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

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

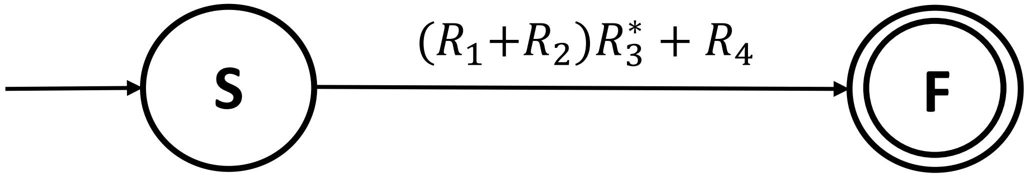
HYDERABAD

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.



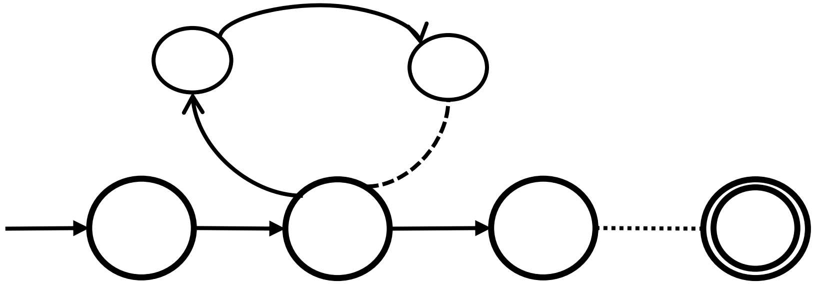
DFA, NFA, Regular Expressions have equal power and all of them correspond to Regular Languages

Pumping Lemma

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

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

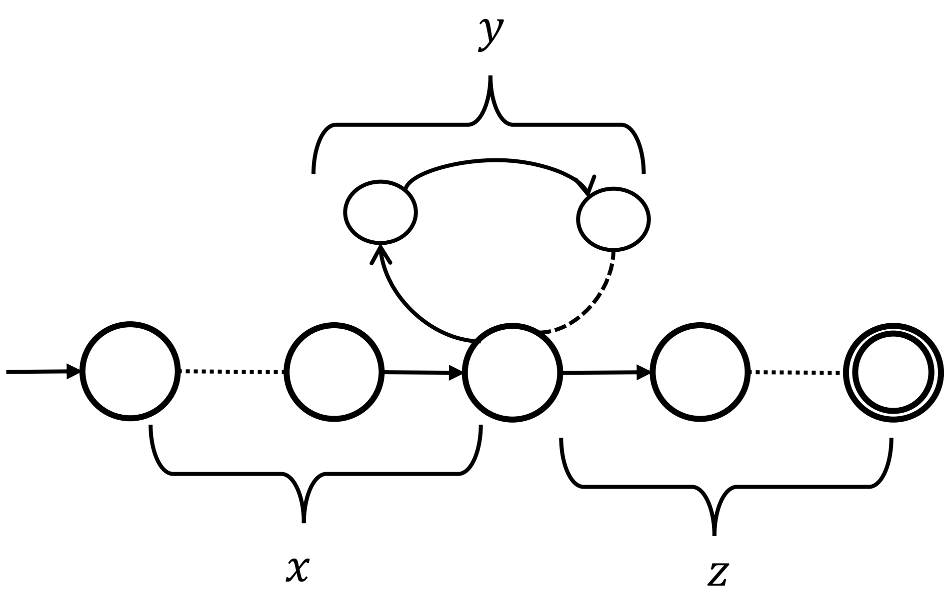
Mil: I have a DFA for L.
Karl: How many states are there?
Mil: n-states (say $n = 10$)
Karl: Then $0^{10}1^{10}$ must be accepted. By the **pigeonhole principle**, while reading the first $(n = 10)$ symbols, some states need to be revisited. Otherwise $n + 1 = 11$ states would have been present. Hence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

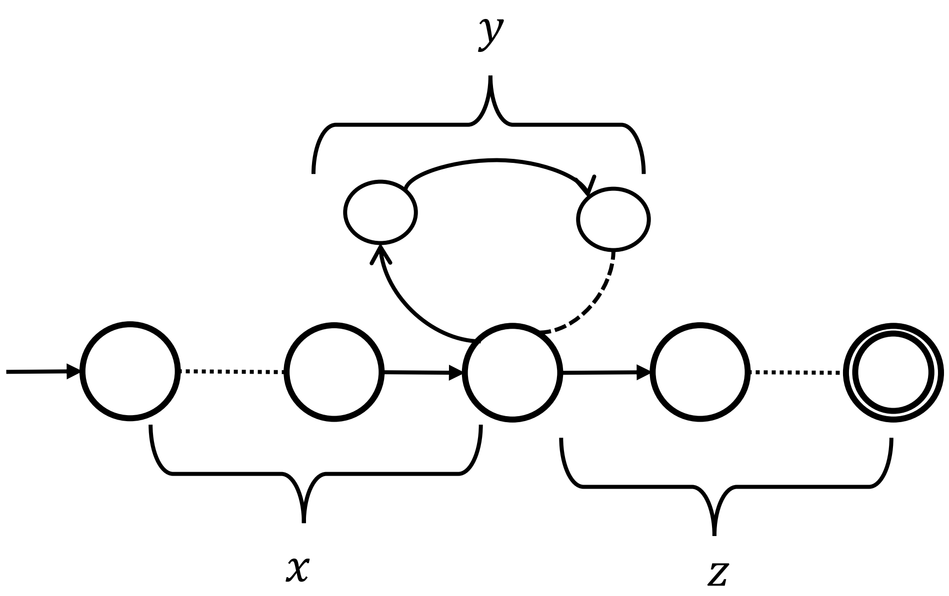


Pumping Lemma

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

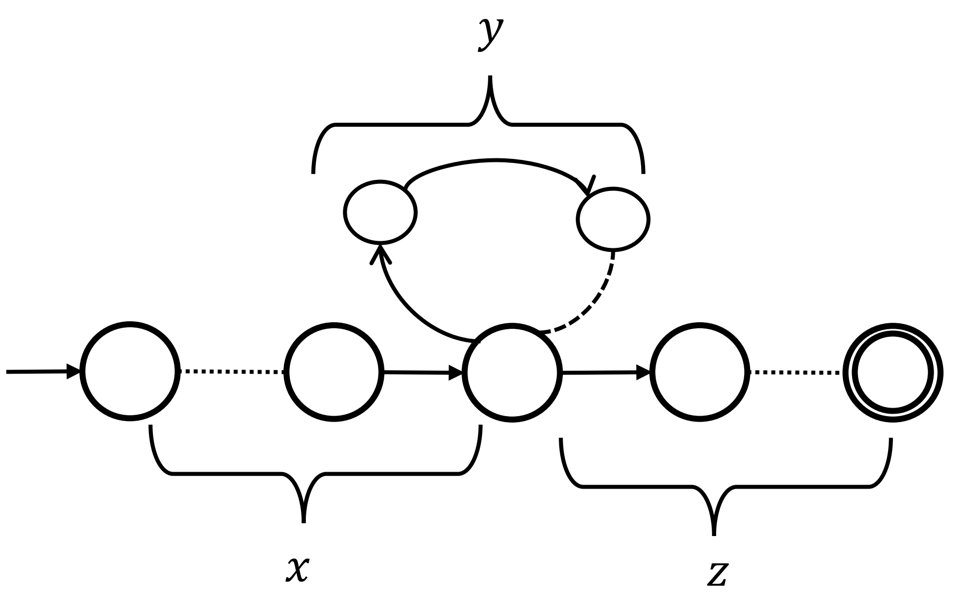
1. $|xy| \leq p$.
2. $|y| \geq 1$
3. $\forall i \geq 0, xy^i z \in L$.

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

If L is regular then, pumping property is satisfied

$\equiv$

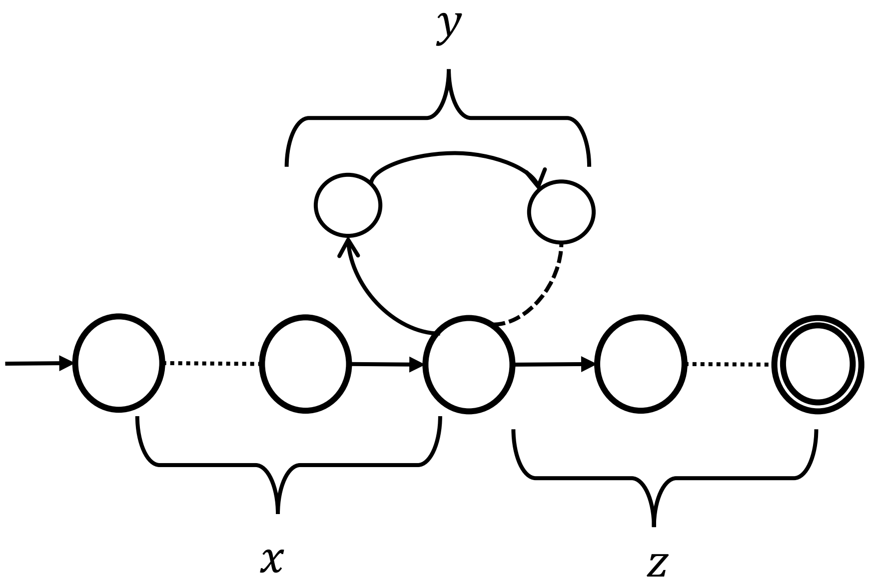
If pumping property is NOT satisfied, then L is NOT regular.



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.



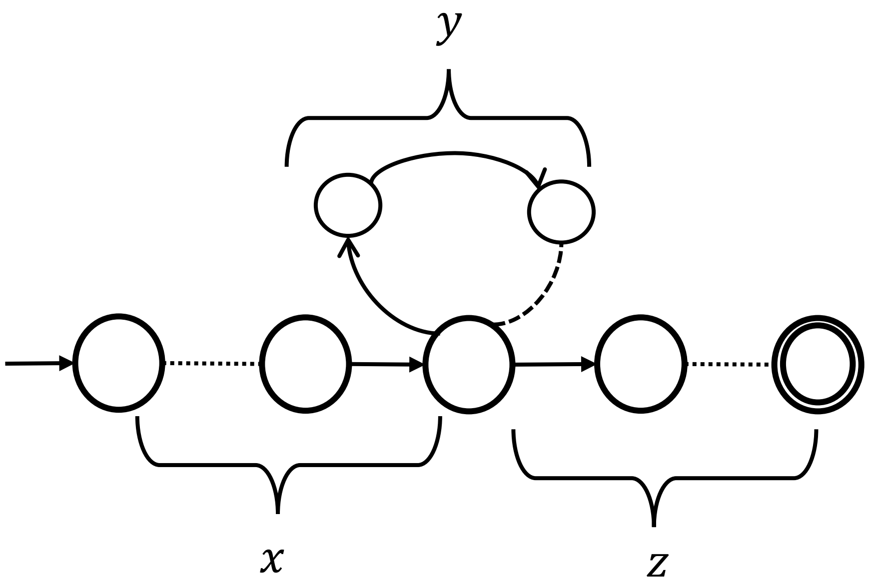
Pumping Lemma

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.

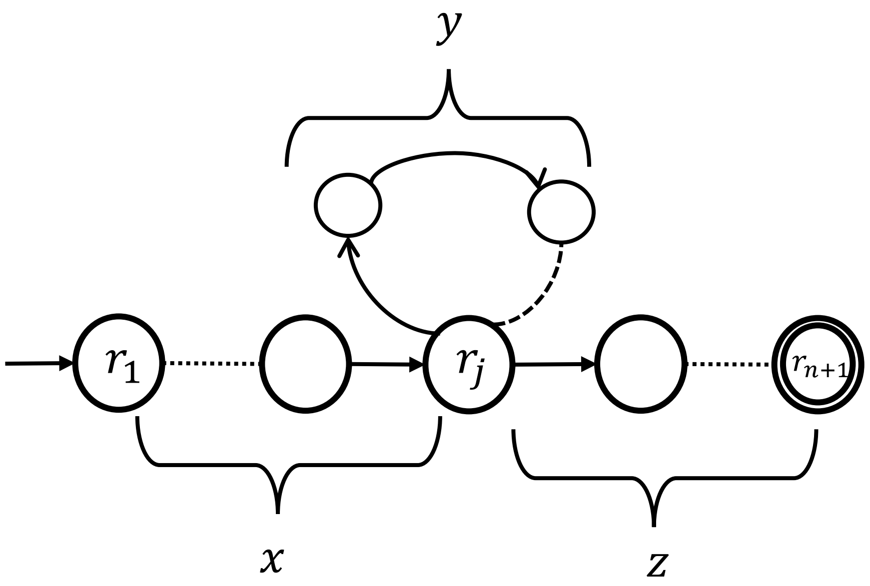
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are p distinct states in the DFA.

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.

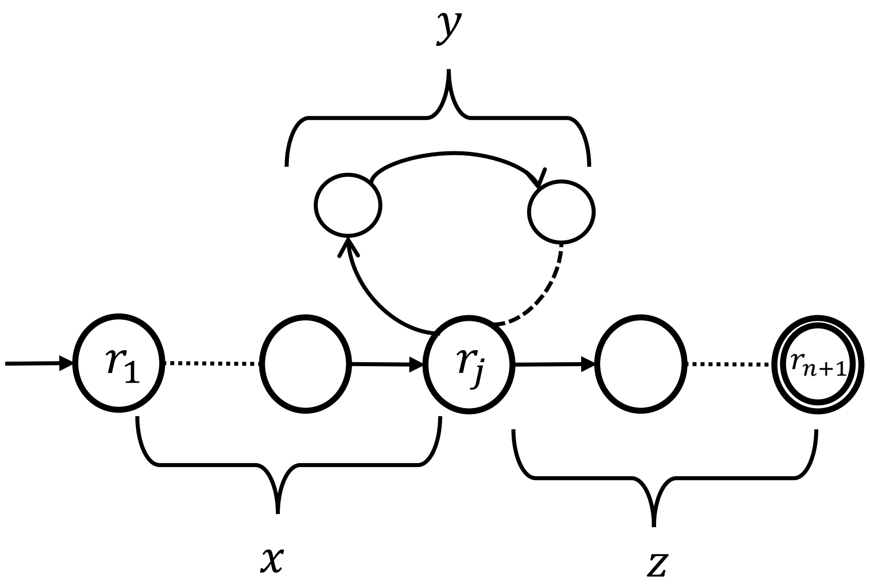
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are p distinct states in the DFA.

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.

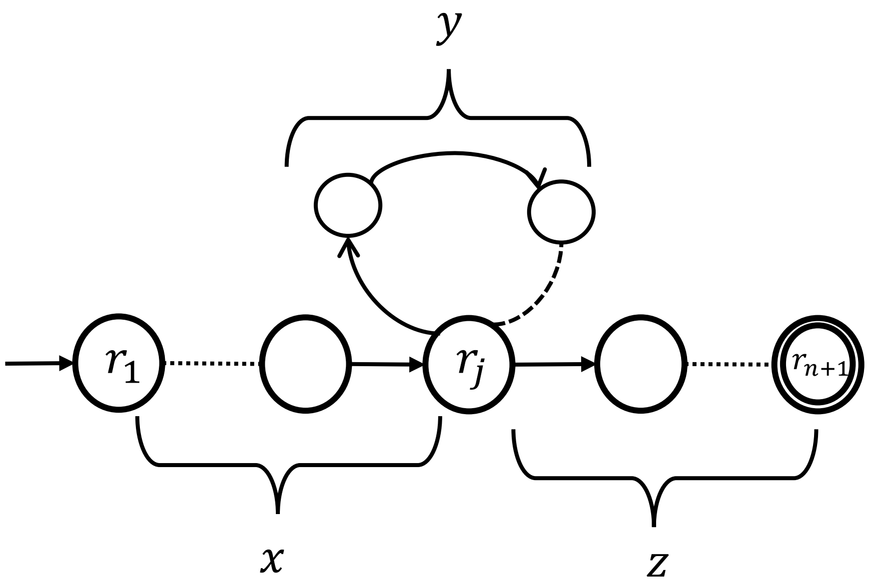
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are p distinct states in the DFA.

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

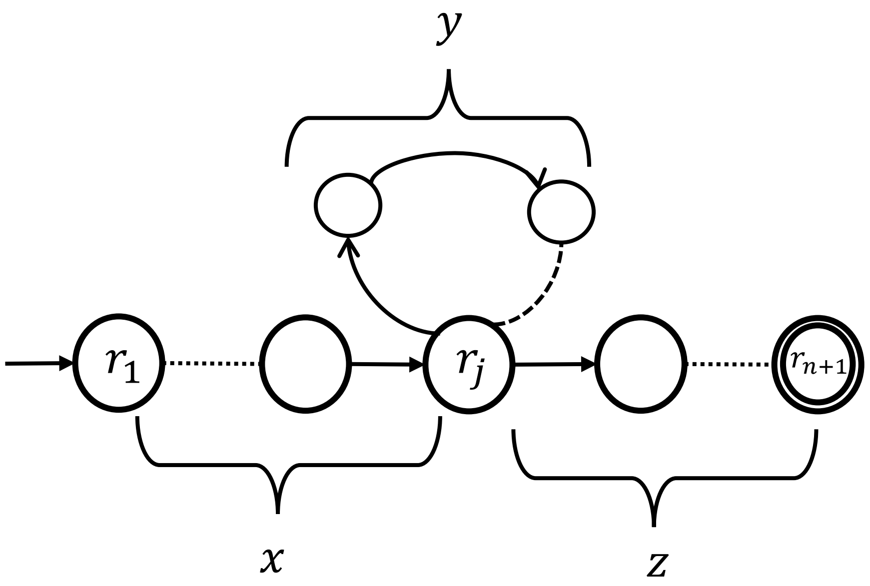
Proof sketch: Suppose that we have a DFA M of p states. Then any run in the DFA corresponding to strings of length at least p, some states are repeated.

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:

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

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

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

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

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

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

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.

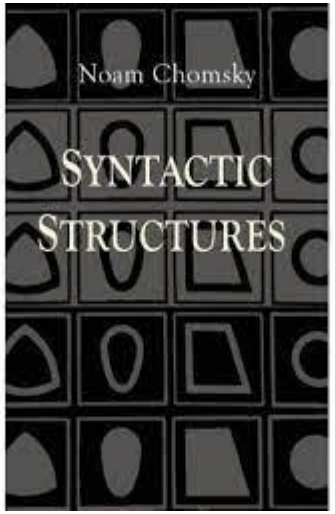
- Consider these rules

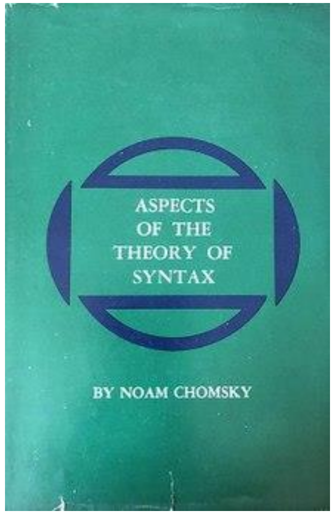
$$
\begin{aligned}
&Sentence \to Subject\ Verb\ Object\\
&Subject \to Noun.phrase\\
&Object \to Noun.phrase\\
&Noun.phrase \to Article\ Noun|Noun\\
&Article \to \boldsymbol{the}\\
&Noun \to \boldsymbol{boy|girl|soccer|poetry}\\
&Verb \to \boldsymbol{loves|plays}
\end{aligned}
$$

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

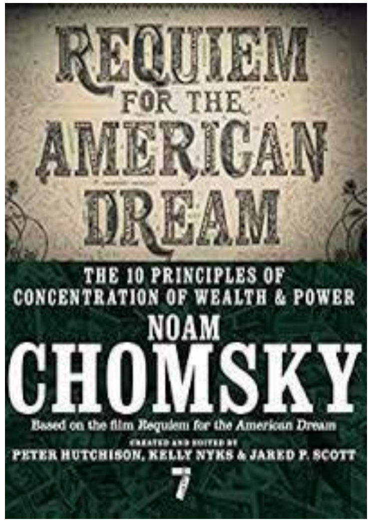
Variables: $\{Sentence, Subject, Verb, Object, Noun, Noun.phrase, Article\}$, **Terminals:** $\{The, girl, loves, plays, soccer, poetry\}$
Start Variable: $Sentence$

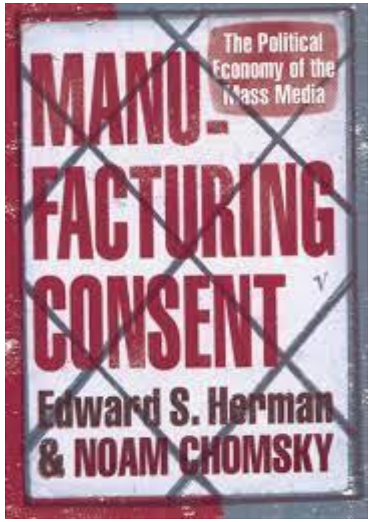
Grammars





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 5-tuple (V, Σ, P, S) such that

- V is the set of **Variables**
- Σ 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: $\{\epsilon, 0, 1\}$

Grammars

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

- V is the set of **Variables**
- Σ 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, Σ, P, S) such that

- V is the set of **Variables**
- Σ 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 $\boldsymbol{w} \in \boldsymbol{L}(\boldsymbol{G})$, 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 $\boldsymbol{S} \overset{*}{\Rightarrow} \boldsymbol{w}$.

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

Grammars

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

- V is the set of **Variables**
- Σ 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 $\boldsymbol{w} \in \boldsymbol{L}(\boldsymbol{G})$, 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 $\boldsymbol{S} \overset{*}{\Rightarrow} \boldsymbol{w}$.

- The **language of the grammar, $\boldsymbol{L}(\boldsymbol{G})$** 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 $\mathbf{1101} \in \boldsymbol{L}(\boldsymbol{G})$ 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** (all variables are to the right of terminals in the RHS).

Right linear Grammar to DFA

Eg: Consider the grammar G

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

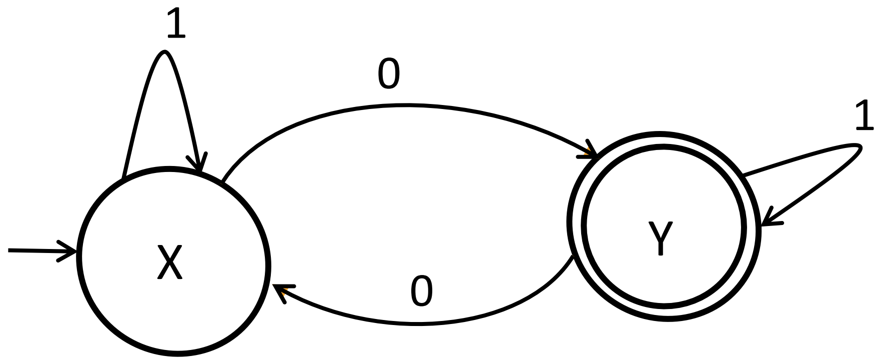
$$Var \rightarrow Ter\ \boldsymbol{Var}$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

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

Right linear Grammar to DFA

Eg: Consider the grammar G

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

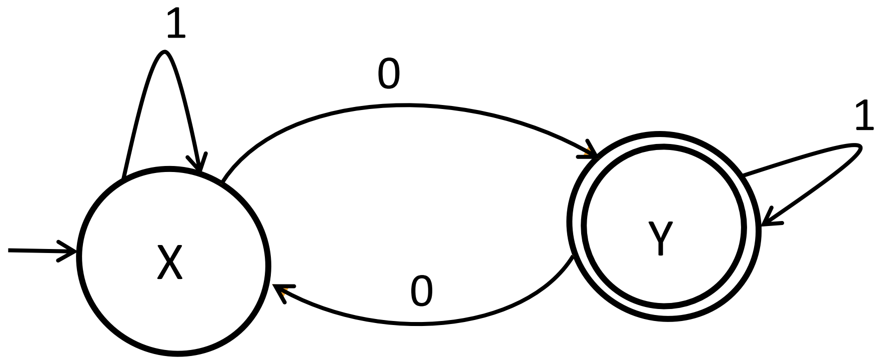
$$Var \rightarrow Ter\ \boldsymbol{Var}$$
$$Var \rightarrow Ter$$
$$Var \rightarrow \boldsymbol{\epsilon}$$

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

Right linear Grammar to DFA

Eg: Consider the grammar G

$X \rightarrow 1X$
$X \rightarrow 0Y$
$Y \rightarrow 1Y$
$Y \rightarrow 0X$
$Y \rightarrow \epsilon$ (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: $X \rightarrow 1X \rightarrow 11X \rightarrow 110Y \rightarrow 1101Y \rightarrow 1101$. 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

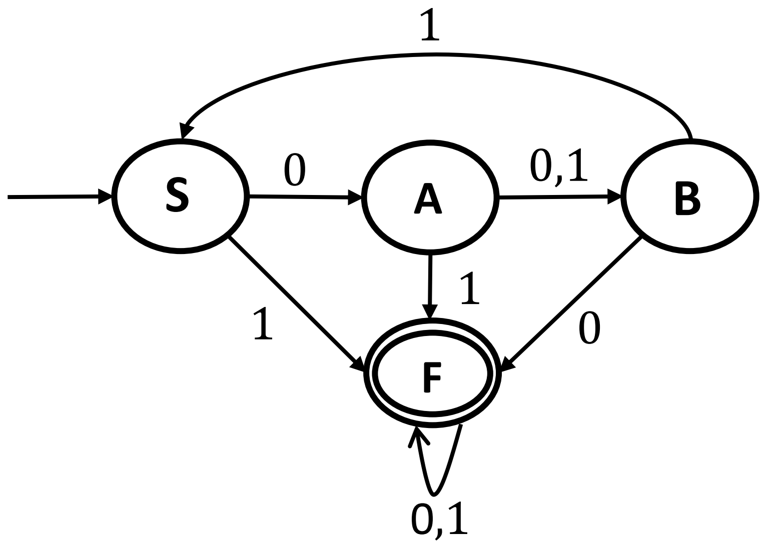
Regular grammar: If the *rules* of the underlying grammar G are of the form

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

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

$S \to 0A$
$A \to 01B$
$B \to 1S$
$F \to 01F$
$A \to 1F$
$B \to 0F$
$S \to 1F$
$F \to \epsilon$

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

Context free Grammars

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

- V is the set of **Variables**
- Σ 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** $\subset$ **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

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

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:

$$S \to 0S1|\epsilon$$

What is the language generated by this grammar?

Strings that can be derived from G:

$$S \to \epsilon$$

$$\{\epsilon\}$$

Context free Grammars

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

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

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:

$$S \to 0S1|\epsilon$$

What is the language generated by this grammar?

Strings that can be derived from G:

$$S \to 0S1 \to 01$$

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

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

Consider the Grammar G with the following rules:

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

Strings that can be derived from G:

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

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.**

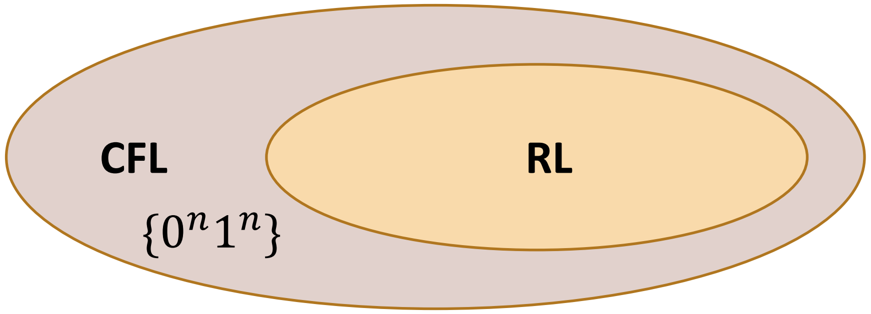
Any language generated by a context-free grammars is called a ***context-free language.***

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

Consider the Grammar G with the following rules:

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

What is the language generated by this grammar?



$$L(G) = \{\omega | \omega = 0^n1^n, n \geq 0\}$$

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.**

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

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

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

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

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$S \to 0\boldsymbol{S}1 \to 0\boldsymbol{0S1}1 \ldots$$

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

Context free Grammars

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

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

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

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

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$S \to \boldsymbol{0S1} \to 0\boldsymbol{SS}1 \to 0\boldsymbol{0S1}S1 \to 001S1 \to 001\boldsymbol{0S1}1 \to 001011$$

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

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.**

Regular languages ⊂ **Context Free Languages**.

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$S \rightarrow \boldsymbol{0S1} \rightarrow 0\boldsymbol{SS}1 \rightarrow 0\boldsymbol{0S1}S1 \rightarrow 001S1 \rightarrow 001\boldsymbol{0S1}1 \rightarrow 001011$$

$$\{\epsilon, 01, 0011, \ldots 0^n1^n, 001011, \ldots .\}$$

Show that the string $\mathbf{010101} \in \boldsymbol{L}(\boldsymbol{G})$.

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.**

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:

$$S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{SS}S \rightarrow \boldsymbol{0S1}SS \rightarrow 0S1\boldsymbol{0S1}S \rightarrow 0S10S1\boldsymbol{0S1} \rightarrow 010101$$

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

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.**

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:

$$S \rightarrow \boldsymbol{SS} \rightarrow \boldsymbol{SS}S \rightarrow \boldsymbol{0S1}SS \rightarrow 0S1\boldsymbol{0S1}S \rightarrow 0S10S1\boldsymbol{0S1} \rightarrow 010101$$

$$\{\epsilon, 01, 0011, \ldots 0^n1^n, 001011, 010101, \ldots.\}$$

What is $\boldsymbol{L(G)}$?

Context free Grammars

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

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

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:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Context free Grammars

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \ldots 0^n1^n, 001011, 010101, \ldots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \ldots, 0^n1^n, 001011, 010101, \ldots.\}$

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

Context free Grammars

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots.\}$

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

Context free Grammars

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots.\}$

$$\left\{\epsilon, (\quad), ((\quad)), \dots, \left(\left(((\dots.))\right)\right)\right\}$$

Context free Grammars

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots.\}$

$$\left\{\epsilon, (\quad), ((\quad)), \dots, \left(\left(\left((\dots.)\right)\right)\right), ((\quad)(\quad)), \dots.\right\}$$

Context free Grammars

Consider the Grammar G with the following rules:

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

Strings that can be derived by G:

$$\{\epsilon, 01, 0011, \dots 0^n1^n, 001011, 010101, \dots.\}$$

What is $L(G)$?

You can see what the language is, if you replace $\mathbf{0}$ with (and $\mathbf{1}$ with)

Strings that can be derived by G: $\{\epsilon, 01, 0011, \dots, 0^n1^n, 001011, 010101, \dots.\}$

$$\left\{\epsilon, (\ \ ), ((\ \ )), \dots, \left(\left(\left((\dots.)\right)\right)\right), ((\ \ )(\ \ )), ()()(), \dots.\right\}$$

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 the rules of G:

$$S \rightarrow S_1 | S_2$$
$$S_1 \rightarrow \cdots \cdots$$
$$S_2 \rightarrow \cdots \cdots$$

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 the rules of G:

$$S \rightarrow S_1 | S_2$$
$$S_1 \rightarrow \cdots \cdots$$
$$S_2 \rightarrow \cdots \cdots$$

- 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 $\boldsymbol{L(G) = \{\omega | \omega}$ **has equal number of** $\mathbf{0}$**'s and** $\mathbf{1}$**'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 $S \to aSb$ help generate where the portions of *a* and *b* are equal.

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

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

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$

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 $\boldsymbol{L(G)} = \{\boldsymbol{\omega}|\boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

- Grammar for $L_1 \cup L_2$:

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$

- **Is that all? Is $\boldsymbol{L_1 \cup L_2 = L(G)}$?** $L_1 \cup L_2$ 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 $\boldsymbol{L(G)} = \{\boldsymbol{\omega} | \boldsymbol{\omega}$ **has equal number of** $\mathbf{0}$**'s and** $\mathbf{1}$**'s**$\}$

- Grammar for $L_1 \cup L_2$:

$$S \rightarrow S_1 | S_2$$
$$S_1 \rightarrow 0S_1 1 | \epsilon$$
$$S_2 \rightarrow 1S_2 0 | \epsilon$$

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

$$0\boldsymbol{S_1}1 \rightarrow 0\boldsymbol{S_2}1 \rightarrow 0\mathbf{1}\boldsymbol{S_2}\mathbf{0}1 \rightarrow \cdots$$

Context free Grammars

Constructing CFG corresponding to a Language.

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

- Grammar for $L_1 \cup L_2$:

$$S \to S_1 | S_2$$
$$S_1 \to 0S_1 1 | \epsilon$$
$$S_2 \to 1S_2 0 | \epsilon$$

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

$$S \to S_1 | S_2$$
$$S_1 \to 0S_1 1 | \epsilon$$
$$S_2 \to 1S_2 0 | \epsilon$$
$$S_1 \to S_2$$
$$S_2 \to S_1.$$

Context free Grammars

Constructing CFG corresponding to a Language.

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

$$S \to S_1|S_2$$
$$S_1 \to 0S_1 1|\epsilon$$
$$S_2 \to 1S_2 0|\epsilon$$
$$S_1 \to S_2$$
$$S_2 \to S_1$$

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

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

Context free Grammars

Constructing CFG corresponding to a Language.

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

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_1 1|\epsilon$$
$$S_2 \rightarrow 1S_2 0|\epsilon$$
$$S_1 \rightarrow S_2$$
$$S_2 \rightarrow S_1$$

- Simplified grammar:

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

Context free Grammars

Constructing CFG corresponding to a Language.

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

$$S \rightarrow S_1|S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$
$$S_1 \rightarrow S_2$$
$$S_2 \rightarrow S_1$$

- Simplified grammar:

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

- Is that all? What about strings such as $\{\mathbf{0110}, \mathbf{00111100}\}$?

- 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 $\boldsymbol{L(G)} = \{\boldsymbol{\omega} | \boldsymbol{\omega}$ **has equal number of 0's and 1's**$\}$

$$S \to S_1 | S_2$$
$$S_1 \to 0S_1 1 | \epsilon$$
$$S_2 \to 1S_2 0 | \epsilon$$
$$S_1 \to S_2$$
$$S_2 \to S_1$$

- Simplified grammar:

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

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

Context free Grammars

Constructing CFG corresponding to a Language.

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

$$S \rightarrow S_1|S_2|S_1S_2$$
$$S_1 \rightarrow 0S_11|\epsilon$$
$$S_2 \rightarrow 1S_20|\epsilon$$
$$S_1 \rightarrow S_2$$
$$S_2 \rightarrow S_1$$

- Simplify this further.

G: $\boldsymbol{S \rightarrow SS|0S1|1S0|\epsilon}$

Thank You!