

# 15-453

## FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY

**HOMEWORK DUE: THURSDAY**  
**MIDTERM: TUESDAY**  
**PROJECT REPORT: TODAY!**

TUESDAY Feb 5

### Chomsky Normal Form and TURING MACHINES

TUESDAY Feb 5

### CHOMSKY NORMAL FORM

A context-free grammar is in Chomsky normal form if every rule is of the form:

$A \rightarrow BC$  B and C aren't start variables

$A \rightarrow a$  a is a terminal

$S \rightarrow \epsilon$  S is the start variable

Any variable A that is not the start variable can only generate strings of length  $> 0$

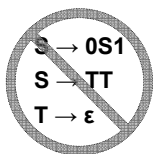
### CHOMSKY NORMAL FORM

A context-free grammar is in Chomsky normal form if every rule is of the form:

$A \rightarrow BC$  B and C aren't start variables

$A \rightarrow a$  a is a terminal

$S \rightarrow \epsilon$  S is the start variable



$S_0 \rightarrow TU \mid TV \mid \epsilon$

$T \rightarrow 0$

$U \rightarrow SV$

$S \rightarrow TU \mid TV$

$V \rightarrow 1$



Theorem: If G is in CNF,  $w \in L(G)$  and  $|w| > 0$ , then any derivation of w in G has length  $2|w| - 1$

Proof (by induction on  $|w|$ ):

Base Case: If  $|w| = 1$ , then any derivation of w must have length 1

Inductive Step: Assume true for any string of length at most  $k \geq 1$ , and let  $|w| = k+1$

Since  $|w| > 1$ , derivation starts with  $S \rightarrow AB$

So  $w = xy$  where  $A \Rightarrow^* x$ ,  $|x| > 0$  and  $B \Rightarrow^* y$ ,  $|y| > 0$

By the inductive hypothesis, the length of any derivation of w must be

$$1 + (2|x| - 1) + (2|y| - 1) = 2(|x| + |y|) - 1$$

**Theorem: Any context-free language can be generated by a context-free grammar in Chomsky normal form**

**“Can transform any CFG into Chomsky normal form”**

**Theorem: Any context-free language can be generated by a context-free grammar in Chomsky normal form**

**Proof Idea:**

1. Add a new start variable
2. Eliminate all  $A \rightarrow \epsilon$  rules. Repair grammar
3. Eliminate all  $A \rightarrow B$  rules. Repair
4. Convert  $A \rightarrow u_1 u_2 \dots u_k$  to  $A \rightarrow u_1 A_1, A_1 \rightarrow u_2 A_2, \dots$   
If  $u_i$  is a terminal, replace  $u_i$  with  $U_i$  and add  $U_i \rightarrow u_i$

1. Add a new start variable  $S_0$  and add the rule  $S_0 \rightarrow S$

$S_0 \rightarrow S$   
 $S \rightarrow OS1$   
 $S \rightarrow T\#T$   
 $S \rightarrow T$   
 $T \rightarrow \epsilon$

2. Remove all  $A \rightarrow \epsilon$  rules (where  $A$  is not  $S_0$ )

For each occurrence of  $A$  on right hand side of a rule, add a new rule with the occurrence deleted

If we have the rule  $B \rightarrow A$ , add  $B \rightarrow \epsilon$ , unless we have previously removed  $B \rightarrow \epsilon$

$S_0 \rightarrow S$   
 $S \rightarrow OS1$   
 $S \rightarrow T\#T$   
 $S \rightarrow T$   
  
 $S \rightarrow T\#$   
 $S \rightarrow \#T$   
 $S \rightarrow \#$   
  
 $S \rightarrow 01$   
 $S_0 \rightarrow \epsilon$

2. Remove all  $A \rightarrow \epsilon$  rules (where  $A$  is not  $S_0$ )

For each occurrence of  $A$  on right hand side of a rule, add a new rule with the occurrence deleted

If we have the rule  $B \rightarrow A$ , add  $B \rightarrow \epsilon$ , unless we have previously removed  $B \rightarrow \epsilon$

3. Remove unit rules  $A \rightarrow B$

Whenever  $B \rightarrow w$  appears, add the rule  $A \rightarrow w$  unless this was a unit rule previously removed

$S_0 \rightarrow S$   
 $S \rightarrow OS1$   
 $S \rightarrow T\#T$   
  
 $S \rightarrow T\#$   
 $S \rightarrow \#T$   
 $S \rightarrow \#$   
  
 $S \rightarrow 01$   
 $S_0 \rightarrow \epsilon$   
 $S_0 \rightarrow OS1$

4. Convert all remaining rules into the proper form:

$S_0 \rightarrow OS1$        $S_0 \rightarrow 01$   
 $S_0 \rightarrow A_1 A_2$        $S_0 \rightarrow A_1 A_3$   
 $A_1 \rightarrow 0$        $S \rightarrow 01$   
 $A_2 \rightarrow SA_3$        $S \rightarrow A_1 A_3$   
 $A_3 \rightarrow 1$

$S_0 \rightarrow \epsilon$   
 $S_0 \rightarrow OS1$   
 $S_0 \rightarrow T\#T$   
 $S_0 \rightarrow T\#$   
 $S_0 \rightarrow \#T$   
 $S_0 \rightarrow \#$   
 $S_0 \rightarrow 01$   
 $S \rightarrow OS1$   
 $S \rightarrow T\#T$   
 $S \rightarrow T\#$   
 $S \rightarrow \#T$   
 $S \rightarrow \#$   
 $S \rightarrow 01$

Convert the following into Chomsky normal form:

$$A \rightarrow BAB \mid B \mid \epsilon$$

$$B \rightarrow 00 \mid \epsilon$$

$$S_0 \rightarrow A$$

$$A \rightarrow BAB \mid B \mid \epsilon \rightarrow A \rightarrow BAB \mid B \mid BB \mid AB \mid BA$$

$$B \rightarrow 00 \mid \epsilon \rightarrow B \rightarrow 00$$

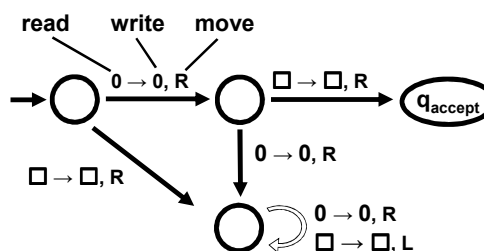
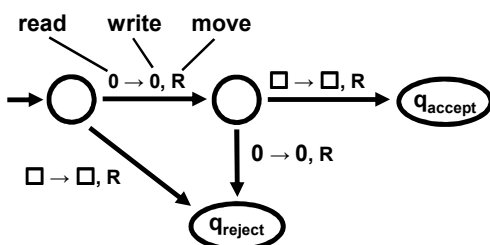
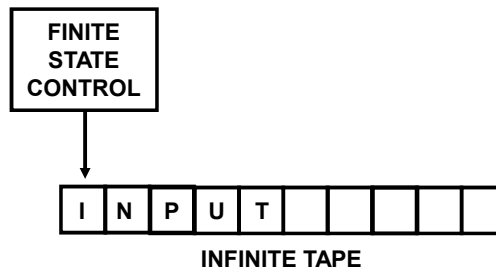
$$S_0 \rightarrow BAB \mid 00 \mid BB \mid AB \mid BA \mid \epsilon$$

$$A \rightarrow BAB \mid 00 \mid BB \mid AB \mid BA$$

$$B \rightarrow 00$$

$$S_0 \rightarrow BC \mid DD \mid BB \mid AB \mid BA \mid \epsilon, \quad C \rightarrow AB, \\ A \rightarrow BC \mid DD \mid BB \mid AB \mid BA, \quad B \rightarrow DD, \quad D \rightarrow 0$$

## TURING MACHINE



## TMs VERSUS FINITE AUTOMATA

TM can both *write* to and read from the tape

The head can move *left and right*

The input doesn't have to be read entirely, and the computation can continue after all the input has been read

Accept and Reject take immediate effect

**Definition:** A Turing Machine is a 7-tuple  $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ , where:

$Q$  is a finite set of states

$\Sigma$  is the input alphabet, where  $\square \notin \Sigma$

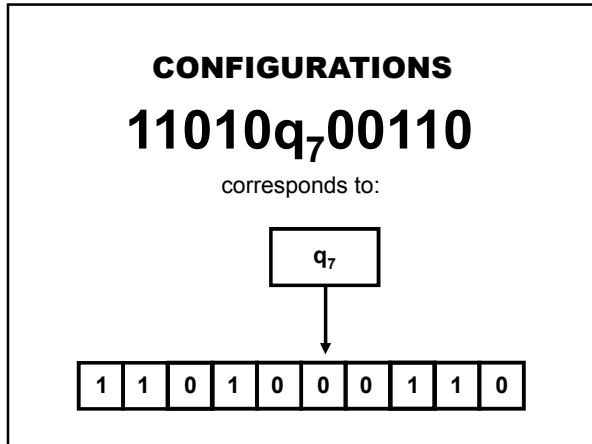
$\Gamma$  is the tape alphabet, where  $\square \in \Gamma$  and  $\Sigma \subseteq \Gamma$

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

$q_0 \in Q$  is the start state

$q_{accept} \in Q$  is the accept state

$q_{reject} \in Q$  is the reject state, and  $q_{reject} \neq q_{accept}$



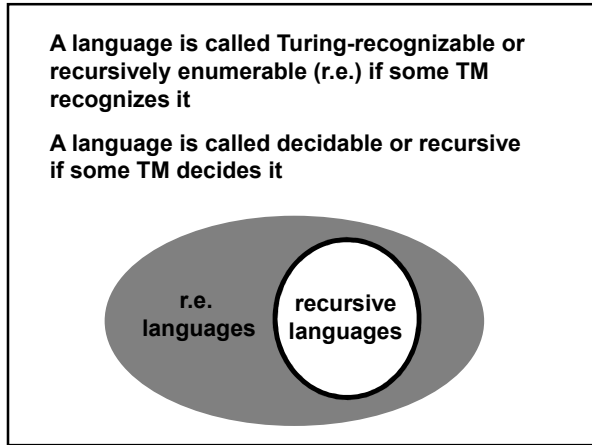
A TM *recognizes* a language iff it accepts all and only those strings in the language

A language L is called Turing-recognizable or recursively enumerable iff some TM recognizes L

---

A TM *decides* a language L iff it accepts all strings in L and rejects all strings not in L

A language L is called decidable or recursive iff some TM decides L



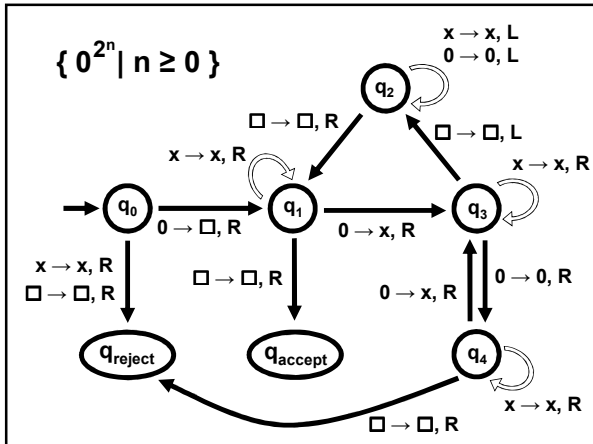
**Theorem:** If A and  $\neg A$  are r.e. then A is recursive

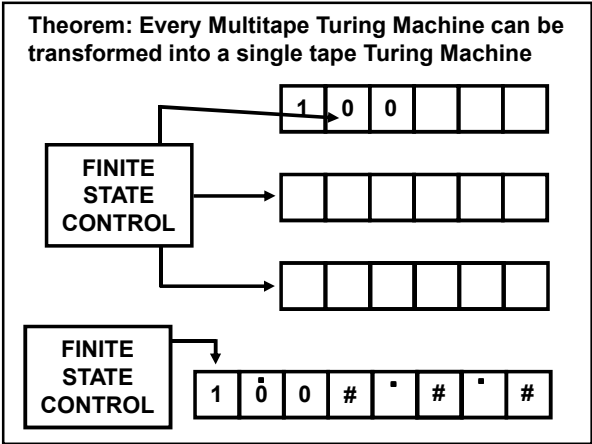
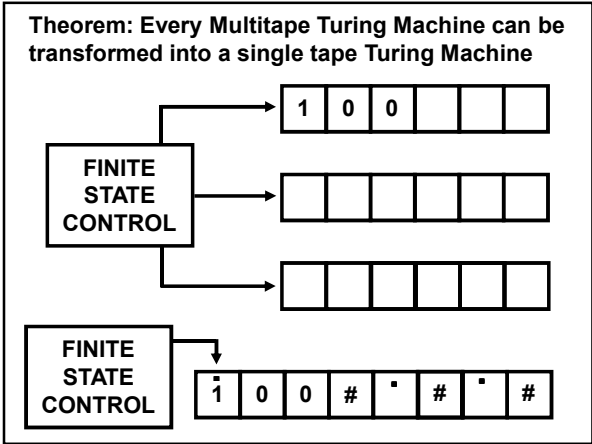
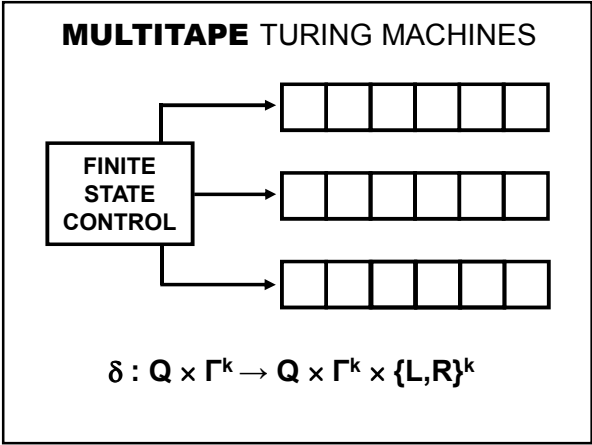
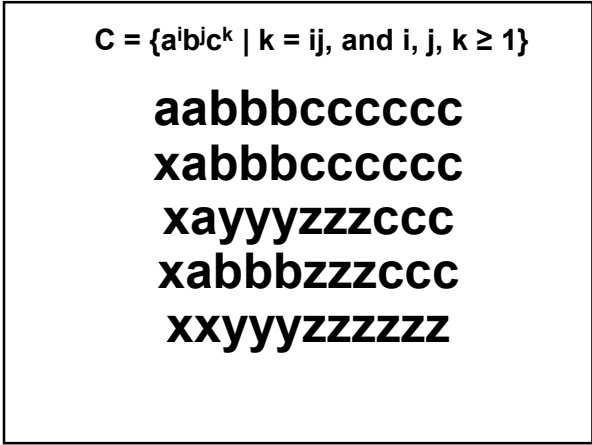
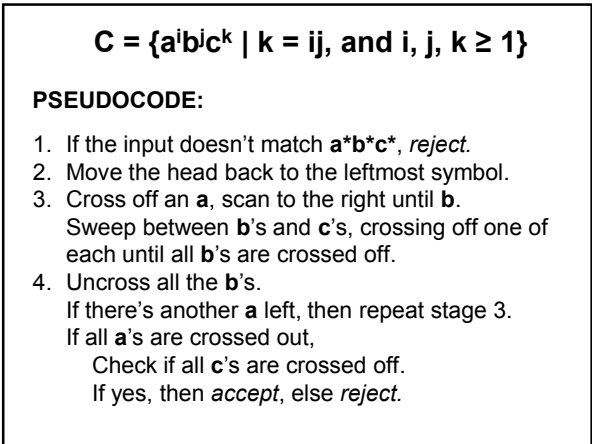
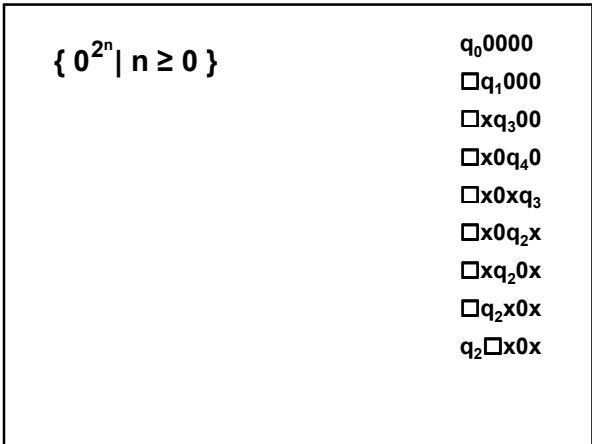
**Given:**  
 a TM that recognizes A and  
 a TM that recognizes  $\neg A$ ,  
 we can build a new machine that *decides* A

$\{0^{2^n} \mid n \geq 0\}$

**PSEUDOCODE:**

1. Sweep from left to right, cross out every other 0
2. If in stage 1, the tape had only one 0, *accept*
3. If in stage 1, the tape had an odd number of 0's, *reject*
4. Move the head back to the first input symbol.
5. Go to stage 1.

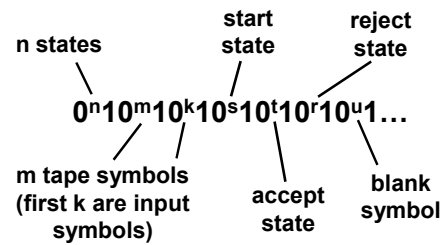




## THE CHURCH-TURING THESIS

Intuitive Notion of Algorithms  
EQUALS  
Turing Machines

We can encode a TM as a string of 0s and 1s



$$((p, a), (q, b, L)) = 0^p 10^a 10^q 10^b 10$$

$$((p, a), (q, b, R)) = 0^p 10^a 10^q 10^b 11$$

Similarly, we can encode DFAs, NFAs,  
CFGs, etc. into strings of 0s and 1s

So we can define the following languages:

$$A_{\text{DFA}} = \{ (B, w) \mid B \text{ is a DFA that accepts string } w \}$$

$$A_{\text{NFA}} = \{ (B, w) \mid B \text{ is an NFA that accepts string } w \}$$

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

$$A_{\text{DFA}} = \{ (B, w) \mid B \text{ is a DFA that accepts string } w \}$$

Theorem:  $A_{\text{DFA}}$  is decidable

Proof Idea: Simulate B on w

$$A_{\text{NFA}} = \{ (B, w) \mid B \text{ is an NFA that accepts string } w \}$$

Theorem:  $A_{\text{NFA}}$  is decidable

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

Theorem:  $A_{\text{CFG}}$  is decidable

Proof Idea: Transform G into Chomsky Normal Form. Try all derivations of length up to  $2|w|-1$

# WWW.FLAC.WS

Read Chapter 3 of the book for next time