# 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.

| Regular Expression | Regular Language | Comment |
|---|---|---|
| . $\Phi$ | $\{\}$ | The empty set |
| $\epsilon$ | $\{\epsilon\}$ | The set containing $\epsilon$ only |
| $a$ | $\{a\}$ | Any $a \in \Sigma$ |
| $R_1 + R_2$ | $L(R_1) \cup L(R_2)$ | For regular expressions $R_1$ and $R_2$ |
| $R_1 R_2$ | $L(R_1).L(R_2)$ | For regular expressions $R_1$ and $R_2$ |
| $R^*$ | $\left(L(R)\right)^*$ | For regular expressions $R$ |
| $(R)$ | $L(R)$ | For regular expressions $R$ |

**Some algebraic properties of Regular Expressions:**

- $(R_1^*)^* = R_1^*$
- $R\epsilon = \epsilon R = R$
- $R\Phi = \Phi R = \Phi$
- $R + \Phi = R$
- $\epsilon + RR^* = \epsilon + R^*R = R^*$
- $(R_1 + R_2)^* = (R_1^* R_2^*)^* = (R_1^* + R_2^*)^*$

**Claim: A language $L$ is regular if and only if for some regular expression $R$, $L(R) = L$.**

We saw that it is possible to construct **NFAs given a Regular Expression**.

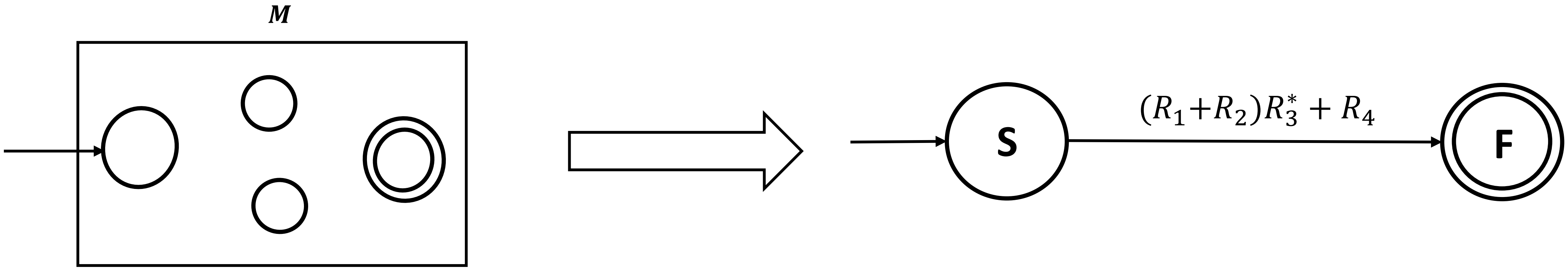
# DFA to Regular Expressions

If a language is regular then it accepts a regular expression. We could draw equivalent NFAs for Regular Expressions.

How can we obtain Regular expressions given a DFA?

Given a DFA $M$, we **recursively** construct a  two-state **Generalized NFA** (GNFA) with

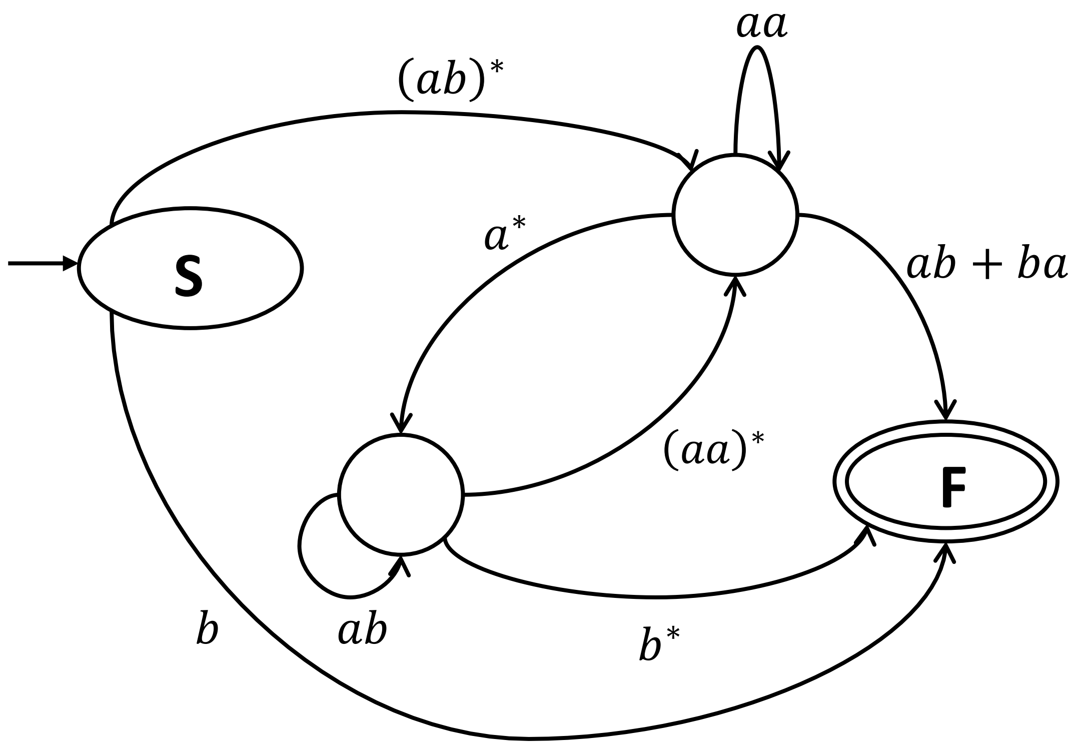
- A start state and a final state
- A single arrow goes from the start state to the final state
- The label of this arrow is the regular expression corresponding to the language accepted by the DFA $M$.



# DFA to Regular Expressions: GNFA

What are GNFAs? They are simply NFAs such that

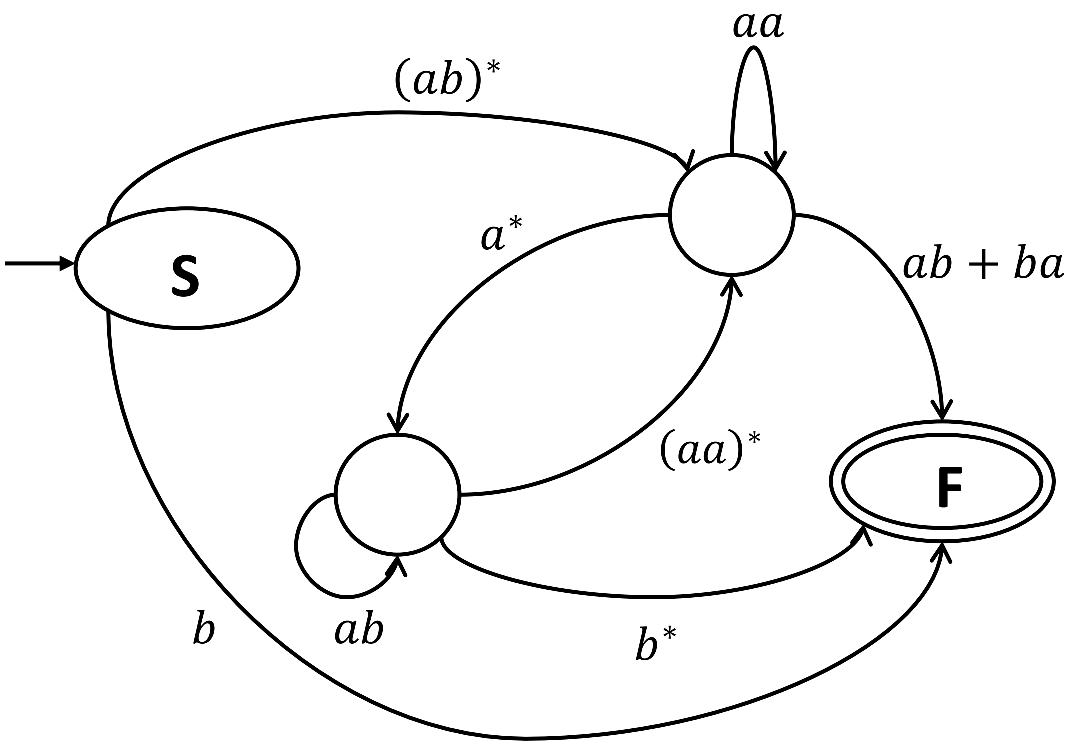
- The transitions may have regular expressions
- A unique start state that has arrows going to other states, but has no incoming arrows
- A unique final state that has arrows incoming from other states, but has no outgoing arrows
- For an input string, **runs** on a GNFA are similar to that of an NFA, except now a block of symbols are read corresponding to the Regular Expressions on the transitions.
- $b, abababab, aaabba$ are some input strings that have accepting runs for the GNFA on the right



# DFA to Regular Expressions: GNFA

What are GNFAs? They are simply NFAs such that

- The transitions may have regular expressions
- A unique start state that has arrows going to other states, but has no incoming arrows
- A unique final state that has arrows incoming from other states, but has no outgoing arrows
- For an input string, **runs** on a GNFA are similar to that of an NFA, except now a block of symbols are read corresponding to the Regular Expressions on the transitions.
- $b, abababab, aaabba$ are some input strings that have accepting runs for the GNFA on the right

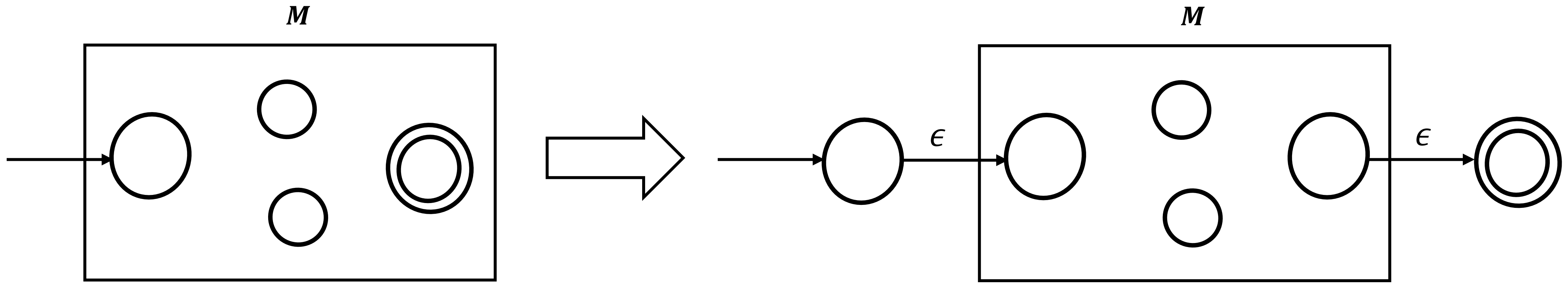



Starting from a DFA we will begin by constructing a GNFA with $k$ states. We then outline a recursive procedure by which at each step, we will construct a GNFA with one less state. This step will be repeated until we obtain the **2-state GNFA**.

# DFA to Regular Expressions: GNFA

Starting from the DFA $M$,

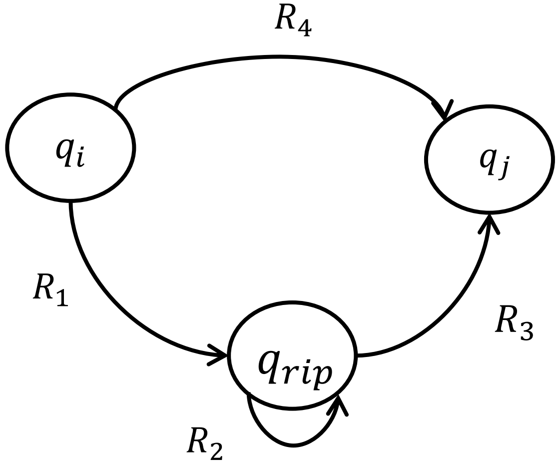
- Add a new start state with an $\epsilon$ arrow to the old start state.
- Add a new final state by with an $\epsilon$ arrow to the old final state.



# DFA to Regular Expressions: GNFA

The crucial step is to convert a GNFA with $k$ (>2) states to a GNFA with $k-1$ states. This is what we shall show next.

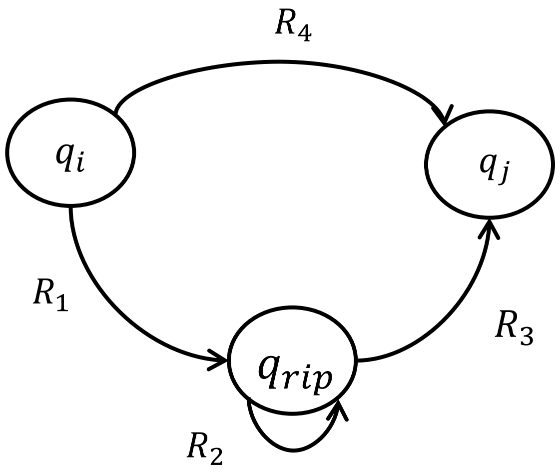
- Start by picking any state of the GNFA (except the new start and final states)
- Let us call this state $q_{rip}$. We "rip" $q_{rip}$ out of the machine and create a GNFA with $k-1$ states.
- Of course, we need to "repair" the machine by altering the regular expressions that label each of the remaining arrows.
- The new labels compensate for the loss of $q_{rip}$.

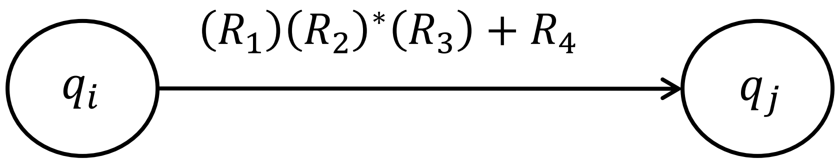


# DFA to Regular Expressions: GNFA

The crucial step is to convert a GNFA with $k$ (>2) states to a GNFA with $k-1$ states. This is what we shall show next.

- Start by picking any state of the GNFA (except the new start and final states)
- Let us call this state $q_{rip}$. We "rip" $q_{rip}$ out of the machine and create a GNFA with $k-1$ states.
- Of course, we need to "repair" the machine by altering the regular expressions that label each of the remaining arrows.
- The new labels compensate for the loss of $q_{rip}$.



# DFA to Regular Expressions: GNFA

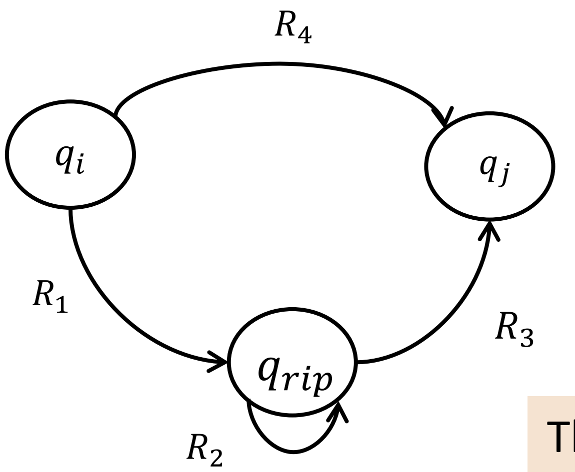
The crucial step is to convert a GNFA with $k$ (>2) states to a GNFA with $k-1$ states.

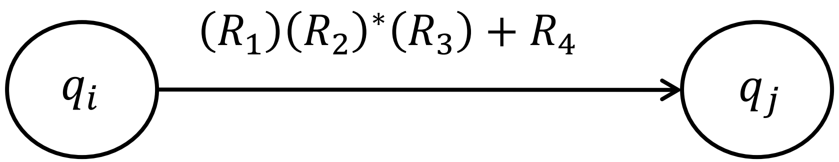
How do we remove $q_{rip}$? In the old machine if

- $q_i$ goes to $q_{rip}$ with an arrow labelled $R_1$
- $q_{rip}$ goes to itself with an arrow labelled $R_2$
- $q_{rip}$ goes to $q_j$ with an arrow labelled $R_3$
- $q_i$ goes to $q_j$ with an arrow labelled $R_4$

**Repeat this until $\boldsymbol{k = 2}$**

then in the new machine, the arrow from $q_i$ to $q_j$ has the label $(R_1)(R_2)^*(R_3) + R_4$

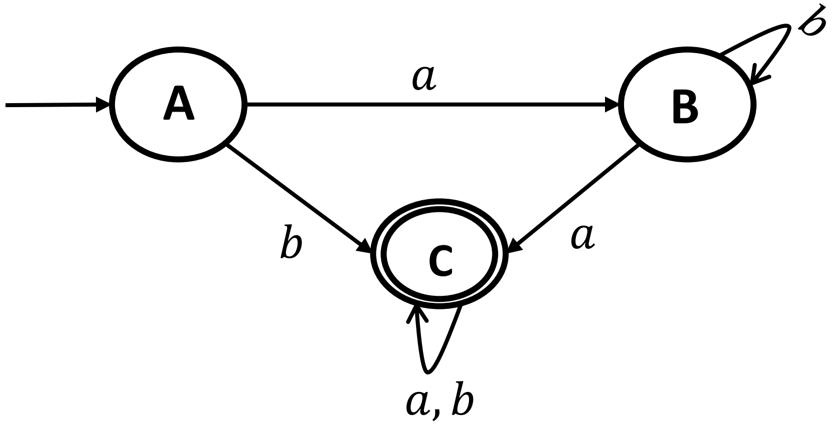


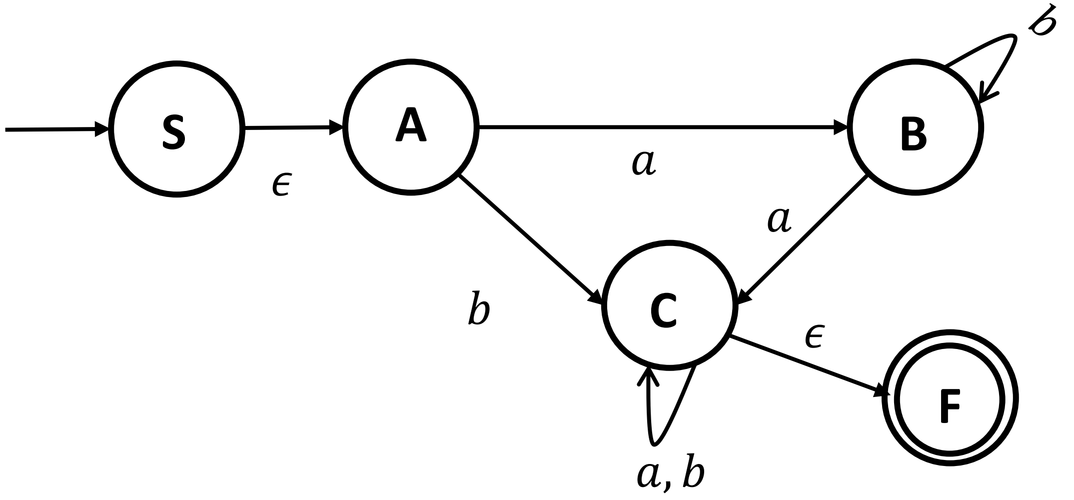

This should be done for **every pair** of arrows outgoing from and incoming to $q_{rip}$

# DFA to Regular Expressions: GNFA

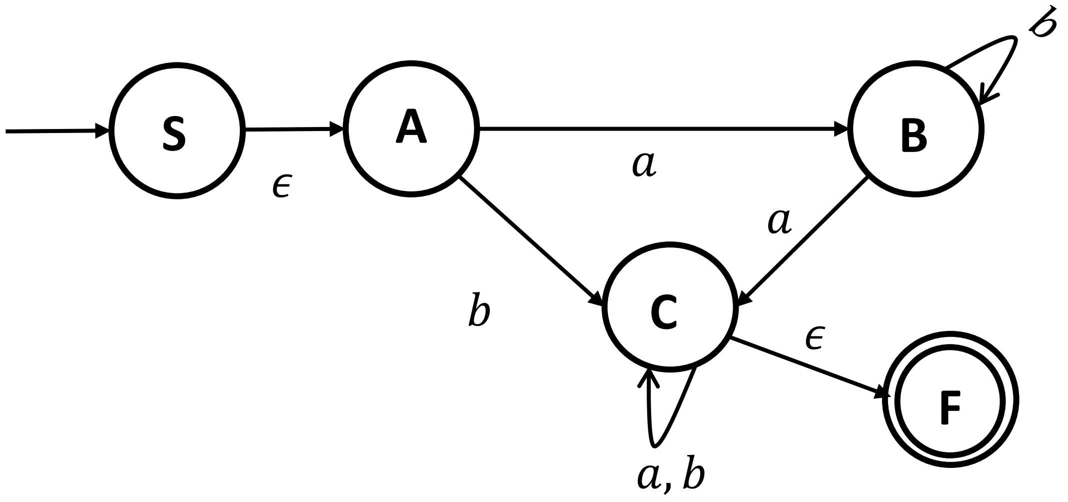
Let us look at an example. Consider the original DFA $M$ below and find the regular expression corresponding to $L(M)$.



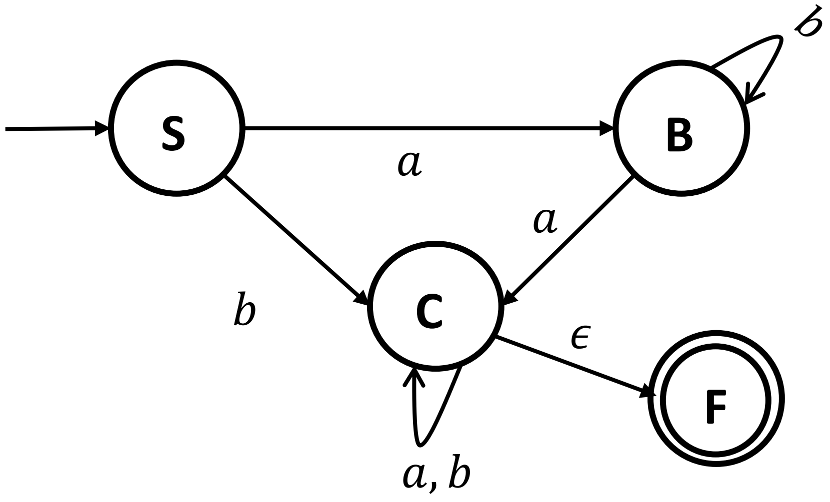

**Step 1: Add new start and final states**



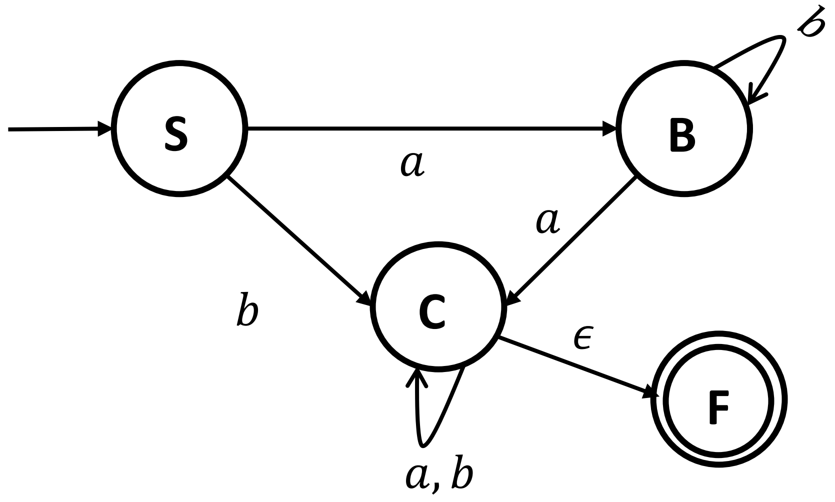
# DFA to Regular Expressions: GNFA




**Step 2: Eliminate $A$**

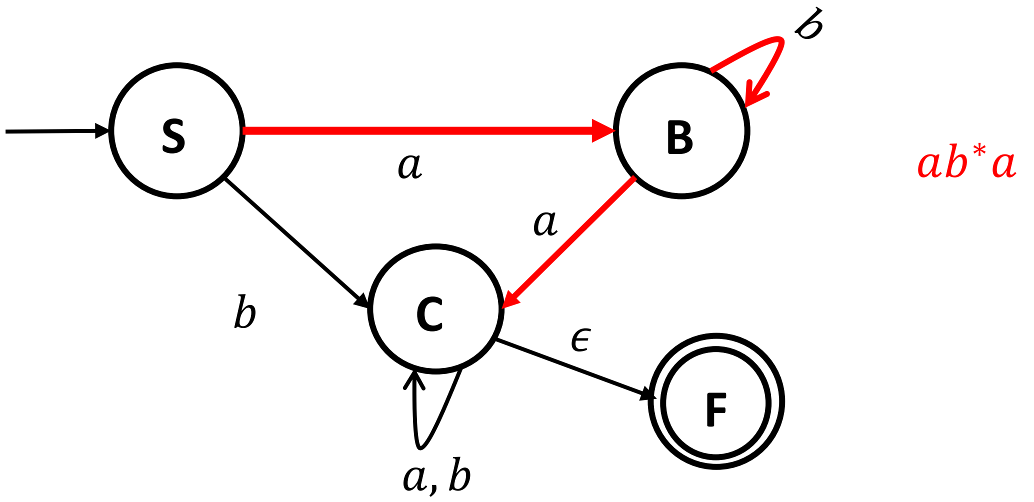


# DFA to Regular Expressions: GNFA

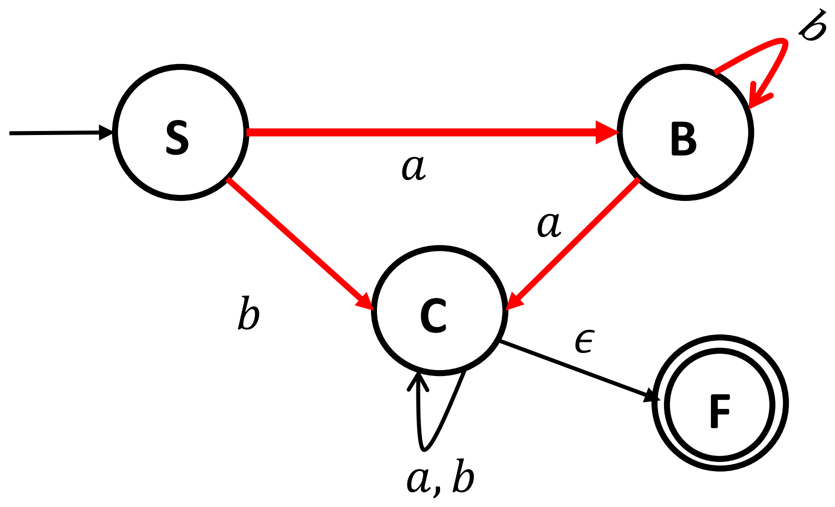



**Step 2: Eliminate $\boldsymbol{B}$**

$S \rightarrow C$ via $B$, RE: $ab^*a$

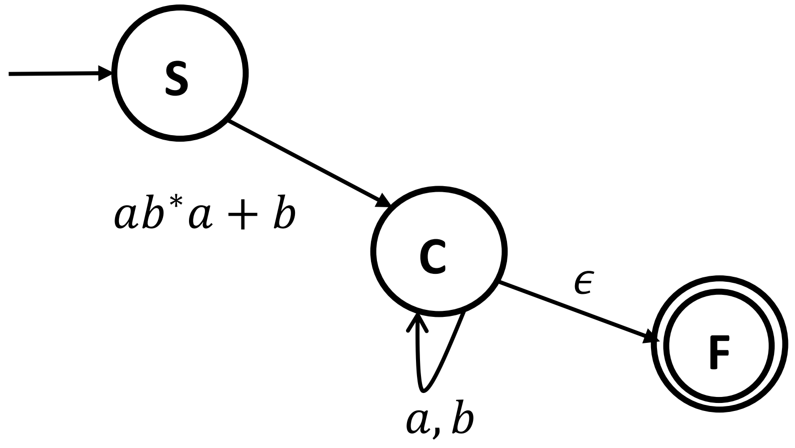


# DFA to Regular Expressions: GNFA

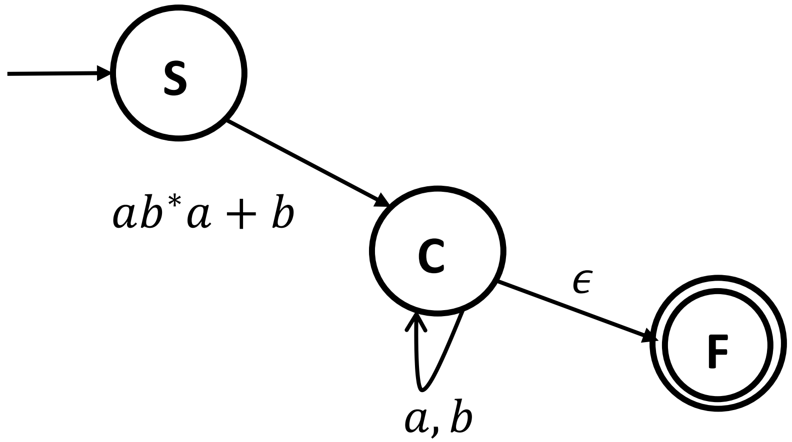



**Step 2: Eliminate $\boldsymbol{B}$**

$S \to C$ via $B$, RE: $ab^*a$
Overall RE for $S \to C$: $\boldsymbol{ab^*a + b}$

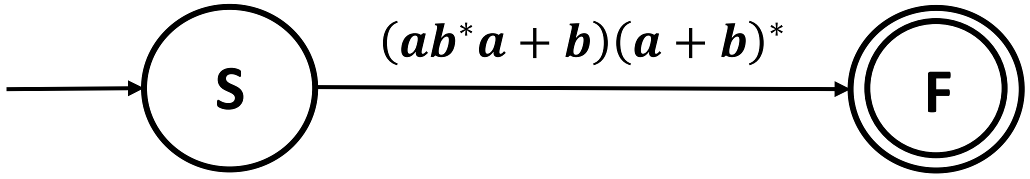


# DFA to Regular Expressions: GNFA

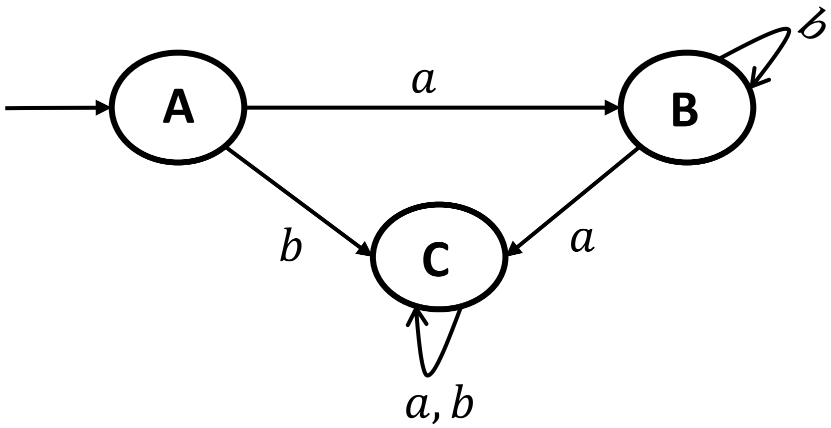


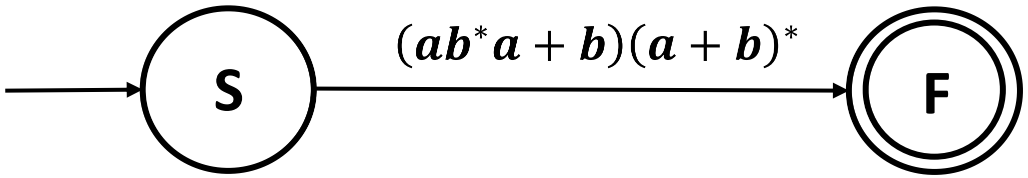

**Step 2: Eliminate $\boldsymbol{C}$**

$S \rightarrow F$ via C, RE: $(\boldsymbol{ab^*a} + \boldsymbol{b})(\boldsymbol{a} + \boldsymbol{b})^*$



# DFA to Regular Expressions: GNFA




Recursively, we managed to convert the DFA $M$ to a 2-state GNFA such that the label from of the arrow from the start state to the final state of the GNFA is the Regular Expression corresponding to $L(M)$.

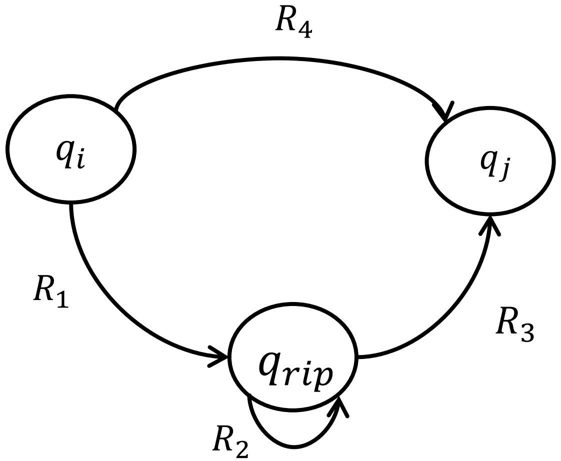
# DFA to Regular Expressions: GNFA

Formally, a GNFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states.
- $\Sigma$ is the input alphabet.
- $\delta: Q - \{F\} \times Q - \{q_0\} \mapsto \mathcal{R}$ is the transition function.
- $q_0$ is the start state.
- $F$ is the final state.

**Convert $k$-state GNFA to a 2-state GNFA:**

We provide a recursive algorithm CONVERT($G$) for this.




**CONVERT($G$):**

1. Let $k$ be the number of states of $G$.
2. If $k = 2$, then return the label $R$ of the arrow between the start and the final state.
3. If $k > 2$, select any state $q_{rip} \in Q$ different from $q_0$ and $F$ and let $G'$ be the GNFA$(Q', \Sigma, \delta', q_0, F)$, where

$$Q' = Q - \{q_{rip}\},$$

and for any $q_i \in Q' - \{F\}$ and any $q_j \in Q' - \{q_0\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) + R_4,$$

for $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$ and $R_4 = \delta(q_i, q_j)$

4. Compute CONVERT($G'$) and return its value.

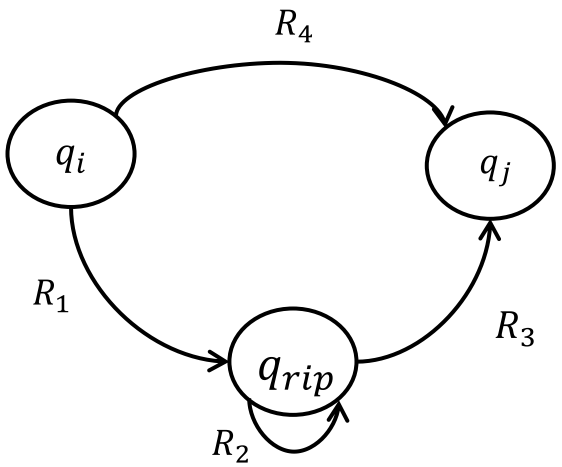
# DFA to Regular Expressions: GNFA

Formally, a GNFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states.
- $\Sigma$ is the input alphabet.
- $\delta: Q - \{F\} \times Q - \{q_0\} \mapsto \mathcal{R}$ is the transition function.
- $q_0$ is the start state.
- $F$ is the final state.

**Convert $k$-state GNFA to a $2$-state GNFA:**

We provide a recursive algorithm CONVERT($G$) for this.




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

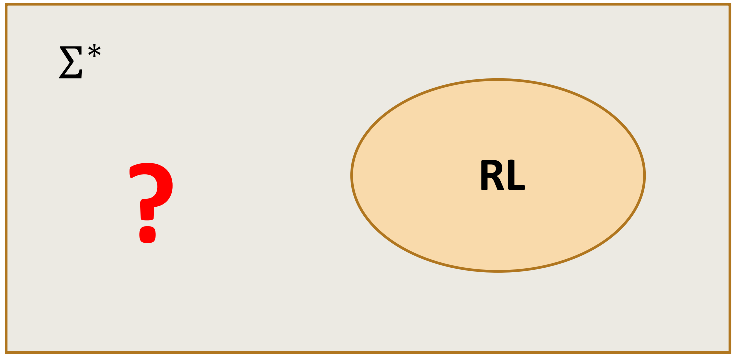
**How do Non-regular languages look like?**
**How can we prove that certain languages are not regular?**

# 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.

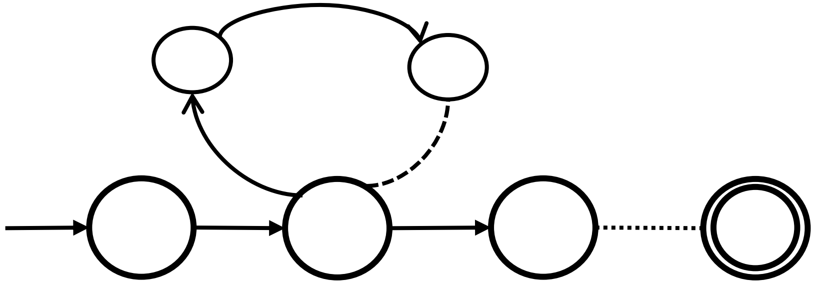
**Mil:** I have a DFA for $L$.
**Karl:** How many states are there?
**Mil:** $n$-states (say $n = 10$)
**Karl:** Then $0^{10}1^{10}$ must be accepted.
By the **pigeonhole principle**, while reading the first $(n = 10)$ symbols, some states need to be revisited. Otherwise $n + 1 = 11$ states would have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

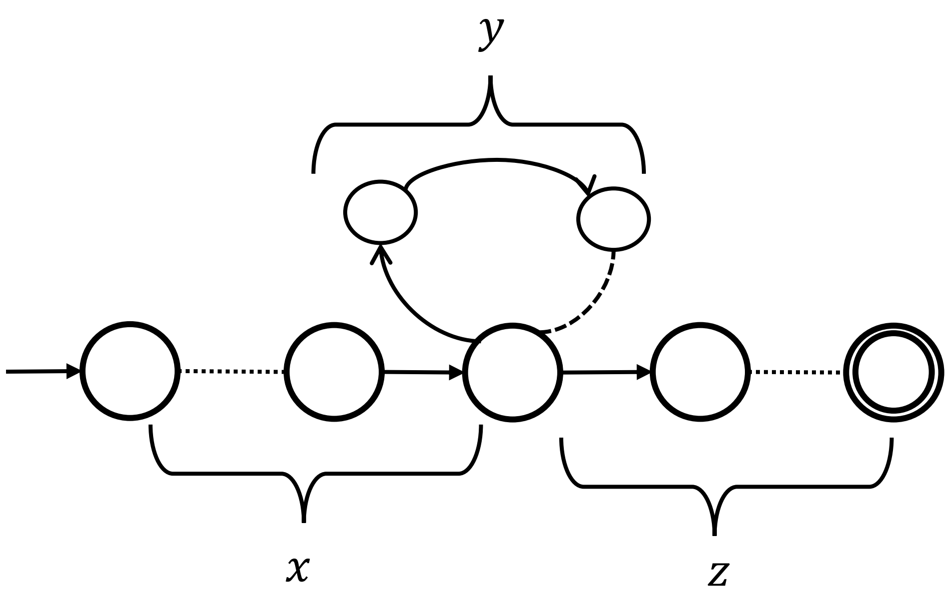
**Mil:** I have a DFA for $L$.
**Karl:** How many states are there?
**Mil:** $n$-states (say $n = 10$)
**Karl:** Then $0^{10}1^{10}$ must be accepted. By the **pigeonhole principle**, while reading the first $(n = 10)$ symbols, some states need to be revisited. Otherwise $n + 1 = 11$ states would have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

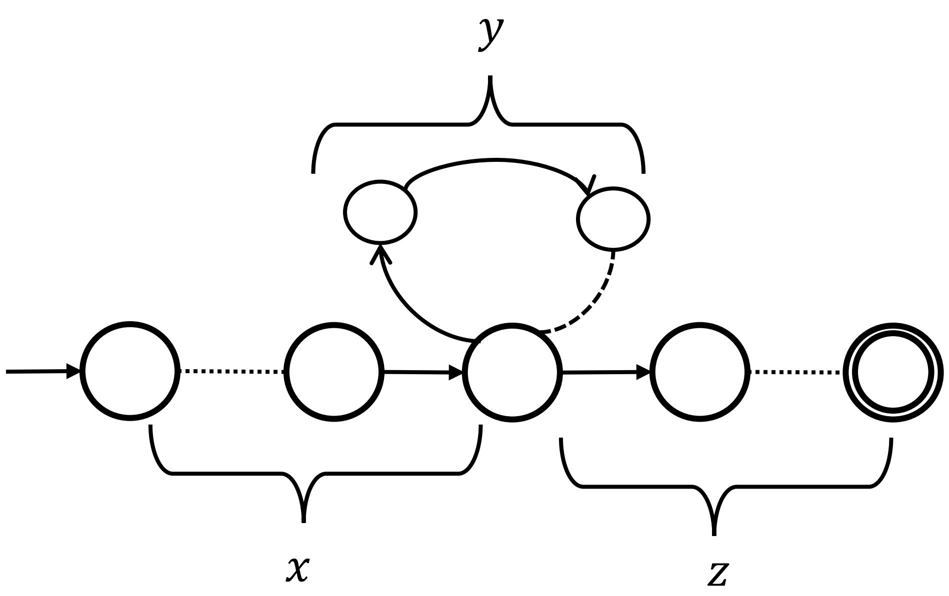


# Pumping Lemma

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

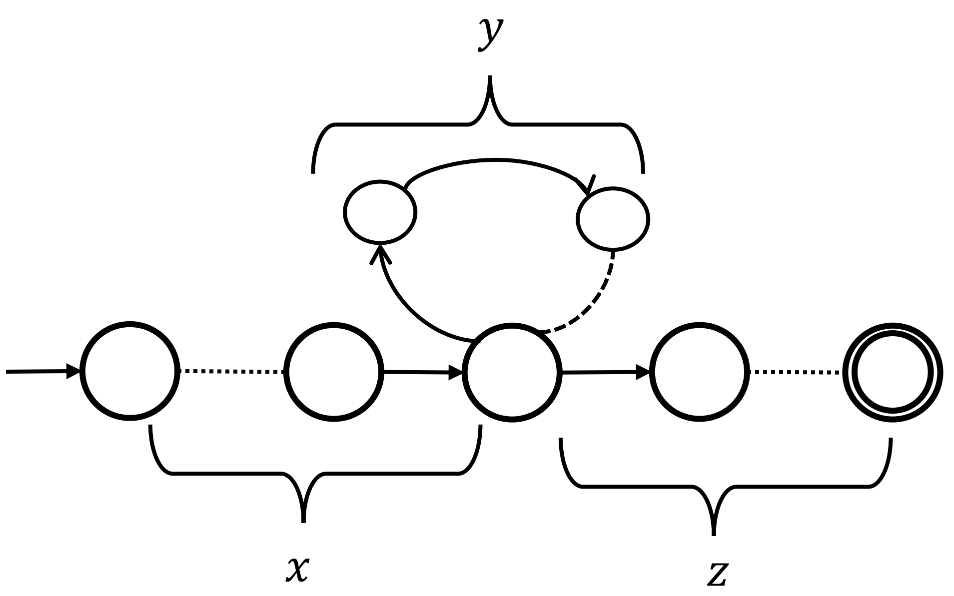
1. $|xy| \leq p$.
2. $|y| \geq 1$
3. $\forall i \geq 0, xy^i z \in L$.

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

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

$\equiv$

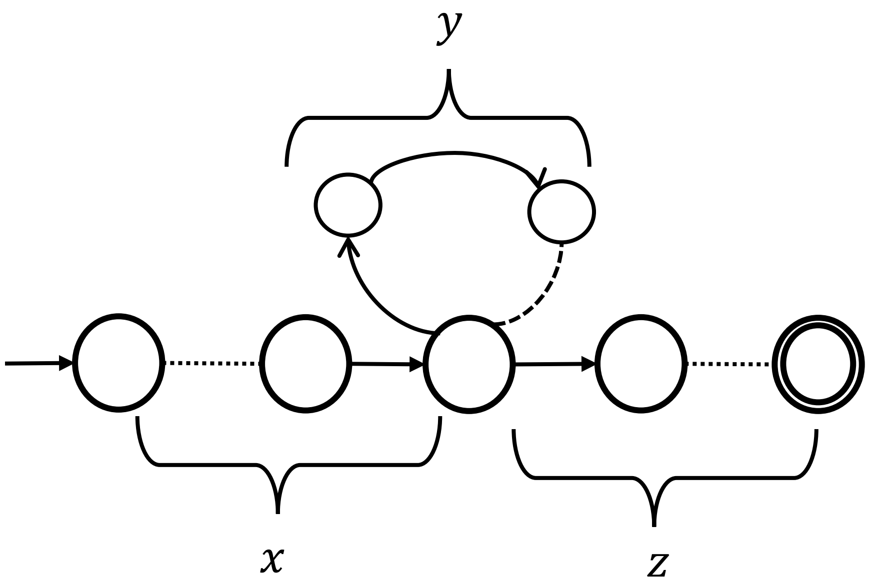
**If pumping property is NOT satisfied, then $L$ is NOT regular.**



# 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.



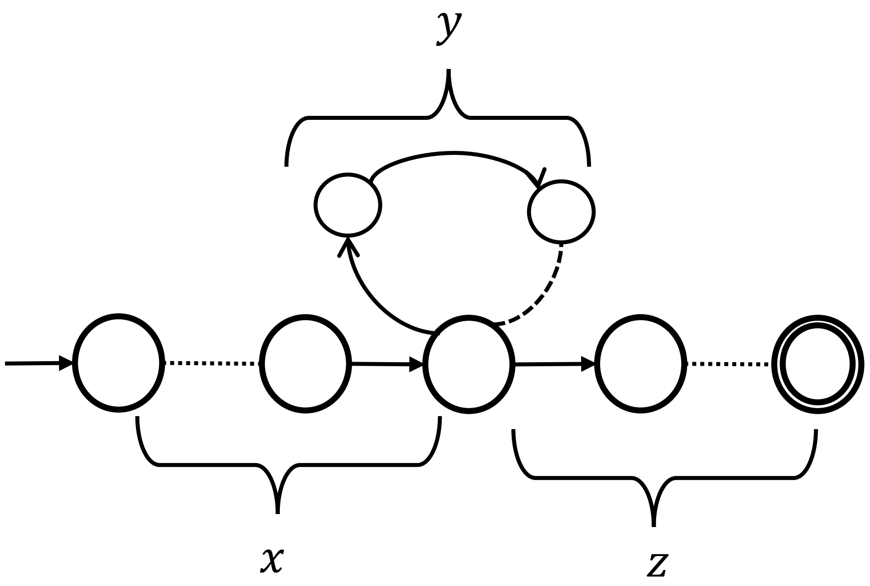
# Pumping Lemma

**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.

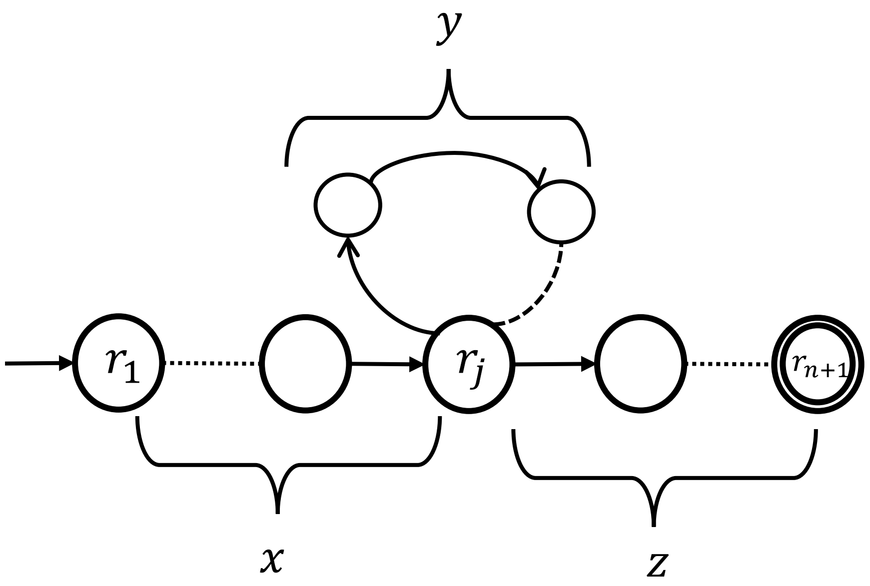
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are $p$ distinct states in the DFA.

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

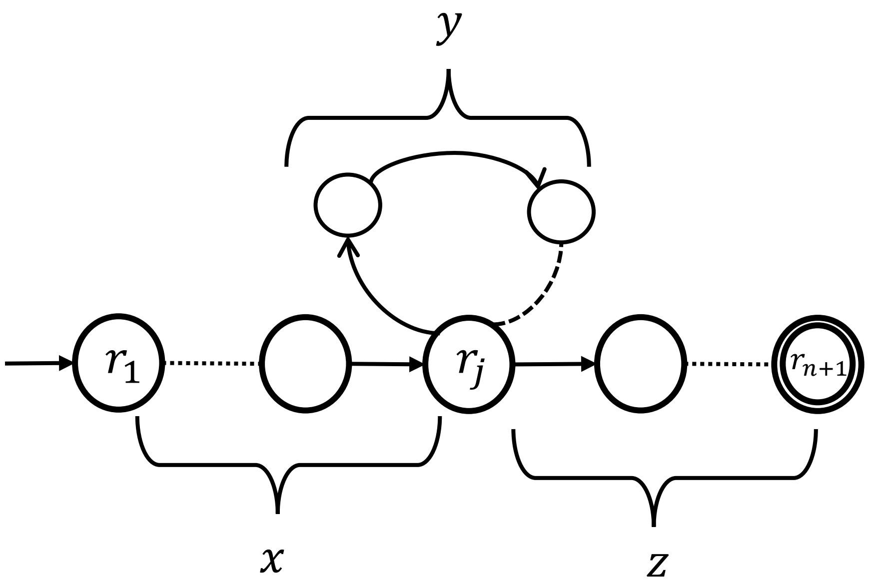
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are $p$ distinct states in the DFA.

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

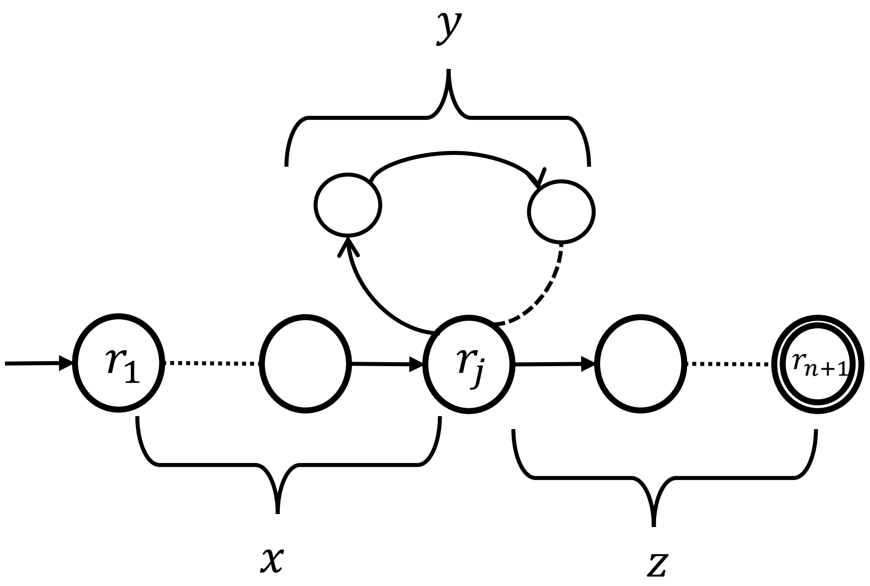
# Pumping Lemma

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

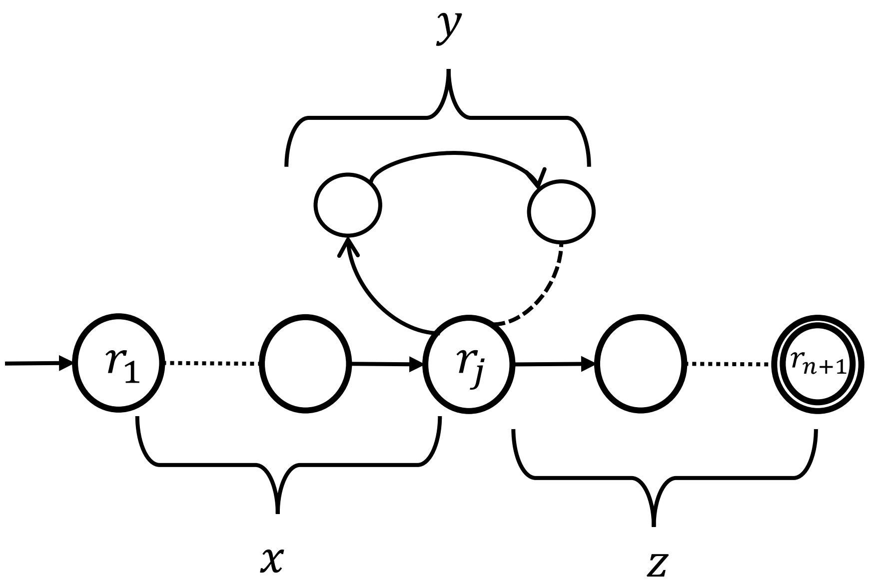
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are $p$ distinct states in the DFA.

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, $x = s_1 \dots s_{j-1}$, $y = s_j \dots .\, s_{l-1}$, $z = s_l \dots s_n$. 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 $j \neq l$, $|y| \geq 1$, 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 $\equiv$ 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.

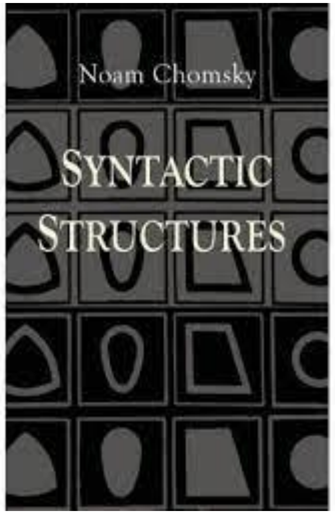
- Consider these rules

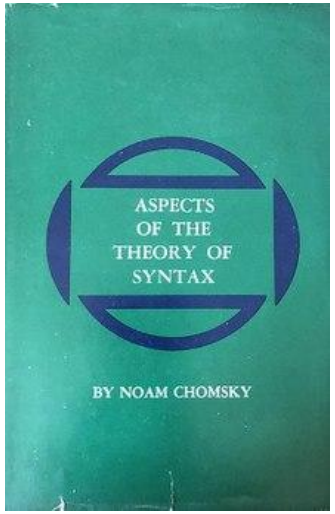
$$
\begin{aligned}
&Sentence \rightarrow Subject\ Verb\ Object\\
&Subject \rightarrow Noun.phrase\\
&Object \rightarrow Noun.phrase\\
&Noun.phrase \rightarrow Article\ Noun|Noun\\
&Article \rightarrow \boldsymbol{the}\\
&Noun \rightarrow \boldsymbol{boy}|\boldsymbol{girl}|\boldsymbol{soccer}|\boldsymbol{poetry}\\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

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

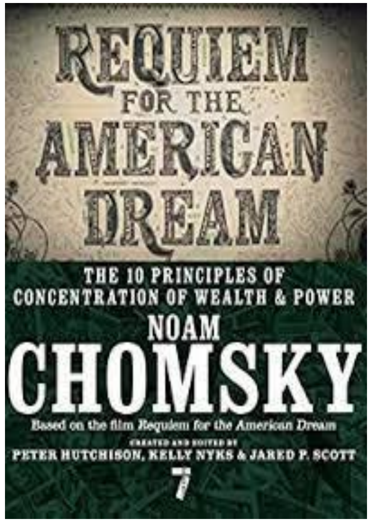
**Variables**: $\{Sentence, Subject, Verb, Object, Noun, Noun.phrase, Article\}$, **Terminals:** $\{The, girl, loves, plays, soccer, poetry\}$
**Start Variable:** $Sentence$

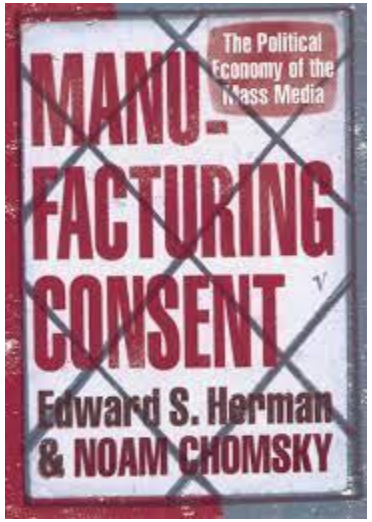
# Grammars




**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.

# Thank You!