

# CS 302.1 - Automata Theory

## Lecture 04

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad

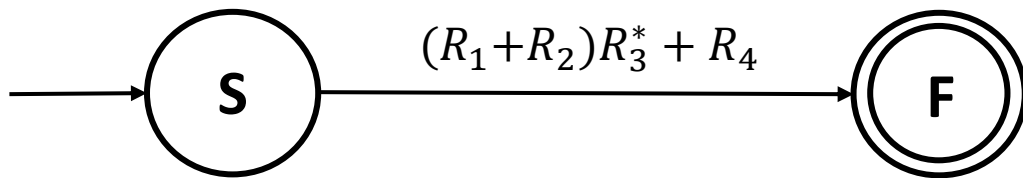


# Quick Recap

- RL can also be derived from first principles.
- Regular expressions provide an elegant algebraic framework to represent regular languages.
- We can construct NFAs given a Regular Expression.

A Generalized NFA (GNFA) is similar to an NFA except that transitions contain regular expressions.

Given a DFA  $M$ , we obtain the regular expression corresponding to  $L(M)$  by constructing a 2-state GNFA via a recursive algorithm.



**DFA, NFA, Regular Expressions  
have equal power and all of them  
correspond to Regular Languages**

# Pumping Lemma

How do we prove that certain languages are non-regular? We start with an example

Let  $\Sigma = \{0,1\}$ . Consider the language  $L = \{0^n 1^n \mid n \geq 0\}$  and the following conversation between Karl and Mil.

**Mil:** I have a DFA for  $L$ .

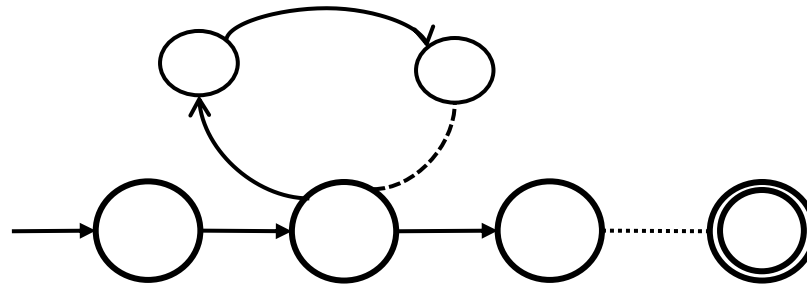
**Karl:** How many states are there?

**Mil:**  $n$ -states (say  $n = 10$ )

**Karl:** Then  $0^{10}1^{10}$  must be accepted. By the **pigeonhole principle**, while reading the first ( $n = 10$ ) symbols, some states need to be revisited. Otherwise  $n + 1 = 11$  states would have been present. Hence some loop must be present. How many states are there in the loop?

**Mil:**  $t$ -states (say  $t = 3$ ).

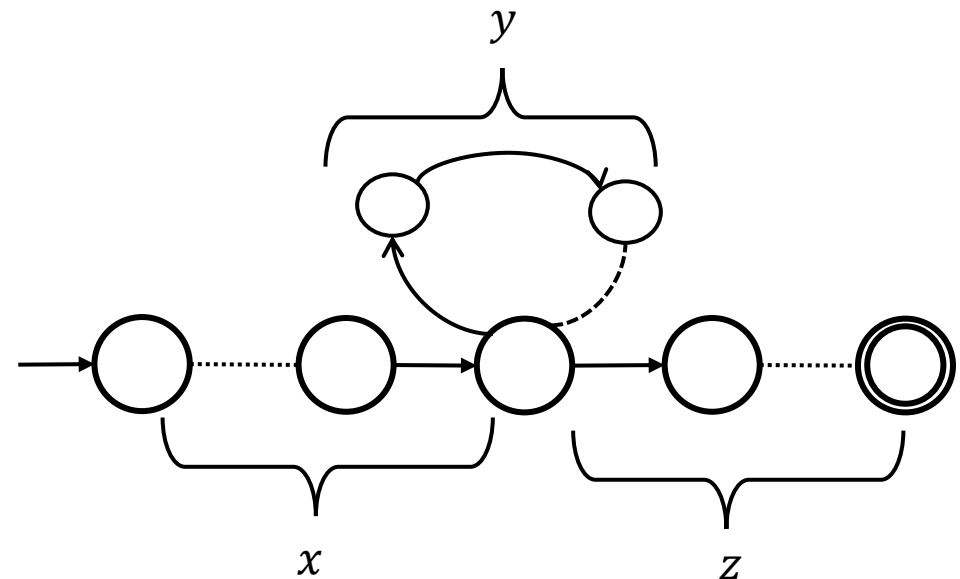
**Karl:** If your DFA accepts  $0^n 1^n$ , it must also accept  $0^{n+t} 1^n$ . This is because, if we take the loop one extra time, we read  $t$  more 0's.



**Contradiction as  $0^{n+t}1^n \notin L$ .** So Mil, you never had a DFA for  $L$  and in fact,  **$L$  is not regular.**

# Pumping Lemma

If  $L$  is a regular language, all strings in the language, larger than a certain length (pumping length), can be *pumped*: the string contains a certain section that can be repeated *any number of times* and the resulting string still  $\in L$ .

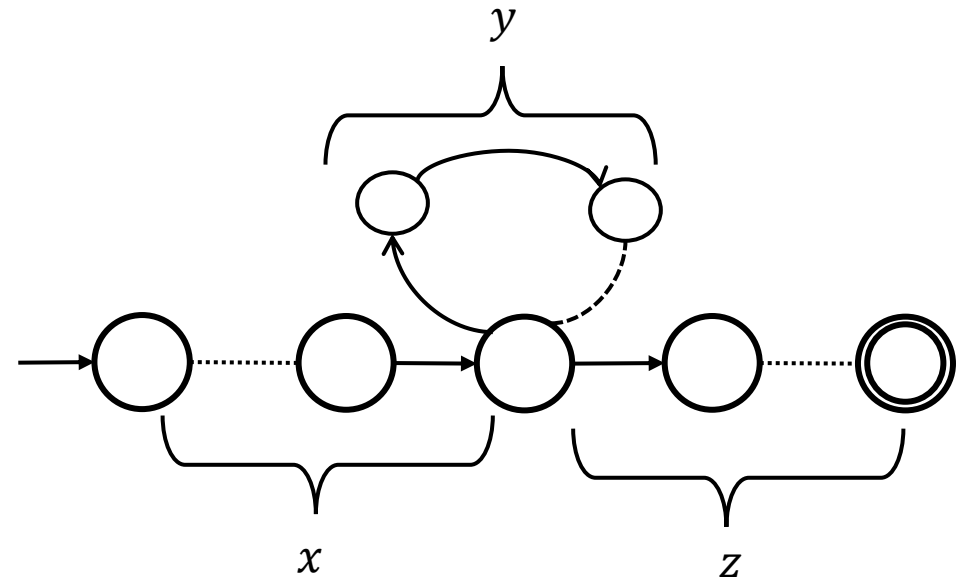


# Pumping Lemma

If  $L$  is a regular language, all strings in the language, larger than a certain length (pumping length), can be *pumped*: the string contains a certain section that can be repeated *any number of times* and the resulting string still  $\in L$ .

**(Pumping Lemma)** If  $L$  is a regular language, then there exists a number  $p$  (the pumping length) where for all  $s \in L$  of length at least  $p$ , there exists  $x, y, z$  such that  $s = xyz$ , such that

1.  $|xy| \leq p$ .
2.  $|y| \geq 1$
3.  $\forall i \geq 0, xy^iz \in L$ .



# Pumping Lemma

If  $L$  is a regular language, all strings in the language, larger than a certain length (pumping length), can be *pumped*: the string contains a certain section that can be repeated *any number of times* and the resulting string still  $\in L$ .

**(Pumping Lemma)** If  $L$  is a regular language, then there exists a number  $p$  (the pumping length) where for all  $s \in L$  of length at least  $p$ , there exists  $x, y, z$  such that  $s = xyz$ , such that

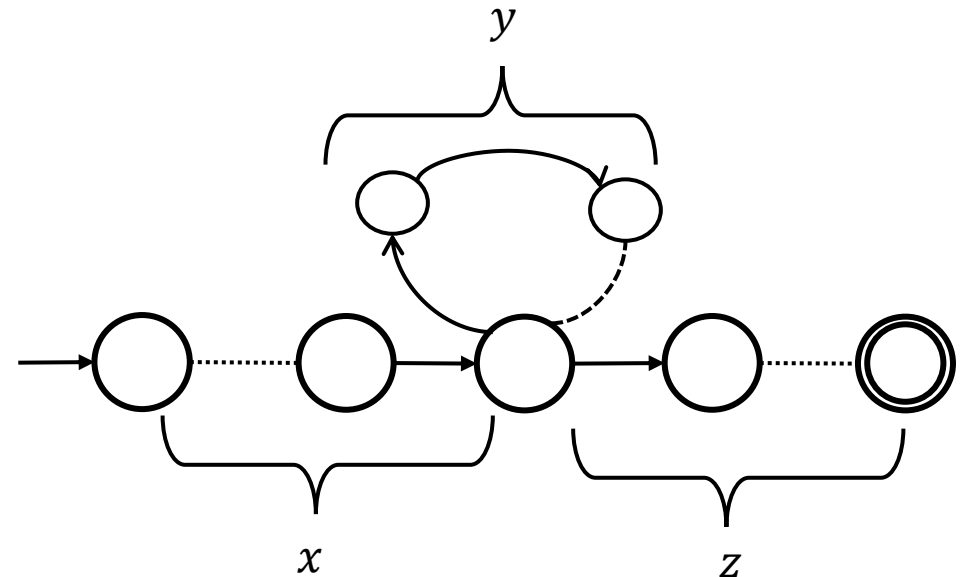
1.  $|xy| \leq p$ .
2.  $|y| \geq 1$
3.  $\forall i \geq 0, xy^iz \in L$ .

**Note:**  $(A \Rightarrow B) \equiv (\neg B) \Rightarrow (\neg A)$

If  $L$  is regular then, pumping property is satisfied

$\equiv$

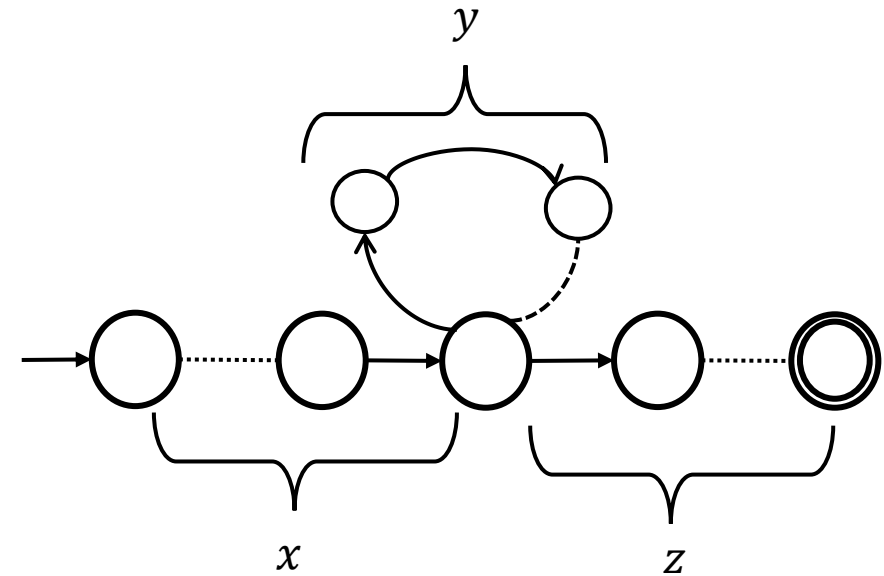
If pumping property is NOT satisfied, then  $L$  is NOT regular.



# Pumping Lemma

**Proof sketch:** Suppose that we have a DFA  $M$  of  $p$  states. Then any run in the DFA corresponding to strings of length at least  $p$ , some states are repeated.

This is because of the **pigeonhole principle**: any such run would encounter  $p + 1$  states, but there are  $p$  distinct states in the DFA.



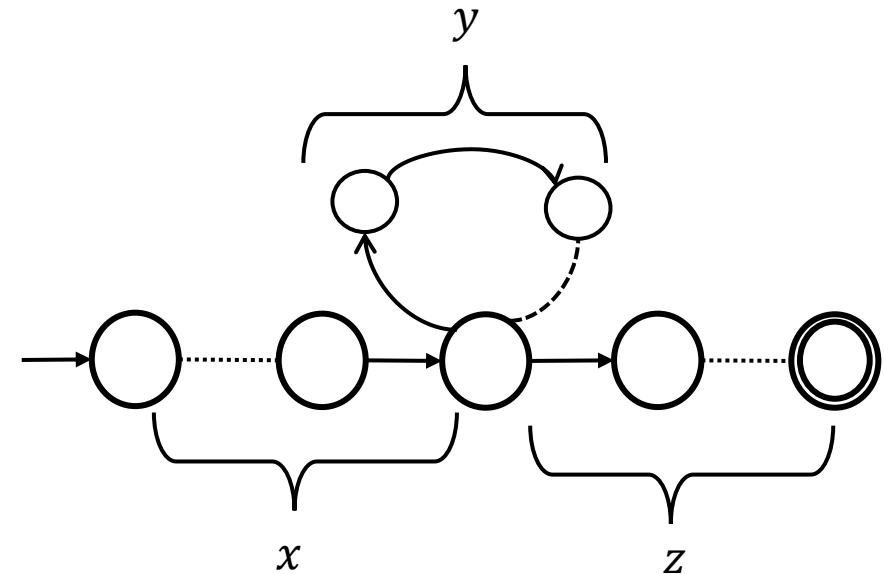
# Pumping Lemma

**Proof sketch:** Suppose that we have a DFA  $M$  of  $p$  states. Then any run in the DFA corresponding to strings of length at least  $p$ , some states are repeated.

This is because of the **pigeonhole principle**: any such run would encounter  $p + 1$  states, but there are  $p$  distinct states in the DFA.

Suppose  $s = s_1s_2 \cdots s_n$  be any such string of length  $n (\geq p)$  and suppose  $r_1r_2 \cdots r_{n+1}$  be the sequence of states encountered, while implementing a run of  $s$  in  $M$ .

As  $n + 1 \geq p + 1$ , in the above sequence at least two states must be repeated. Let them be  $r_j$  and  $r_l$ , i.e.,  $r_j = r_l$ , but  $j \neq l$ .





# Pumping Lemma

**Proof sketch:** Suppose that we have a DFA  $M$  of  $p$  states. Then any run in the DFA corresponding to strings of length at least  $p$ , some states are repeated.

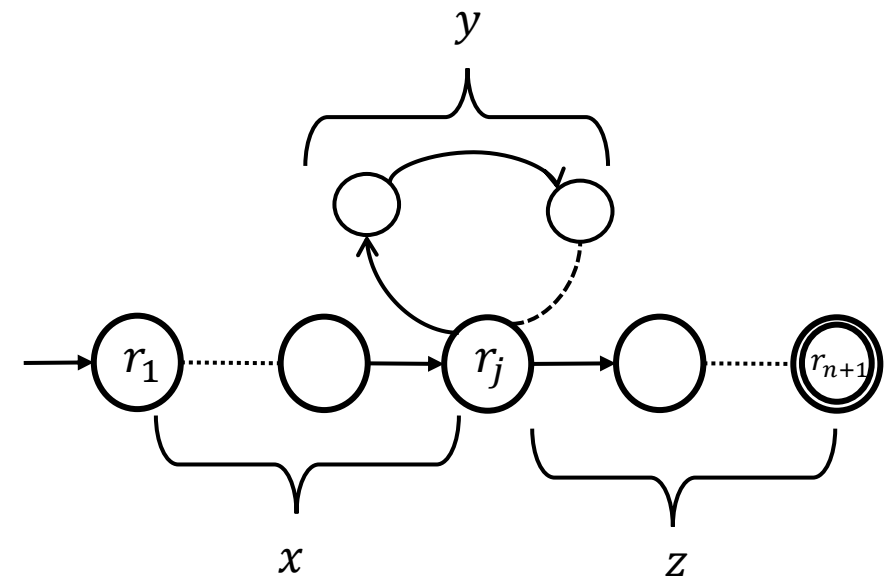
This is because of the **pigeonhole principle**: any such run would encounter  $p + 1$  states, but there are  $p$  distinct states in the DFA.

Suppose  $s = s_1 s_2 \cdots s_n$  be any such string of length  $n (\geq p)$  and suppose  $r_1 r_2 \cdots r_{n+1}$  be the sequence of states encountered, while implementing a run of  $s$  in  $M$ .

As  $n + 1 \geq p + 1$ , in the above sequence at least two states must be repeated. Let them be  $r_j$  and  $r_l$ , i.e.,  $r_j = r_l$ , but  $j \neq l$ .

So we can divide the  $s$  into three parts,  $x = s_1 \dots s_{j-1}$ ,  $y = s_j \dots s_{l-1}$ ,  $z = s_l \dots s_n$ . For a run on  $M$ , due to  $s$

- the  $x$  part takes us from  $r_1$  to  $r_j$
- the  $y$  part belongs to the loop part (we go from  $r_j$  to  $r_j$ )
- $z$  takes us from  $r_j$  to  $r_{n+1}$ , which is a final state if  $s \in L$ .



# Pumping Lemma

**Proof sketch:** Suppose that we have a DFA  $M$  of  $p$  states. Then any run in the DFA corresponding to strings of length at least  $p$ , some states are repeated.

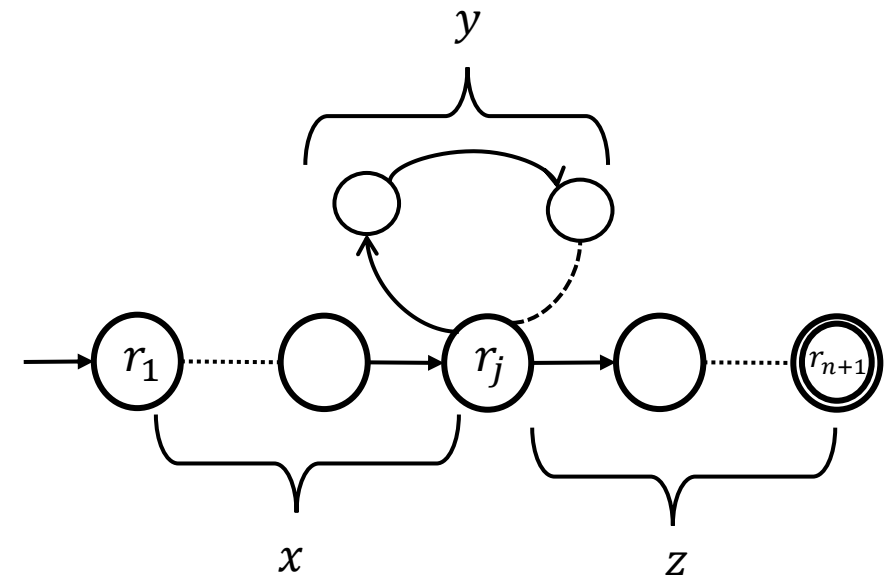
This is because of the **pigeonhole principle**: any such run would encounter  $p + 1$  states, but there are  $p$  distinct states in the DFA.

Suppose  $s = s_1 s_2 \cdots s_n$  be any such string of length  $n (\geq p)$  and suppose  $r_1 r_2 \cdots r_{n+1}$  be the sequence of states encountered, while implementing a run of  $s$  in  $M$ .

As  $n + 1 \geq p + 1$ , in the above sequence at least two states must be repeated. Let them be  $r_j$  and  $r_l$ , i.e.,  $r_j = r_l$ , but  $j \neq l$ .

So we can divide the  $s$  into three parts,  $x = s_1 \dots s_{j-1}$ ,  $y = s_j \dots s_{l-1}$ ,  $z = s_l \dots s_n$ . For a run on  $M$ , due to  $s$

- the  $x$  part takes us from  $r_1$  to  $r_j$
- the  $y$  part belongs to the loop part (we go from  $r_j$  to  $r_j$ )
- $z$  takes us from  $r_j$  to  $r_{n+1}$ , which is a final state if  $s \in L$ .



- We can traverse the loop bit any number of times and so  $\forall i \geq 0, xy^i z \in L$ .

# Pumping Lemma

**Proof sketch:** Suppose that we have a DFA  $M$  of  $p$  states. Then any run in the DFA corresponding to strings of length at least  $p$ , some states are repeated.

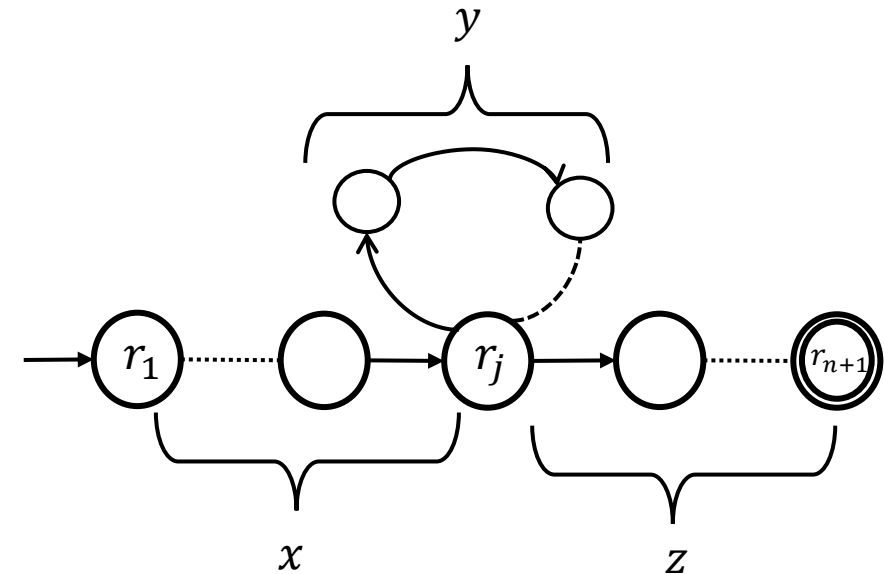
This is because of the **pigeonhole principle**: any such run would encounter  $p + 1$  states, but there are  $p$  distinct states in the DFA.

Suppose  $s = s_1 s_2 \cdots s_n$  be any such string of length  $n (\geq p)$  and suppose  $r_1 r_2 \cdots r_{n+1}$  be the sequence of states encountered, while implementing a run of  $s$  in  $M$ .

As  $n + 1 \geq p + 1$ , in the above sequence at least two states must be repeated. Let them be  $r_j$  and  $r_l$ , i.e.,  $r_j = r_l$ , but  $j \neq l$ .

So we can divide the  $s$  into three parts,  $x = s_1 \dots s_{j-1}$ ,  $y = s_j \dots s_{l-1}$ ,  $z = s_l \dots s_n$ . For a run on  $M$ , due to  $s$

- the  $x$  part takes us from  $r_1$  to  $r_j$
- the  $y$  part belongs to the loop part (we go from  $r_j$  to  $r_j$ )
- $z$  takes us from  $r_j$  to  $r_{n+1}$ , which is a final state if  $s \in L$ .



- We can traverse the loop bit any number of times and so  $\forall i \geq 0, xy^i z \in L$ .
- Also, as  $j \neq l$ ,  $|y| \geq 1$
- While reading the input, within the first  $p$  symbols of  $s$ , some state must be repeated.

# Pumping Lemma

**Proof sketch:** Suppose that we have a DFA  $M$  of  $p$  states. Then any run in the DFA corresponding to strings of length at least  $p$ , some states are repeated.

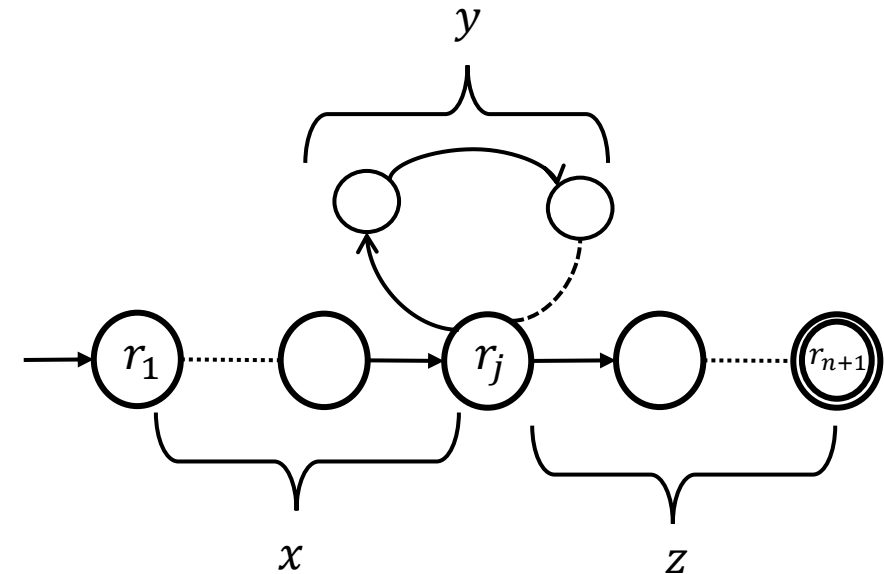
This is because of the **pigeonhole principle**: any such run would encounter  $p + 1$  states, but there are  $p$  distinct states in the DFA.

Suppose  $s = s_1s_2 \cdots s_n$  be any such string of length  $n (\geq p)$  and suppose  $r_1r_2 \cdots r_{n+1}$  be the sequence of states encountered, while implementing a run of  $s$  in  $M$ .

As  $n + 1 \geq p + 1$ , in the above sequence at least two states must be repeated. Let them be  $r_j$  and  $r_l$ , i.e.,  $r_j = r_l$ , but  $j \neq l$ .

So we can divide the  $s$  into three parts,  $x = s_1 \dots s_{j-1}$ ,  $y = s_j \dots s_{l-1}$ ,  $z = s_l \dots s_n$ . For a run on  $M$ , due to  $s$

- the  $x$  part takes us from  $r_1$  to  $r_j$
- the  $y$  part belongs to the loop part (we go from  $r_j$  to  $r_j$ )
- $z$  takes us from  $r_j$  to  $r_{n+1}$ , which is a final state if  $s \in L$ .



- We can traverse the loop bit any number of times and so  $\forall i \geq 0, xy^iz \in L$ .
- Also, as  $j \neq l$ ,  $|y| \geq 1$ , and
- The DFA reads  $|xy|$  by then and so  $|xy| \leq p$ .

# Pumping Lemma

In order to prove that a language is non-regular,

- Assume that it is regular and obtain a contradiction.
- Find a string in the language of length  $\geq p$  (pumping length) that cannot be pumped.

Examples of languages that are NOT regular:

- $\{0^p \mid p \text{ is prime}\}$
- $\{0^n 1^n \mid n \geq 0\}$
- $\{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$
- $\{\omega \mid \omega \text{ is palindrome}\}$
- $\vdots$
- $\vdots$

Refer to Sipser (or some other textbook) for proofs using Pumping lemma

# The story so far...

- We have built devices (DFAs/NFAs) that decides some languages.
- Regular languages are precisely the ones that are accepted by finite automata.
- For any  $L \in RL$ , we have DFA/NFA  $M$  such that  $L(M) = L$ .
- Regular expressions describe regular languages algebraically.
- There are languages that are not regular.

**DFA  $\equiv$  NFA  $\equiv$  Regular Expressions**

Next up:

- How do we generate the strings in a language?
- **Syntax:** What are the set of legal strings in a language?
- Think of the English language (Rules of **grammar**)

# Grammars

- **Grammars** provide a way to generate strings belonging to a language. The set of all strings generated by the grammar is the *language* of the grammar.
- ***Grammars generate languages:*** Grammars consist of a set of ***rules*** that allow you to construct strings of the language.
- For some classes of grammars, one can build automata that recognizes the language generated by the grammar.
- In fact, these concepts have been fundamental in attempts to formalize natural languages.

# Grammars

- **Grammars** provide a way to generate strings belonging to a language. The set of all strings generated by the grammar is the *language* of the grammar.
- ***Grammars generate languages:*** Grammars consist of a set of ***rules*** that allow you to construct strings of the language.
- For some classes of grammars, one can build automata that recognizes the language generated by the grammar.
- Consider these rules

*Sentence* → *Subject Verb Object*

*Subject* → *Noun.phrase*

*Object* → *Noun.phrase*

*Noun.phrase* → *Article Noun|Noun*

*Article* → ***the***

*Noun* → ***boy|girl|soccer|poetry***

*Verb* → ***loves|plays***



# Grammars

- **Grammars** provide a way to generate strings belonging to a language. The set of all strings generated by the grammar is the *language* of the grammar.
- ***Grammars generate languages:*** Grammars consist of a set of ***rules*** that allow you to construct strings of the language.
- For some classes of grammars, one can build automata that recognizes the language generated by the grammar.
- Consider these rules

*Sentence* → *Subject Verb Object*

*Subject* → *Noun.phrase*

*Object* → *Noun.phrase*

*Noun.phrase* → *Article Noun|Noun*

*Article* → ***the***

*Noun* → ***boy|girl|soccer|poetry***

*Verb* → ***loves|plays***

**Terminals** consist of strings over the alphabet corresponding to the language that the Grammar generates

**Variables:** {*Sentence, Subject, Verb, Object, Noun, Noun.phrase, Article*}, **Terminals:** {*the, girl, loves, plays, soccer, poetry*}

**Start Variable:** *Sentence*

# Grammars

- **Grammars** provide a way to generate strings belonging to a language. The set of all strings generated by the grammar is the *language* of the grammar.
- ***Grammars generate languages:*** Grammars consist of a set of ***rules*** that allow you to construct strings of the language.
- For some classes of grammars, one can build automata that recognizes the language generated by the grammar.
- Consider these rules

*Sentence* → *Subject Verb Object*

*Subject* → *Noun.phrase*

*Object* → *Noun.phrase*

*Noun.phrase* → *Article Noun|Noun*

*Article* → ***the***

*Noun* → ***boy|girl|soccer|poetry***

*Verb* → ***loves|plays***

The sentence “**the girl plays soccer**” can be derived from this set of rules.

**Variables:** {*Sentence, Subject, Verb, Object, Noun, Noun.phrase, Article*}, **Terminals:** {*the, girl, loves, plays, soccer, poetry*}

**Start Variable:** *Sentence*

# Grammars

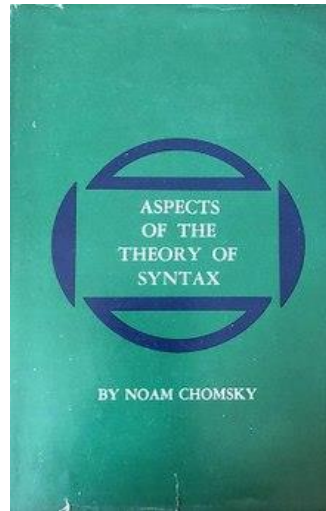
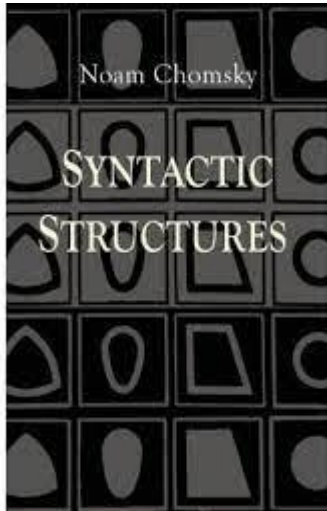
- **Grammars** provide a way to generate strings belonging to a language. The set of all strings generated by the grammar is the *language* of the grammar.
- **Grammars generate languages:** Grammars consist of a set of **rules** that allow you to construct strings of the language.
- For some classes of grammars, one can build automata that recognizes the language generated by the grammar.
- Consider these rules

*Sentence* → *Subject Verb Object*  
*Subject* → *Noun.phrase*  
*Object* → *Noun.phrase*  
*Noun.phrase* → *Article Noun|Noun*  
*Article* → ***the***  
*Noun* → ***boy|girl|soccer|poetry***  
*Verb* → ***loves|plays***

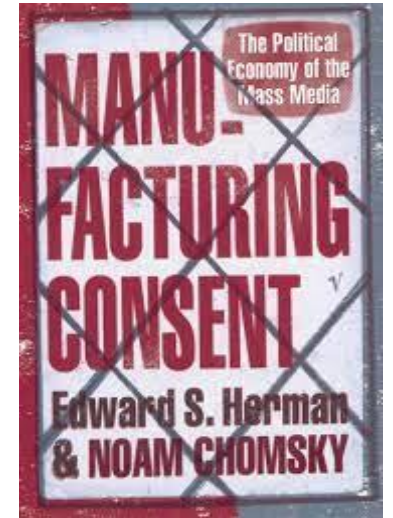
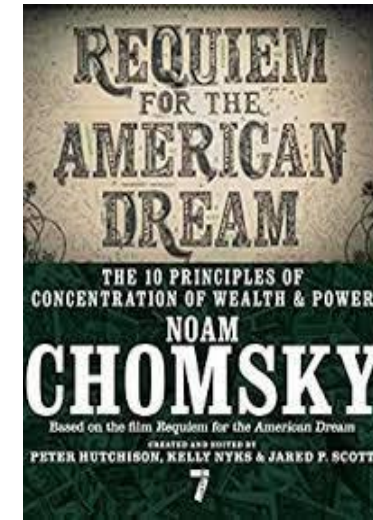
*Sentence* → *Subject Verb Object*  
→ *Noun.phrase Verb Object*  
→ *Article Noun Verb Object*  
→ ***the*** *Noun Verb Object*  
→ ***the girl*** *Verb Object*  
→ ***the girl plays*** *Object*  
→ ***the girl plays*** *Noun.phrase*  
→ ***the girl plays*** *Noun*  
→ ***the girl plays soccer***

**Variables:** {*Sentence, Subject, Verb, Object, Noun, Noun.phrase, Article*}, **Terminals:** {*The, girl, loves, plays, soccer, poetry*}  
**Start Variable:** *Sentence*

# Grammars



**Noam Chomsky**



- Noam Chomsky did pioneering work on linguistics and formalized many of these concepts.
- Also made great contributions to political economy and has been a champion of anti-imperialist, anti-capitalist, social justice struggles across the globe.

# Grammars

**(Grammar)** Formally, a *Grammar*  $G$  is a 5-tuple  $(V, \Sigma, P, S)$  such that

- $V$  is the set of **Variables**
- $\Sigma$  is the set of **Terminals** (disjoint from  $V$ )
- $P$  is the set of production **Rules**  $[(V \cup \Sigma)^* V (V \cup \Sigma)^* \rightarrow (V \cup \Sigma)^*]$
- $S$  is the **Start Variable**  $[ \text{The variable in the LHS of the first rule is generally the start variable} ]$

Eg: Consider the grammar  $G$

$X \rightarrow 1X$

$X \rightarrow 0Y$

$Y \rightarrow 0X$

$Y \rightarrow 1Y$

$Y \rightarrow \epsilon$

**X is the start variable of the Grammar.** Variables:  $\{X, Y\}$ , Terminals:  $\{\epsilon, 0, 1\}$

# Grammars

**(Grammar)** Formally, a *Grammar*  $G$  is a 5-tuple  $(V, \Sigma, P, S)$  such that

- $V$  is the set of **Variables**
- $\Sigma$  is the set of **Terminals** (disjoint from  $V$ )
- $P$  is the set of production **Rules**  $[(V \cup \Sigma)^* V (V \cup \Sigma)^* \rightarrow (V \cup \Sigma)^*]$
- $S$  is the **Start Variable** [ The variable in the LHS of the first rule is generally the start variable ]

## Grammars can be used to derive strings.

The sequence of **substitutions** (using the rules of  $G$ ) required to obtain a certain string is called a **derivation**.

- Begin the **derivation** from the **Start variable**.
- Replace any variable according to a rule. Repeat until only terminals remain.
- The generated string is **derived by the grammar**.

Eg: Consider the grammar  $G$

$X \rightarrow 1X$

$X \rightarrow 0Y$

$Y \rightarrow 1Y$

$Y \rightarrow 0X$

$Y \rightarrow \epsilon$

$X$ : Start Variable

$\{X, Y\}$ : Variables

$\{\epsilon, 0, 1\}$ : Terminals

The following is a derivation

$X \rightarrow 1X \rightarrow 11X \rightarrow 110Y \rightarrow 1101Y \rightarrow \mathbf{1101}$

# Grammars

**(Grammar)** Formally, a *Grammar*  $G$  is a 4-tuple  $(V, \Sigma, P, S)$  such that

- $V$  is the set of **Variables**
- $\Sigma$  is the set of **Terminals**
- $P$  is the set of production **Rules**       $[(V \cup T)^* V (V \cup T)^* \rightarrow (V \cup T)^*]$
- $S$  is the **Start Variable**      [ The variable in the LHS of the first rule is generally the start variable ]

- To show that a string  $w \in L(G)$ , we show that there exists a **derivation ending up in  $w$** . The fact that  $w$  can be derived using the rules of  $G$ , is expressed as  $S \xRightarrow{*} w$ .
- The **language of the grammar**,  $L(G)$  is  $\{w \in \Sigma^* \mid S \xRightarrow{*} w\}$

# Grammars

**(Grammar)** Formally, a *Grammar*  $G$  is a 5-tuple  $(V, \Sigma, P, S)$  such that

- $V$  is the set of **Variables**
- $\Sigma$  is the set of **Terminals**
- $P$  is the set of production **Rules**       $[(V \cup T)^* V (V \cup T)^* \rightarrow (V \cup T)^*]$
- $S$  is the **Start Variable**      [ The variable in the LHS of the first rule is generally the start variable ]

- To show that a string  $w \in L(G)$ , we show that there exists a **derivation ending up in  $w$** . The fact that  $w$  can be derived using the rules of  $G$ , is expressed as  $S \xRightarrow{*} w$ .
- The **language of the grammar**,  $L(G)$  is  $\{w \in \Sigma^* | S \xRightarrow{*} w\}$

Eg: Consider the grammar  $G$

$X \rightarrow 1X$   
 $X \rightarrow 0Y$   
 $Y \rightarrow 1Y$   
 $Y \rightarrow 0X$   
 $Y \rightarrow \epsilon$

**The string  $1101 \in L(G)$  because there exists the following derivation**

$X \rightarrow 1X \rightarrow 11X \rightarrow 110Y \rightarrow 1101Y \rightarrow 1101$



# Grammars for Regular Languages

**Regular grammar:** If the *rules* of the underlying grammar  $G$  are of the form

$$Var \rightarrow Ter \mathbf{Var}$$

$$Var \rightarrow Ter$$

$$Var \rightarrow \epsilon$$

then the language of the grammar is **regular**. Also known as **Right-linear grammar** (all variables are to the right of terminals in the RHS).

## Right linear Grammar to DFA

Eg: Consider the grammar  $G$

$$X \rightarrow 1X$$

$$X \rightarrow 0Y$$

$$Y \rightarrow 1Y$$

$$Y \rightarrow 0X$$

$$Y \rightarrow \epsilon \text{ (indicates that } Y \text{ is the final state)}$$

# Grammars for Regular Languages

**Regular grammar:** If the *rules* of the underlying grammar  $G$  are of the form

$$Var \rightarrow Ter \mathbf{Var}$$

$$Var \rightarrow Ter$$

$$Var \rightarrow \epsilon$$

then the language of the grammar is **regular**. Also known as **Right-linear grammar** (all variables are to the right of terminals in the RHS).

## Right linear Grammar to DFA

Eg: Consider the grammar  $G$

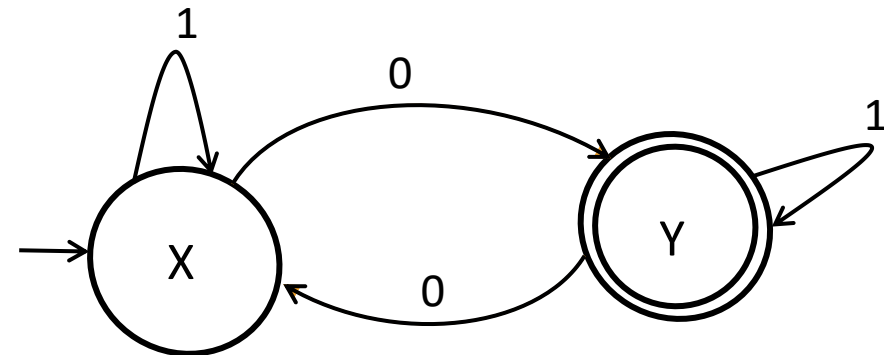
$$X \rightarrow 1X$$

$$X \rightarrow 0Y$$

$$Y \rightarrow 1Y$$

$$Y \rightarrow 0X$$

$$Y \rightarrow \epsilon \text{ (indicates that } Y \text{ is the final state)}$$



# Grammars for Regular Languages

**Regular grammar:** If the *rules* of the underlying grammar  $G$  are of the form

$$Var \rightarrow Ter \mathbf{Var}$$

$$Var \rightarrow Ter$$

$$Var \rightarrow \epsilon$$

then the language of the grammar is **regular**. Also known as **Right-linear grammar** (all variables are to the right of terminals in the RHS).

## Right linear Grammar to DFA

Eg: Consider the grammar  $G$

$$X \rightarrow 1X$$

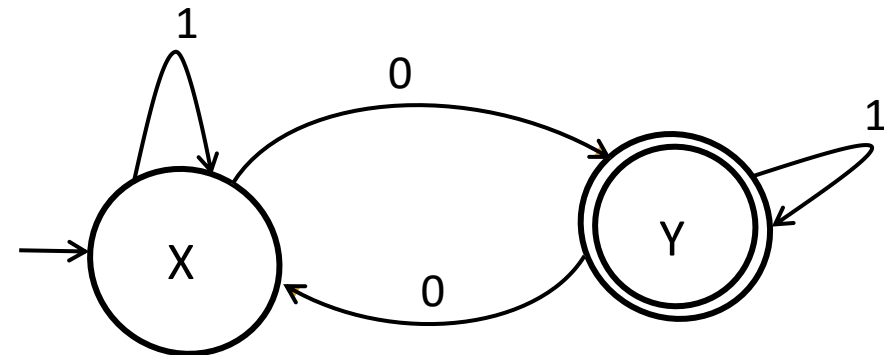
$$X \rightarrow 0Y$$

$$Y \rightarrow 1Y$$

$$Y \rightarrow 0X$$

$$Y \rightarrow \epsilon \text{ (indicates that } Y \text{ is the final state)}$$

A **run** in a DFA model is analogous to a **derivation** in a linear grammar.



For the string **1101**:

**Derivation:**  $X \rightarrow 1X \rightarrow 11X \rightarrow 110Y \rightarrow 1101Y \rightarrow 1101$ . So  $1101 \in L(G)$

**Run:**  $X \xrightarrow{1} X \xrightarrow{1} X \xrightarrow{0} Y \xrightarrow{1} Y$  (Accepting Run and so  $1101 \in L(M)$ ).

# Grammars for Regular Languages

**Regular grammar:** If the *rules* of the underlying grammar  $G$  are of the form

$$Var \rightarrow Ter \mathbf{Var}$$

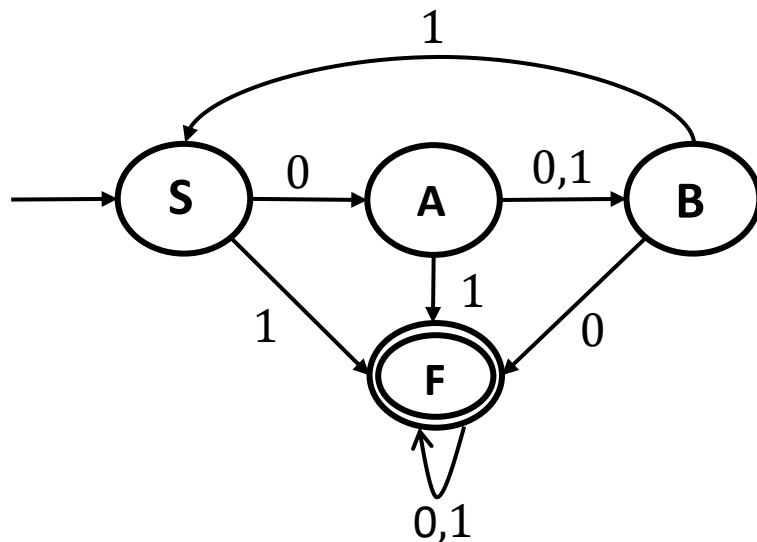
$$Var \rightarrow Ter$$

$$Var \rightarrow \epsilon$$

then the language of the grammar is **regular**. Also known as **Right-linear grammar** (all variables are to the right of terminals in the RHS).

## DFA to Right linear Grammar

Consider the following DFA  $M$



The right-linear grammar  $G$  for  $M$

$$S \rightarrow 0A$$

$$A \rightarrow 01B$$

$$B \rightarrow 1S$$

$$F \rightarrow 01F$$

$$A \rightarrow 1F$$

$$B \rightarrow 0F$$

$$S \rightarrow 1F$$

$$F \rightarrow \epsilon$$

# Grammars for Regular Languages

Right-linear grammar  $\equiv$  DFA  $\equiv$  NFA  $\equiv$  Regular Expressions

**Left linear grammar:** If the *rules* of the underlying grammar  $G$  are of the form

$$Var \rightarrow \mathbf{Var} Ter$$

$$Var \rightarrow Ter$$

$$Var \rightarrow \epsilon$$

then such a grammar is called **Left-linear** (all Variables are to the left of terminals in the RHS).

**Right linear grammars are equivalent to Left-linear grammar** (We won't be proving it here )

# Grammars for Regular Languages

**Right-linear grammar  $\equiv$  DFA  $\equiv$  NFA  $\equiv$  Regular Expressions**

**Left linear grammar:** If the *rules* of the underlying grammar  $G$  are of the form

$$Var \rightarrow \mathbf{Var} Ter$$

$$Var \rightarrow Ter$$

$$Var \rightarrow \epsilon$$

then such a grammar is called **Left-linear** (all Variables are to the left of terminals in the RHS).

**Right linear grammars are equivalent to Left-linear grammar** (We won't be proving it here)

**Right-linear grammars and Left-linear grammars generate Regular Languages.**

Note that mixing left-linear grammars and right-linear grammars in the same set of rules **won't generate regular languages**. (e.g:  $S \rightarrow aX, X \rightarrow Sb, S \rightarrow \epsilon$ )

**Left-linear grammar  $\equiv$  Right-linear grammar  $\equiv$  DFA  $\equiv$  NFA  $\equiv$  Regular Expressions**

# Context free Grammars

**(Grammar)** Formally, a *Grammar*  $G$  is a 4-tuple  $(V, \Sigma, P, S)$  such that

- $V$  is the set of **Variables**
- $\Sigma$  is the set of **Terminals**
- $P$  is the set of production **Rules**
- $S$  is the **Start Variable**

$$[ (V \cup T)^* V (V \cup T)^* \rightarrow (V \cup T)^* ]$$

[ The variable in the LHS of the first rule is generally the start variable ]

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (V \cup T)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammar is called a ***context-free language***.

Immediately we find that the *rules* are less restrictive than left-linear grammars and right-linear grammars. Context free grammars allow

$$Var \rightarrow Anything$$

$$Var \rightarrow \text{String of Variables} \mid \text{String of Terminals} \mid \text{Strings of Variables and Terminals} \mid \epsilon$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

Immediately we find that the *rules* are less restrictive than left-linear grammars and right-linear grammars. Context free grammars allow

$$Var \rightarrow Anything$$

$$Var \rightarrow String\ of\ Variables | String\ of\ Terminals | Strings\ of\ Variables\ and\ Terminals | \epsilon$$

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**



# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1$$

$$S \rightarrow \epsilon$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | \epsilon$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|\epsilon$$

What is the language generated by this grammar?

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|\epsilon$$

What is the language generated by this grammar?

Strings that can be derived from  $G$ :

$$S \rightarrow \epsilon$$

$$\{\epsilon\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | \epsilon$$

What is the language generated by this grammar?

Strings that can be derived from  $G$ :

$$S \rightarrow 0S1 \rightarrow 01$$

$$\{\epsilon, 01\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | \epsilon$$

What is the language generated by this grammar?

Strings that can be derived from  $G$ :

$$S \rightarrow 0S1 \rightarrow 00S11 \rightarrow 0011$$

$$\{\epsilon, 01, 0011\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | \epsilon$$

What is the language generated by this grammar?

Strings that can be derived from  $G$ :

$$S \rightarrow 0S1 \rightarrow 00S11 \rightarrow 000S111 \rightarrow 000111$$

$$\{\epsilon, 01, 0011, 000111\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | \epsilon$$

Strings that can be derived from  $G$ :

$$\{\epsilon, 01, 0011, 000111, 0^4 1^4, \dots\}$$

What is the language generated by this grammar?

$$L(G) = \{\omega | \omega = 0^n 1^n, n \geq 0\}$$

**So although  $L(G)$  is not regular, it is context-free.**



# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

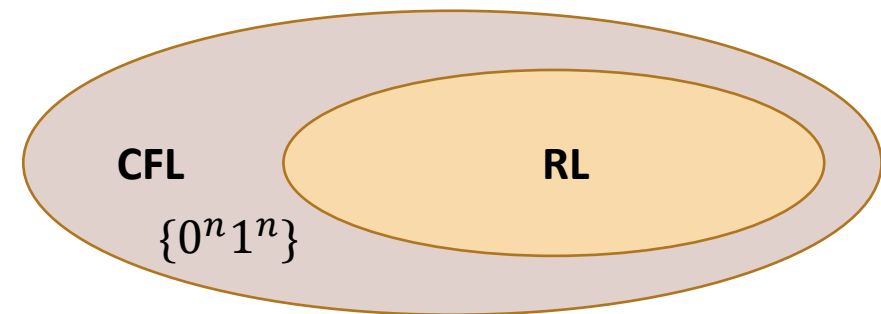
Any language generated by a context-free grammars is called a ***context-free language***.

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages  $\subset$  Context Free Languages.**

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | \epsilon$$

What is the language generated by this grammar?



$$L(G) = \{\omega | \omega = 0^n 1^n, n \geq 0\}$$

**So although  $L(G)$  is not regular, it is context-free.**

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

**Regular languages**  $\subset$  **Context Free Languages**.

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$S \rightarrow \epsilon$$

$$\{\epsilon\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

**Regular languages**  $\subset$  **Context Free Languages**.

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | SS | \epsilon$$

Strings that can be derived by  $G$ :

$$S \rightarrow 0S1 \rightarrow 00S11 \dots$$

$$\{\epsilon, 01, 0011, \dots 0^n 1^n\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

**Regular languages**  $\subset$  **Context Free Languages**.

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$S \rightarrow \mathbf{0S1} \rightarrow \mathbf{0SS1} \rightarrow \mathbf{00S1S1} \rightarrow \mathbf{001S1} \rightarrow \mathbf{0010S11} \rightarrow \mathbf{001011}$$

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, \dots\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

**Regular languages**  $\subset$  **Context Free Languages**.

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$S \rightarrow \mathbf{0S1} \rightarrow \mathbf{0SS1} \rightarrow \mathbf{00S1S1} \rightarrow \mathbf{001S1} \rightarrow \mathbf{0010S11} \rightarrow \mathbf{001011}$$

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, \dots\}$$

**Show that the string  $010101 \in L(G)$ .**

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

**Regular languages**  $\subset$  **Context Free Languages**.

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$S \rightarrow \mathbf{SS} \rightarrow \mathbf{SSS} \rightarrow \mathbf{0S1SS} \rightarrow 0S1\mathbf{0S1S} \rightarrow 0S10S1\mathbf{0S1} \rightarrow 010101$$

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

# Context free Grammars

**Context-Free Grammars:** If the *rules* of the underlying grammar  $G$  are of the form

$$V \rightarrow (VUT)^*$$

then such a grammar is called **Context-Free**.

**Regular languages**  $\subset$  **Context Free Languages**.

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$S \rightarrow SS \rightarrow SSS \rightarrow 0S1SS \rightarrow 0S10S1S \rightarrow 0S10S10S1 \rightarrow 010101$$

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**

# Context free Grammars

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**



# Context free Grammars

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**

You can see what the language is, if you replace **0** with ( and **1** with )

# Context free Grammars

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | SS | \epsilon$$

Strings that can be derived by  $G$ :

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**

You can see what the language is, if you replace **0** with ( and **1** with )

Strings that can be derived by  $G$ :  $\{\epsilon, 01, 0011, \dots, 0^n 1^n, 001011, 010101, \dots\}$

$$\{\epsilon, ()\}$$

# Context free Grammars

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 \mid SS \mid \epsilon$$

Strings that can be derived by  $G$ :

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**

You can see what the language is, if you replace **0** with ( and **1** with )

Strings that can be derived by  $G$ :  $\{\epsilon, 01, 0011, \dots, 0^n 1^n, 001011, 010101, \dots\}$

$$\{\epsilon, ( ), (( )), \dots, \}$$

# Context free Grammars

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1 | SS | \epsilon$$

Strings that can be derived by  $G$ :

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**

You can see what the language is, if you replace **0** with ( and **1** with )

Strings that can be derived by  $G$ :  $\{\epsilon, 01, 0011, \dots, 0^n 1^n, 001011, 010101, \dots\}$

$$\{\epsilon, ( ), (( )), \dots, ((((((\dots))))))\}$$

# Context free Grammars

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**

You can see what the language is, if you replace **0** with ( and **1** with )

Strings that can be derived by  $G$ :  $\{\epsilon, 01, 0011, \dots, 0^n 1^n, 001011, 010101, \dots\}$

$$\{\epsilon, ( ), (( )), \dots, ((((((\dots)))))), (( ) ( )), \dots\}$$

# Context free Grammars

Consider the Grammar  $G$  with the following rules:

$$S \rightarrow 0S1|SS|\epsilon$$

Strings that can be derived by  $G$ :

$$\{\epsilon, 01, 0011, \dots 0^n 1^n, 001011, 010101, \dots\}$$

**What is  $L(G)$ ?**

You can see what the language is, if you replace **0** with ( and **1** with )

Strings that can be derived by  $G$ :  $\{\epsilon, 01, 0011, \dots, 0^n 1^n, 001011, 010101, \dots\}$

$$\{\epsilon, ( ), (( )), \dots, ((((((\dots)))))), (( ) ( )), 000, \dots\}$$

**So,  $L(G)$  is the language of all strings of properly nested parentheses.**

$$L(G) = \{\omega | \omega \text{ is a correctly nested parenthesis}\}$$

# Context free Grammars

## Constructing CFG corresponding to a Language.

There is no fixed recipe for doing this. Requires some level of creativity.

Some tips might come in handy:

- Check if the CFL is a union of simpler languages. If  $L(G) = L(G_1) \cup L(G_2)$  and  $G_1$  and  $G_2$  are known. If  $S_1$  is the start variable for  $G_1$  and  $S_2$  is the start variable for  $G_2$  then the rules of  $G$ :

$$S \rightarrow S_1 | S_2$$

$$S_1 \rightarrow \dots$$

$$S_2 \rightarrow \dots$$

# Context free Grammars

## Constructing CFG corresponding to a Language.

There is no fixed recipe for doing this. Requires some level of creativity.

Some tips might come in handy:

- Check if the CFL is a union of simpler languages. If  $L(G) = L(G_1) \cup L(G_2)$  and  $G_1$  and  $G_2$  are known. If  $S_1$  is the start variable for  $G_1$  and  $S_2$  is the start variable for  $G_2$  then the rules of  $G$ :

$$S \rightarrow S_1 | S_2$$

$$S_1 \rightarrow \dots$$

$$S_2 \rightarrow \dots$$

- Grammars with rules such as  $S \rightarrow aSb$  help generate strings where the corresponding machine would need unbounded memory to *remember* the number of  $a$ 's needed to verify that there are an equal number of  $b$ 's. This was not possible with regular expressions/linear grammars.



# Context free Grammars

## Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as  $S \rightarrow aSb$  help generate where the portions of  $a$  and  $b$  are equal.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega | \omega \text{ has equal number of 0's and 1's}\}$

# Context free Grammars

## Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as  $S \rightarrow aSb$  help generate where the portions of  $a$  and  $b$  are equal.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

- The first thing to notice is that  $L_1 = \{0^n 1^n, n \geq 0\} \subset L(G)$ . We know the grammar for this language.
- Any string  $\omega \in L_1$  has a series of 0's followed by an equal number of 1's.
- Again, consider  $L_2$  to comprise all strings that start with a series of 1's followed by an equal number of 0's, i.e.

$$L_2 = \{1^n 0^n, n \geq 0\}$$

- The grammar for  $L_2$  is similar to that of  $L_1$ : replace the 0's with 1's and vice versa. Importantly,  $L_2 = \{1^n 0^n, n \geq 0\} \subset L(G)$  also.
- Also,  $L_1 \cup L_2 \subset L(G)$

# Context free Grammars

## Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as  $S \rightarrow aSb$  help generate where the portions of  $a$  and  $b$  are equal.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

- So  $L'(G') = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\} \subset L(G)$
- Grammar for  $L_1$ :  $S \rightarrow 0S1 \mid \epsilon$
- Grammar for  $L_2$ :  $S \rightarrow 1S0 \mid \epsilon$
- Grammar for  $L_1 \cup L_2$ :

$$\begin{aligned} S &\rightarrow S_1 \mid S_2 \\ S_1 &\rightarrow 0S_1 1 \mid \epsilon \\ S_2 &\rightarrow 1S_2 0 \mid \epsilon \end{aligned}$$

# Context free Grammars

## Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as  $S \rightarrow aSb$  help generate where the portions of  $a$  and  $b$  are equal.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

- Grammar for  $L_1 \cup L_2$ :

$$\begin{aligned} S &\rightarrow S_1 | S_2 \\ S_1 &\rightarrow 0S_11 | \epsilon \\ S_2 &\rightarrow 1S_20 | \epsilon \end{aligned}$$

- **Is that all? Is  $L_1 \cup L_2 = L(G)$ ?**  $L_1 \cup L_2$  contains all strings that have equal number 0's followed by equal number of 1's or vice versa.

# Context free Grammars

## Constructing CFG corresponding to a Language.

- Check if the CFL is a union of simpler languages.
- Grammars with rules such as  $S \rightarrow aSb$  help generate where the portions of  $a$  and  $b$  are equal.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

- Grammar for  $L_1 \cup L_2$ :

$$\begin{aligned} S &\rightarrow S_1 | S_2 \\ S_1 &\rightarrow 0S_1 1 | \epsilon \\ S_2 &\rightarrow 1S_2 0 | \epsilon \end{aligned}$$

- **Is that all? Is  $L_1 \cup L_2 = L(G)$ ?**  $L_1 \cup L_2$  contains all strings that have equal number 0's followed by equal number of 1's or vice versa.
- What about strings such as  $s_1 = 0101 \dots$  and  $s_2 = 1010 \dots$ ? For this we need to be able to go from

$$0S_1 1 \rightarrow 0S_2 1 \rightarrow 01S_2 01 \rightarrow \dots$$

# Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

- Grammar for  $L_1 \cup L_2$ :

$$\begin{aligned} S &\rightarrow S_1 | S_2 \\ S_1 &\rightarrow 0S_1 1 | \epsilon \\ S_2 &\rightarrow 1S_2 0 | \epsilon \end{aligned}$$

- What about strings such as  $s_1 = 0101 \dots$  and  $s_2 = 1010 \dots$ ? Add transitions  $S_1 \rightarrow S_2$  and  $S_2 \rightarrow S_1$ .

$$\begin{aligned} S &\rightarrow S_1 | S_2 \\ S_1 &\rightarrow 0S_1 1 | \epsilon \\ S_2 &\rightarrow 1S_2 0 | \epsilon \\ S_1 &\rightarrow S_2 \\ S_2 &\rightarrow S_1. \end{aligned}$$

# Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

$$\begin{aligned} S &\rightarrow S_1 | S_2 \\ S_1 &\rightarrow 0S_1 1 | \epsilon \\ S_2 &\rightarrow 1S_2 0 | \epsilon \\ S_1 &\rightarrow S_2 \\ S_2 &\rightarrow S_1 \end{aligned}$$

- Can't we simplify this? We can replace  $S_1$  and  $S_2$  with a single Start variable as follows:  $S \rightarrow 0S1 | 1S0 | \epsilon$
- What kind of strings does the grammar generate? Well if we use Rule  $S \rightarrow 0S1$ ,  $m$  times, we get to rules such as  $0^m S 1^m$ .
- Now applying the rule  $S \rightarrow 1S0$ ,  $k$  times, we get  $0^m 1^k S 0^k 1^m$ .
- So the strings we obtain are of the form:

$$\{0^{m_1} 1^{n_1} 0^{m_2} 1^{n_2} \dots 0^{n_2} 1^{m_2} 0^{n_1} 1^{m_1}\} \cup \{1^{m_1} 0^{n_1} 1^{m_2} 0^{n_2} \dots 1^{n_2} 0^{m_2} 1^{n_1} 0^{m_1}\} \in L(G)$$

# Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

$$\begin{aligned} S &\rightarrow S_1 S_2 \\ S_1 &\rightarrow 0 S_1 1 \mid \epsilon \\ S_2 &\rightarrow 1 S_2 0 \mid \epsilon \\ S_1 &\rightarrow S_2 \\ S_2 &\rightarrow S_1 \end{aligned}$$

- Simplified grammar:

$$S \rightarrow 0 S 1 \mid 1 S 0 \mid \epsilon$$



# Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

$$\begin{aligned} S &\rightarrow S_1 | S_2 \\ S_1 &\rightarrow 0S_11 | \epsilon \\ S_2 &\rightarrow 1S_20 | \epsilon \\ S_1 &\rightarrow S_2 \\ S_2 &\rightarrow S_1 \end{aligned}$$

- Simplified grammar:

$$S \rightarrow 0S1 | 1S0 | \epsilon$$

- Is that all? What about strings such as  $\{0110, 00111100\}$ ?
- More generally, what about strings that are a concatenation of  $L_1$  and  $L_2$ ?

# Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

$$\begin{aligned} S &\rightarrow S_1 S_2 \\ S_1 &\rightarrow 0S_1 1 \mid \epsilon \\ S_2 &\rightarrow 1S_2 0 \mid \epsilon \\ S_1 &\rightarrow S_2 \\ S_2 &\rightarrow S_1 \end{aligned}$$

- Simplified grammar:

$$S \rightarrow 0S1 \mid 1S0 \mid \epsilon$$

- Is that all? What about strings such as  $\{0110, 00111100\}$ ?
- More generally, what about strings that are a concatenation of  $L_1$  and  $L_2$ ?
- Adding transitions like  $S \rightarrow S_1 S_2$  incorporates this.

# Context free Grammars

Constructing CFG corresponding to a Language.

Example: Construct the grammar  $G$  such that  $L(G) = \{\omega \mid \omega \text{ has equal number of 0's and 1's}\}$

$$S \rightarrow S_1 S_2 \mid S_1 S_2 S_1$$

$$S_1 \rightarrow 0S_11 \mid \epsilon$$

$$S_2 \rightarrow 1S_20 \mid \epsilon$$

$$S_1 \rightarrow S_2$$

$$S_2 \rightarrow S_1$$

- Simplify this further.

$$G: S \rightarrow SS \mid 0S1 \mid 1S0 \mid \epsilon$$

Thank You!