Prof. Ganesh Ramakrishna

Handouts on:

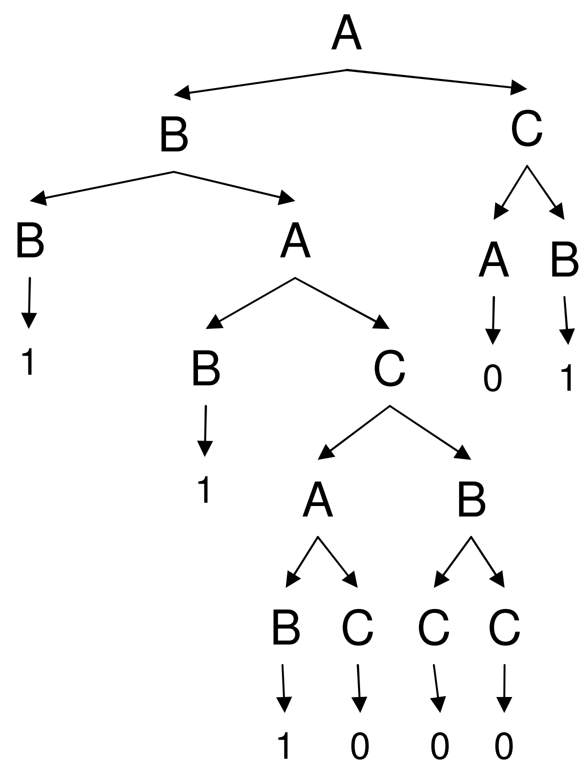
Pumping Lemma, CYK algorithm, Undecidability, Reduction and Turing Machines

Pumping Lemma [Bar Hillel lemma]

- For FSMs we used a diagonalization tool called the pumping lemma
- Similar trick for PDMs may not work
  - If 1101001 is accepted by an FSM, then 1101001001001001 should also be accepted
  - But same does not hold for PDMs, because the transitions are determined not only based on tape symbol but also based on symbol on top of the stack.
- Ironically, you look at pumping lemma for Context free languages through the grammars (CFGs) rather than the machine equivalents (PDMs)

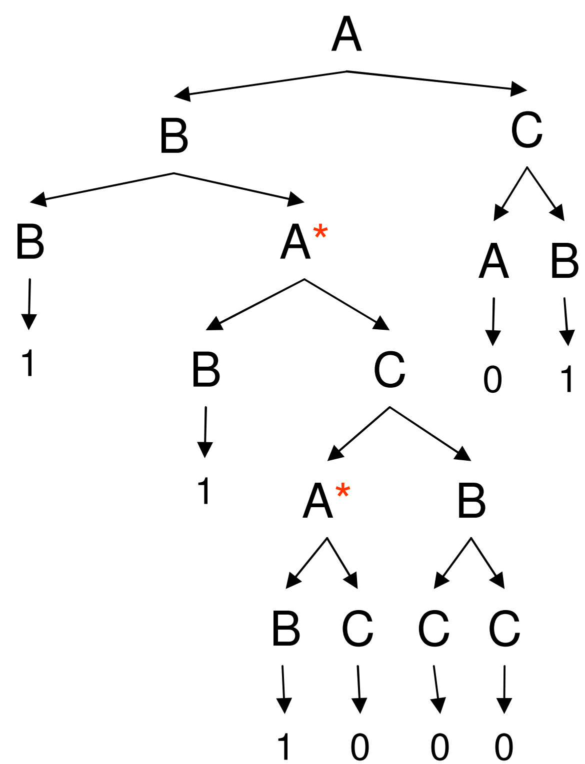
Pumping lemma and parse trees

- Consider the grammar
  - A → BC | 0
  - B → BA | 1 | CC
  - C → AB | 0
- Consider the string 11100001 and the corresponding parse tree



Loop in the parse tree

- Since there are only 3 non-terminals, if we find a path of length >= 4 starting from a leaf, we will find a duplicate
  - The first non-terminal that repeats itself on a path form a leaf to the root marks a duplicate (marked with a * in the parse tree below)

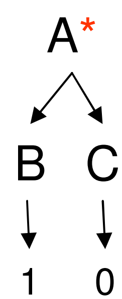


Loop in parse tree

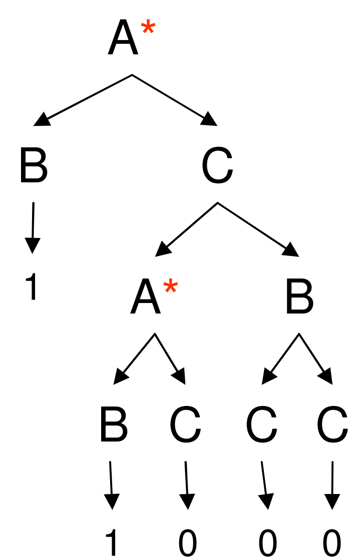
- How long should the string be to ensure duplicates?
- If the grammar has *n* non-terminals, the string should be at least of exponential length (around 2^n) to ensure a duplicate in some path.
- We can hide this 2^n conversion by asking the adversary for $m=2^n$ and work directly with m.

Loop in parse tree

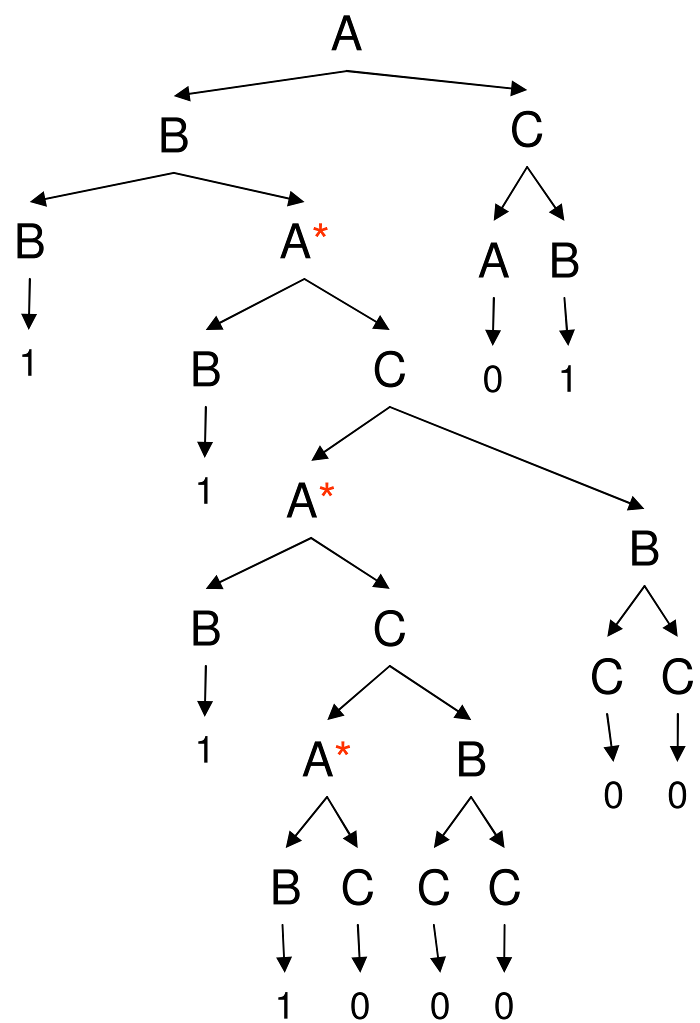
- We can insist that instead of the following subtree



- We have..



- To get…



- Which is a parse tree for another valid string 1111000001 as against the original string 11100001
- Any thing that got doubled is marked in green.

Pumping at multiple places!

- If we repeat this, we will find that two loops on either side of the center get pumped up an arbitrary number of times
  - $1\{1^n\}10\{00\}^n01$
- If the string is long enough, there will be some recursion
  - Due to some repeated substitution of some larger tree
  - Some substring *vwx* of *uvwxy* can get pumped up as $v^n w x^n$
  - *Task: Identify the parts of uvwxy*
- Examples of non-CFLs are
  - ww
  - $0^n 1^n 0^n$

Pumping Lemma: Formal statement
[Not a characterization, it is an implication]

- If a language L is infinite and context-free, then
- There exists some integer $p > 0$ [generally, exponential in number of states k] such that
- For every w in L with $|w| \geq p$ (where p is a pumping length)
- There exists a decomposition $w = uvwxz$ with strings u, v, x, y and z, such that $|vwx| \leq p$, $|vx| \geq 1$, (Note that both v and x cannot be empty) then
- For every integer $i \geq 0$, uv^iwx^iz is in L.

$0^n 1^n 0^n$ is not CFL

- Adversary: I have a CFG of *k* non-terminals
  - Equivalently pumping length=p=0(2^k)
- Prover: Consider the string $0^p 1^p 0^p$
- Adversary: v= 0^q w=$0^s 1^t$ x=1^b
  - $q+s+t+b \leq p$, $q+b \geq 1$
  - $0^{p-q-s} 0^{q*i} 0^s 1^t 1^{b*i} 1^{p-b-t} 0^p$ is not in L!
- Similarly, for other choices of v, w, x

ww is not CFL

- Adversary: I have a CFG of *k* non-terminals
  - Equivalently pumping length=p=0(2^k)
- Prover: Consider the string $0^p1^p0^p1^p$
- Adversary: v= 0^q w=0^s1^t x=1^b
  - $q+s+t+b \le p$, $q+b \ge 1$
  - $0^{p-q-s}0^{q*i}0^s1^t1^{b*i}1^{p-b-t}0^p1^p$ is not in L!
- Similarly, for other choices of v, w, x

Argument fails for CFLs!

- Try for palindromes !
- Adversary: I have a CFG of *k* non-terminals
  - Equivalently pumping length=p=0(2^k)
- Prover can consider the string $0^p1^p0^p$ or $0^p1^p0^p1^p0^p$ or any thing else! In each case, you can pump!

What about 0^{x^2}

- Adversary: I have a CFG of *k* non-terminals
  - Equivalently pumping length=p=0(2^k)
- Prover: Consider the string 0^{p^2}
- Adversary: $v=0^{a}$ $w=0^{b}$ $x=1^{c}$
  - $a+b+c \leq p$, $a+c \geq 1$
  - $0^{p\text{^}2-(i-1)*k}$ is not in L for some *i*!
- You need Turing machines for these languages and others such as 0^{primes} *etc.*

Pumping property may hold for non-CFLs!

- Eg: $0^{composite}$ satisfies pumping lemma
- But it is actually not context free

Use of closure to prove non-CFL

- Context Free Languages are closed under intersection with regular sets!
  - $Equal(0,1,2) \cap 0^*1^*2^* = 0^n1^n2^n$
  - You cannot have one stack simulate two stacks, but you can have one state simulate two states (one from an FSM and another from a PDM).
  - Another example: Let L consist of all strings of a's and b's with equal numbers of a's and b's but containing no substring abaa or babb. Then L is context-free, since it is the intersection of the language accepted a the pushdown automaton with the regular language $\{a, b\}^* - \{a, b\}^*(abaa \cup babb)\{a, b\}^*$
- Recall that PDMs are closed under unions

Theorem : *The intersection of a context-free language with a regular language is a context-free language.*

Proof: If L is a context-free language and R is a regular language, then $L = L(M_1)$ for some pushdown automaton $M_1 = (K_1, \Sigma, \Gamma_1, \Delta_1, s_1, F_1)$, and $R = L(M_2)$ for some deterministic finite automaton $M_2 = (K_2, \Sigma, \delta, s_2, F_2)$. The idea is to combine these machines into a single pushdown automaton M that carries out computations by M_1 and M_2 *in parallel* and accepts only if both would have accepted. Specifically, let $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where

$K = K_1 \times K_2$, the Cartesian product of the state sets of M_1 and M_2;
$\Gamma = \Gamma_1$;
$s = (s_1, s_2)$;
$F = F_1 \times F_2$, and
Δ, the transition relation, is defined as follows. For each transition of the pushdown automaton $((q_1, a, \beta), (p_1, \gamma)) \in \Delta_1$, and for each state $q_2 \in K_2$, we add to Δ the transition $(((q_1, q_2), a, \beta), ((p_1, \delta(q_2, a)), \gamma))$; and for each transition of the form $((q_1, e, \beta), (p_1, \gamma)) \in \Delta_1$ and each state $q_2 \in K_2$, we add to Δ the transition $(((q_1, q_2), e, \beta), ((p_1, q_2), \gamma))$. That is, M passes from state (q_1, q_2) to state (p_1, p_2) in the same way that M_1 passes from state q_1 to p_1, except that in addition M keeps track of the change in the state of M_2 caused by reading the same input.

CYK Algorithm for Parsing

- Deriving a string of length n takes 2n-1 productions using CNF
- Thus, there are 2^{2n-1} trees that generate strings of length 2n-1
  - Brute force: Given a string of length n, try all 2^{2n-1} trees! Impractical!!
- CYK: efficient algorithm that runs in $\Theta(n^3)$ time
  - Determines whether a string can be generated by a given context-free grammar and, if so, how it can be generated. This is known as parsing the string. Uses concept of dynamic programming
  - Avoid repeated computations
- CYK Stands for the author names
  - **Cocke-Younger-Kasami**

- Not used practically – there are more efficient algorithms
- Can do parsing for LRK, or any form of CFGs – not restricted to deterministic subsets of CFGs
- Useful to have the grammar in CNF
  – Makes description of grammar consice

CYK: Illustration

- Consider the grammar
  - A → BC | AB | 1
  - B → AA | 0
  - C → CB | 1 | 0
- Consider the string: 110100
- Basic Idea: Determine what parts of the string can be generated by what non-terminal

CYK: continued

- We will build all substrings of the string and determine which non-terminal can generate them
- Notation *V[i,j]*: All non-terminals that generate j symbols of the string *s*, starting from symbol *i*.
  - E.g: V[4,1]: All non-terminals that can generate the string of length 1, starting with position 4.
    - With CNF, this is easy to determine; it is {A,C}
  - Similarly, V[3,1] = {B,C}
- It will be harder as "*j*" gets bigger
  - But V[i,j] will be based on smaller cases of *j* as *j* gets bigger.

CYK continued

- Bottom-up dynamic programming style
  - Start from smaller values of j and build V[i,j] for larger j's successively.
  - Start *i* from the left.
- Computational Time
  - Number of entries to be computed = $\Theta(n^3)$
  - Computation of each cell requires n computations in worst case and is therefore $\Theta(n)$
  - So total time is $\Theta(n^3)$
- Parsing of LRK grammars (which are equivalent to deterministic PDMs) takes $\Theta(n)$ time!
  - Determining whether a given CFG is LRK is mechanical but complicated

Step 1

i \ j	1	2	3	4	5	6
1	(A,C)					
2	(A,C)					
3	(B,C)					
4	(A,C)					
5	(B,C)					
6	(B,C)					

Step 2

i \ j	1	2	3	4	5	6
1	(A,C)	(B)				
2	(A,C)	(A,C)				
3	(B,C)	(A)				
4	(A,C)	(A,C)				
5	(B,C)	(A,C)				
6	(B,C)					

- V[1,2] depends on two values of V; V[1,1] and V[2,1]
  - Only way you can have a non-terminal from {A,C} followed by a non-terminal from {A,C} is B → AA
- Thus, any value in the second column depends on two adjacent values from the previous column
- Note that some cells can be empty, corresponding to impossible productions

Step 3

i \ j	1	2	3	4	5	6
1	(A,C)	(B)	(B,A)			
2	(A,C)	(A,C)	(B)			
3	(B,C)	(A)	(A)			
4	(A,C)	(A,C)	(B,A,C)			
5	(B,C)	(A,C)				
6	(B,C)					

- V[1,3] depends on two values only [because of CNF]
  - V[1,1] and V[2,2] OR
  - V[1,2] and V[3,1]
- In general, V[i,j] depends on
  - V[i,k] and V[i+k,j-k]
- Note how we are using Dynamic Programming to save computations
  - We are computing V[1,1] only once, but using it twice

Step 4

i \ j	1	2	3	4	5	6
1	(A,C)	(B)	(B,A)	(A,C,B)		
2	(A,C)	(A,C)	(B)	(A,B)		
3	(B,C)	(A)	(A)	(A,C,B)		
4	(A,C)	(A,C)	(B,A,C)			
5	(B,C)	(A,C)				
6	(B,C)					

Step 5

i \ j	1	2	3	4	5	6
1	(A,C)	(B)	(B,A)	(A,C,B)	(A,C,B)	
2	(A,C)	(A,C)	(B)	(A,B,C)	(A,C,B)	
3	(B,C)	(A)	(A)	(A,C,B)		
4	(A,C)	(A,C)	(B,A,C)			
5	(B,C)	(A,C)				
6	(B,C)					

Step 6

i \ j	1	2	3	4	5	6
1	(A,C)	(B)	(B,A)	(A,C,B)	(A,C,B)	(A,C,B)
2	(A,C)	(A,C)	(B)	(A,B,C)	(A,C,B)	
3	(B,C)	(A)	(A)	(A,C,B)		
4	(A,C)	(A,C)	(B,A,C)			
5	(B,C)	(A,C)				
6	(B,C)					

- V[1,6] contains A and hence, we can conclude that the grammar generates the string *s.*

Closure + Decision Algorithms for Context Free Languages

- Closure:
  - Useful for determining decidability as well as for determining languages that are not context free
- Trivial Questions:
  - Is the language given by a left linear grammar regular? (trivially yes).
  - Is the language given by a context free grammar context free? (trivially yes).
  - Is the complement of a given regular language regular? (trivially yes).
  - Is the complement of a given context sensitive language context sensitive? (it turns out to be true)
- Hard Questions:
  - Is the complement of a given context free language context free? (not trivial, since context free languages are not closed under complement – it might be, it might not be).

Closure properties of CFLs

- Context Free Languages are closed under union
  - Given two grammars for the two CFLs, with start symbols S1 and S2, create a new grammar with all the rules of the two grammars along with a new rule, $S \rightarrow S1 \mid S2$
- Context Free Languages are also closed under string reversal
  - If L is context free then $L^R = \{w^R \mid w \in R, w^R$ is reverse of $w\}$ is also context free
  - Can get a CFG G^R for L^R using the CFG G for L by reversing the order of symbols on the right hand side of every production in G^R

Closure Property: Summary

Context Free Languages	Deterministic context free languages
Union	Complement [Toggle final and non-final states of corresponding deterministic PDM]
Reversal	Not closed under union E.g: $0^n 1^n \cup 0^n 1^{2n}$ is not deterministic CFL
Concatenation	

Undecidability problems in CFLs: Bridge to Turing Machines

- One of the highest level of undecidable problems
  - Post Correspondence Problem
  - Introduced by Emil Post in 1946
- Many problems in CFL world are undecidable if this problem is undecidable
  - Can be proved using the concept of “reduction”
  - If a problem can be ***reduced*** to the “Post Correspondence Problem”, then we know that the problem is undecidable

Post Correspondence Problem

- The input of the problem consists of two finite lists $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ of words over some alphabet *A* having at least two symbols. A solution to this problem is a non-empty sequence of indices,
  $i_1, \ldots, i_k, 1 \le i_k \le n$ such that
  - $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$
- The decision problem then is to decide whether such a solution exists or not for any two given finite lists
- Example:
  - Consider the following two lists:

u_1	u_2	u_3	u_4
aba	*bbb*	*aab*	*bb*

,

v_1	v_2	v_3	v_4
a	*aaa*	*abab*	*babba*

  - A solution to this problem would be the sequence 1, 4, 3, 1 because
    - $u_1 u_4 u_3 u_1$ = *aba* + *bb* + *aab* + *aba* = *ababbaababa* = *a* + *babba* + *abab* + *a* = $v_1 v_4 v_3 v_1$
  - However, if the two lists had consisted of only u_1, u_2, u_3 and v_1, v_2, v_3, then there would have been no solution.

Post Correspondence Problem contd

- A solution corresponds to some way of laying blocks next to each other so that the string in the top cells corresponds to the string in the bottom cells. Then the solution to the above example corresponds to:

aba	bb	aab	aba
a	$babba$	$abab$	a
$i_1 = 1$	$i_2 = 4$	$i_3 = 3$	$i_4 = 1$

Post Correspondence Problem contd

- Consider the following two examples. Which of them has a solution?
- Note that Problem 2 has an issue which we saw in the class…
- There are 3^k string of length k

Problem 1

String index	List A	List B
(1)	1	111
(2)	10111	10
(3)	10	0

Problem 2

String index	List A	List B
(1)	10	101
(2)	011	11
(3)	101	011

Decidability: Context free Languages

- Let L be the language generated by a CFG

Question	Context Free Languages	Deterministic context free languages
Is L empty?	Convert CFG to CNF and see if the only production is that start symbol goes to empty symbol	Convert CFG to CNF and see if the only production is that start symbol goes to empty symbol
Is L the universe?	Undecidable	Closed under complement. So check if complement PDM accepts any string.
Is complement of L CFL?	Undecidable	Trivially decidable
Is L1 = L2?	Undecidable	Nobody knows!
Is L1 intersection with L2 empty?	Undecidable	Undecidable

Undecidability and Reduction
Empty Intersection Problem [EIP]

- Is L1 intersection with L2 empty?
  - L1 and L2 are both context free
- We will prove that if there exists a solution for EIP, there will exist one for PCP
- Reduction:
  - Given a two finite lists $u_1,\ldots, u_n$ and $v_1,\ldots, v_n$ of words over some alphabet A having at least two symbols, we will describe a CFG problem

Reduction Example

- Grammar for Problem1 defining language L1
  - $S_A \rightarrow 1S_A\, a \mid 10111S_A\, b \mid 10S_A\, c \mid 1a \mid 10111b \mid 10c$
- Grammar for Problem2 defining language L2
  - $S_B \rightarrow 111a \mid 10b \mid 0c$
- a,b and c keep record of which strings were concatenated
- If there exists a string in common between the two grammar, it implies that there is a solution to the post-correspondence problem

Problem 1

String index	List A	List B
(1)	1	111
(2)	10111	10
(3)	10	0

So?

- If someone claims that the EIP problem is decidable, then the PCP problem should also be decidable!
- Thus, we made use of "reduction" to prove that a problem is undecidable

Is a CFL L $=\Sigma^*$ (the universe of strings)?

- This problem is undecidable. Let us call this problem the UNIVERSE problem
  - Why not take the complement and check if it is CFL?
  - No! The complement may be a Turing machine
  - Checking for membership in a turing machine is undecidable!
- Proof: By reducing the PCP problem to the UNIVERSE problem

Undecidability of the UNIVERSE problem

[In brief]

- Covert a PCP problem to two grammars
- Note that the grammars from PCP reduction are *deterministic context free* and closed under complement
  - You can define a deterministic push-down machines for these problems
- That is, in the previous case with
  - Grammar for Problem1 defining language L1
    - $S_A \rightarrow 1S_A a \mid 10111S_A b \mid 10S_A c \mid 1a \mid 10111b \mid 10c$
  - Grammar for Problem2 defining language L2
    - $S_B \rightarrow 111a \mid 10b \mid 0c$
- $(L1 \cap L2)^c = L1^c \cup L2^c$
  - Give this as input to the EMPTY problem

AMBIGUITY PROBLEM

- Is a given grammar ambiguous
  - *That is, does there exist a string σ that can be generated using two different left-most derivations by the grammar?*
- It is also undecidable
  - Provable by reduction from PCP
- Given a PCP, convert it into the grammars with start symbols S_A and S_B as before
- Write a new grammar $S \rightarrow S_A \mid S_B$
- If this grammar is detected to be ambiguous, it means a string has two left-most derivations, starting through S_A and S_B respectively
  - Which means the PCP problem has a string/solution!
- Thus proved by reduction!

REGULAR problem

- Is a given CFG "G" regular?
- Solution by **reduction** from UNIVERSE problem
- Reduce the particular problem
  - Is L == Σ^* (the universe of strings)
  - To an instance of REGULAR problem
  - Is L == $\{0,1\}^*$

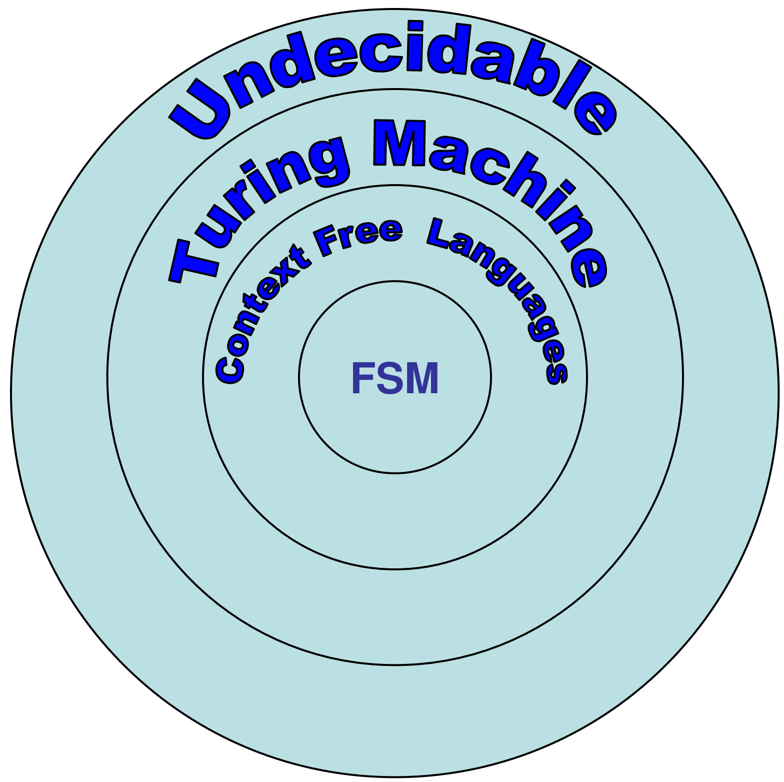
EQUALITY problem

- Given CFGs L1 and L2, is L1==L2?
- Trivial reduction from UNIVERSE problem
  - Give it the input L1 and L2 = Σ^*

Some more interesting properties

- Let L be a CFG and R be a regular language
  - Is $R \subset L$?
    - Is an undecidable problem
  - Is $L \subset R$
    - Is a decidable problem!!
    - Provable because CFLs are closed under intersection with regular languages

Turing Machines = Decidable



Turing Machines

- Very robust and powerful model of computation
- Adding stacks, queues etc to a Turing machine adds no more power
- Any procedure/computation we write using CFL is
  - E.g.: Procedure to add two numbers is a turing machine

Church and Turing

- Formalized independently
  – Equivalent representations in the form of **Church's** lambda calculus and **Turing**' machine
- Church Turing hypothesis
  – Anything we do using a procedure is a turing machine
- You can write Turing machines for all decidable problems….

Undecidable vs partially decidable

- Partially decidable
  - Also called Recursively enumerable problems
  - If there exists a program for a problem that answers `yes' when the actual answer is `yes' and may not answer `no' when the actual answer is `no'
- Decidable == Turing acceptable

Turing Machines

- Neither finite automata nor pushdown automata can be regarded as truly general models for computers, since they are not capable of recognizing even such simple languages as $\{a^n b^n c^n : n > 0\}$
  - Turing Machines are most powerful models of computations that recognize languages such as $\{a^n b^n c^n : n > 0\}$ and many more
- A Turing machine consists of a finite control, a tape, and a head that can be used for reading or writing on that tape.
- The formal definitions of Turing machines and their operation are in the same mathematical style as those used for finite and pushdown automata.

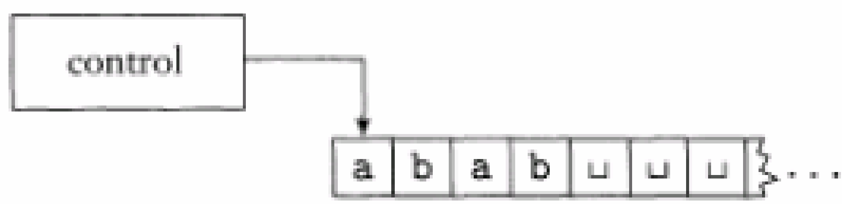
Turing Machine components

1. A Finite state control Unit
   - The control unit operates in discrete steps; at each step it performs two functions in a way that depends on its current state and the tape symbol currently scanned by the read/write head:
     1. Put the control unit in a new state.
     2. Either:

        (a) Write a symbol in the tape square currently scanned, replacing the one already there; or

        (b) Move the read/write head one tape square to the left or right.
2. A tape
   - Communication between the tape and the control unit is provided by a single head, which reads symbols from the tape and is also used to change the symbols on the tape.

Pictorial representation of a Turing Machine



Definition of Turing Machine

DEFINITION

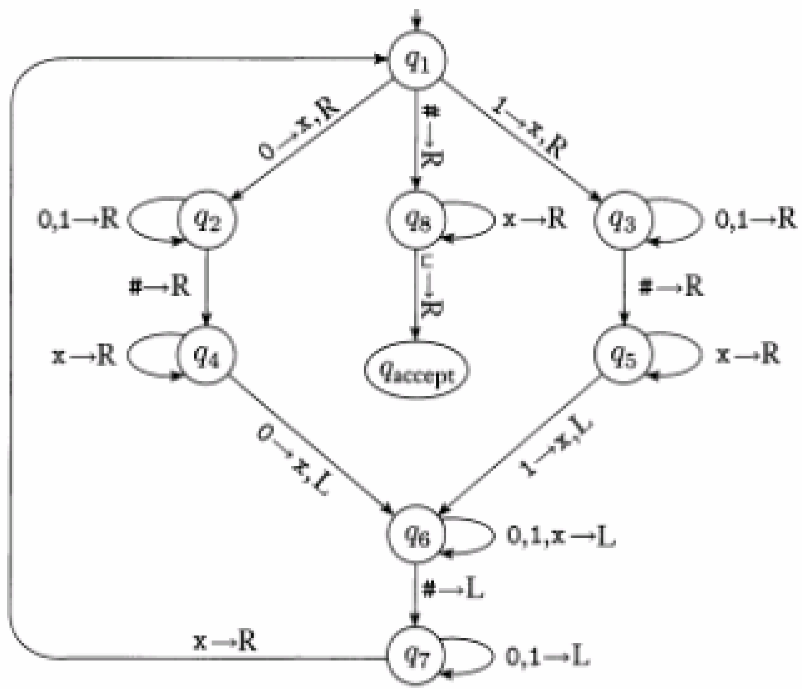
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Example Turing machine that accepts the language

$$L = \{w\#w \mid w \in \{0,1\}^*\}$$

- $Q = \{q_1, \ldots, q_{14}, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.
- We describe δ with a state diagram
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}.

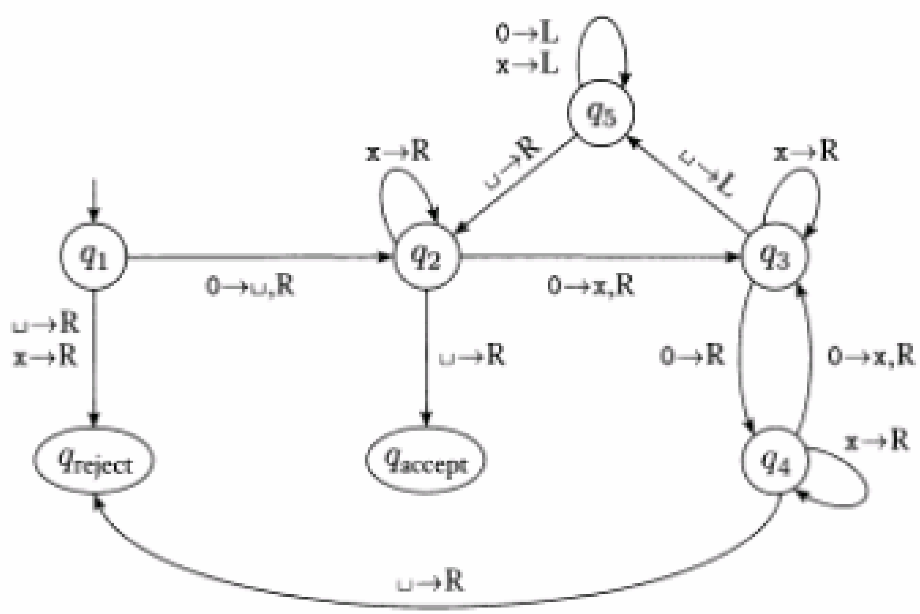


Example Turing machine M_2 that accepts the language

$$L = \left\{0^{2^n} \mid n \geq 0\right\}$$

The machine begins by writing a blank symbol over the leftmost 0 on the tape so that it can find the left-hand end of the tape in subsequent steps

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$,
- $\Sigma = \{0\}$, and
- $\Gamma = \{0, x, \sqcup\}$.
- We describe δ with a state diagram (see Figure 3.8).



Sample run of M_2 on input 0000

- The starting configuration is $q_1 0000$. The sequence of configurations the machine enters appears as follows; read down the columns and left to right.

$q_1\mathtt{0000}$	$\sqcup q_5\mathtt{x0x}\sqcup$	$\sqcup\mathtt{x}q_5\mathtt{xx}\sqcup$
$\sqcup q_2\mathtt{000}$	$q_5\sqcup\mathtt{x0x}\sqcup$	$\sqcup q_5\mathtt{xxx}\sqcup$
$\sqcup\mathtt{x}q_3\mathtt{00}$	$\sqcup q_2\mathtt{x0x}\sqcup$	$q_5\sqcup\mathtt{xxx}\sqcup$
$\sqcup\mathtt{x0}q_4\mathtt{0}$	$\sqcup\mathtt{x}q_2\mathtt{0x}\sqcup$	$\sqcup q_2\mathtt{xxx}\sqcup$
$\sqcup\mathtt{x0x}q_3\sqcup$	$\sqcup\mathtt{xx}q_3\mathtt{x}\sqcup$	$\sqcup\mathtt{x}q_2\mathtt{xx}\sqcup$
$\sqcup\mathtt{x0}q_5\mathtt{x}\sqcup$	$\sqcup\mathtt{xxx}q_3\sqcup$	$\sqcup\mathtt{xx}q_2\mathtt{x}\sqcup$
$\sqcup\mathtt{x}q_5\mathtt{0x}\sqcup$	$\sqcup\mathtt{xx}q_5\mathtt{x}\sqcup$	$\sqcup\mathtt{xxx}q_2\sqcup$
		$\sqcup\mathtt{xxx}\sqcup q_{\mathrm{accept}}$