# CS 302.1 - Automata Theory

## Lecture 04


## Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

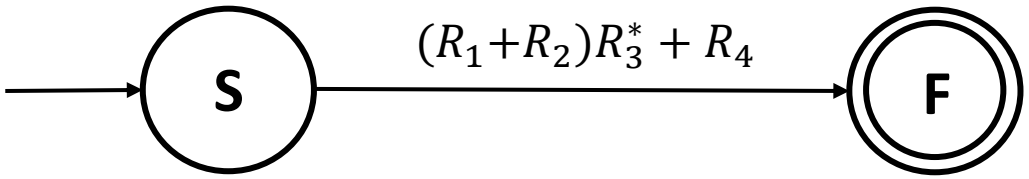
IIIT Hyderabad


# 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.



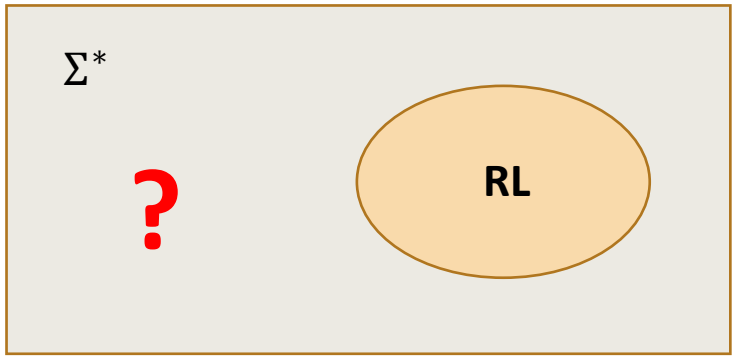

**DFA, NFA, Regular Expressions have equal power and all of them correspond to Regular Languages**

# Pumping Lemma

Recall that so far, we have proven that the following statements are all equivalent:

- $L$ is a regular language.
- There is a DFA $D$ such that $\mathcal{L}(D) = L$.
- There is an NFA $N$ such that $\mathcal{L}(N) = L$.
- There is a regular expression $R$ such that $\mathcal{L}(R) = L$.

- Not all languages are regular.




# Pumping Lemma

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

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

# Pumping Lemma

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

Let $\Sigma = \{0,1\}$. Consider the language $L = \{0^n 1^n | n \geq 0\}$ 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 $n = 10$)

# Pumping Lemma

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

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

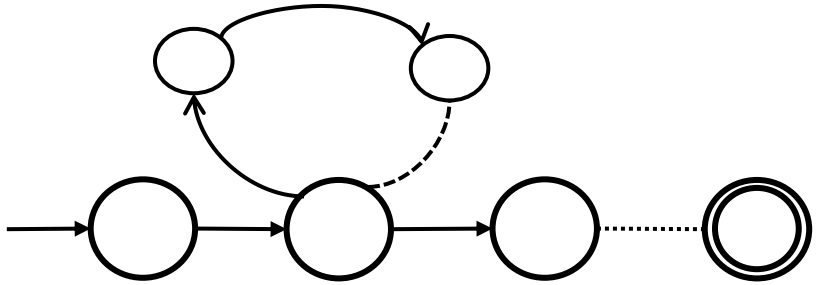
**Mil:** I have a DFA for $L$.
**Karl:** How many states are there?
**Mil:** $n$-states (say $n = 10$)
**Karl:** Then $0^{10}1^{10}$ must be accepted.
By the **pigeonhole principle**, while reading the first $(n = 10)$ symbols, some states need to be revisited. Otherwise $n + 1 = 11$ states would have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

# Pumping Lemma

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

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

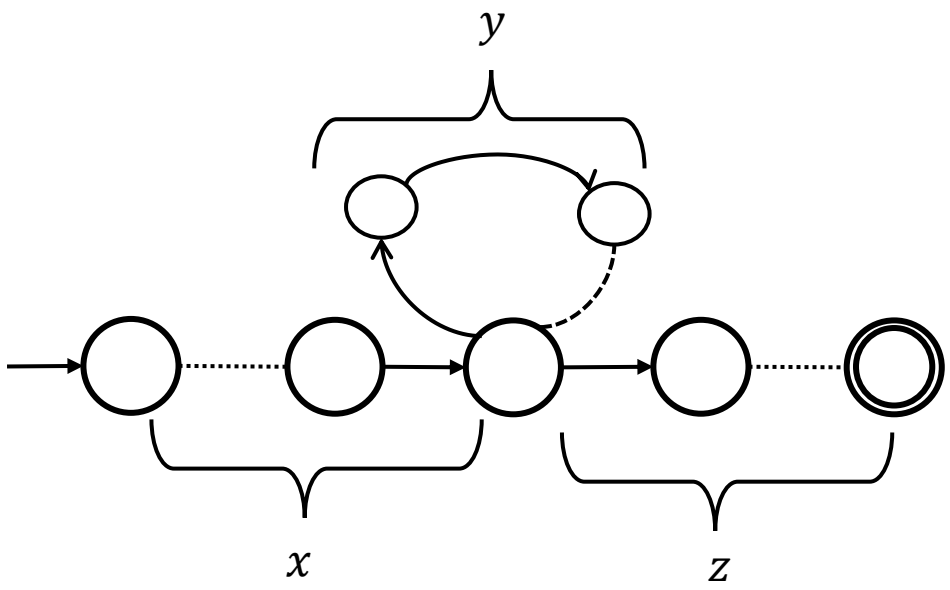
**Mil:** I have a DFA for $L$.
**Karl:** How many states are there?
**Mil:** $n$-states (say $n = 10$)
**Karl:** Then $0^{10}1^{10}$ must be accepted. By the **pigeonhole principle**, while reading the first $(n = 10)$ symbols, some states need to be revisited. Otherwise $n + 1 = 11$ states would have 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, $\boldsymbol{L}$ **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 $\in L$.

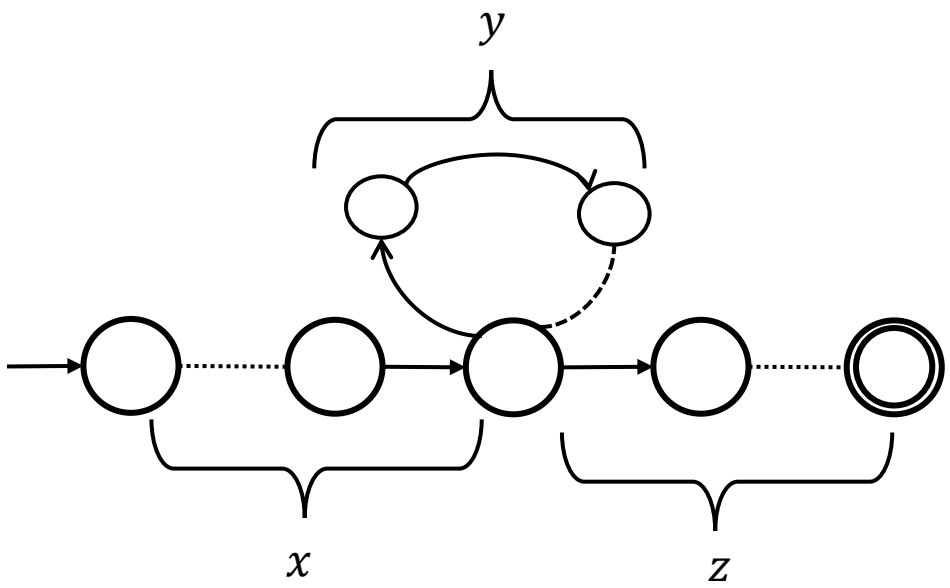


# Pumping Lemma

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. $\forall i \geq 0, xy^i z \in L$.



# 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

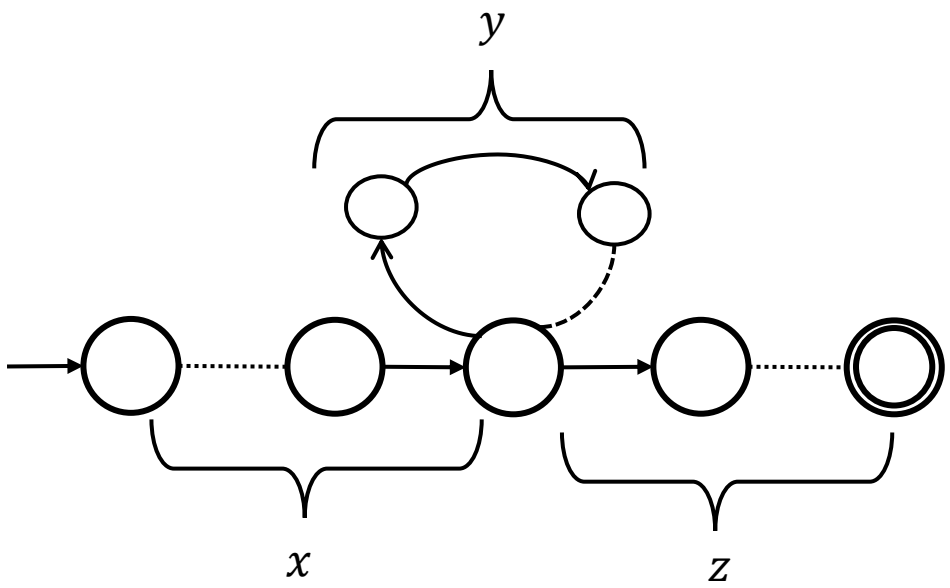
1. $|xy| \leq p$.
2. $|y| \geq 1$
3. $\forall i \geq 0, xy^i z \in L$.

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

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

$\equiv$

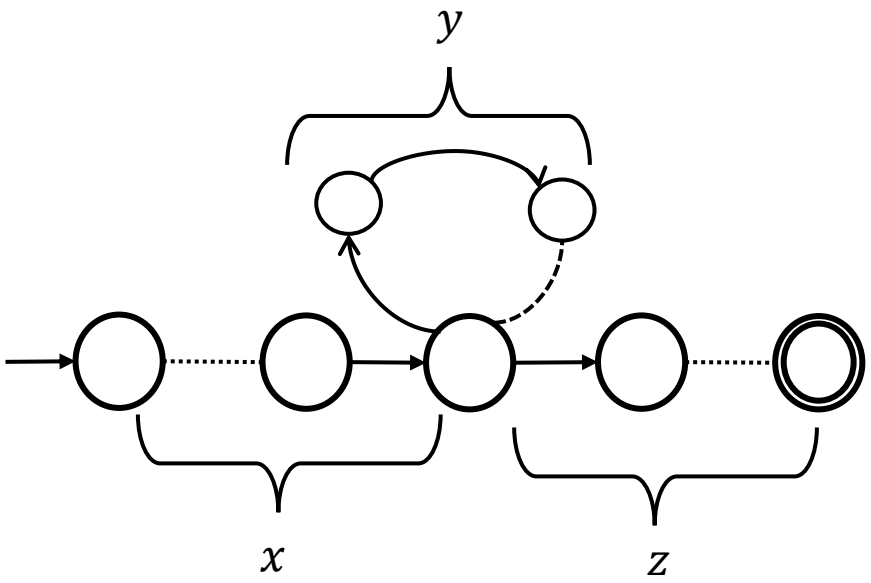
**If pumping property is NOT satisfied, then $L$ is NOT regular.**



# 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.



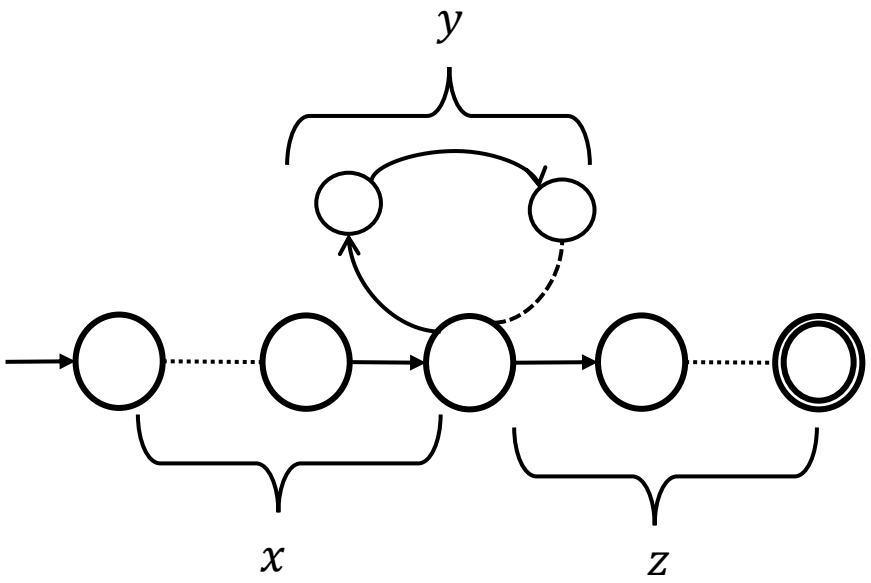
# Pumping Lemma

**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 $s = s_1 s_2 \cdots s_n$ 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 $j \neq l$.



# 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.

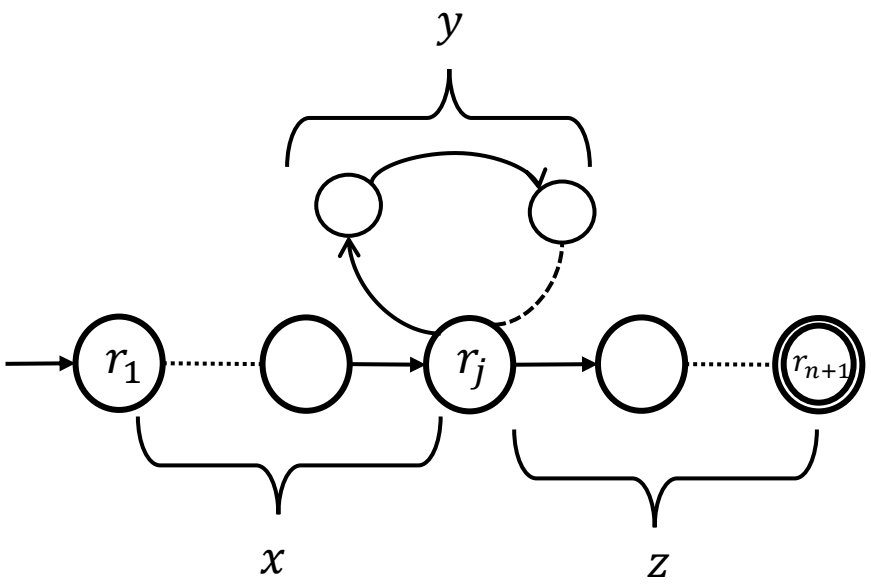
This is because of the ***pigeonhole principle***: any such run would encounter $p + 1$ states, but there are $p$ distinct states in the DFA.

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, $x = s_1 \ldots s_{j-1}$, $y = s_j \ldots . s_{l-1}$, $z = s_l \ldots s_n$. 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.

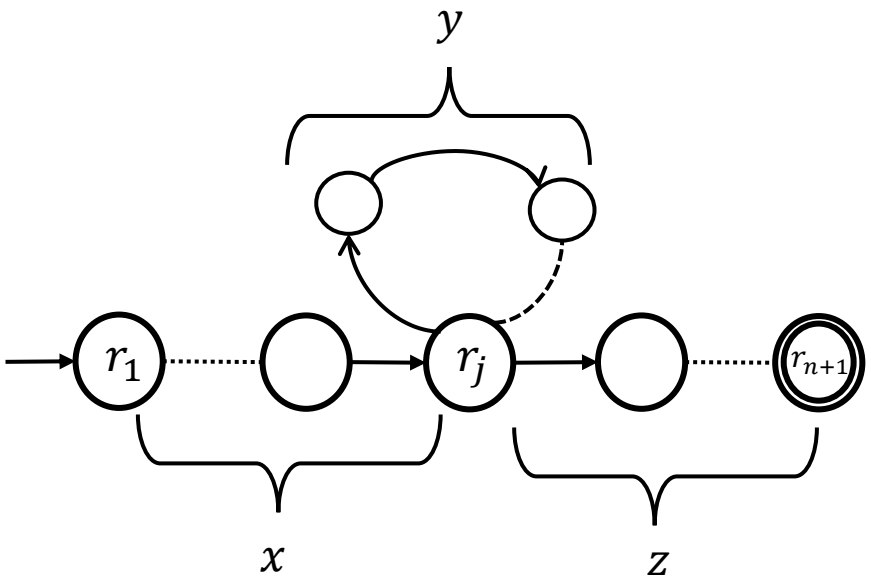
This is because of the ***pigeonhole principle***: any such run would encounter $p + 1$ states, but there are $p$ distinct states in the DFA.

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, $x = s_1 \ldots s_{j-1}$, $y = s_j \ldots . s_{l-1}$, $z = s_l \ldots s_n$. 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.

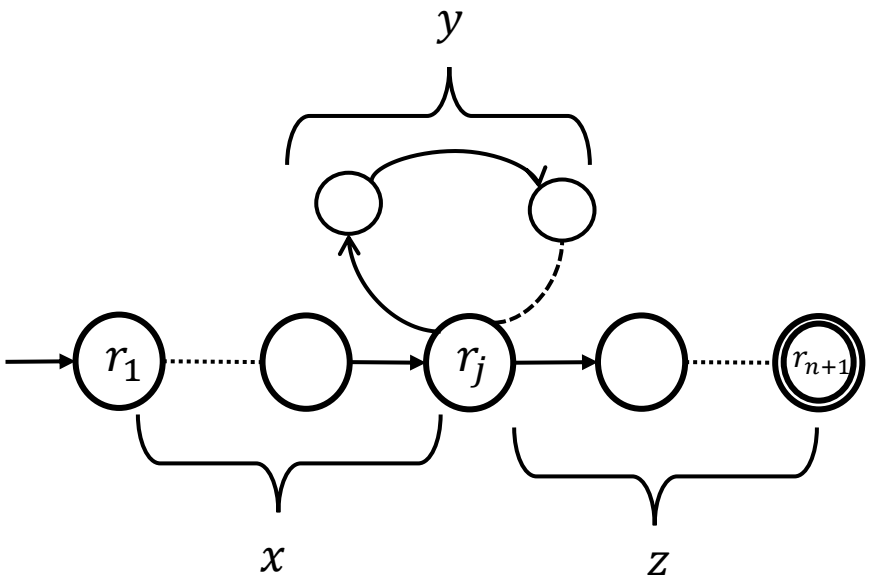
This is because of the ***pigeonhole principle***: any such run would encounter $p + 1$ states, but there are $p$ distinct states in the DFA.

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, $x = s_1 \ldots s_{j-1}$, $y = s_j \ldots . s_{l-1}$, $z = s_l \ldots s_n$. 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.

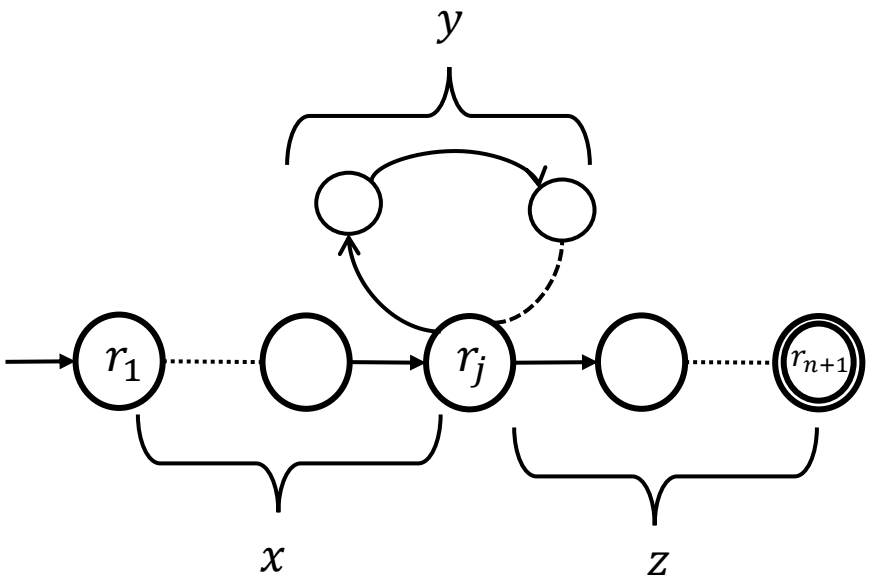
This is because of the ***pigeonhole principle***: any such run would encounter $p + 1$ states, but there are $p$ distinct states in the DFA.

Suppose $s = s_1 s_2 \cdots s_n$ 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, $x = s_1 \ldots s_{j-1}$, $y = s_j \ldots . s_{l-1}$, $z = s_l \ldots s_n$. 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 $j \neq l$, $|y| \geq 1$, 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:

- $\{0^n 1^n | n \geq 0\}$
- $\{\omega | \omega$ has equal number of 0's and $1's\}$
- $\{\omega | \omega$ is palindrome$\}$

⋮

⋮

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

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

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 ≡ NFA ≡ 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

$$Sentence \rightarrow Subject\ Verb\ Object$$
$$Subject \rightarrow Noun.phrase$$
$$Object \rightarrow Noun.phrase$$
$$Noun.phrase \rightarrow Article\ Noun | Noun$$
$$Article \rightarrow \boldsymbol{the}$$
$$Noun \rightarrow \boldsymbol{boy} | \boldsymbol{girl} | \boldsymbol{soccer} | \boldsymbol{poetry}$$
$$Verb \rightarrow \boldsymbol{loves} | \boldsymbol{plays}$$

# 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

$$Sentence \rightarrow Subject\ Verb\ Object$$
$$Subject \rightarrow Noun.phrase$$
$$Object \rightarrow Noun.phrase$$
$$Noun.phrase \rightarrow Article\ Noun | Noun$$
$$Article \rightarrow \boldsymbol{the}$$
$$Noun \rightarrow \boldsymbol{boy | girl | soccer | poetry}$$
$$Verb \rightarrow \boldsymbol{loves | plays}$$

**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

$$Sentence \rightarrow Subject\ Verb\ Object$$
$$Subject \rightarrow Noun.phrase$$
$$Object \rightarrow Noun.phrase$$
$$Noun.phrase \rightarrow Article\ Noun | Noun$$
$$Article \rightarrow \boldsymbol{the}$$
$$Noun \rightarrow \boldsymbol{boy} | \boldsymbol{girl} | \boldsymbol{soccer} | \boldsymbol{poetry}$$
$$Verb \rightarrow \boldsymbol{loves} | \boldsymbol{plays}$$

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

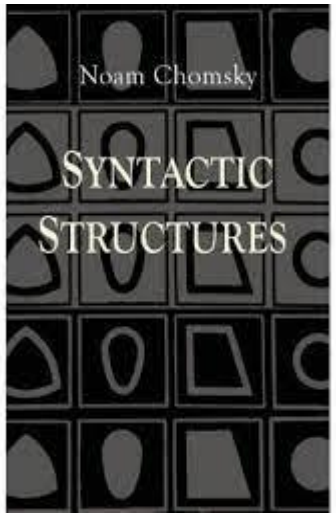
- **Grammars** provide a way to generate strings belonging to a language. The set of all strings generated by the grammar is the *language* of the grammar.
- ***Grammars generate languages:*** Grammars consist of a set of ***rules*** that allow you to construct strings of the language.
- For some classes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

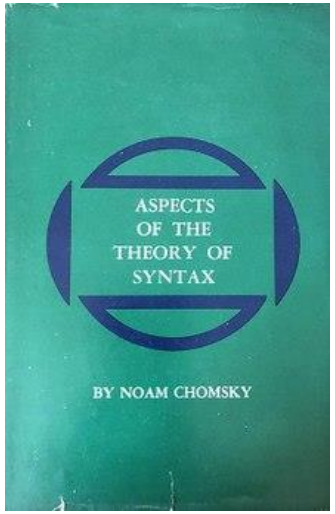
$$
\begin{aligned}
&Sentence \rightarrow Subject\ Verb\ Object\\
&Subject \rightarrow Noun.phrase\\
&Object \rightarrow Noun.phrase\\
&Noun.phrase \rightarrow Article\ Noun | Noun\\
&Article \rightarrow \boldsymbol{the}\\
&Noun \rightarrow \boldsymbol{boy}|\boldsymbol{girl}|\boldsymbol{soccer}|\boldsymbol{poetry}\\
&Verb \rightarrow \boldsymbol{loves}|\boldsymbol{plays}
\end{aligned}
$$

$$
\begin{aligned}
Sentence &\rightarrow Subject\ Verb\ Object\\
&\rightarrow Noun.phrase\ Verb\ Object\\
&\rightarrow Article\ Noun\ Verb\ Object\\
&\rightarrow \boldsymbol{the}\ Noun\ Verb\ Object\\
&\rightarrow \boldsymbol{the\ girl}\ Verb\ Object\\
&\rightarrow \boldsymbol{the\ girl\ plays}\ Object\\
&\rightarrow \boldsymbol{the\ girl\ plays}\ Noun.phrase\\
&\rightarrow \boldsymbol{the\ girl\ plays}\ Noun\\
&\rightarrow \boldsymbol{the\ girl\ plays\ soccer}
\end{aligned}
$$

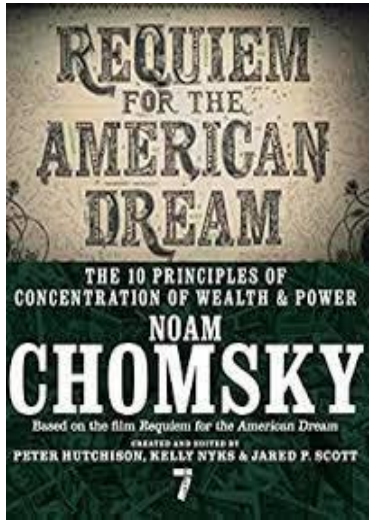
**Variables**: $\{Sentence, Subject, Verb, Object, Noun, Noun.phrase, Article\}$, **Terminals:** $\{The, girl, loves, plays, soccer, poetry\}$
**Start Variable:** $Sentence$

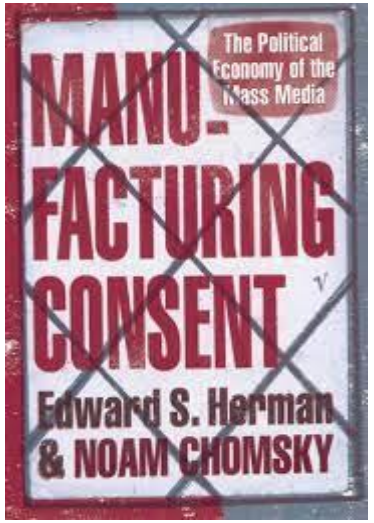
# Grammars




**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 4-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 ]

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: {0, 1}

# 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** (disjoint from $V$)
- $P$ is the set of production **Rules** $\quad [\ (V \cup \Sigma)^* V (V \cup \Sigma)^* \rightarrow (V \cup \Sigma)^*]$
- $S$ 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
$\{0,1\}$: 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 $\boldsymbol{w} \in \boldsymbol{L(G)}$, we show that there exists a **derivation ending up in $\boldsymbol{w}$**. The fact that $w$ can be derived using the rules of $G$, is expressed as $\boldsymbol{S \overset{*}{\Rightarrow} w}$.

- The **language of the grammar, $\boldsymbol{L(G)}$** is $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$

# 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 $\boldsymbol{w} \in \boldsymbol{L(G)}$, we show that there exists a **derivation ending up in $\boldsymbol{w}$**. The fact that $w$ can be derived using the rules of $G$, is expressed as $\boldsymbol{S \overset{*}{\Rightarrow} w}$.

- The **language of the grammar, $\boldsymbol{L(G)}$** is $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$

Eg: Consider the grammar $G$

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

**The string $\boldsymbol{1101} \in \boldsymbol{L(G)}$ 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

$$Var \rightarrow Ter\ \boldsymbol{Var}$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

then the language of the grammar is **regular.** Also known as **Right-linear grammar** (a single variable to the right of terminals in the RHS).

**Right linear Grammar to DFA**

Eg: Consider the grammar $G$

$X \rightarrow 1X$
$X \rightarrow 0Y$
$Y \rightarrow 1Y$
$Y \rightarrow 0X$
$Y \rightarrow \epsilon$ (indicates that $Y$ is the final state)

# Grammars for Regular Languages

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

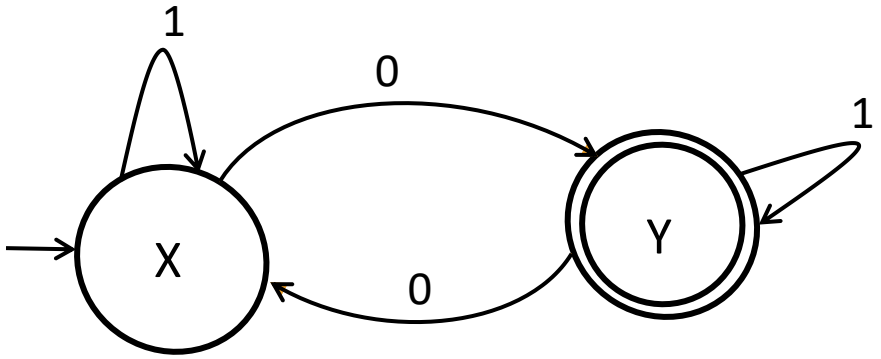
$$Var \rightarrow Ter\ \boldsymbol{Var}$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

then the language of the grammar is **regular.** Also known as **Right-linear grammar** (a single variable to the right of terminals in the RHS).

**Right linear Grammar to DFA**

Eg: Consider the grammar $G$

$X \rightarrow 1X$
$X \rightarrow 0Y$
$Y \rightarrow 1Y$
$Y \rightarrow 0X$
$Y \rightarrow \epsilon$ (indicates that $Y$ is the final state)



# Grammars for Regular Languages

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

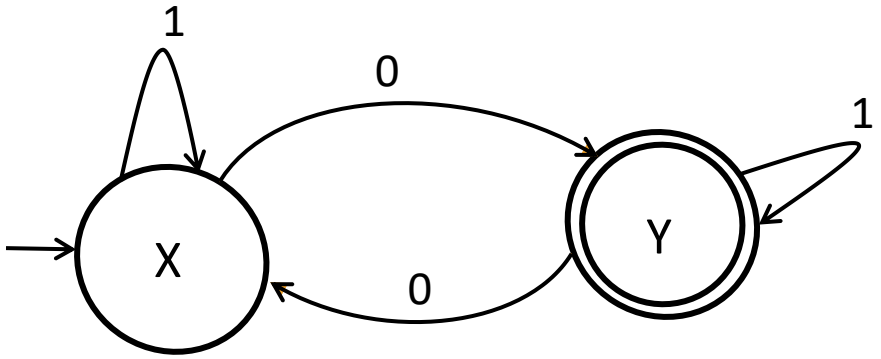
$$Var \to Ter\ \boldsymbol{Var}$$
$$Var \to Ter$$
$$Var \to \boldsymbol{\epsilon}$$

then the language of the grammar is **regular.** Also known as **Right-linear grammar** (a single variable to the right of terminals in the RHS).

**Right linear Grammar to DFA**

Eg: Consider the grammar $G$

$X \to 1X$
$X \to 0Y$
$Y \to 1Y$
$Y \to 0X$
$Y \to \epsilon$ (indicates that $Y$ is the final state)




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

For the string $\mathbf{1101}$:

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

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

# Grammars for Regular Languages

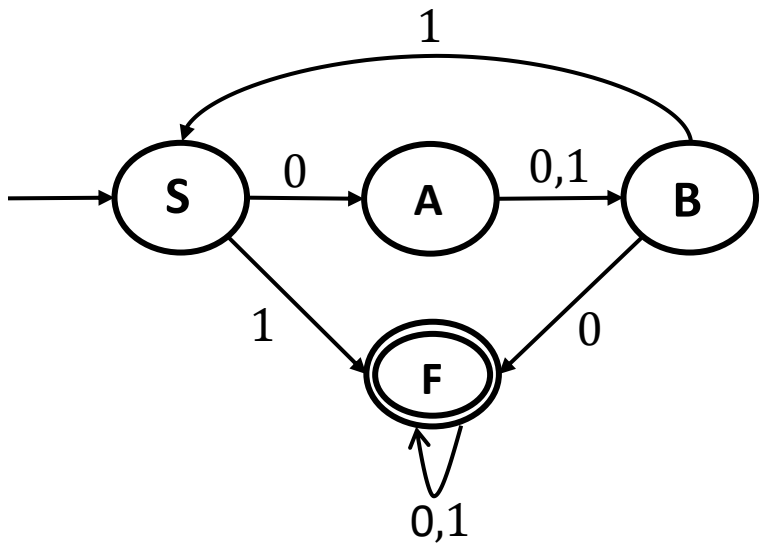
**Regular grammar:** If the *rules* of the underlying grammar $G$ are of the form

$$Var \rightarrow Ter\ \boldsymbol{Var}$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \epsilon$$

then the language of the grammar is **regular.** Also known as **Right-linear grammar** (a single variable to the right of terminals in the RHS).

**DFA to Right linear Grammar**

Consider the following DFA $M$




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

$S \rightarrow 0A|1F$
$A \rightarrow 0B|1B$
$B \rightarrow 0F|1S$
$F \rightarrow 0F|1F|\epsilon$

# Grammars for Regular Languages

**Right-linear grammar ≡ DFA ≡ NFA ≡ Regular Expressions**

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

$$Var \rightarrow \boldsymbol{Var}\, Ter$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

then such a grammar is called **Left-linear** (a single variable 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 ≡ DFA ≡ NFA ≡ Regular Expressions**

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

$$Var \rightarrow \boldsymbol{Var}\, Ter$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

then such a grammar is called **Left-linear** (a single variable 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 ≡ Right-linear grammar ≡ DFA ≡ NFA ≡ 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** [ $(\boldsymbol{V} \cup \boldsymbol{T})^* \boldsymbol{V} (\boldsymbol{V} \cup \boldsymbol{T})^* \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$ ]
- $S$ is the **Start Variable** [ 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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$Var \rightarrow Anything$$

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

# Context free Grammars

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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 ⊂ Context Free Languages**.

# Context free Grammars

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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 ⊂ Context Free Languages**.

Consider the Grammar $G$ with the following rules:

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

Strings that can be derived from $G$:

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

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

$$\boldsymbol{V} \rightarrow (\boldsymbol{V} \cup \boldsymbol{T})^*$$

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

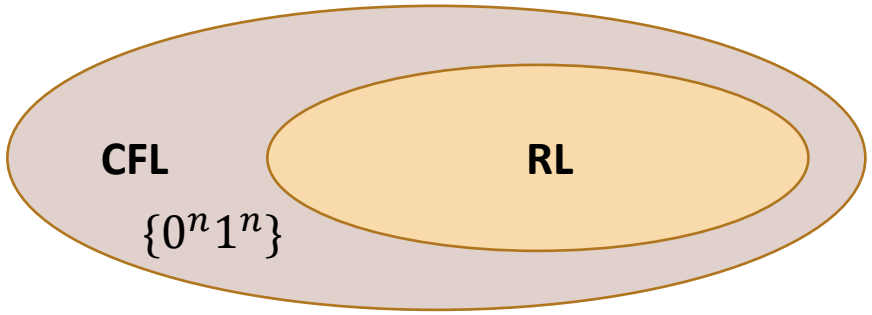
Any language generated by a context-free grammars is called a ***context-free language.***

- So Left linear grammars and Right linear grammars are also context-free grammars.
- **Regular languages ⊂ 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.**

# Thank You!