

# Closure properties of CFL

- Find satisfactory answer to the following question;  
When we perform operations (union, intersection, etc.) with CFLs, do we get as a result a CFL?
- We raised similar questions about RLs, and had easy answers.
- When we ask the same questions about CFLs, we encounter some difficulties.

# The Closure Theorems

**Theorem:** CFLs are closed under **Union, Concatenation, Kleene Star, and Reverse.**

Let  $G_1 = (V_1, T_1, S_1, R_1)$ ,  $G_2 = (V_2, T_2, S_2, R_2)$ , such that  $V_1 \cap V_2 = \{ \}$ .

**Union, Proof:** We can construct

$G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, S_3, P_3)$ , where

$$P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1 | S_2\}$$

Clearly  $G_3$  is CF, and it is easy to see that  $L(G_3) = L(G_1) \cup L(G_2)$ .

**Concatenation, proof:** Similarly we can construct  $G_3$  where

$$P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1 S_2\}$$

Clearly  $G_3$  is CF and  $L(G_3) = L(G_1)L(G_2)$ .

# The Closure Theorems (cnt.)

**Kleene \*, Proof:** We can construct  $G_3 = (V_1 \cup \{S_3\}, T_1, S_3, P_3)$ , where

$$P_3 = P_1 \cup \{S_3 \rightarrow \epsilon, S_3 \rightarrow S_1 S_3\}$$

Clearly  $G_3$  is CF and  $L(G_3) = L(G_1)^*$ .

**Reverse, proof:** Given  $G = (V, T, P, S)$  in CNF, we can construct  $G^R = (V, T, P^R, S)$ , where  $P^R$  is obtained as follows:

- For every rule in  $P$  of the form  $X \rightarrow AB$ , add to  $P^R$  the rule  $X \rightarrow BA$ .
- For every rule in  $P$  of the form  $X \rightarrow a$ , add to  $P^R$  the rule  $X \rightarrow a$ .

It is easy to see that  $G^R$  is CF and  $L(G^R) = L(G)^R$ .

# The Closure Theorems (cnt.)

**Theorem.:** CFLs are not closed under **Intersection**, **Complement**, and **Difference**.

**Intersection, proof:** If  $L_1$  and  $L_2$  are CF, then  $L_1 \cap L_2$  is not necessarily CF.

- Consider two languages CFLs

$$L_1 = \{a^n b^n c^m : n \geq 0, m \geq 0\}, L_2 = \{a^n b^m c^m : n \geq 0, m \geq 0\}$$

- Notice that  $L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0, m \geq 0\}$
- But  $\{a^n b^n c^n : n \geq 0, m \geq 0\}$  is not CF.
- Using the PL: Let  $p$  be a number and  $z = a^p b^p c^p \in L_1 \cap L_2$ . Considering possible decompositions for  $z = uvwxy$ , there is no decomposition (obeying the PL reqs.) that would allow to pump  $v$  and  $x$  and produce a string in the language.

# The Closure Theorems (cnt.)

**Complement, proof:** If  $\overline{L}$  always was CF, it would follow that

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \text{ always would be CF.}$$

However, CFLs are closed under union. If they were closed under complement, then they would necessarily be closed under intersection.

**Difference, proof:** Given any language  $L$  and  $M$ ,

$$L - M = L \cap \overline{M}.$$

If  $L$  and  $M$  are CFLs and they were closed under difference, then they would necessarily be closed under intersection and complement.

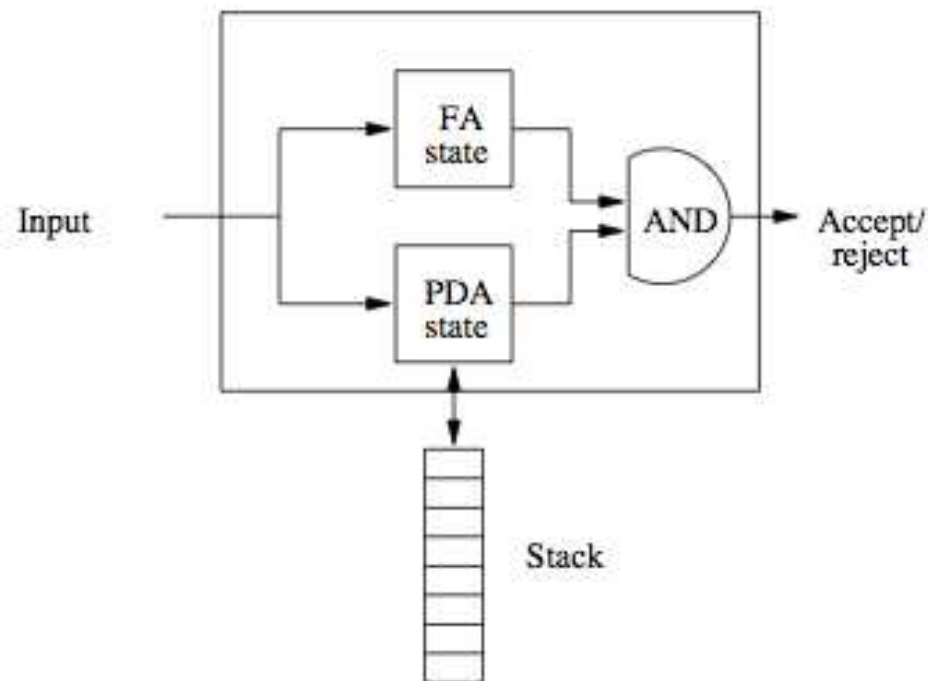
# CFL $\cap$ Regular is a CFL

**Theorem** : If  $L$  is CF and  $R$  is regular, then  $L \cap R$  is CF.

**Proof:** Let  $L$  be accepted by PDA

$P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$  by final state, and let  $R$  be accepted by DFA  $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$

We'll construct a PDA for  $L \cap R$  according to the picture



# Proof idea

Formally, we define

$$P' = (Q_P \times Q_A, \Sigma, \Gamma, \Delta, (q_P, q_A), Z_0, F_P \times F_A)$$

where

$$\delta((q, p), a, X) = \{((r, \delta_A(p, a)), \gamma) : (r, \gamma) \in \delta_P(q, a, X)\}$$

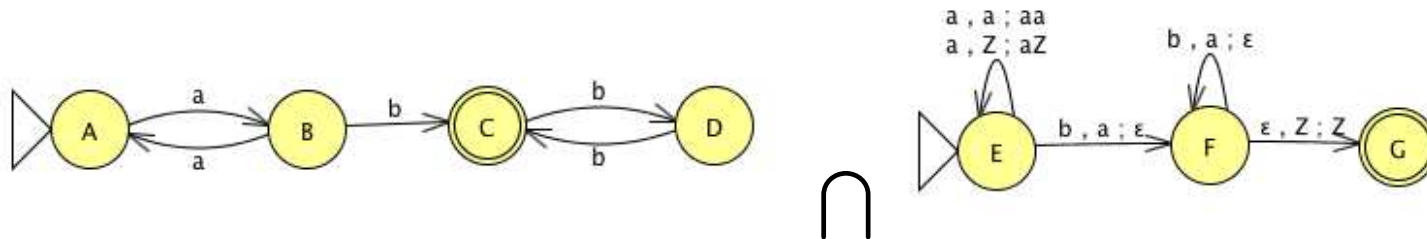
Then carry out a proof by induction on  $\vdash^*$  that

$$(q_P, w, Z_0) \vdash_P^* (q, \epsilon, \gamma) \text{ if and only if}$$

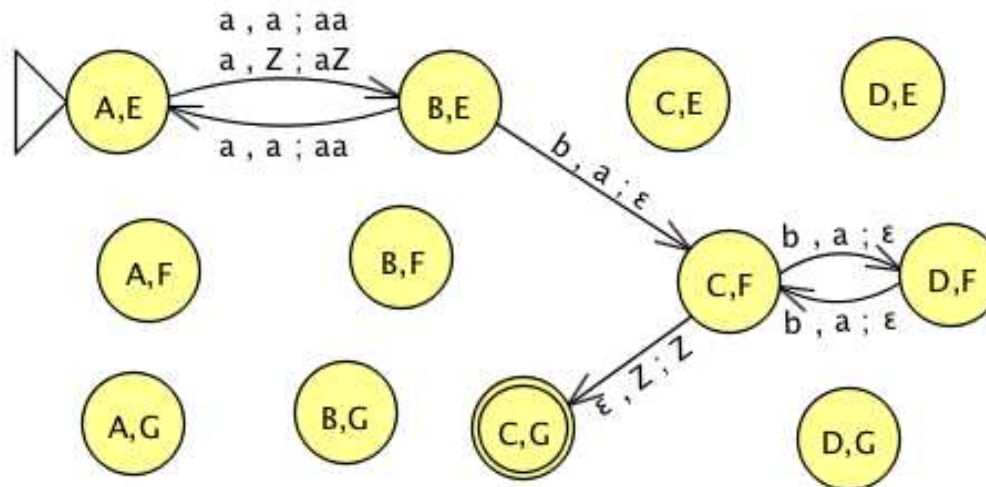
$$((q_P, q_A), w, Z_0) \vdash_{P'}^* ((q, \hat{\delta}(q_A, w)), \epsilon, \gamma)$$

# Example

Is  $L = \{a^i b^j : i, j \text{ are odd}\} \cap \{a^n b^n : n \geq 1\}$  context free?



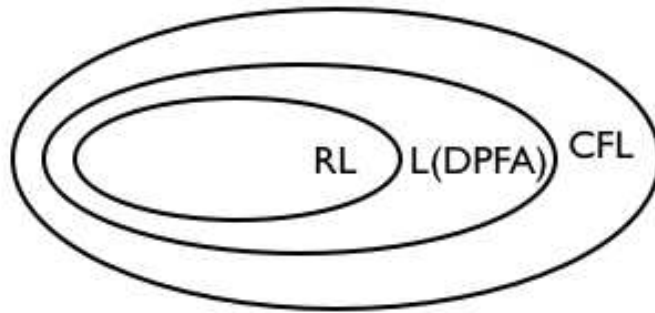
For clarity sake, I have not included transitions that lead to "trap states."





# Our recent studies

- So far, we have been studying simple **classes of languages** (problems related to searching text, analysis of protocols, and parsing).
- We saw that PDAs are more powerful than finite automata.
- We saw that CFL, while fundamental to the study of programming languages, are limited in scope; e.g.,  $a^n b^n c^n$  and  $ww$  although quite simple, are not CF.
- What is then beyond CFL? And how can we define new language families that include these examples?



# In search for a more powerful automaton

- **FA vs PDA:** the nature of the temporary storage captures the difference between them.
  - if there is no storage, we have an FA; if the storage is a stack, then we have the more powerful PDA
- By extrapolation, we can expect to discover even more powerful languages if we give the automaton more flexible storage, e.g., two stacks, three stacks, a queue or some other storage device, etc.
- What can we say of the most powerful automaton and the limits of computation?
- The **Turing Thesis** maintains that the **Turing Machine** is such automaton.

# Turing Machines



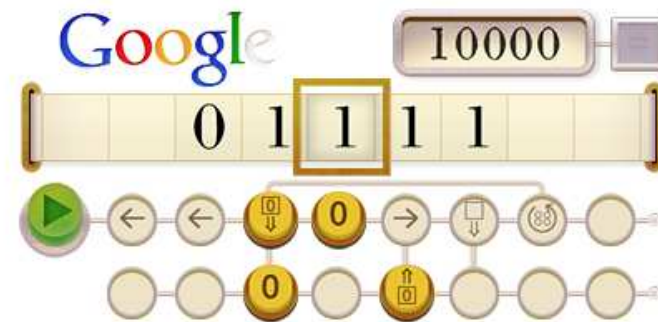
*From Google's "Turing Machine doodle" (June 23, 2012), in commemoration of Alan Turing's 100th Birthday.*

# Motivation for studying this stuff

- To provide guidance to programmers on what they might or might not be able to accomplish through programming.
- How do Turing Machines fit in this context?
  - It is a tool that will allow us to prove everyday questions undecidable or intractable (those problems that are decidable but require large amounts of time to solve them).
- To prove a problem undecidable, we'll reduce it to a problem that the TM cannot decide.

# The Turing Machine (1936)

- Has a **finite-state control** unit, like all automata.
- One infinite read-write **tape** serves as both input and unbounded storage device.
- The tape:
  - divided into **cells**;
  - each cell holds one symbol from the **tape alphabet**;
  - the head marks the *current* cell, which is the only cell that can influence the move of the TM.
  - Initially, tape holds  $a_1a_2 \cdots a_nBB \cdots$ , where  $a_1a_2 \cdots a_n$  is the input, chosen from an **input alphabet** (subset of the tape alphabet) and  $B$  is the “blank” symbol.



# A move of a TM

- A move of a Turing machine (TM) is a function of **the state** of the finite control and **the tape symbol** just scanned.
- In one move, the TM will:
  1. Change state.
  2. **Write a tape symbol** in the cell scanned.
  3. Move the tape head **left or right**.

# TMs: formally

A TM is a 7-tuple  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ , where:

- $Q, \Sigma, \delta, q_0$  and  $F$  are our old friends.
- $\Gamma$  is the set of tape symbols,  $\Sigma \subset \Gamma$ .
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ , e.g.:  $\delta(q, X) = (p, Y, \mathcal{D})$ , where  $q$  is a state,  $X$  is a tape symbol
  - $p$  is the next state,
  - $Y$  is the symbol written in the cell being scanned, replacing whatever was there, and
  - $\mathcal{D}$  is the direction, either  $L$  or  $R$ , of the next move.
- $B \in \Gamma$  is the blank symbol;  $B \notin \Sigma$

# Instantaneous description for TMs

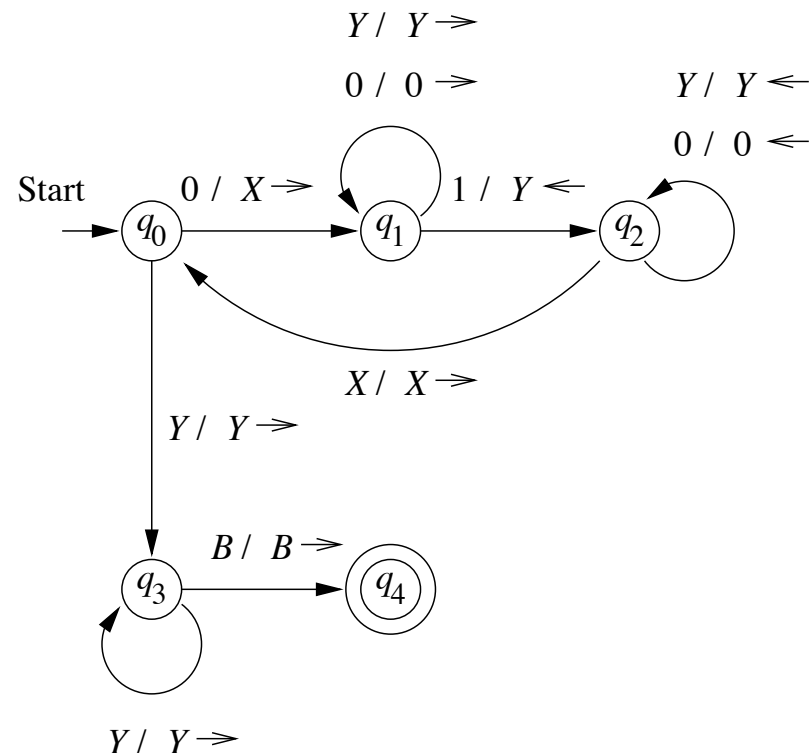
- An ID, denoted  $\alpha q \beta$ , represents what is going on at any moment with a TM:
  - $\alpha$  is the tape contents to the left of the head.
  - $q$  is the state the TM is at the moment.
  - $\beta$  is the nonblank tape contents at or to the right of the tape head.
- E.g.:  $XXq_3Y1BB$
- as before,  $\vdash$  denotes one move;  $\vdash^*$  denotes zero, one, or more moves.



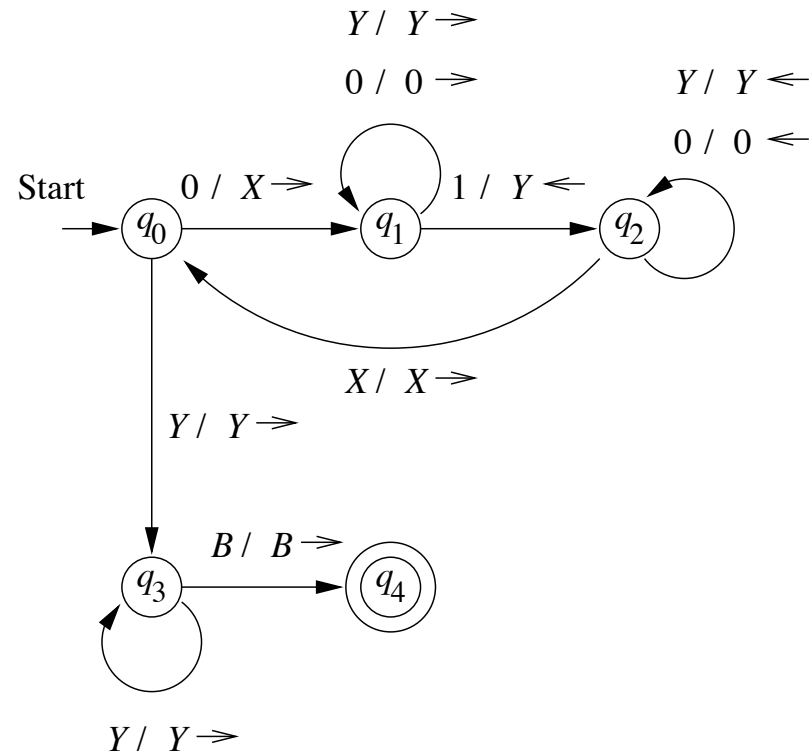
# Example: a TM for $\{0^n 1^n : n \geq 1\}$

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$   
 where  $\delta$  is given by the following table:

	0	1	X	Y	B
$\rightarrow q_0$	$(q_1, X, R)$			$(q_3, Y, R)$	
$q_1$	$(q_1, 0, R)$	$(q_2, Y, L)$		$(q_1, Y, R)$	
$q_2$	$(q_2, 0, L)$		$(q_0, X, R)$	$(q_2, Y, L)$	
$q_3$				$(q_3, Y, R)$	
$\star q_4$					$(q_4, B, R)$

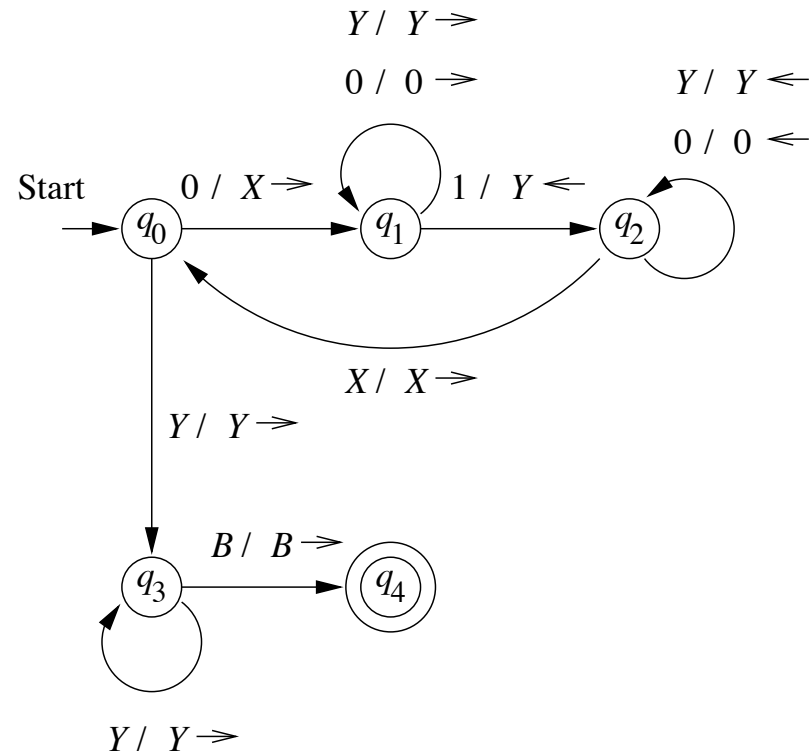


# An accepting computation by $M$



$q_0 0 0 1 1 \vdash X q_1 0 1 1 \vdash X 0 q_1 1 1 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1$   
 $X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash$   
 $X X q_0 Y Y \vdash X X Y q_3 Y \vdash X X Y Y q_3 B \vdash X X Y Y B q_4 B$

# A non-accepting computation by $M$



$q_0 0 0 1 0 \vdash X q_1 0 1 0 \vdash X 0 q_1 1 0 \vdash X q_2 0 Y 0 \vdash q_2 X 0 Y 0$   
 $X q_0 0 Y 0 \vdash X X q_1 Y 0 \vdash X X Y q_1 0 \vdash X X Y 0 q_1 B$

# The language of a TM

- Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  be a TM. The language **accepted** by  $M$  is defined as

$L(M) = \{w : w \in \Sigma^*, q_0 w \vdash^* \alpha p \beta\}$ , where  $p \in F$  and any tape strings  $\alpha$  and  $\beta$ .

- A language is called **recursively enumerable** (*Turing-recognizable*) if a TM accepts it.
- But what if a TM does not halt (loops) on a given input? (it may be hard to determine whether it is actually looping or just taking too long to compute).

# TM: acceptance by halting

- A Turing machine halts if it enters a state  $q$ , scanning a tape symbol  $X$ , and there is no move in this situation, i.e.,  $\delta(q, X)$  is undefined.
- A TM that halts on **all** inputs is called a **decider**, as it decides to accept an input or not.
  - A language is called **decidable** if a TM *decides* it.

# Exercise

Provide a TM that decides the language

$$L = \{w\#w : |w| \geq 0, w \in \{0, 1\}^*\}$$