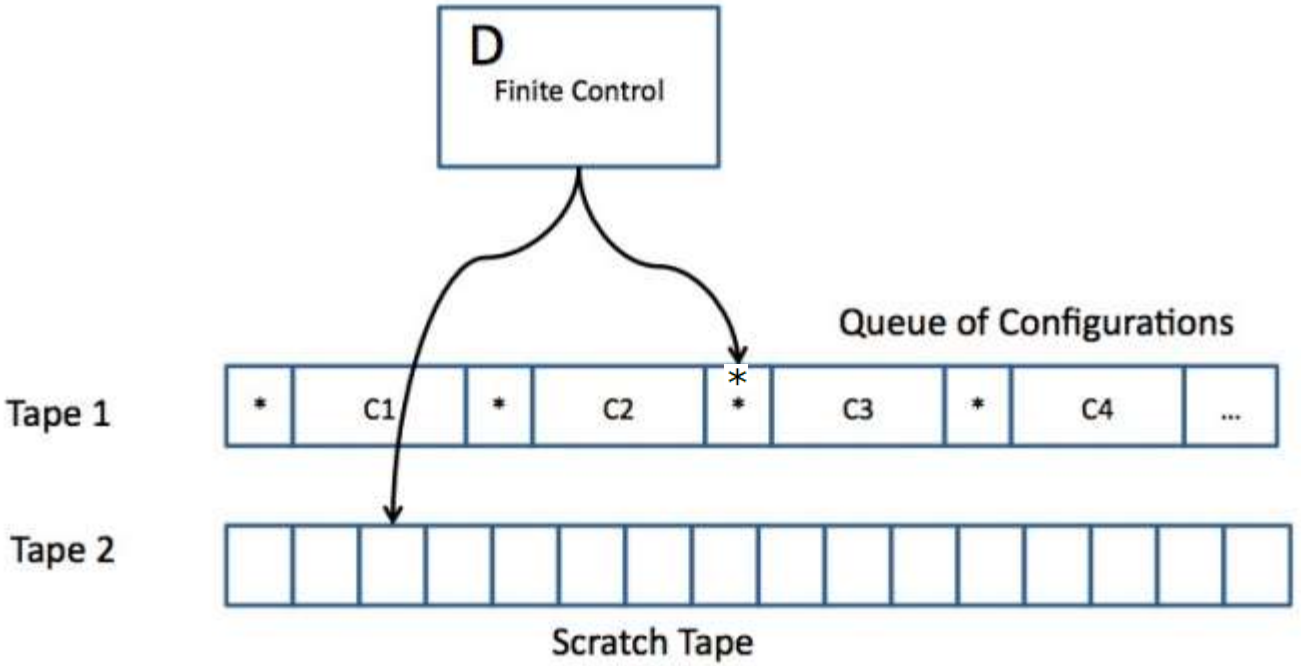
Simulating Nondeterministic Computation



- During simulation, *D* processes the configurations of *N* in a breadth-first fashion.
- Thus *D* needs to maintain a queue of *N*'s configurations (Remember queues?)
- *D* gets the next configuration from the head of the queue.
- *D* creates copies of this configuration (as many as needed)
- On each copy, *D* simulates one of the nondeterministic moves of *N*.
- *D* places the resulting configurations to the back of the queue.

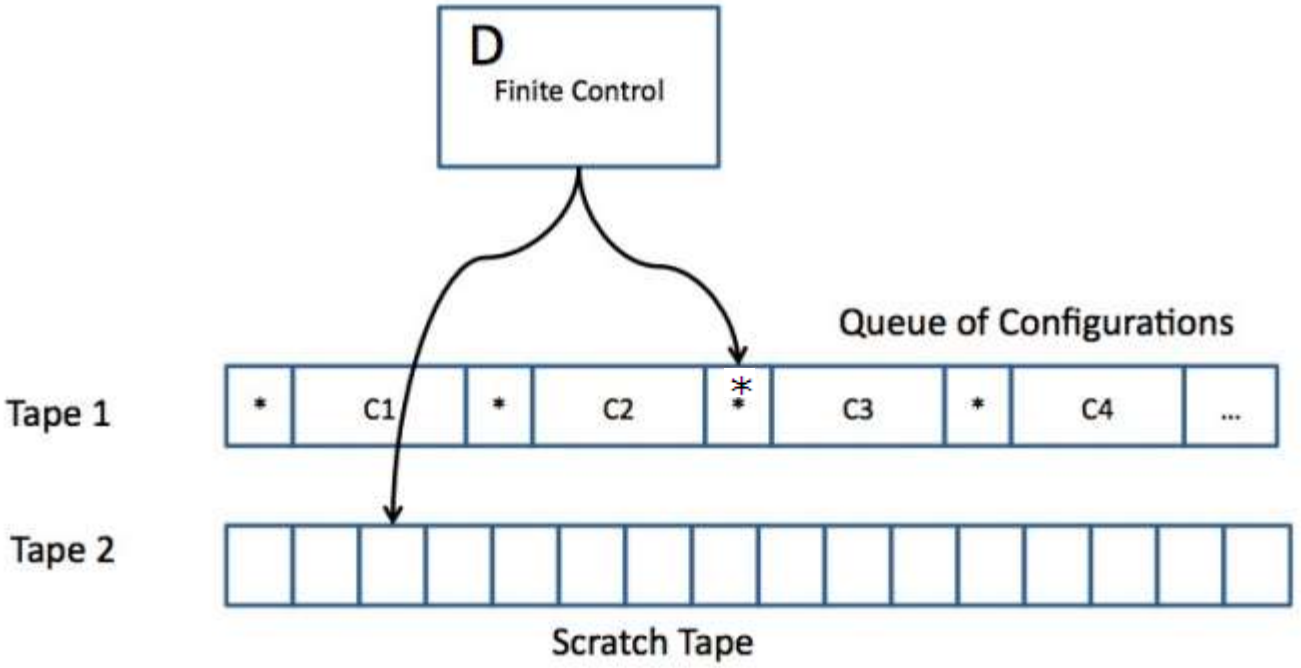
STRUCTURE OF THE SIMULATING DTM

- *N* is simulated with 2-tape DTM, *D*



Structure of the Simulating DTM

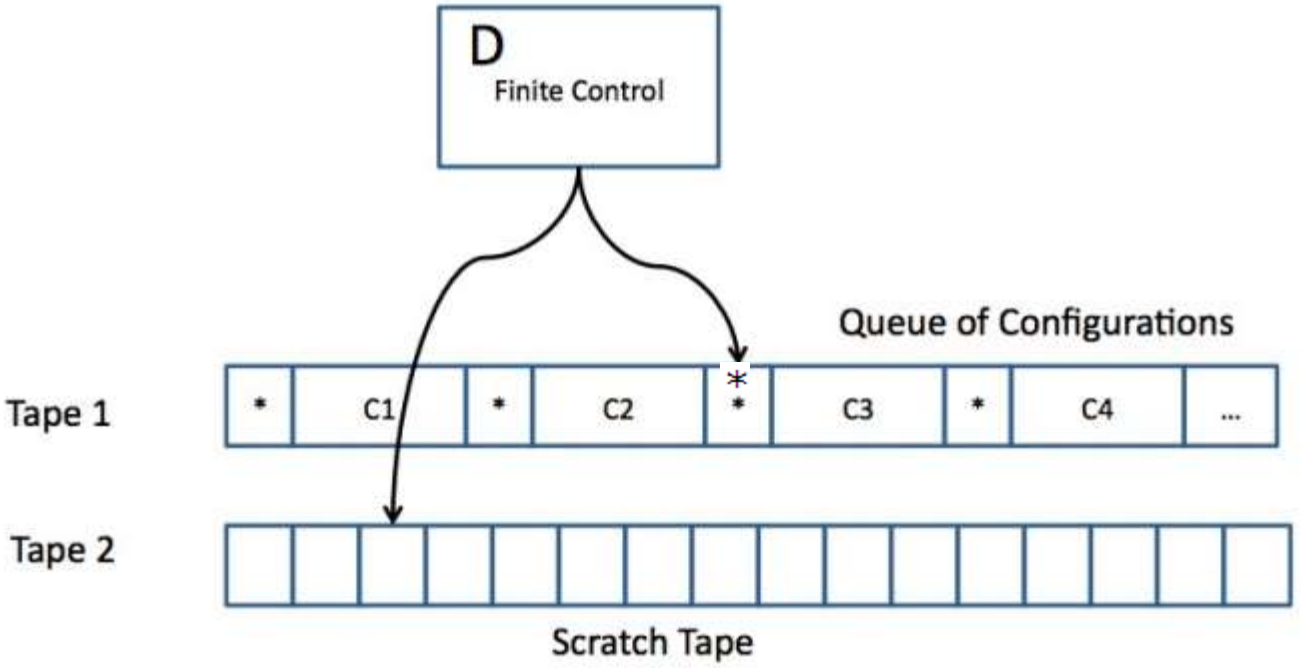
- *N* is simulated with 2-tape DTM, *D*



- Built into the finite control of *D* is the knowledge of what choices of moves *N* has for each state and input.

Structure of the Simulating DTM

- *N* is simulated with 2-tape DTM, *D*



1. *D* examines the state and the input symbol of the current configuration (right after the dotted separator)
2. If the state of the current configuration is the accept state of *N*, then *D* accepts the input and stops simulating *N*.

How D Simulates N

- Let m be the maximum number of choices N has for any of its states.
- Then, after n steps, N can reach at most $1 + m + m^2 + \cdots + m^n$ configurations (which is at most nm^n)
- Thus D has to process at most this many configurations to simulate n steps of N.
- Thus the simulation can take exponentially more time than the nondeterministic TM.
- It is not known whether or not this exponential slowdown is necessary.

Implications

Corollary

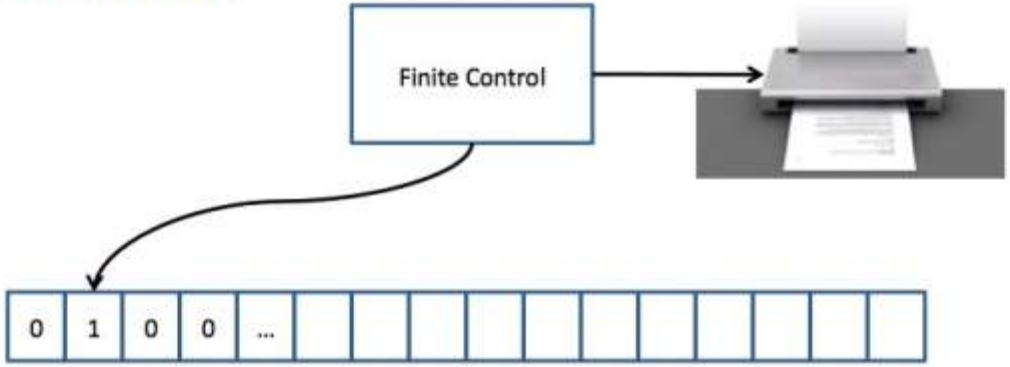
A language is Turing-recognizable if and only if some nondeterministic TM recognizes it.

Corollary

A language is decidable if and only of some nondeterministic TM decides it.

ENUMERATORS

- Remember we noted that some books used the term recursively enumerable for Turing-recognizable.
- This term arises from a variant of a TM called an enumerator.



- TM generates strings one by one.
- Everytime the TM wants to add a string to the list, it sends it to the printer.

ENUMERATORS

- The enumerator E starts with a blank input tape.
- If it does not halt, it may print an infinite list of strings.
- The strings can be enumerated in any order; repetitions are possible.
- The language of the enumerator is the collection of strings it eventually prints out.

- We can assume that the enumerator E writes one string at a time over a tape (it can use a tape symbol # to separate strings).

ENUMERATORS

THEOREM

A language is Turing recognizable if and only if some enumerator enumerates it.

PROOF.

The If-part: If an enumerator E enumerates the language A then a TM M recognizes A.

M = "On input w

1. Run E. Everytime E outputs a string, compare it with w.
2. If w ever appears in the output of E, *accept.*"

Clearly M accepts only those strings that appear on E's list.

□

The TM M accepts w only when E produces w as one of its output strings.

ENUMERATORS

THEOREM

A language is Turing recognizable if and only if some enumerator enumerates it.

PROOF.

The Only-If-part: If a TM *M* recognizes a language *A*, we can construct the following enumerator for *A*.

- For each possible string $s \in \Sigma^*$ we can verify whether M accepts s, if so output s on to the tape .

Following attempt, does not work.

- Assume that there is an enumerator which enumerates Σ^* in a standard order.
- For each possible string $s \in \Sigma^*$ we can verify whether M accepts s, if so output s on to the tape .
- The problem with this is, for some s the TM M can enter in to an infinite loop and never returns.
  - There may be other strings which are accepted by M but will never gets a chance to be verified (and thus never is outputted).

- A feasible way of doing this (without falling in to an infinite loop) is given in the next slide.
- Basic idea:
- For example if there are two strings w1 and w2 and one of them makes the TM to loop infinitely. Do the following.
  1. **k = 1**
  2. **Run TM for k steps on w1 and if accept occurs then output "accept" and stop.**
  3. **Run TM for k steps on w2 and if accept occurs then output "accept" and stop.**
  4. **k++; goto step 2.**

Enumerators

Theorem

A language is Turing recognizable if and only if some enumerator enumerates it.

Proof.

The Only-If-part: If a TM M recognizes a language A, we can construct the following enumerator for A. Assume $s_1, s_2, s_3, \ldots$ is a list of possible strings in Σ^*.

$E =$ "Ignore the input

1. Repeat the following for $i = 1, 2, 3, \ldots$
2. Run M for i steps on each input $s_1, s_2, s_3, \ldots s_i$.
3. If any computations accept, print out corresponding s_j."

If M accepts a particular string, it will appear on the list generated by E (in fact infinitely many times)

The Definition of Algorithm - History

- in 1900, Hilbert posed the following problem:

  "Given a polynomial of several variables with integer coefficients, does it have an integer root – an assignment of integers to variables, that make the polynomial evaluate to 0"

- For example, $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root at $x = 5, y = 3, z = 0$.
- Hilbert explicitly asked that an algorithm/procedure to be "devised". He assumed it existed; somebody needed to find it!
- 70 years later it was shown that no algorithm exists.
- The intuitive notion of an algorithm may be adequate for giving algorithms for certain tasks, but was useless for showing no algorithm exists for a particular task.

This is known as Hilbert's Tenth Problem.

The Definition of Algorithm - History

- In early 20^{th} century, there was no formal definition of an algorithm.
- In 1936, Alonzo Church and Alan Turing came up with formalisms to define algorithms. These were shown to be equivalent, leading to the

Church-Turing Thesis

Intutitive notion of algorithms $\equiv$ Turing Machine Algorithms

THE DEFINITION OF AN ALGORITHM

- Let $D = \{p \mid p \text{ is a polynomial with integral roots}\}$
- Hilbert's 10^{th} problem in TM terminology is "Is D decidable?" (No!)
- However D is Turing-recognizable!
- Consider a simpler version
  $D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with integral roots}\}$
- M_1 = "The input is polynomial p over x.
  1. Evaluate p with x successively set to 0, 1, -1, 2, -2, 3, -3, ...
  2. If at any point, p evaluates to 0, *accept*."
- D_1 is actually decidable since only a finite number of x values need to be tested (math!)
- D is also recognizable: just try systematically all integer combinations for all variables.

- For D_1 (polynomial of single variable) there is a bound and we can abandon the search beyond that and declare “No”.
- For D such a bound cannot exist (proof is given by Matijasevich (1971)).
  - So, if answer is “No” we enter in to an infinite loop.

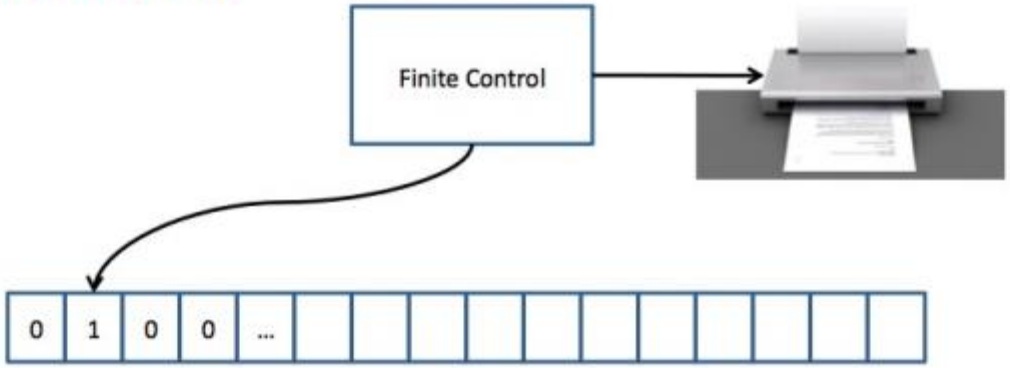
In 1971, Yuri Matijasevich gave a resounding negative answer to Hilbert’s tenth problem.

- This is called undecidability.
- Hilbert’s tenth problem is undecidable.

- We will see the theory behind this in the next …

ENUMERATORS

- Remember we noted that some books used the term recursively enumerable for Turing-recognizable.
- This term arises from a variant of a TM called an enumerator.



- TM generates strings one by one.
- Everytime the TM wants to add a string to the list, it sends it to the printer.

Loosely defined, an enumerator is a Turing machine with an attached printer.

ENUMERATORS

- The enumerator E starts with a blank input tape.
- If it does not halt, it may print an infinite list of strings.
- The strings can be enumerated in any order; repetitions are possible.
- The language of the enumerator is the collection of strings it eventually prints out.

- We can assume that the enumerator E writes one string at a time over a tape (it can use a tape symbol # to separate strings).

Enumerators

Theorem

A language is Turing recognizable if and only if some enumerator enumerates it.

Proof.

The If-part: If an enumerator E enumerates the language A then a TM M recognizes A.

M = "On input w

1. Run E. Everytime E outputs a string, compare it with w.
2. If w ever appears in the output of E, *accept.*"

Clearly M accepts only those strings that appear on E's list.

□

The TM M accepts w only when E produces w as one of its output strings.

ENUMERATORS

THEOREM

A language is Turing recognizable if and only if some enumerator enumerates it.

PROOF.

The Only-If-part: If a TM *M* recognizes a language *A*, we can construct the following enumerator for *A*.

- For each possible string $s \in \Sigma^*$ we can verify whether M accepts s, if so output s on to the tape .

Following attempt, does not work.

- Assume that there is an enumerator which enumerates Σ^* in a standard order.
- For each possible string $s \in \Sigma^*$ we can verify whether M accepts s, if so output s on to the tape .
- The problem with this is, for some s the TM M can enter in to an infinite loop and never returns.
  - There may be other strings which are accepted by M but will never gets a chance to be verified (and thus never is outputted).

- A feasible way of doing this (without falling in to an infinite loop) is given in the next slide.
- Basic idea:
- For example if there are two strings w1 and w2 and one of them makes the TM to loop infinitely. Do the following.
  1. **k = 1**
  2. **Run TM for k steps on w1 and if accept occurs then output "accept" and stop.**
  3. **Run TM for k steps on w2 and if accept occurs then output "accept" and stop.**
  4. **k++; goto step 2.**

Enumerators

Theorem

A language is Turing recognizable if and only if some enumerator enumerates it.

Proof.

The Only-If-part: If a TM M recognizes a language A, we can construct the following enumerator for A. Assume $s_1, s_2, s_3, \ldots$ is a list of possible strings in Σ^*.

$E =$ "Ignore the input

1. Repeat the following for $i = 1, 2, 3, \ldots$
2. Run M for i steps on each input $s_1, s_2, s_3, \ldots s_i$.
3. If any computations accept, print out corresponding s_j."

If M accepts a particular string, it will appear on the list generated by E (in fact infinitely many times)

The Definition of Algorithm - History

- in 1900, Hilbert posed the following problem:

  "Given a polynomial of several variables with integer coefficients, does it have an integer root – an assignment of integers to variables, that make the polynomial evaluate to 0"

- For example, $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root at $x = 5, y = 3, z = 0$.
- Hilbert explicitly asked that an algorithm/procedure to be "devised". He assumed it existed; somebody needed to find it!
- 70 years later it was shown that no algorithm exists.
- The intuitive notion of an algorithm may be adequate for giving algorithms for certain tasks, but was useless for showing no algorithm exists for a particular task.

This is known as Hilbert's Tenth Problem.

The Definition of Algorithm - History

- In early 20^{th} century, there was no formal definition of an algorithm.
- In 1936, Alonzo Church and Alan Turing came up with formalisms to define algorithms. These were shown to be equivalent, leading to the

Church-Turing Thesis

Intutitive notion of algorithms $\equiv$ Turing Machine Algorithms

THE DEFINITION OF AN ALGORITHM

- Let $D = \{p \mid p \text{ is a polynomial with integral roots}\}$
- Hilbert's 10^{th} problem in TM terminology is "Is D decidable?" (No!)
- However D is Turing-recognizable!
- Consider a simpler version
  $D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with integral roots}\}$
- M_1 = "The input is polynomial p over x.
  1. Evaluate p with x successively set to 0, 1, -1, 2, -2, 3, -3, ...
  2. If at any point, p evaluates to 0, *accept*."
- D_1 is actually decidable since only a finite number of x values need to be tested (math!)
- D is also recognizable: just try systematically all integer combinations for all variables.

- For D_1 (polynomial of single variable) there is a bound and we can abandon the search beyond that and declare "No".
- The roots of such a polynomial must lie between the values:

$$\pm k \frac{C_{max}}{C_1}$$

- k is the number of terms in the polynomial, cmax is the coefficient with the largest absolute value, and c1 is the coefficient of the highest order term.
- For D such a bound cannot exist (proof is given by Matijasevich (1971)).
  - So, if answer is "No" we enter in to an infinite loop.

In 1971, Yuri Matijasevich gave a resounding negative answer to Hilbert's tenth problem.

- This is called undecidability.
- Hilbert’s tenth problem is undecidable.

Decidability

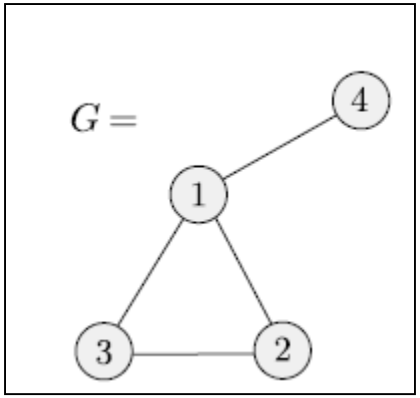
Theory behind existence of undecidable problems – Halting Problem --

Ref: https://www.andrew.cmu.edu/user/ko/pdfs/lecture-15.pdf

Describing Turing Machines and Their Inputs

- For the rest of the course we will have a rather standard way of describing TMs and their inputs.
- The input to TMs have to be strings.
- Every object O that enters a computation will be represented with an string $\langle O \rangle$, encoding the object.
- For example if G is a 4 node undirected graph with 4 edges $\langle O \rangle =$ `(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`
- Then we can define problems over graphs,e.g., as:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$



Describing Turing Machines and Their Inputs

- A TM for this problem can be given as:
- M = "On input $\langle G \rangle$, the encoding of a graph G:
  1. Select the first node of G and mark it.
  2. Repeat 3) until no new nodes are marked
  3. For each node in G, mark it, if there is edge attaching it to an already marked node.
  4. Scan all the nodes in G. If all are marked, the *accept*, else *reject*"

- Important thing to understand is that a machine (computing machine) can be represented as a string in some language.
  - It will be a string over some alphabet.
  - This is, in some sense, equivalent to say that program is also data.

Representation Vs. Real Thing

- DNA represents a living thing (like a human-being).
- Some people believe (!) that this is a complete description of a human being.
- You can know his personality, you can even know what he will do in future?
  - Some Hollywood movies captured this idea.

- A representation in itself is lifeless.
- A program in itself cannot process the data.
- It has to be realized through a processing unit or DNA has to be realized through a laboratory test tube or so.

- But the processing unit, now can be entirely independent of the program.
- It can execute any program.

- We can give <program, data> to a general purpose computer which runs the program over the data and gives the output.

- A TM M also can be represented as a string.
- Call this string $< M >$.
- $< M >$ is like a program.
- Working of TM M on a string w can be realized by a *general purpose computer* like machine.
- This machine which can take input $< M, w >$ and outputs what the TM M does with w, is called the **U**niversal **T**uring **M**achine.

DECIDABILITY

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be.
- Why discuss unsolvability?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.

Decidability

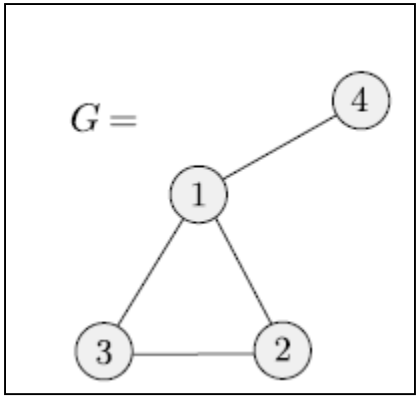
Theory behind existence of undecidable problems – Halting Problem --

Ref: https://www.andrew.cmu.edu/user/ko/pdfs/lecture-15.pdf

Describing Turing Machines and Their Inputs

- For the rest of the course we will have a rather standard way of describing TMs and their inputs.
- The input to TMs have to be strings.
- Every object O that enters a computation will be represented with an string $\langle O \rangle$, encoding the object.
- For example if G is a 4 node undirected graph with 4 edges $\langle O \rangle =$ `(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`
- Then we can define problems over graphs,e.g., as:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$



Describing Turing Machines and Their Inputs

- A TM for this problem can be given as:
- M = "On input $\langle G \rangle$, the encoding of a graph G:
  1. Select the first node of G and mark it.
  2. Repeat 3) until no new nodes are marked
  3. For each node in G, mark it, if there is edge attaching it to an already marked node.
  4. Scan all the nodes in G. If all are marked, the *accept*, else *reject*"

- Important thing to understand is that a machine (computing machine) can be represented as a string in some language.
  - It will be a string over some alphabet.
  - This is, in some sense, equivalent to say that program is also data.

Representation Vs. Real Thing

- DNA represents a living thing (like a human-being).
- Some people believe (!) that this is a complete description of a human being.
- You can know his personality, you can even know what he will do in future?
  - Some Hollywood movies captured this idea.

- A representation in itself is lifeless.
- A program in itself cannot process the data.
- It has to be realized through a processing unit or DNA has to be realized through a laboratory test tube or so.

- But the processing unit, now can be entirely independent of the program.
- It can execute any program.

- We can give <program, data> to a general purpose computer which runs the program over the data and gives the output.

- A TM M also can be represented as a string.
- Call this string $< M >$.
- $< M >$ is like a program.
- Working of TM M on a string w can be realized by a *general purpose computer* like machine.
- This machine which can take input $< M, w >$ and outputs what the TM M does with w, is called the **U**niversal **T**uring **M**achine.

DECIDABILITY

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be.
- Why discuss unsolvability?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.

Overview

- Decidable Languages
- Diagonalization
- Halting Problem as a undecidable problem
- Turing-unrecognizable languages.

Decidable Languages

Some notational details

- $\langle B \rangle$ represents the encoding of the description of an automaton (DFA/NFA).
- We need to encode Q, Σ, δ and F.

Encoding Finite Automata as Strings

- Here is one possible encoding scheme:
- Encode Q using unary encoding:
  - For $Q = \{q_0, q_1, \ldots q_{n-1}\}$, encode q_i using $i + 1$ 0's, i.e., using the string 0^{i+1}.
  - We assume that q_0 is always the start state.
- Encode Σ using unary encoding:
  - For $\Sigma = \{a_1, a_2, \ldots a_m\}$, encode a_i using i 0's, i.e., using the string 0^i.
- With these conventions, all we need to encode is δ and F!
- Each entry of δ, e.g., $\delta(q_i, a_j) = q_k$ is encoded as

$$\underbrace{0^{i+1}}_{q_i} 1 \underbrace{0^{j}}_{a_j} 1 \underbrace{0^{k+1}}_{q_k}$$

Encoding Finite Automata as Strings

- The whole δ can now be encoded as

$$\underbrace{00100001000}_{transition_1}1\underbrace{000001001000000}_{transition_2}\cdots 1\underbrace{000000100000010}_{transition_t}$$

- F can be encoded just as a list of the encodings of all the final states. For example, if states 2 and 4 are the final states, F could be encoded as

$$\underbrace{000}_{q_2}1\underbrace{00000}_{q_4}$$

- The whole DFA would be encoded by

$$11\underbrace{00100010000100000\cdots 0}_{\text{encoding of the transitions}}11\underbrace{000000010000000}_{\text{encoding of the final states}}11$$

Encoding Finite Automata as Strings

- $\langle B \rangle$ representing the encoding of the description of an automaton (DFA/NFA) would be something like

$$\langle B \rangle = 11\underbrace{00100010000100000\cdots0}_{\textit{encoding of the transitions}}11\underbrace{000000010000000}_{\textit{encoding of the final states}}11$$

- In fact, the description of all DFAs could be described by a regular expression like

$$11(0^+10^+10^+1)^*1(0^+1)^+1$$

- Similarly strings over Σ can be encoded with (the same convention)

$$\langle w \rangle = \underbrace{0000}_{a_4}1\underbrace{000000}_{a_6}1\cdots\underbrace{0}_{a_1}$$

Encoding Finite Automata as Strings

- $\langle B, w \rangle$ represents the encoding of a machine followed by an input string, in the manner above (with a suitable separator between $\langle B \rangle$ and $\langle w \rangle$.
- Now we can describe our problems over languages and automata as problems over strings (representing automata and languages).

Decidable Problems

Regular Languages

- Does B accept w?
- Is $L(B)$ empty?
- Is $L(A) = L(B)$?

The Acceptance Problem for DFAs

Theorem 4.1

$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ is a decidable language.

Proof

- Simulate with a two-tape TM.
  - One tape has $\langle B, w\rangle$
  - The other tape is a work tape that keeps track of which state of B the simulation is in.
- $M =$ "On input $\langle B, w\rangle$
  1. Simulate B on input w
  2. If the simulation ends in an accept state of B, *accept*; if it ends in a nonaccepting state, *reject*."

The Acceptance Problem for NFAs

Theorem 4.2

$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$ is a decidable language.

Proof

- Convert NFA to DFA and use Theorem 4.1
- N = "On input $\langle B, w\rangle$ where B is an NFA
  1. Convert NFA B to an equivalent DFA C, using the determinization procedure.
  2. Run TM M in Thm 4.1 on input $\langle C, w\rangle$
  3. If M accepts *accept*; otherwise *reject*."

The Generation Problem for Regular Expressions

Theorem 4.3

$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular exp. that generates string } w\}$ is a decidable language.

Proof

- Note R is already a string!!
- Convert R to an NFA and use Theorem 4.2
- $P =$ "On input $\langle R, w\rangle$ where R is a regular expression
  1. Convert R to an equivalent NFA A, using the Regular Expression-to-NFA procedure
  2. Run TM N in Thm 4.2 on input $\langle A, w\rangle$
  3. If N accepts *accept*; otherwise *reject*."

The Emptiness Problem for DFAs

Theorem 4.4

$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \Phi\}$ is a decidable language.

Proof

- Use the DFS algorithm to mark the states of DFA
- T = "On input $\langle A \rangle$ where A is a DFA.
  1. Mark the start state of A
  2. Repeat until no new states get marked.
     - Mark any state that has a transition coming into it from any state already marked.
  3. If no final state is marked, *accept*; otherwise *reject*."

- Is the following a decidable language?

$$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

The Equivalence Problem for DFAs

Theorem 4.5

$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ is a decidable language.

Proof

- Construct the machine for $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ and check if $L(C) = \Phi$.
- T = "On input $\langle A, B\rangle$ where A and B are DFAs.
  1. Construct the DFA for $L(C)$ as described above.
  2. Run TM T of Theorem 4.4 on input $\langle C\rangle$.
  3. If T accepts, *accept*; otherwise *reject*."

Decidable Problems

Context-free Languages

- Does grammar G generate w?
- Is $L(G)$ empty?

- Is the following language decidable?

$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$$

The Generation Problem for CFGs

Theorem 4.7

$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ is a decidable language.

Proof

- Convert G to Chomsky Normal Form and use the CYK algorithm.
- $C =$ "On input $\langle G, w\rangle$ where G is a CFG
  1. Convert G to an equivalent grammar in CNF
  2. Run CYK algorithm on w of length n
  3. If $S \in V_{i,n}$ *accept*; otherwise *reject*."

The Generation Problem for CFGs

Alternative Proof

- Convert G to Chomsky Normal Form and check all derivations up to a certain length (Why!)
- $S =$ "On input $\langle G, w \rangle$ where G is a CFG
  1. Convert G to an equivalent grammar in CNF
  2. List all derivations with $2n - 1$ steps where n is the length of w. If $n = 0$ list all derivations of length 1.
  3. If any of these strings generated is equal to w, *accept*; otherwise *reject*."

- This works because every derivation using a CFG in CNF either increase the length of the sentential form by 1 (using a rule like $A \to BC$ or leaves it the same (using a rule like $A \to a$)
- Obviously this is not very efficient as there may be exponentially many strings of length up to $2n - 1$.

Decidability of CFLs

Theorem 4.9

Every context free language is decidable.

Proof

- Design a TM M_G that has G built into it and use the result of A_{CFG}.
- $M_G =$ "On input w
  1. Run TM S (from Theorem 4.7) on input $\langle G, w \rangle$
  2. If S accepts, *accept*, otherwise *reject*.

The Emptiness Problem for CFGs

Theorem 4.8

$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Phi\}$ is a decidable language.

Proof

- Mark variables of G systematically if they can generate terminal strings, and check if S is unmarked.
- $R =$ "On input $\langle G \rangle$ where G is a CFG.
  1. Mark all terminal symbols G
  2. Repeat until no new variable get marked.
     - Mark any variable A such that G has a rule $A \to U_1 U_2 \cdots U_k$ and $U_1, U_2, \ldots U_k$ are already marked.
  3. If start symbol is NOT marked, *accept*; otherwise *reject*."

The Equivalence Problem for CFGs

$EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- Is this decidable?

The Equivalence Problem for CFGs

$EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- It turns out that EQ_{DFA} is not a decidable language.
- The construction for DFAs does not work because CFLs are NOT closed under intersection and complementation.
- Proof comes later.

Acceptance Problem for TMs

Theorem 4.11

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ is undecidable.

- Note that A_{TM} is Turing-recognizable. Thus this theorem when proved, shows that recognizers are more powerful than deciders.
- We can encode TMs with strings just like we did for DFA's (How?)

- We shall assume the states are $q_1, q_2, \ldots, q_r$ for some r. The start state will always be q_1, and q_2 will be the only accepting state. Note that, since we may assume the TM halts whenever it enters an accepting state, there is never any need for more than one accepting state.

- We shall assume the tape symbols are $X_1, X_2, \ldots, X_s$ for some s. X_1 always will be the symbol 0, X_2 will be 1, and X_3 will be B, the blank. However, other tape symbols can be assigned to the remaining integers arbitrarily.

- We shall refer to direction L as D_1 and direction R as D_2.

- We shall assume the states are $q_1, q_2, \ldots, q_r$ for some r. The start state will always be q_1, and q_2 will be the only accepting state. Note that, since we may assume the TM halts whenever it enters an accepting state, there is never any need for more than one accepting state.

- We shall assume the tape symbols are $X_1, X_2, \ldots, X_s$ for some s. X_1 always will be the symbol 0, X_2 will be 1, and X_3 will be B, the blank. However, other tape symbols can be assigned to the remaining integers arbitrarily.

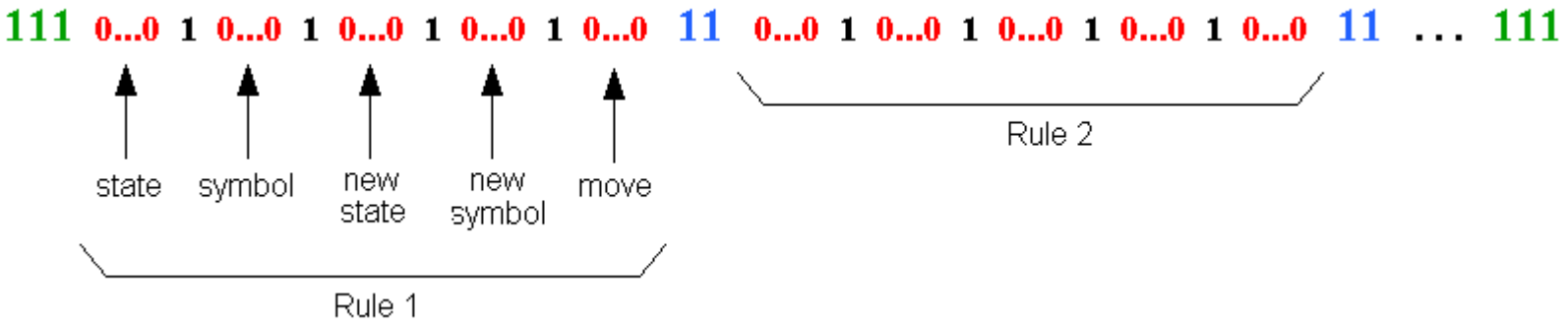
- We shall refer to direction L as D_1 and direction R as D_2.

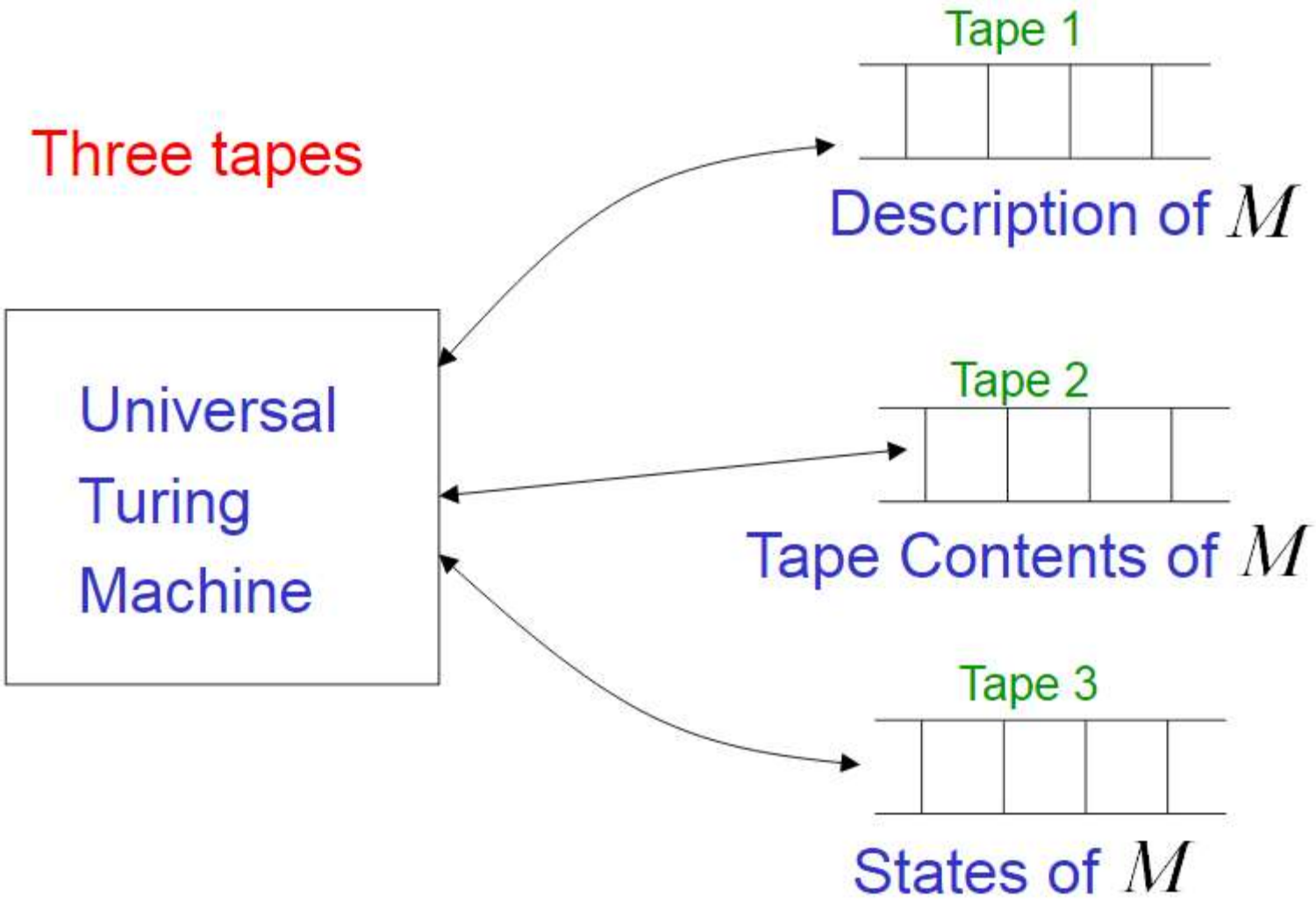
Suppose one transition rule

is $\delta(q_i, X_j) = (q_k, X_l, D_m)$, for some integers i, j, k, l, and m.

We shall code this rule by the string $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.

Encoding of a TM




Three tapes
Universal
Turing
Machine
Tape 1
Description of M
Tape 2
Tape Contents of M
Tape 3
States of M

- Turing Machine M can be encoded as a string $< M >$
- These encodings, of various Turing Machines, can be seen as a language.
- Language of TMs = $\{ < M > | < M > \text{ is an encoding of a TM } M \}$.
- Strings of the above language can be lexicographically ordered.
- As per the above ordering, one can say the 1st TM, the 2nd TM, so on.
  - Same machine may be encoded in multiple ways
  - 2nd TM may be same as 10032nd TM

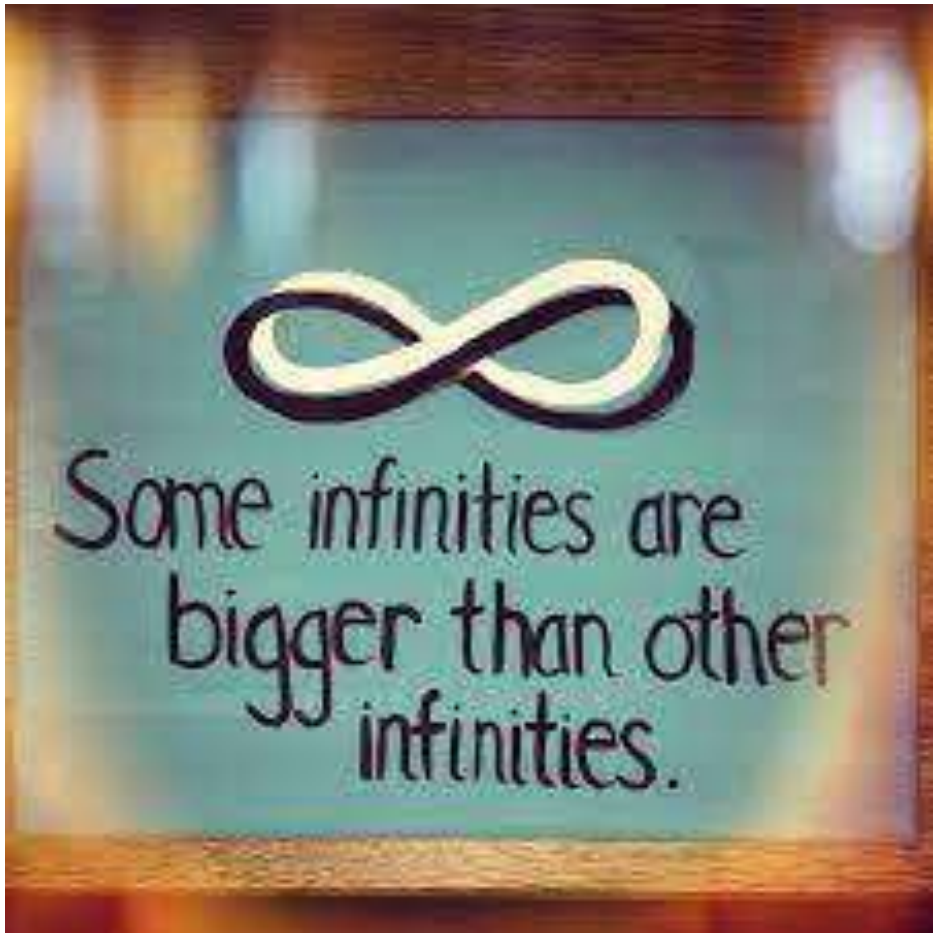
Acceptance Problem for TMs

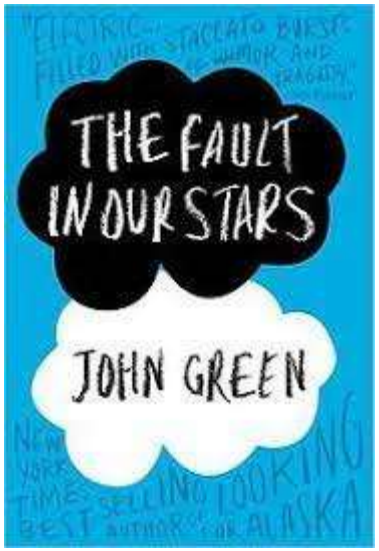
- The TM U recognizes A_{TM}
- $U =$ "On input $\langle M, w \rangle$ where M is a TM and w is a string:
  1. Simulate M on w
  2. If M ever enters its accepts state, *accept*; if M ever enters its reject state, *reject*.
- Note that if M loops on w, then U loops on $\langle M, w \rangle$, which is why it is NOT a decider!
- U can not detect that M halts on w.
- A_{TM} is also known as the Halting Problem
- U is known as the Universal Turing Machine because it can simulate every TM (including itself!)

Acceptance Problem for TMs

- The TM U recognizes A_{TM}
- $U =$ "On input $\langle M, w \rangle$ where M is a TM and w is a string:
  1. Simulate M on w
  2. If M ever enters its accepts state, *accept*; if M ever enters its reject state, *reject*.
- Note that if M loops on w, then U loops on $\langle M, w \rangle$, which is why it is NOT a decider!
- U can not detect that M halts on w.
- A_{TM} is also known as the Halting Problem
- U is known as the Universal Turing Machine because it can simulate every TM (including itself!)

- The proof of the undecidability of ATM uses a technique called *diagonalization* discovered by mathematician Georg Cantor in 1873.
- Cantor was concerned with the problem of measuring the sizes of infinite sets.
- If we have two infinite sets, how can we tell whether one is larger than the other or whether they are of the same size?


Some infinities are
bigger than other
infinities.


THE FAULT
IN OUR STARS
JOHN GREEN

The Diagonalization Method

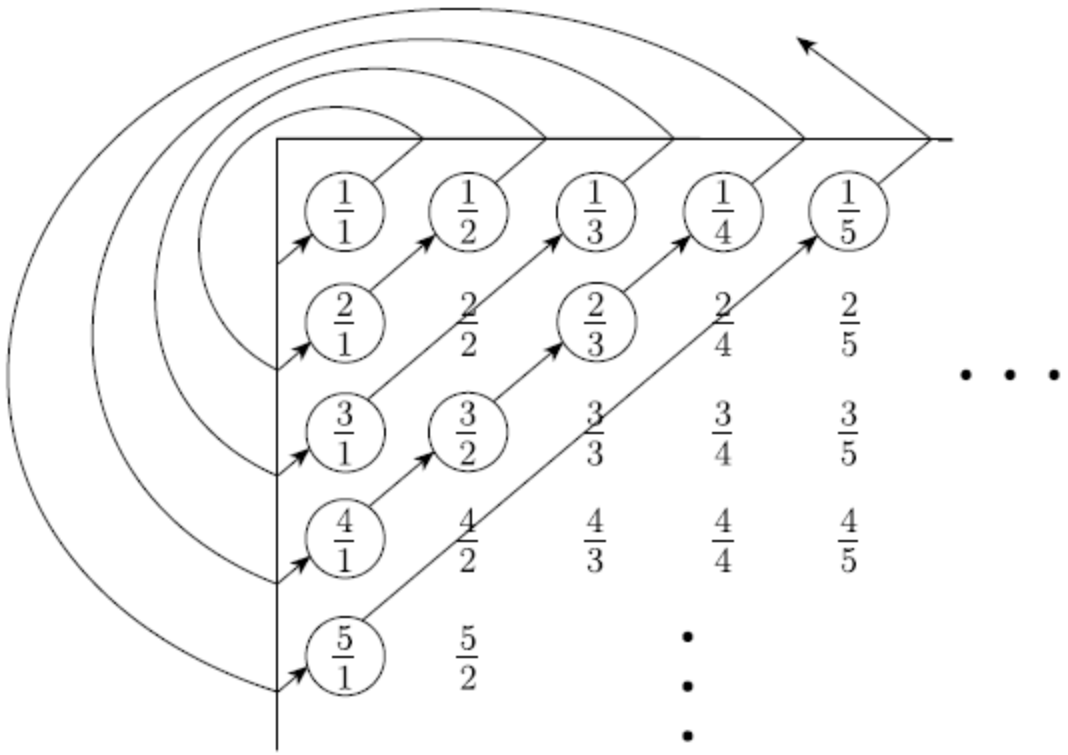
Some basic definitions

- Let A and B be any two sets (not necessarily finite) and f be a function from A to B.
- f is one-to-one if $f(a) \neq f(b)$ whenever $a \neq b$.
- f is onto if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$.
- We say A and B are the same size if there is a one-to-one and onto function $f : A \longrightarrow B$.
- Such a function is called a correspondence for pairing A and B.
  - Every element of A maps to a unique element of B
  - Each element of B has a unique element of A mapping to it.

The Diagonalization Method

- Let $\mathcal{N}$ be the set of natural numbers $\{1, 2, \ldots\}$ and let $\mathcal{E}$ be the set of even numbers $\{2, 4, \ldots\}$.
- $f(n) = 2n$ is a correspondence between $\mathcal{N}$ and $\mathcal{E}$.
- Hence, $\mathcal{N}$ and $\mathcal{E}$ have the same size (though $\mathcal{E} \subset \mathcal{N}$).
- A set A is countable if it is either finite or has the same size as $\mathcal{N}$.
- $\mathcal{Q} = \{\frac{m}{n} \mid m, n \in \mathcal{N}\}$ is countable!
- Z the set of integers is countable:

$$f(n) = \begin{cases} \frac{n}{2} & n \text{ even} \\ -\frac{n+1}{2} & n \text{ odd} \end{cases}$$


1/1 1/2 1/3 1/4 1/5
2/1 2/2 2/3 2/4 2/5
3/1 3/2 3/3 3/4 3/5
4/1 4/2 4/3 4/4 4/5
5/1 5/2

The Diagonalization Method

Theorem

$\mathcal{R}$ is uncountable

Proof.

- Assume f exists and every number in $\mathcal{R}$ is listed.
- Assume $x \in \mathcal{R}$ is a real number such that x differs from the j^{th} number in the j^{th} decimal digit.
- If x is listed at some position k, then it differs from itself at k^{th} position; otherwise the premise does not hold
- f does not exist

n	$f(n)$
1	3.14159...
2	55.77777...
3	0.12345...
4	0.50000...
⋮	⋮

$x = .4527\ldots$ defined as such, can not be on this list. □

- Pi: “ it keeps on going, forever, without ever repeating. Which means that contained within this string of decimals, is every single other number. Your birthdate, combination to your locker, your social security number, it's all in there, somewhere. And if you convert these decimals into letters, you would have every word that ever existed in every possible combination; the first syllable you spoke as a baby, the name of your latest crush, your entire life story from beginning to end, everything we ever say or do; all of the world's infinite possibilities rest within this one simple circle.”

Diagonalization over Languages

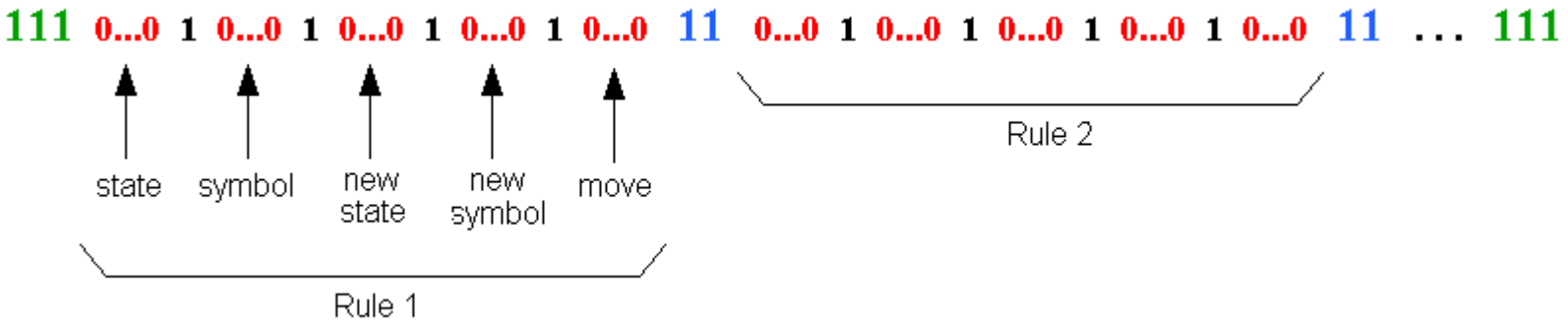
Corollary

Some languages are not Turing-recognizable.

Proof

- For any alphabet Σ, Σ^* is countable. Order strings in Σ^* by length and then alphanumerically, so $\Sigma^* = \{s_1, s_2, \ldots, s_i, \ldots\}$
- The set of all TMs is a countable language.
  - Each TM M corresponds to a string $\langle M \rangle$.
  - Generate a list of strings and remove any strings that do not represent a TM to get a list of TMs.

Encoding of a TM



Diagonalization over Languages

Proof (continued)

- The set of infinite binary sequences, $\mathcal{B}$, is uncountable. (Exactly the same proof we gave for uncountability of $\mathcal{R}$)
- Let $\mathcal{L}$ be the set of all languages over Σ.
- For each language $A \in \mathcal{L}$ there is unique infinite binary sequence χ_A
  - The i^{th} bit in χ_A is 1 if $s_i \in A$, 0 otherwise.

$\Sigma^* = \{$	$\epsilon,$	$0,$	$1,$	$00,$	$01,$	$10,$	$11,$	$000,$	$001,$	$\cdots$	$\}$
$A = \{$		$0,$		$00,$	$01,$			$000,$	$001,$	$\cdots$	$\}$
$\chi_A = \{$	0	1	0	1	1	0	0	1	1	$\cdots$	$\}$

Diagonalization over Languages

Proof (continued)

- The function $f : \mathcal{L} \longrightarrow \mathcal{B}$ is a correspondence. Thus $\mathcal{L}$ is uncountable.
- So, there are languages that can not be recognized by some TM. There are not enough TMs to go around.

A_{TM} is undecidable

THEOREM

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$, *is undecidable.*

PROOF

- We assume A_{TM} is decidable and obtain a contradiction.
- Suppose H decides A_{TM}

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

A_{TM} is undecidable

PROOF (CONTINUED)

- We now construct a new TM D
  $D =$ "On input $\langle M \rangle$, where M is a TM
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H accepts, *reject*, if H rejects, *accept*"
- So

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

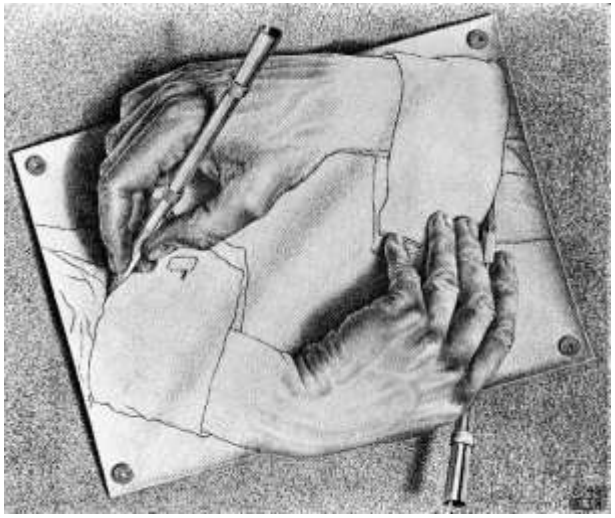
- When D runs on itself we get

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

- Neither D nor H can exist.

Of Paradoxes & Strange Loops

E.g., Barber's paradox, Achilles & the Tortoise (Zeno's paradox)
MC Escher's paintings



A fun book for further reading:

"Godel, Escher, Bach: An Eternal Golden Braid"
by Douglas Hofstadter (Pulitzer winner, 1980)

What Happened to Diagonalization?

Consider the behaviour of all possible deciders:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$		$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$		$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$		$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$		$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$		$\cdots$
$\vdots$		$\vdots$			$\ddots$		
							$\cdots$
$\vdots$		$\vdots$					$\ddots$

What Happened to Diagonalization?

Consider the behaviour of all possible deciders:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$		$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$		$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$		$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$		$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$		$\cdots$
$\vdots$		$\vdots$			$\ddots$		
							$\cdots$
$\vdots$		$\vdots$					$\ddots$

- D computes the opposite of the diagonal entries!

What Happened to Diagonalization?

Consider the behaviour of all possible deciders:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$ $\langle M_j \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
$\vdots$		$\vdots$			$\ddots$		
$D = M_j$	*reject*	*reject*	*accept*	*accept*	$\cdots$	?	$\cdots$
$\vdots$		$\vdots$					$\ddots$

- D computes the opposite of the diagonal entries!

What Happened to Diagonalization?

Consider the behaviour of all possible deciders:

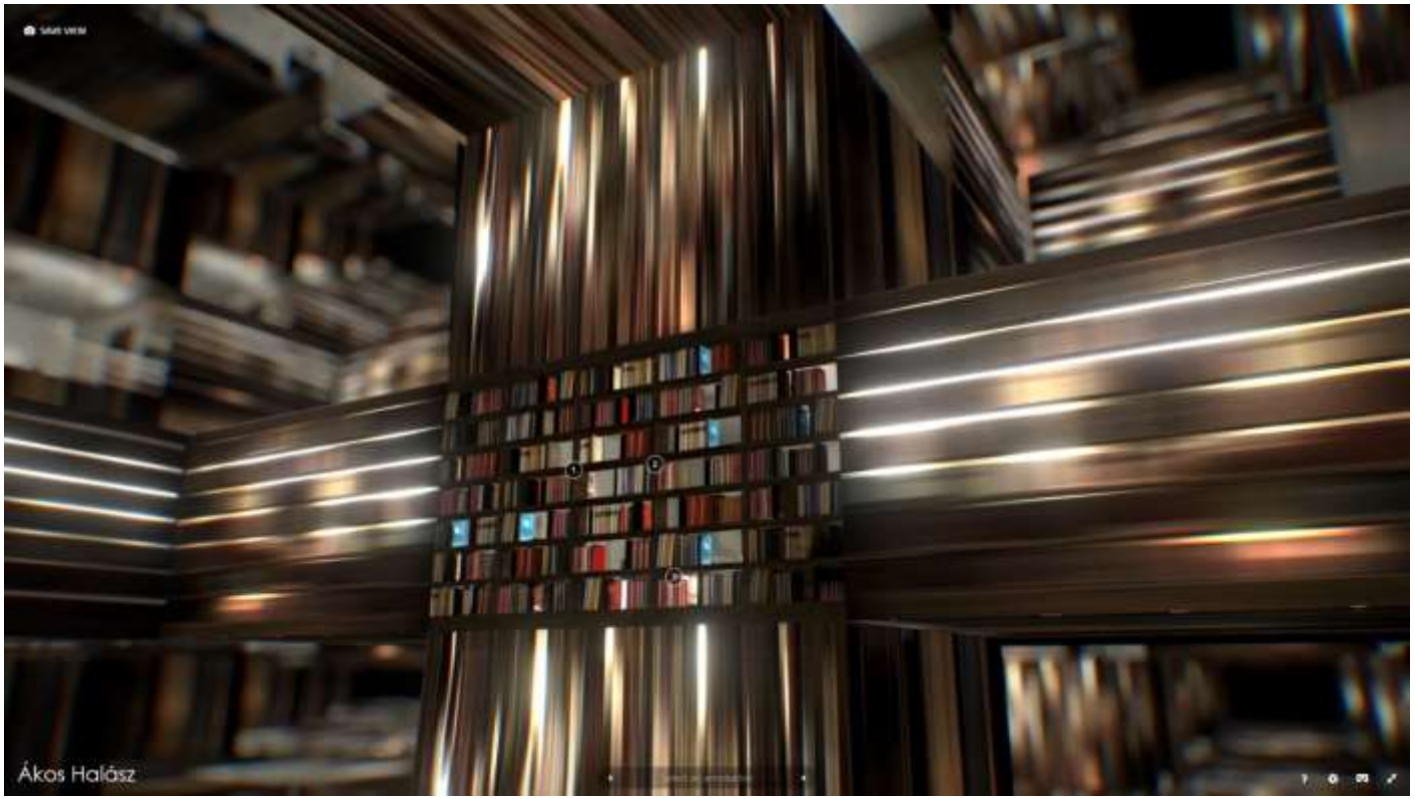
	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$ $\langle M_j \rangle$	$\cdots$
M_1	accept	reject	accept	reject	$\cdots$	accept	$\cdots$
M_2	accept	accept	accept	accept	$\cdots$	accept	$\cdots$
M_3	reject	reject	reject	reject	$\cdots$	reject	$\cdots$
M_4	accept	accept	reject	reject	$\cdots$	accept	$\cdots$
$\vdots$		$\vdots$			$\ddots$		
$D = M_j$	reject	reject	accept	accept	$\cdots$	?	$\cdots$
$\vdots$		$\vdots$					$\ddots$

- D computes the opposite of the diagonal entries!

D cannot exist; After all D used only H; So if H exists then D exists; Hence H (decider for A_{TM}) cannot exist.

- Is Computer Technology stagnant?


AMY ADAMS
JEREMY RENNER
FOREST WHITAKER
WHY ARE THEY HERE?
FROM THE DIRECTOR OF SICARIO AND PRISONERS
ARRIVAL
IN THEATRES 11.11


Ákos Halász


MATTHEW McCONAUGHEY
ANNE HATHAWAY
JESSICA CHASTAIN
MICHAEL CAINE AND
GO FURTHER.
FROM THE DIRECTOR OF THE DARK KNIGHT TRILOGY AND INCEPTION
INTERSTELLAR
IN THEATRES AND IMAX
EVERYWHERE
NOVEMBER 7

Ludwig Wittgenstein

the limits of my language are the limits of my world

- *There are things in existence that are beyond our human ability to imagine or conceive*

https://oregonstate.edu/instruct/phl201/modules/Philosophers/Wittgenstein/wittgenstein.html

Sapir–Whorf hypothesis

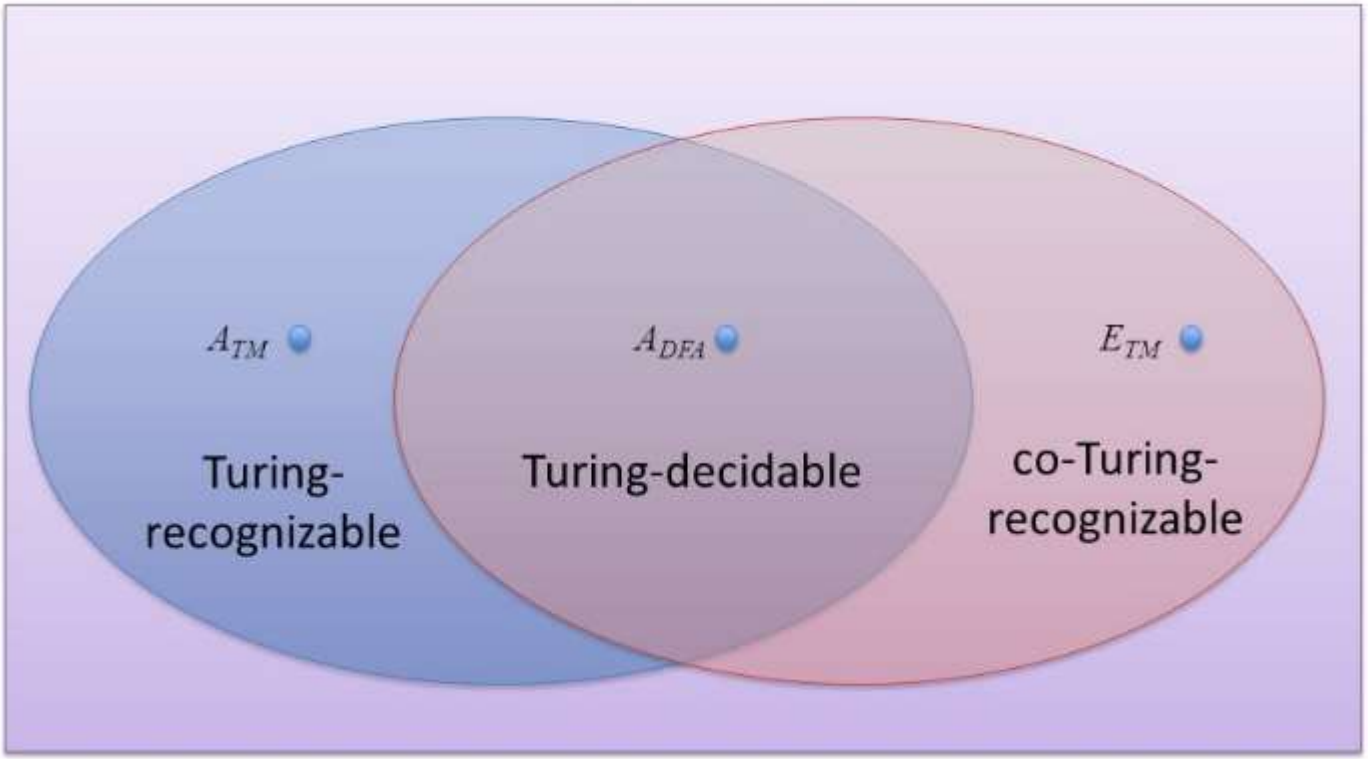
Linguistic relativity

The language you speak determines how you think

https://en.wikipedia.org/wiki/Linguistic_relativity

A Turing Unrecognizable Language

- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.
- A language is decidable if it is Turing-recognizable and co-Turing-recognizable.
- $\overline{A_{TM}}$ is not Turing recognizable.
  - We know A_{TM} is Turing-recognizable.
  - If $\overline{A_{TM}}$ were also Turing-recognizable, A_{TM} would have to be decidable.
  - We know A_{TM} is not decidable.
  - $\overline{A_{TM}}$ must not be Turing-recognizable.



- Can you locate where $\overline{A_{TM}}$ will be?
- Can you locate $\overline{E_{TM}}$ will be?

Preview …

- Are there languages which are neither RE nor co-RE ?? (i.e. both the language and its complement are not in RE).

Preview …

- Are there languages which are neither RE nor co-RE ??
- Yes. $EQ_{TM} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$.
- But, to understand this we need the concept called "mapping reducibility" which is a binary relation between languages and is represented $\leq_m$

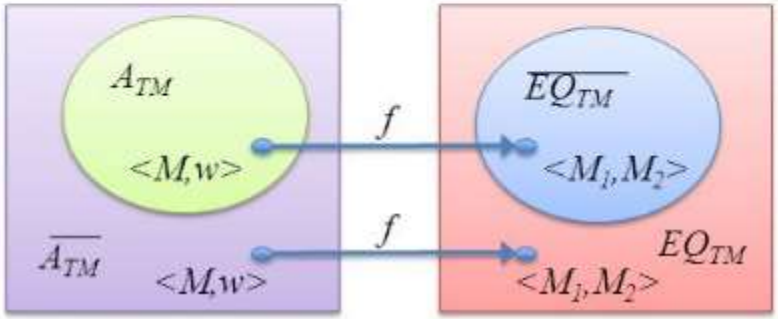
Preview …

- Are there languages which are neither RE nor co-RE ??

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable

- Proof:

Show $A_{TM} \leq_m \overline{EQ_{TM}}$

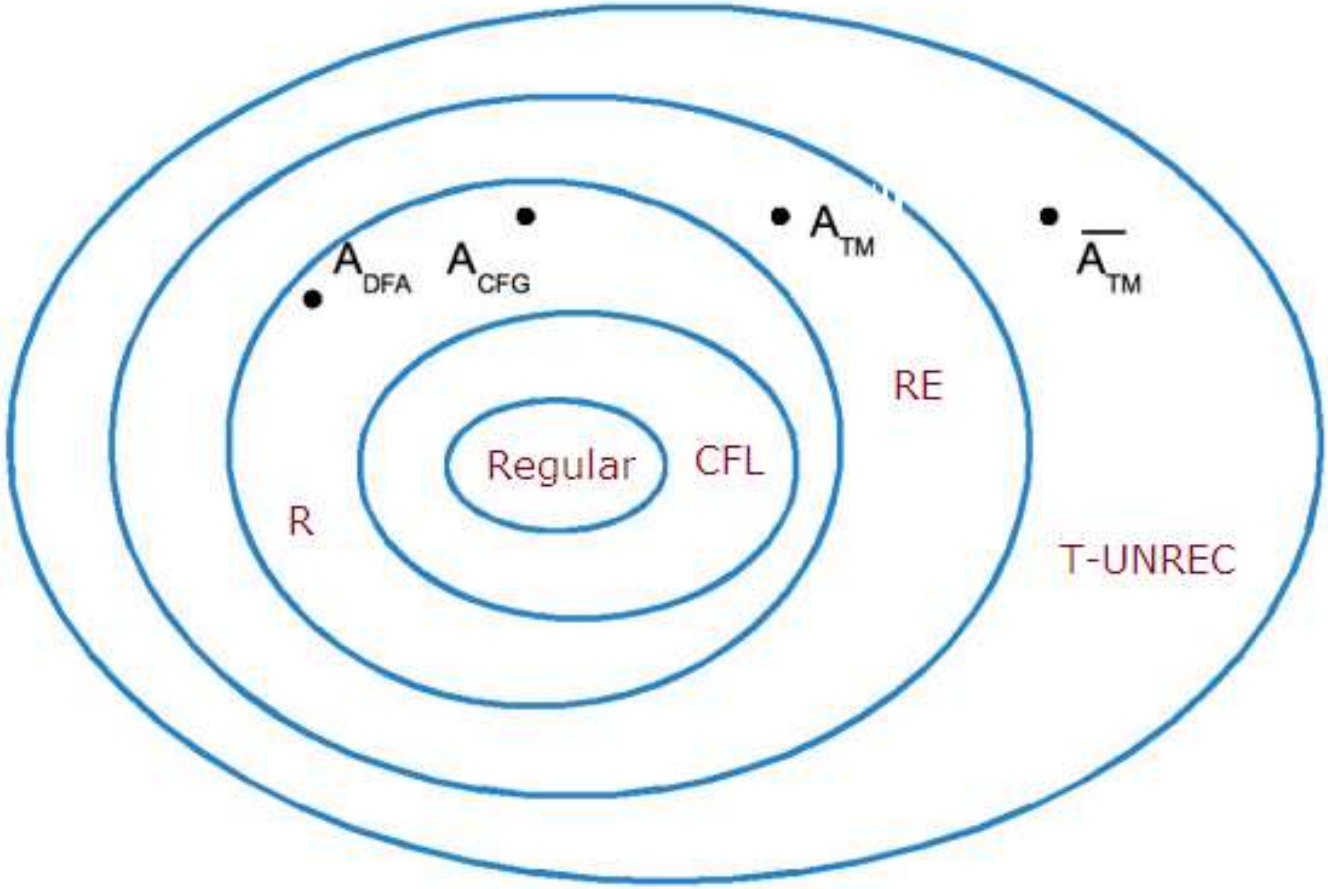


Reducibility

A way to show some languages are undecidable !

https://www.andrew.cmu.edu/user/ko/pdfs/lecture-16.pdf

The Landscape of the Chomsky Hierarchy

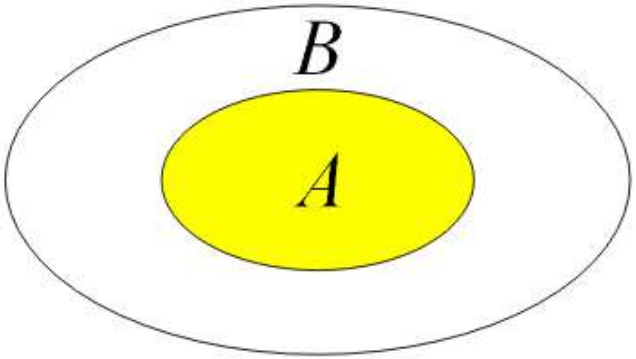


REDUCIBILITY

- A reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.
  - Finding the area of a rectangle, reduces to measuring its width and height
  - Solving a set of linear equations, reduces to inverting a matrix.

Problem A is reduced to problem B

If we can solve problem B then we can solve problem A.



A reduces to B

- $A \leq B$
- Find area of a rectangle $\leq$ find length and find width of rectangle.
- Solving B means you know how to solve the fundamental ingredients (which are needed to solve A).
- A solution to B can be used to solve A.
- Note, a solution to A may not be enough to solve B.
  - Knowing area of a rectangular is not enough to find its length and width !!

A reduces to B

- $A \leq B$
- Solving B means you know how to solve the fundamental ingredients (which are needed to solve A).
- A solution to B can be used to solve A.
- An algorithm that solves B can be converted to an algorithm that solves A

A reduces to B

- $A \leq B$
- If B is decidable, then so is A.
- Contrapositive, if A is undecidable then so is B.

Problem A is reduced to problem B

If B is decidable then A is decidable.

If A is undecidable then B is undecidable.

Proving Undecidability via Reductions

Theorem 5.1

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$ is undecidable.

Proving Undecidability via Reductions

Theorem 5.1

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$ is undecidable.

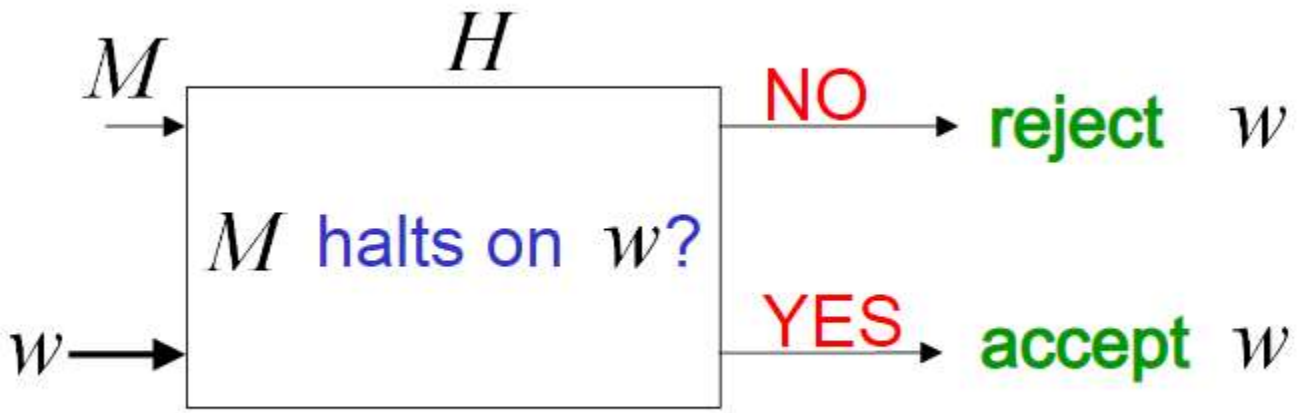
- We show that A_{TM} is reducible to $HALT_{TM}$
- Since A_{TM} is undecidable, so is $HALT_{TM}$

Proving Undecidability via Reductions

Theorem 5.1

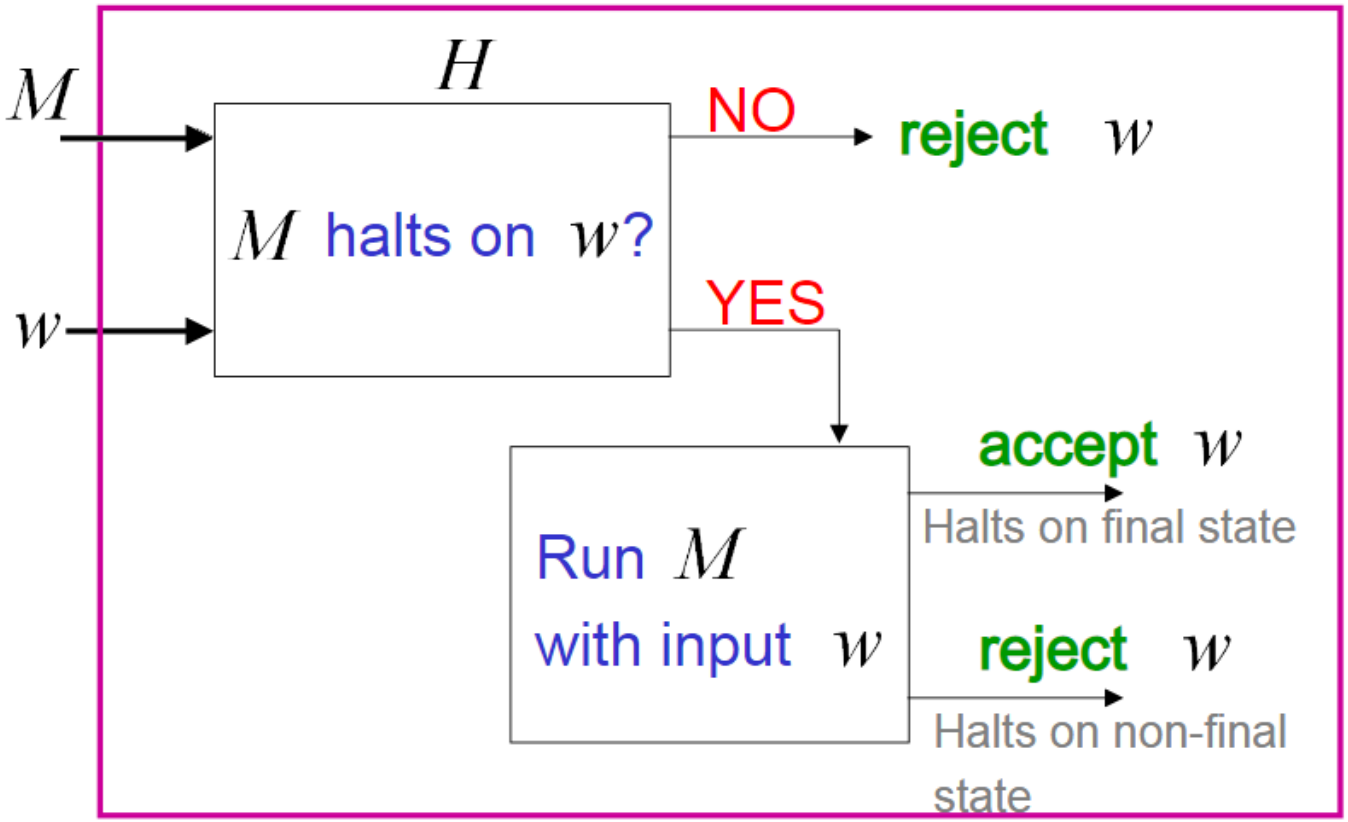
$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$ is undecidable.

- Suppose $HALT_{TM}$ is decidable, this means

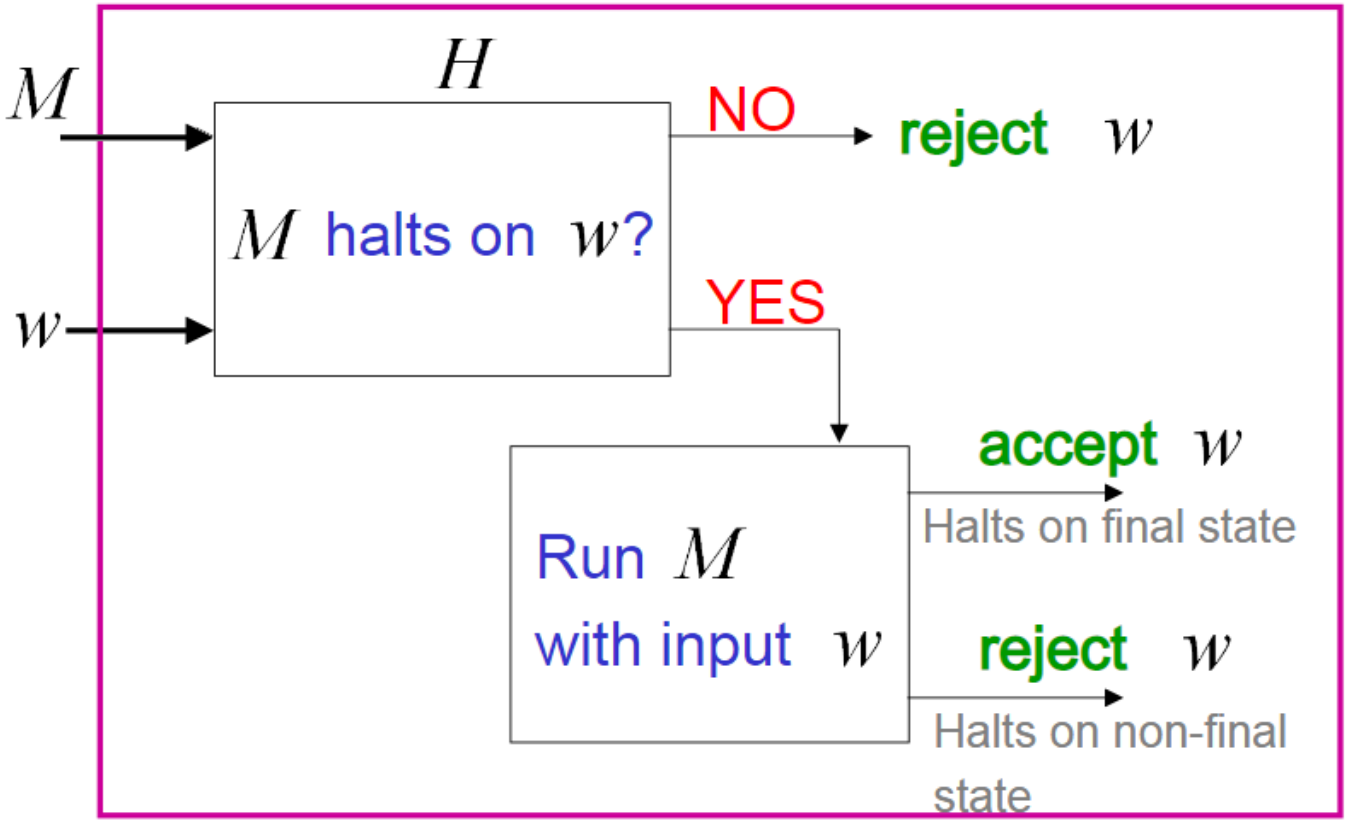


- Then A_{TM} is decidable.

- Then A_{TM} is decidable.



- Then A_{TM} is decidable.



- **Contradiction**

This diagram shows how A_{TM} can be reduced to $HALT_{TM}$

Proving Undecidability via Reductions

Theorem 5.2

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Phi\}$ is undecidable.

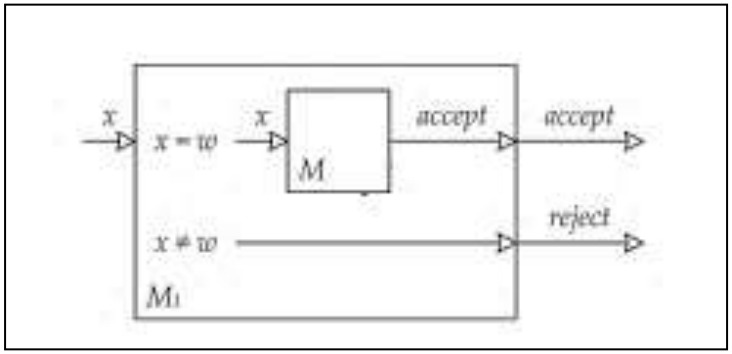
- A decider for A_{TM} via E_{TM} is possible.
- We are given $< M, w >$, and asked to find whether $< M, w > \in A_{TM}$?
- For this, First create M_1

Proving Undecidability via Reductions

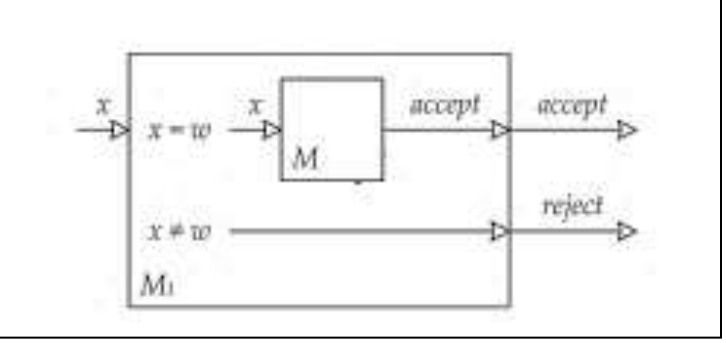
Theorem 5.2

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Phi\}$ is undecidable.

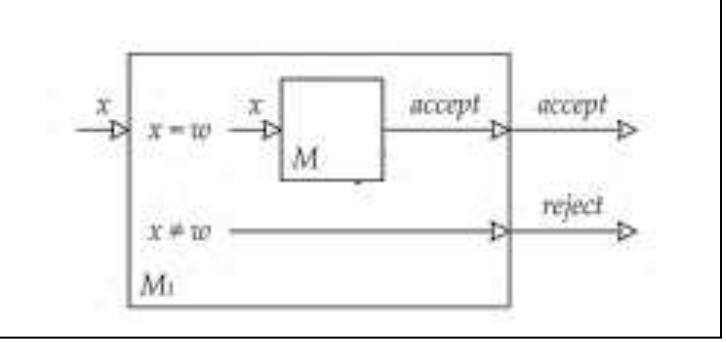
- A decider for A_{TM} via E_{TM} is possible.
- We are given $< M, w >$, and asked to find whether $< M, w > \in A_{TM}$?
- For this, First create M_1



Now, $L(M_1)$ is what?



- Now, $L(M_1)$ is what?



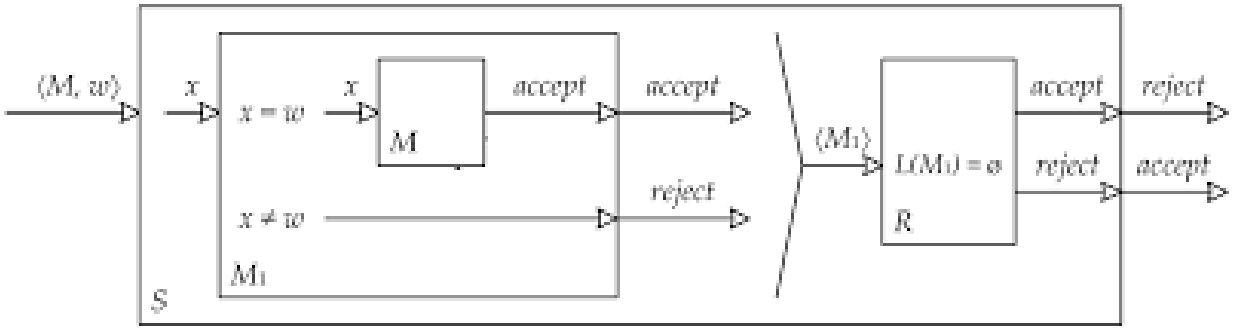
- Now, $L(M_1)$ is what?
- $L(M_1)$ is either $\{w\}$ or is ϕ
- Now, if M_1 is in E_{TM}
  - This means, $L(M_1) = \phi$
  - This means, $< M, w > \notin A_{TM}$
- Now, if M_1 is not in E_{TM}
  - This means, $L(M_1) \neq \phi$
  - This means, $< M, w > \in A_{TM}$

Proving Undecidability via Reductions

Theorem 5.2

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Phi\}$ is undecidable.

- A decider for A_{TM} via E_{TM} is possible.



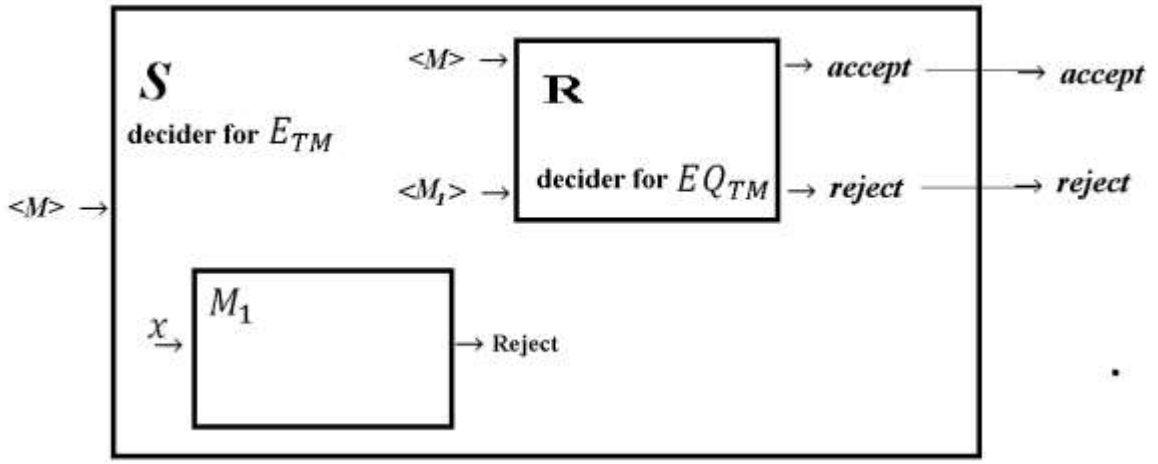
- That is, A_{TM} can be reduced to E_{TM}.
- Contradiction

TESTING FOR LANGUAGE EQUALITY

THEOREM 5.4

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ is undecidable.

- Since, we know E_{TM} is undecidable,
- We can try to reduce E_{TM} to EQ_{TM}
- Let R be a decider for EQ_{TM}
- We can build a decider (call this S) for E_{TM} by using R



This is a decider for E_{TM}

PROOF We let TM R decide EQ_{TM} and construct TM S to decide E_{TM} as follows.

S = "On input $\langle M \rangle$, where M is a TM:

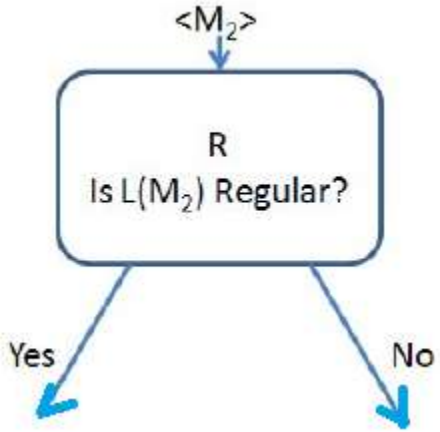
1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, *accept*; if R rejects, *reject*."

If R decides EQ_{TM}, S decides E_{TM}. But E_{TM} is undecidable by Theorem 5.2, so EQ_{TM} also must be undecidable.

Testing for Regularity

$REGULAR_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language } \}$ is undecidable.

- If $REGULAR_{TM}$ is decidable, then this can be used to decide A_{TM} .
- Let R be a decider for $REGULAR_{TM}$.

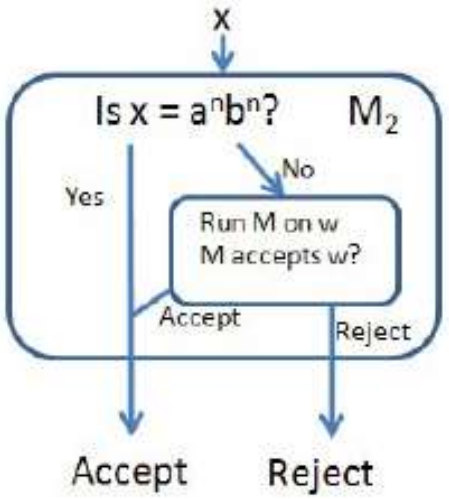


How R can be used to decide $\langle M,w \rangle \in A_{TM}$?

- Build M_2 as shown

We design M_2 to recognize the nonregular language $\{0^n1^n \mid n \geq 0\}$ if M does not accept w, and to recognize the regular language Σ^* if M accepts w.
We must specify how S can construct such an M_2 from M and w.

Here, M_2 works by automatically accepting all strings in $\{0^n1^n \mid n \geq 0\}$. In addition, if M accepts w, M_2 accepts all other strings.

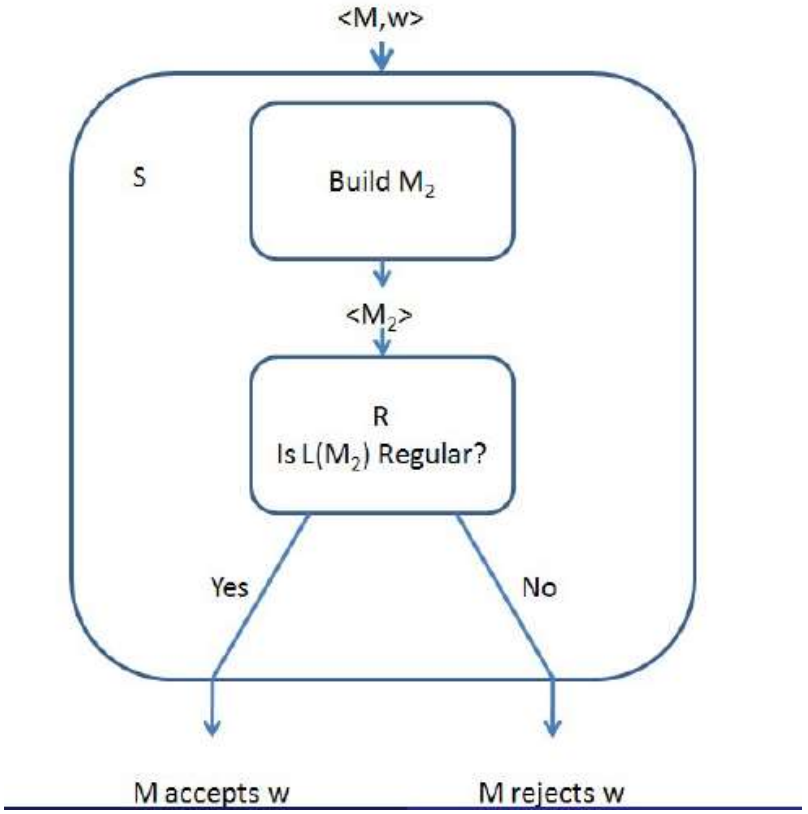


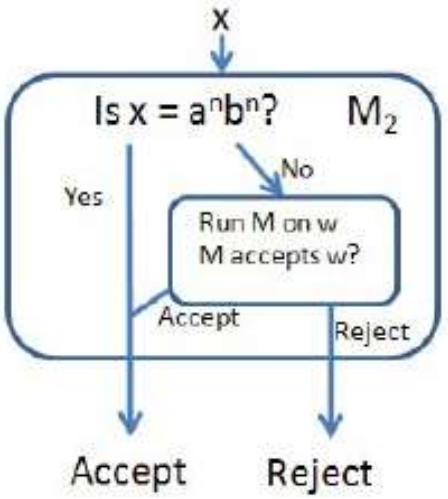
So $L(M_2)$ is = Σ^* if M accepts w
$L(M_2)$ is = $\{a^nb^n\}$ otherwise

We assume that $REGULAR_{TM}$ is decidable by a TM R and use this assumption to construct a TM S that decides A_{TM}.

How R can be used to decide $\langle M,w \rangle \in A_{TM}$?

- Build M_2 as shown





So $L(M_2)$ is = Σ^* if M accepts w
$L(M_2)$ is = $\{a^n b^n\}$ otherwise

We assume that REGULAR_{TM} is decidable by a TM R and use this assumption to construct a TM S that decides A_{TM}.

- Similar to $REGULAR_{TM}$, we can show CFL_{TM} is undecidable.
  - That is, finding whether a TM's language is CFL or not is undecidable.
  - In fact, we can extend this. TM's language is finite or not is undecidable.
  - General theorem in this regard is called ***The Rice's Theorem.***

More formal way of reductions

MAPPING REDUCTION

WE CAN GET MORE REFINED ANSWERS

Computable function

- **DEFINITION 5.17**

  A function $f: \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

- For example,

  we can make a machine that takes input $\langle m, n \rangle$ and returns $m + n$, the sum of m and n.

Mapping Reductions

Definition: Let A and B be two languages. We say that there is a mapping reduction from A to B, and denote

$$A \leq_m B$$

if there is a computable function

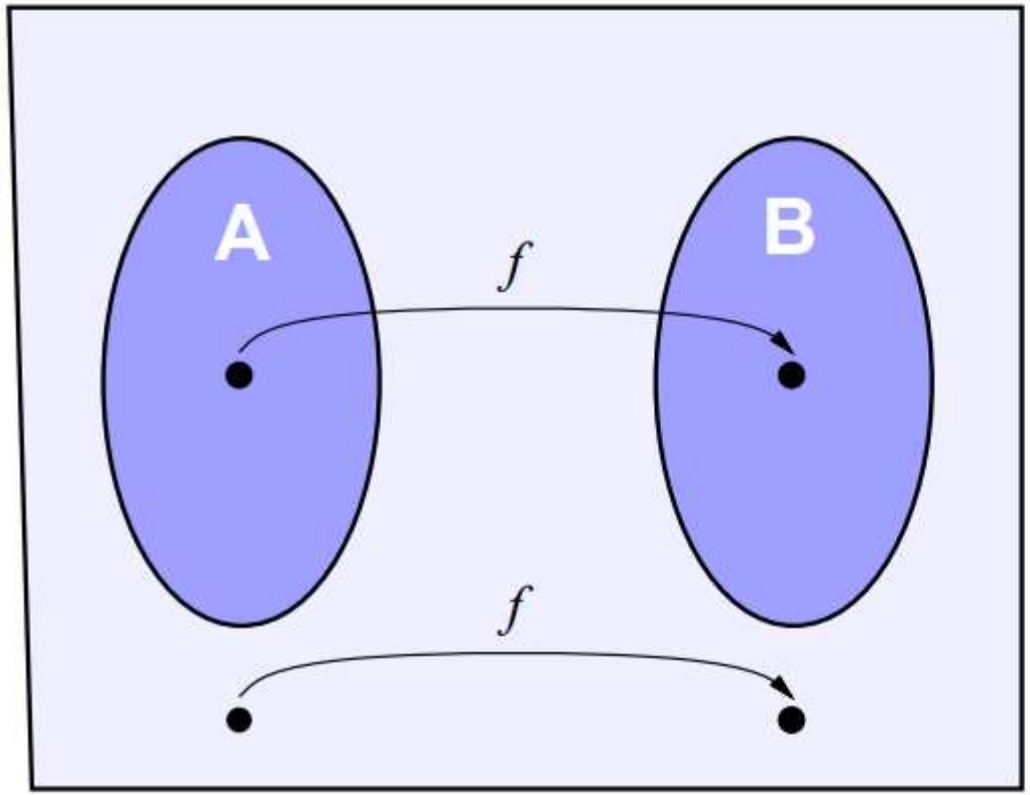
$$f : \Sigma^* \longrightarrow \Sigma^*$$

such that, for every w,

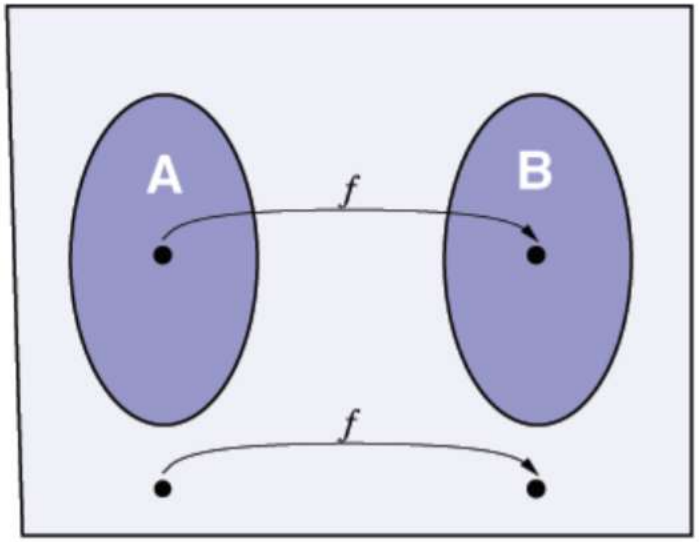
$$w \in A \Longleftrightarrow f(w) \in B.$$

The function f is called the reduction from A to B.

Mapping Reductions



A mapping reduction converts questions about membership in A to membership in B



A mapping reduction converts questions about membership in A to membership in B

Notice that $A \leq_m B$ implies $\overline{A} \leq_m \overline{B}$.

Mapping Reductions

Theorem: If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: Let

- M be the decider for B, and
- f the reduction from A to B.

Define N: On input w

1. compute $f(w)$
2. run M on input $f(w)$ and output whatever M outputs.

Mapping Reductions

Corollary: If $A \leq_m B$ and A is undecidable, then B is undecidable.

In fact, this has been our principal tool for proving undecidability of languages other than A_{TM}.

Example: Halting

Recall that

$$A_{\text{TM}} = \{\langle M, w\rangle | \text{TM } M \text{ accepts input } w\}$$
$$H_{\text{TM}} = \{\langle M, w\rangle | \text{TM } M \text{ halts on input } w\}$$

Earlier we proved that

- H_{TM} undecidable
- by (de facto) reduction from A_{TM}.

Let's reformulate this.

Example: Halting

Define a computable function, f:

- input of form $\langle M, w \rangle$
- output of form $\langle M', w' \rangle$
- where $\langle M, w \rangle \in A_{\mathrm{TM}} \iff \langle M', w' \rangle \in H_{\mathrm{TM}}$.

Example: Halting

The following machine computes this function f.

$F =$ on input $\langle M, w \rangle$:

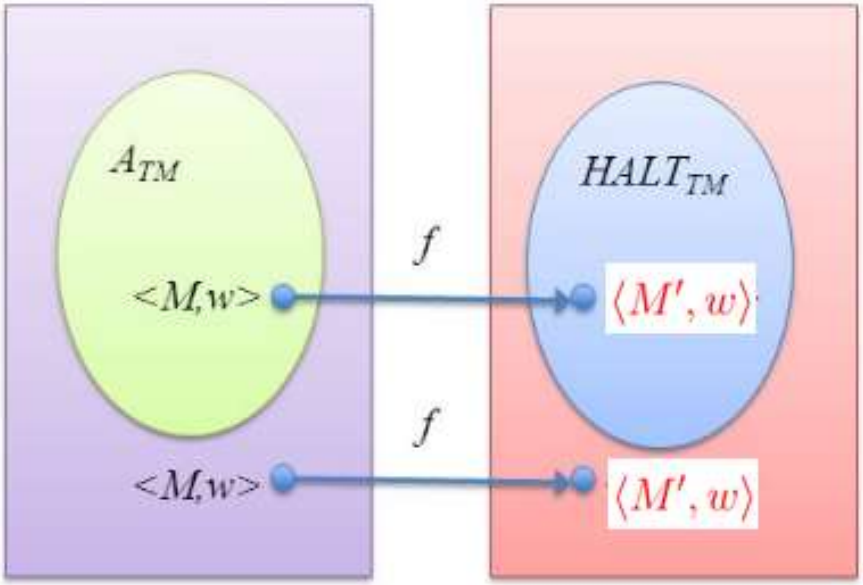
- Construct the following machine M'.

  M': on input x
  - run M on x
  - If M accepts, *accept*.
  - if M rejects, enter a loop.
- output $\langle M', w \rangle$

A_{TM} = {<M,w> | M is a TM and M accepts w}

$\leq_m$

$HALT_{TM}$ = {<M,w> | M is a TM & M halts on input w}



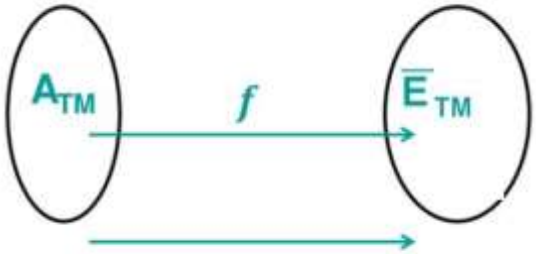
$A_{TM} \leq_m \overline{E_{TM}}$

- $A_{TM} = \{< M, w > | M \text{ is a TM that accepts } w\}$
- $\overline{E_{TM}} = \{< M > | L(M) \neq \phi\}$
- $f: \Sigma^* \rightarrow \Sigma^*$ can be defined as

Create M' : On input x,

if $x \neq w$, output "Reject";

if $x = w$, run w on M, output the result.



More Theorems Re-examined: E_{TM}

Original Method:

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M)=\varnothing\}$

M_1 = "On input x:

1. If $x \neq w$, reject.
2. If $x = w$, run M on input w and accept if M does."

Assume that TM R decides E_{TM} and construct TM S that decides A_{TM} as follows.

S = "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Use the description of M and w to construct the TM M_1 just described.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, reject; if R rejects, accept."

Mapping Reduction:

Problem: The mapping in the proof is actually A_{TM} to $\neg E_{TM}$ (pay attention to the negation).

Notice: Decidability is not affected by complementation. But can we create a pure mapping reduction?

Proof that a Mapping Reduction is impossible: Suppose for a contradiction that $A_{TM} \leq_m E_{TM}$ via reduction f. It follows from the definition of mapping reducibility that $\neg A_{TM} \leq_m \neg E_{TM}$ via the same reduction function f. However, $\neg E_{TM}$ (Exercise 4.5) is Turing-recognizable and $\neg A_{TM}$ is not Turing-recognizable.

The sensitivity of mapping reducibility to complementation is important in the use of reducibility to prove nonrecognizability of certain languages.

We can also use mapping reducibility to show that problems are not Turingrecognizable.

THEOREM 5.28

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

COROLLARY 5.29

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Mapping Reductions: Reminders

Theorem 1:

If $A \leq_m B$ and B is decidable, then A is decidable.

Theorem2 :

If $A \leq_m B$ and B is recursively enumerable, then A is recursively enumerable.

Mapping Reductions: Corollaries

Corollary 1: If $A \leq_m B$ and A is undecidable, then B is undecidable.

Corollary 2: If $A \leq_m B$ and A is not in $\mathcal{RE}$, then B is not in $\mathcal{RE}$.

Corollary 3: If $A \leq_m B$ and A is not in $co\mathcal{RE}$, then B is not in $co\mathcal{RE}$.

THEOREM 5.30

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

PROOF First we show that EQ_{TM} is not Turing-recognizable. We do so by showing that A_{TM} is reducible to $\overline{EQ_{\mathsf{TM}}}$. The reducing function f works as follows.

F = "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.
   M_1 = "On any input:
      1. *Reject*."
   M_2 = "On any input:
      1. Run M on w. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

Here, M_1 accepts nothing. If M accepts w, M_2 accepts everything, and so the two machines are not equivalent. Conversely, if M doesn't accept w, M_2 accepts nothing, and they are equivalent. Thus f reduces A_{TM} to $\overline{EQ_{\mathsf{TM}}}$, as desired.

Key: We know that $\overline{A_{TM}}$ is not enumerable

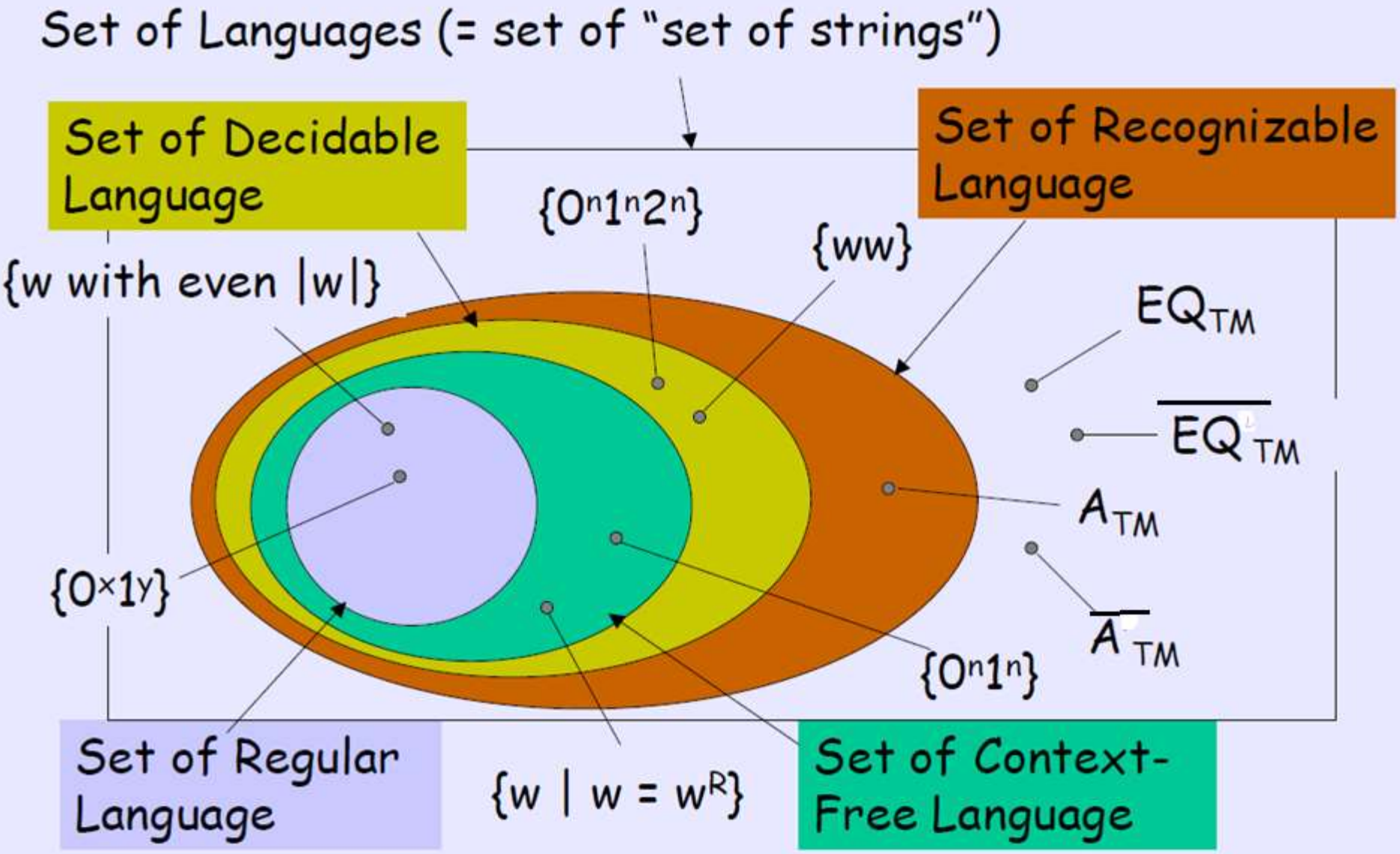
To show that $\overline{EQ_{\mathsf{TM}}}$ is not Turing-recognizable, we give a reduction from A_{TM} to the complement of $\overline{EQ_{\mathsf{TM}}}$—namely, EQ_{TM}. Hence we show that $A_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$. The following TM G computes the reducing function g.

G = "On input $\langle M, w \rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.
   M_1 = "On any input:
      1. *Accept*."
   M_2 = "On any input:
      1. Run M on w.
      2. If it accepts, *accept*."
2. Output $\langle M_1, M_2 \rangle$."

The only difference between f and g is in machine M_1. In f, machine M_1 always rejects, whereas in g it always accepts. In both f and g, M accepts w iff M_2 always accepts. In g, M accepts w iff M_1 and M_2 are equivalent. That is why g is a reduction from A_{TM} to EQ_{TM}.

Language Hierarchy (revisited)



Non Trivial Properties of $\mathcal{RE}$ Languages

A few examples

- L is finite.
- L is infinite.
- L contains the empty string.
- L contains no prime number.
- L is co-finite.
- . . .

All these are non-trivial properties of enumerable languages, since for each of them there is $L_1, L_2 \in \mathcal{RE}$ such that L_1 satisfies the property but L_2 does not.

Are there any trivial properties of $\mathcal{RE}$ languages?

•A property is called to be trivial if either it is not satisfied by any recursively enumerable languages, or if it is satisfied by all recursively enumerable languages.

•A non-trivial property is satisfied by some recursively enumerable languages and are not satisfied by others.

Rice's Theorem

Theorem Let $\mathcal{C}$ be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in $\mathcal{C}$. Then $L_{\mathcal{C}}$ is undecidable.

(See problem 5.22 in Sipser's book)

Proof by reduction from A_{TM}.

Given M and w, we will construct M_0 such that:

- If M accepts w, then $\langle M_0 \rangle \in L_{\mathcal{C}}$.
- If M does not accept w, then $\langle M_0 \rangle \notin L_{\mathcal{C}}$.

http://www.cs.tau.ac.il/~bchor/CM09/Compute9.pdf

http://www.cs.tau.ac.il/~bchor/CM09/Compute9.pdf

Has (towards end) some good slides on Rice's theorem and its consequences.

Post Correspondence Problem

- Undecidability is not just confined to problems concerning automata and languages.
- There are other "natural" problems which can be proved undecidable.
- The Post correspondence problem (PCP) is a tiling problem over strings.
- A tile or a domino contains two strings, t and b; e.g., $\left[\frac{ca}{a}\right]$.
- Suppose we have dominos

$$\left\{\left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right]\right\}$$

- A match is a list of these dominos so that when concatenated the top and the bottom strings are identical. For example,

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right] = \frac{abcaaabc}{abcaaabc}$$

- The set of dominos $\left\{ \left[\frac{abc}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{acc}{ba}\right], \right\}$ does not have a solution.

Post Correspondence Problem

An instance of the PCP

A PCP instance over Σ is a finite collection P of dominos

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\}$$

where for all $i, 1 \leq i \leq k, t_i, b_i \in \Sigma^*$.

Match

Given a PCP instance P, a match is a nonempty sequence

$$i_1, i_2, \ldots, i_\ell$$

of numbers from $\{1, 2, \ldots, k\}$ (with repetition) such that
$t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell}$

Post Correspondence Problem

An instance of the PCP

A PCP instance over Σ is a finite collection P of dominos

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\}$$

where for all $i, 1 \leq i \leq k, t_i, b_i \in \Sigma^*$.

Match

Given a PCP instance P, a match is a nonempty sequence

$$i_1, i_2, \ldots, i_\ell$$

of numbers from $\{1, 2, \ldots, k\}$ (with repetition) such that
$t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell}$

Question:

Does a given PCP instance P have a match?

Post Correspondence Problem

Question:

Does a given PCP instance P have a match?

Language formulation:

$PCP = \{\langle P \rangle \mid P \text{ is a PCP instance and it has a match}\}$

Theorem 5.15

PCP is undecidable.

Post Correspondence Problem

Question:

Does a given PCP instance P have a match?

Language formulation:

$PCP = \{\langle P \rangle \mid P \text{ is a PCP instance and it has a match}\}$

Theorem 5.15

PCP is undecidable.

Proof: By reduction using computation histories. If PCP is decidable then so is A_{TM}. That is, if PCP has a match, then M accepts w.

- That is, A_{TM} can be reduced to PCP.
- This reduction is via a simplified PCP called $MPCP$ (modified PCP).

- Several undecidable properties of CFGs are obtained by reducing them (i.e., the corresponding languages) from PCP.

RE

- Closed under union
- Closed under intersection
- **Not** closed under complementation

R

- Closed under union
- Closed under intersection
- Closed under complementation.