CS 302.1 - Automata Theory

Lecture 03

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

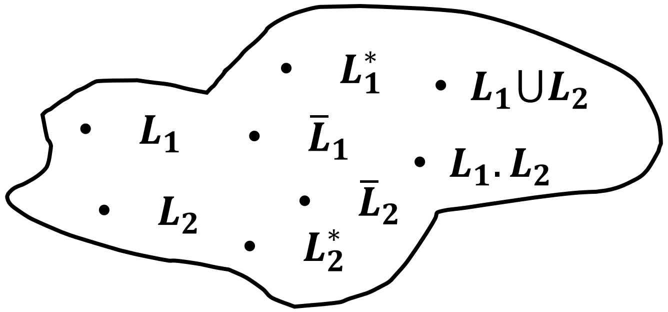
- DFAs and NFAs are equivalent
- For every NFA we can obtain a “Remembering DFA” that accepts the same language.
- The language accepted by finite automata are called Regular Languages.
- Regular operations: Union, Complement, Concatenation, **Star**.

Quick Recap

- DFAs and NFAs are equivalent
- For every NFA we can obtain a “Remembering DFA” that accepts the same language.
- The language accepted by finite automata are called Regular Languages.
- Regular operations: Union, Complement, Concatenation, **Star**.
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0$ and each $x_i \in L_1\}$. Examples:
  - If $\Sigma = \{a\}$, $\Sigma^* = \{\epsilon, a, aa, aaa, \dots\dots\}$

Quick Recap

- DFAs and NFAs are equivalent
- For every NFA we can obtain a "Remembering DFA" that accepts the same language.
- The language accepted by finite automata are called Regular Languages.
- Regular operations: Union, Complement, Concatenation, **Star**.
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0$ and each $x_i \in L_1\}$. Examples:
  - If $\Sigma = \{a\}$, $\Sigma^* = \{\epsilon, a, aa, aaa, \dots \dots\}$
  - If $\Sigma = \{\Phi\}, \Sigma^* = \{\epsilon\}$
- Regular Languages are closed under: Union, Star, Concatenation, Complement,…



Set of all regular Languages

Closure of Regular Languages

Q: Is the set of all regular languages **closed under intersection**? If L_1 and L_2 are regular, then is $L = L_1 \cap \mathrm{L}_2$ also regular?

Proof: We shall use the fact that regular languages are **closed** under union and complement.

Closure of Regular Languages

Q: Is the set of all regular languages **closed under intersection**? If L_1 and L_2 are regular, then is $L = L_1 \cap \mathrm{L}_2$ also regular?

Proof: We shall use the fact that regular languages are **closed** under union and complement.

Note that using De Morgan's laws:

$$\boldsymbol{L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}}$$

Closure of Regular Languages

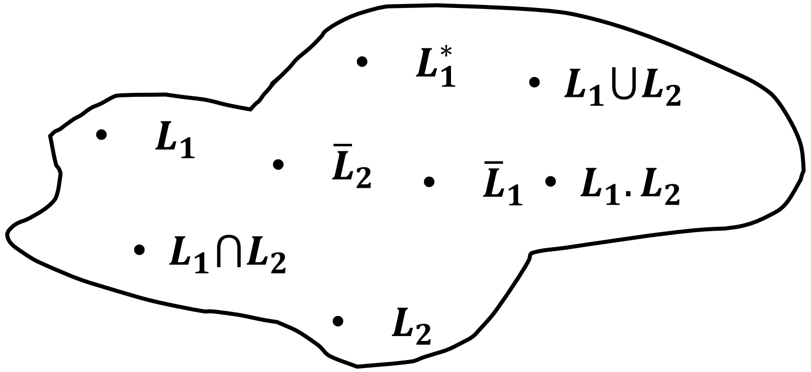
Q: Is the set of all regular languages **closed under intersection**? If L_1 and L_2 are regular, then is $L = L_1 \cap \mathrm{L}_2$ also regular?

Proof: We shall use the fact that regular languages are **closed** under union and complement.

Note that using De Morgan's laws:

$$\boldsymbol{L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}}$$

Given a DFA for L_1 and a DFA for L_2, we know how to construct an NFA for $\overline{L_1}, \overline{L_2}$ as well as for $L_1 \cup L_2$. Using these constructions and the aforementioned relationship, we can construct an NFA for $L = L_1 \cap \mathrm{L}_2$



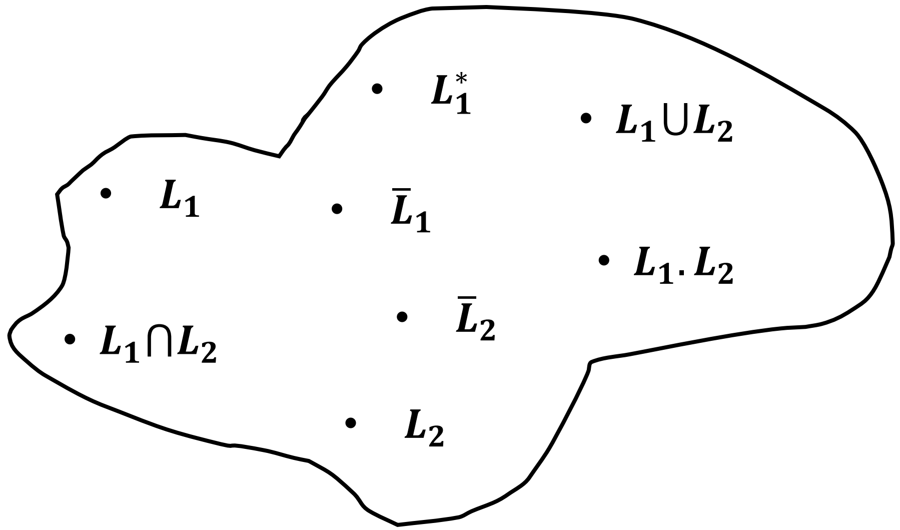
Set of all regular Languages

Closure of Regular Languages

Summary:

Regular Languages are closed under:

- **Union**
- **Intersection**
- **Star**
- **Complement**
- **Concatenation**



Set of all regular Languages

Regular Languages

If Σ is an alphabet, then

- $\Sigma^0 = \{\epsilon\}$
- $\Sigma^2 = \{a_1 a_2 | a_1 \in \Sigma,\ a_2 \in \Sigma\}$
- $\Sigma^k = \{a_1 a_2 \cdots a_k | a_i \in \Sigma \,|1 \leq i \leq k\}$
- $\Sigma^* = \{\bigcup_{i \geq 0} \Sigma^i\} = \{\Sigma^0 \cup \Sigma^1 \cup \Sigma^2\ \cdots\} = \{a_1 a_2 \cdots a_k | k \in \{0,1,\cdots\}\ \&\ a_j \in \Sigma, \forall j \in \{1,2,\cdots,k\}\}$

A Language $L \subset \Sigma^*$ and $L^* = \{\bigcup_{i \geq 0} \mathrm{L}^i\}$

Regular Languages

If Σ is an alphabet, then

- $\Sigma^0 = \{\epsilon\}$
- $\Sigma^2 = \{a_1 a_2 | a_1 \in \Sigma,\ a_2 \in \Sigma\}$
- $\Sigma^k = \{a_1 a_2 \cdots a_k | a_i \in \Sigma\ | 1 \leq i \leq k\}$
- $\Sigma^* = \{\bigcup_{i \geq 0} \Sigma^i\} = \{\Sigma^0 \cup \Sigma^1 \cup \Sigma^2\ \cdots\} = \{a_1 a_2 \cdots a_k | k \in \{0,1,\cdots\}\ \&\ a_j \in \Sigma, \forall j \in \{1,2,\cdots,k\}\}$

A Language $L \subset \Sigma^*$ and $L^* = \{\bigcup_{i \geq 0} \mathrm{L}^i\}$

Regular Language (alternate definition): Let Σ be an alphabet. Then the following are the regular languages over Σ:

- The empty language Φ is regular
- For each $a \in \Sigma, \{a\}$ is regular.
- Let L_1, L_2 be regular languages. Then $L_1 \cup L_2$, $L_1 . L_2$, L_1^* are regular languages.

Regular Expressions

A regular expression describes regular languages algebraically. The algebraic formulation also provides a powerful set of tools which will be leveraged to prove

- languages are regular
- derive properties of regular languages

Regular Expressions

A regular expression describes regular languages algebraically. The algebraic formulation also provides a powerful set of tools which will be leveraged to prove

- languages are regular
- derive properties of regular languages

Syntax for regular expressions (Recursive definition): R is said to be a regular expression if it has one of the following forms:

- Φ is a regular expression, $L(\Phi) = \Phi$
- ϵ is a regular expression, $L(\epsilon) = \{\epsilon\}$
- Any $a \in \Sigma$ is a regular expression, $L(a) = \{a\}$

Regular Expressions

A regular expression describes regular languages algebraically. The algebraic formulation also provides a powerful set of tools which will be leveraged to prove

- languages are regular
- derive properties of regular languages

Syntax for regular expressions (Recursive definition): R is said to be a regular expression if it has one of the following forms:

- Φ is a regular expression, $L(\Phi) = \Phi$
- ϵ is a regular expression, $L(\epsilon) = \{\epsilon\}$
- Any $a \in \Sigma$ is a regular expression, $L(a) = \{a\}$
- $R_1 + R_2$ is a regular expression if R_1 and R_2 are regular expressions, $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- R^* is a regular expression if R is a regular expression, $L(R^*) = \left(L(R)\right)^*$

Regular Expressions

A regular expression describes regular languages algebraically. The algebraic formulation also provides a powerful set of tools which will be leveraged to prove

- languages are regular
- derive properties of regular languages

Syntax for regular expressions (Recursive definition): R is said to be a regular expression if it has one of the following forms:

- Φ is a regular expression, $L(\Phi) = \Phi$
- ϵ is a regular expression, $L(\epsilon) = \{\epsilon\}$
- Any $a \in \Sigma$ is a regular expression, $L(a) = \{a\}$
- $R_1 + R_2$ is a regular expression if R_1 and R_2 are regular expressions, $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- R^* is a regular expression if R is a regular expression, $L(R^*) = \left(L(R)\right)^*$
- $R_1 R_2$ is a regular expression if R_1 and R_2 are regular expressions, $L(R_1 R_2) = L(R_1). L(R_2)$
- (R) is a regular expression if R is a regular expression, $L\left((R)\right) = R$

Regular Expressions

Syntax for regular expressions:

Regular Expression	**Regular Language**	**Comment**
Φ	$\{\}$	The empty set
ϵ	$\{\epsilon\}$	The set containing ϵ 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

Order of precedence: (), *, . , +

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

RE's are equivalent in power to NFAs/DFAs

Regular Expressions

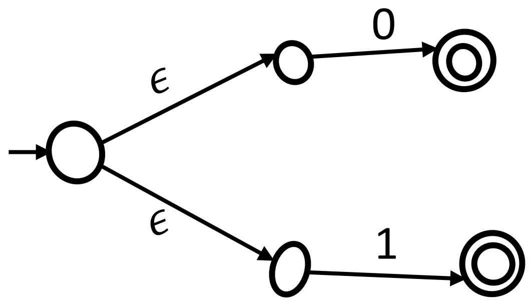
Syntax for regular expressions:

Regular Expression R	$L(R)$
01	$\{01\}$
$01+1$	$\{01,1\}$
$(0+1)^*$	$\{\epsilon, 0, 1, 00, 01, \cdots\}$
$(01+\epsilon)1$	$\{011,1\}$
$(0+1)^*01$	$\{01,001,101,0001, \cdots\}$
$(0+10)^*(\epsilon+1)$	$\{\epsilon, 0, 10, 00, 001, 010, 0101, \cdots\}$

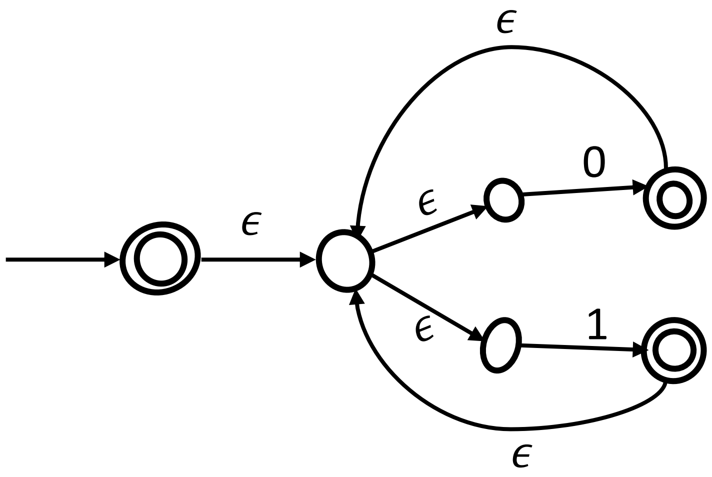
Regular Expressions

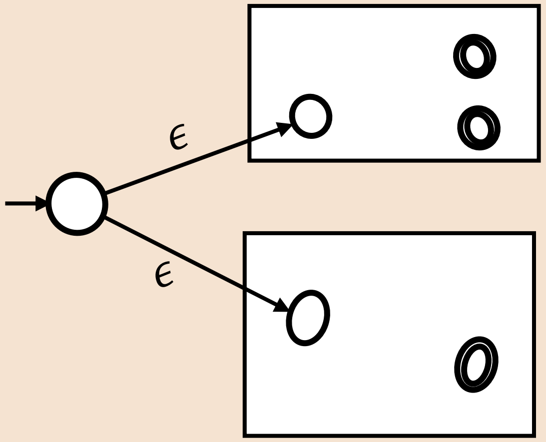
NFA for RE: $(\mathbf{0}+\mathbf{1})^*\mathbf{01}$

(i) NFA for $(0+1)$

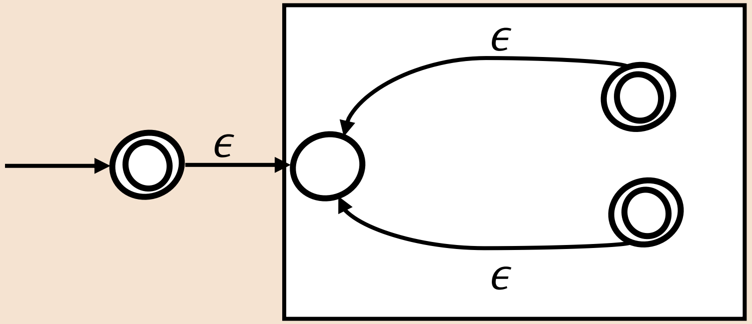


(ii) NFA for $(0+1)^*$





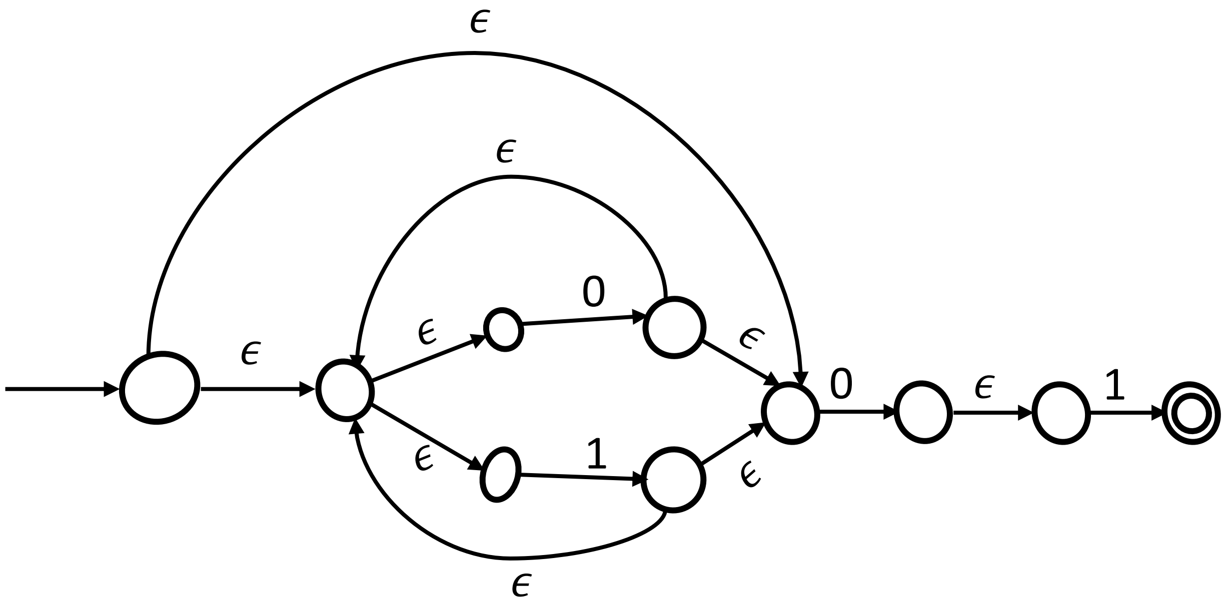
$\boldsymbol{L_1 \cup L_2}$

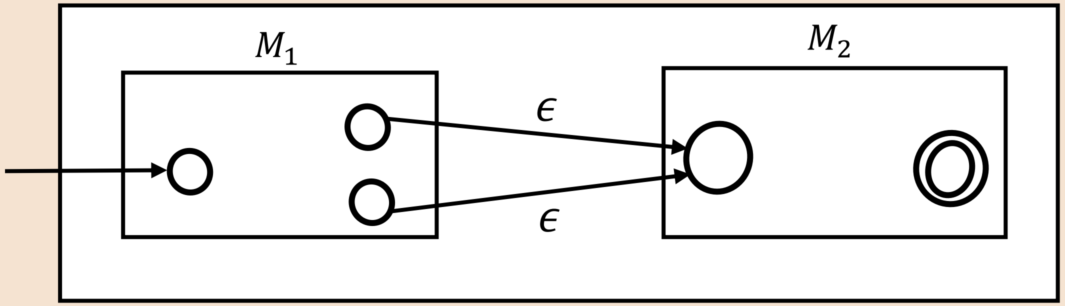


$\boldsymbol{L_1^*}$

Regular Expressions

NFA for $(\mathbf{0}+\mathbf{1})^{*}\mathbf{01}$





$L_1.L_2$

Regular Expressions

Let $\Sigma = \{a, b\}$.

Language	Regular Expression
$\{\omega \mid \omega$ ends in "ab"$\}$	$(a+b)^*ab$
$\{\omega \mid \omega$ has a single $a\}$	b^*ab^*
$\{\omega \mid \omega$ has at most one $a\}$	$b^* + b^*ab^*$
$\{\omega \mid \lvert\omega\rvert$ is even$\}$	$((a+b)(a+b))^* = (aa+bb+ab+ba)^*$
$\{\omega \mid \omega$ has "ab" as a substring$\}$	$(a+b)^*ab(a+b)^*$
$\{\omega \mid \lvert\omega\rvert$ is a multiple of 3$\}$	$\big((a+b)(a+b)(a+b)\big)^*$

Regular Expressions

Let $\Sigma = \{a, b\}$.

Language	**Regular Expression**
$\{\omega \mid \omega$ ends in "ab"$\}$	$(a+b)^*ab$
$\{\omega \mid \omega$ has a single a $\}$	b^*ab^*
$\{\omega \mid \omega$ has at most one $a\}$	$b^* + b^*ab^*$
$\{\omega \mid \ \lvert\omega\rvert$ is even$\}$	$((a+b)(a+b))^* = (aa+bb+ab+ba)^*$
$\{\omega \mid \omega$ has "ab" as a substring$\}$	$(a+b)^*ab(a+b)^*$
$\{\omega \mid \lvert\omega\rvert$ is a multiple of 3$\}$	$\big((a+b)(a+b)(a+b)\big)^*$

Some algebraic properties of Regular Expressions:

- $R_1 + (R_2 + R_3) = (R_1 + R_2) + R_3$
- $R_1(R_2R_3) = (R_1R_2)R_3$
- $R_1(R_2 + R_3) = R_1R_2 + R_1R_3$
- $(R_1 + R_2)R_3 = \mathrm{R_1R_3 + R_2R_3}$
- $R_1 + R_2 = R_2 + R_1$
- $R_1^*R_1^* = R_1^*$
- $(R_1^*)^* = R_1^*$
- $R\epsilon = \epsilon R = R$
- $R\Phi = \Phi R = \Phi$
- $R + \Phi = R$
- $\epsilon + RR^* = \epsilon + R^*R = R^*$
- $(R_1 + R_2)^* = (R_1^*R_2^*)^* = (R_1^* + R_2^*)^*$

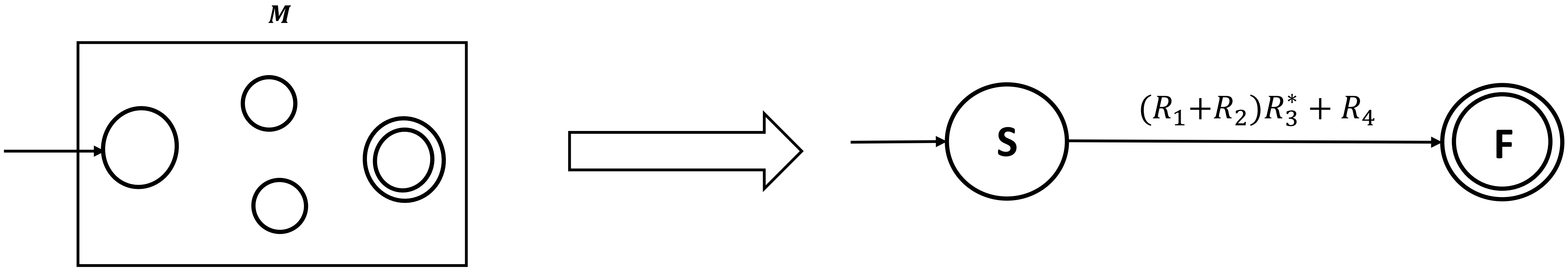
DFA to Regular Expressions

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

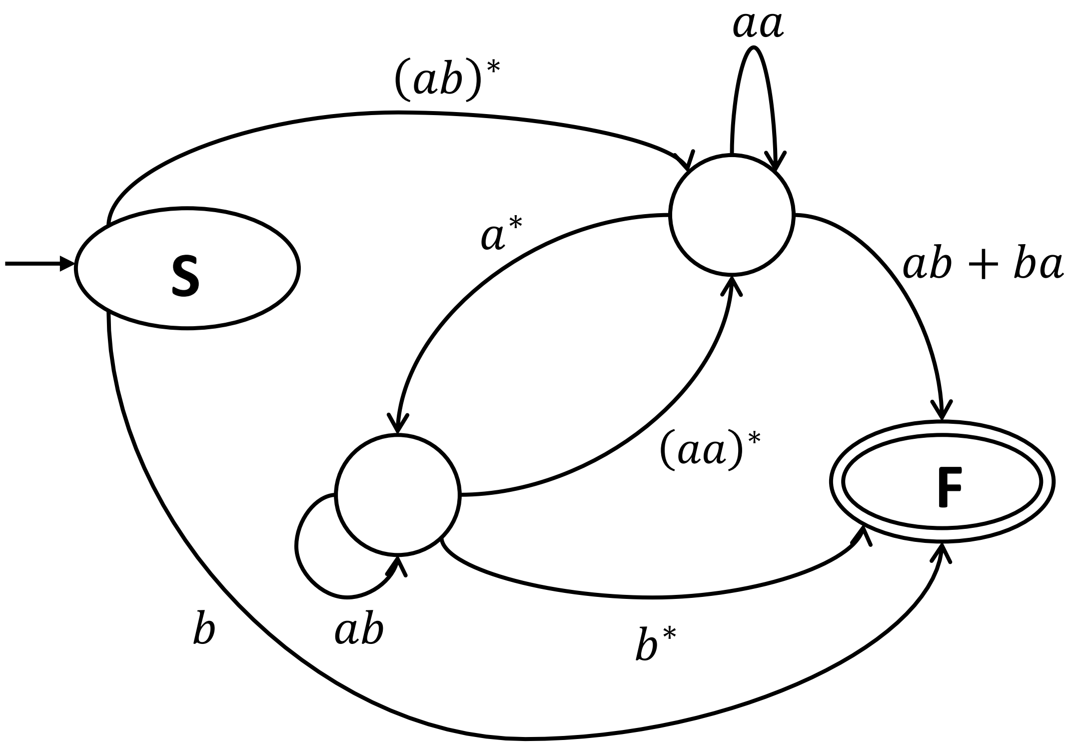
- A start state and a final state
- A single arrow goes from the start state to the final state
- The label of this arrow is the regular expression corresponding to the language accepted by the DFA M.



DFA to Regular Expressions: GNFA

What are GNFAs? They are simply NFAs such that

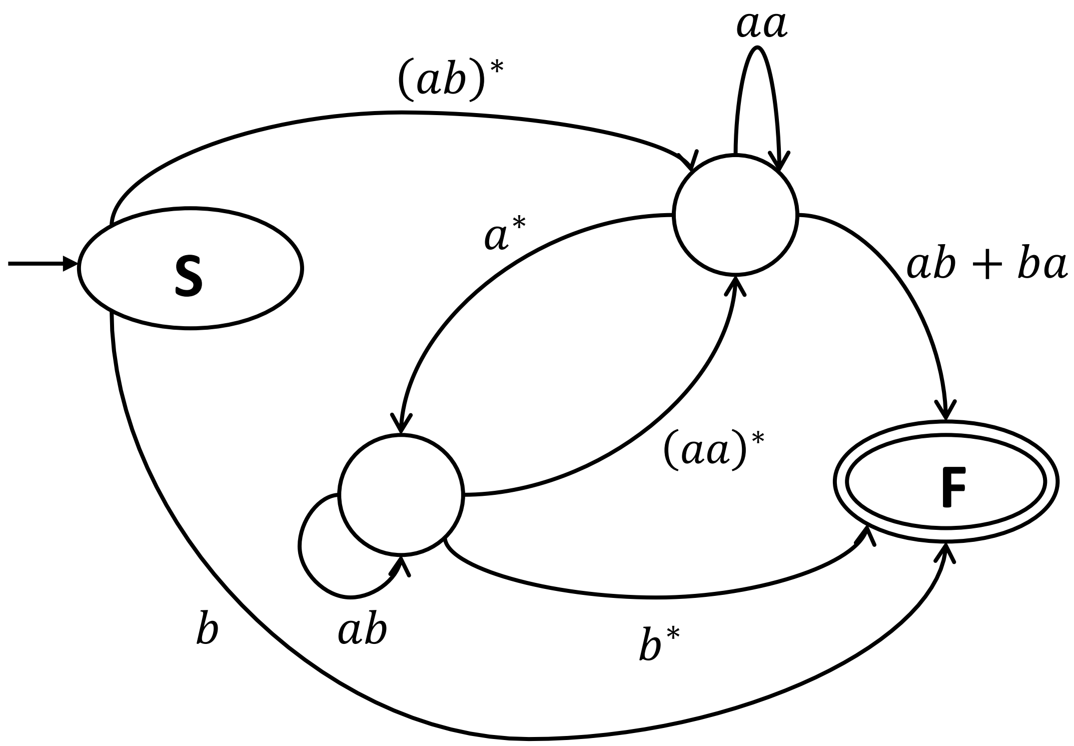
- The transitions may have regular expressions
- A unique start state that has arrows going to other states, but has no incoming arrows
- A unique final state that has arrows incoming from other states, but has no outgoing arrows
- For an input string, **runs** on a GNFA are similar to that of an NFA,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

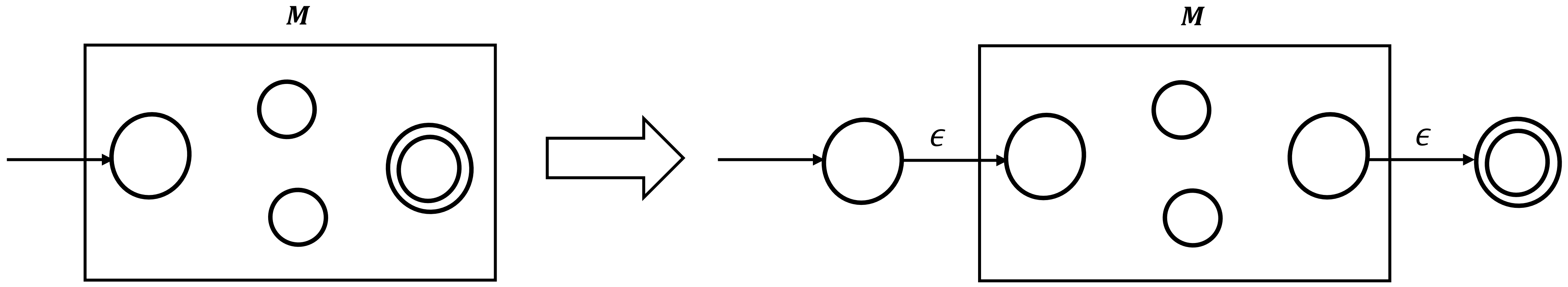


Starting from a DFA we will begin by constructing a GNFA with k states. We then outline a recursive procedure by which at each step, we will construct a GNFA with one less state. This step will be repeated until we obtain the **2-state GNFA**.

DFA to Regular Expressions: GNFA

Starting from the DFA M,

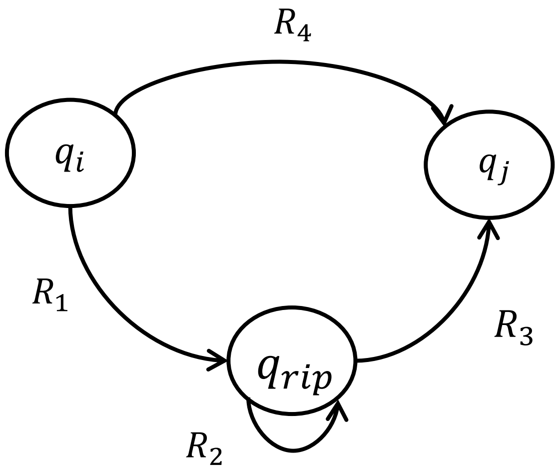
- Add a new start state with an ϵ arrow to the old start state.
- Add a new final state by with an ϵ arrow to the old final state.



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.

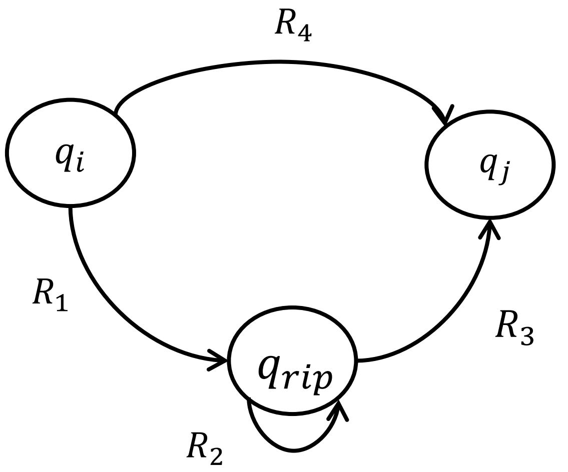
- Start by picking any state of the GNFA (except the new start and final states)
- Let us call this state q_{rip}. We "rip" q_{rip} out of the machine and create a GNFA with $k-1$ states.
- Of course, we need to "repair" the machine by altering the regular expressions that label each of the remaining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

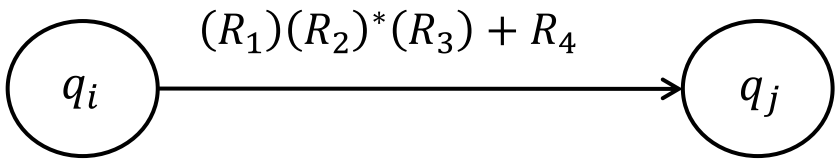


DFA to Regular Expressions: GNFA

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

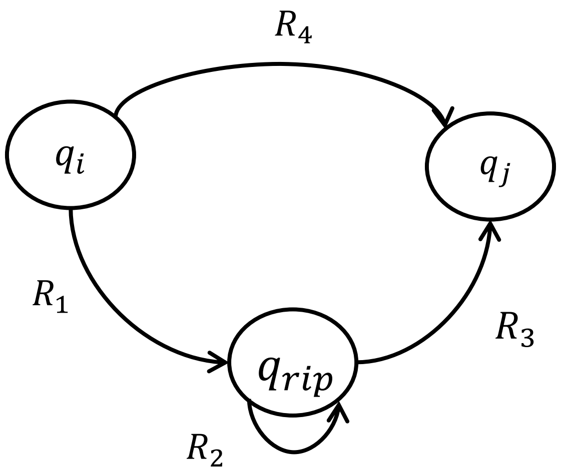
The crucial step is to convert a GNFA with k (>2) states to a GNFA with $k-1$ states.

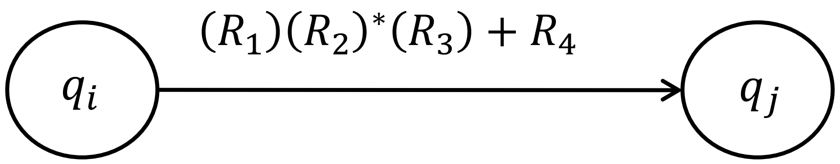
How do we remove q_{rip}? In the old machine if

- 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 $k = 2$

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

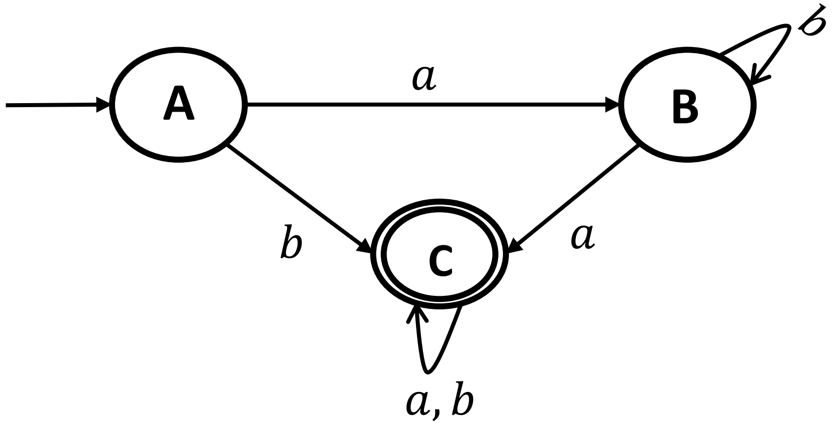




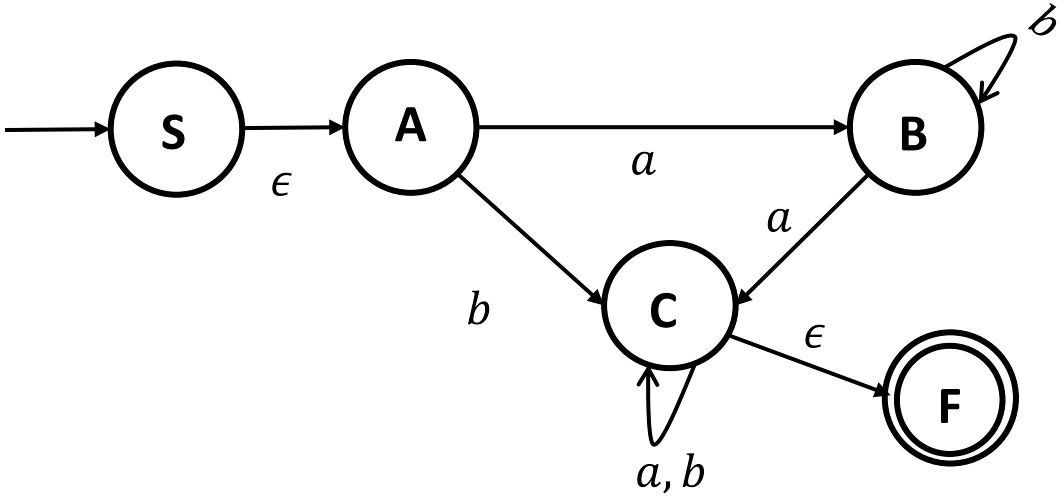
This should be done for **every pair** of arrows outgoing and incoming q_{rip}

DFA to Regular Expressions: GNFA

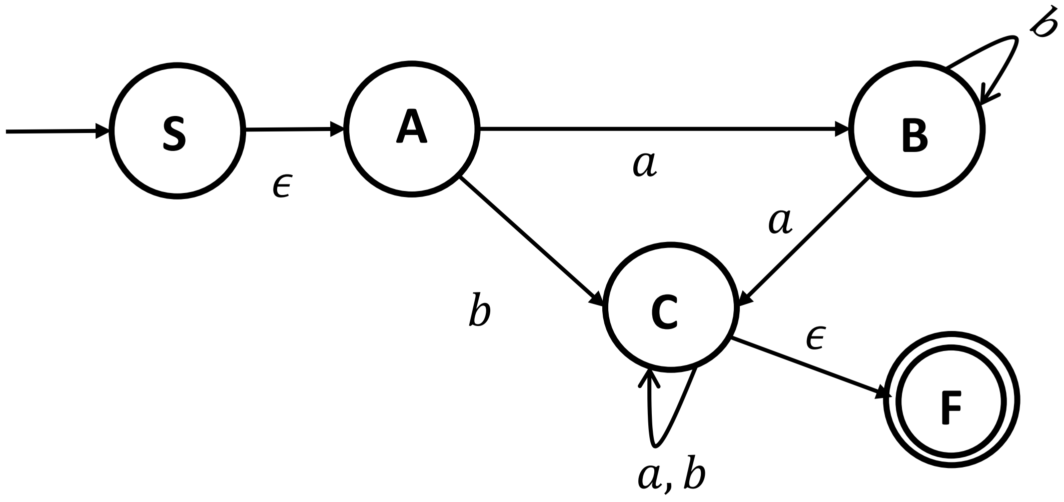
Let us look at an example. Consider the original DFA M below and find the regular expression corresponding to $L(M)$.



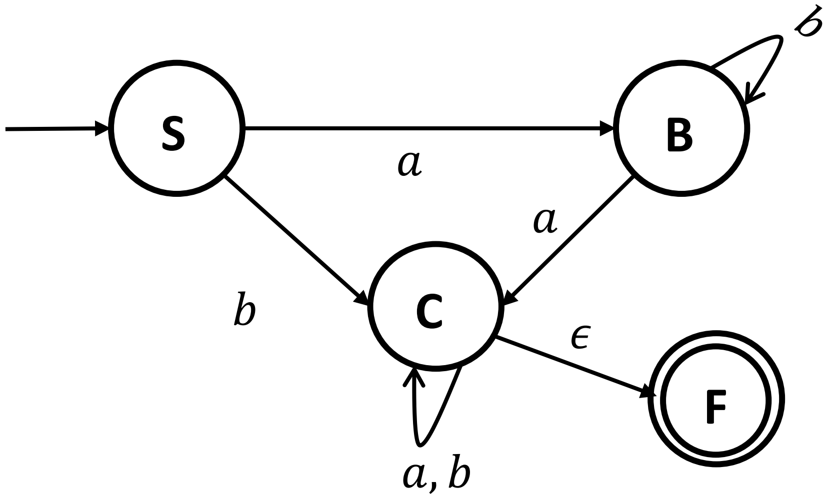
Step 1: Add new start and final states



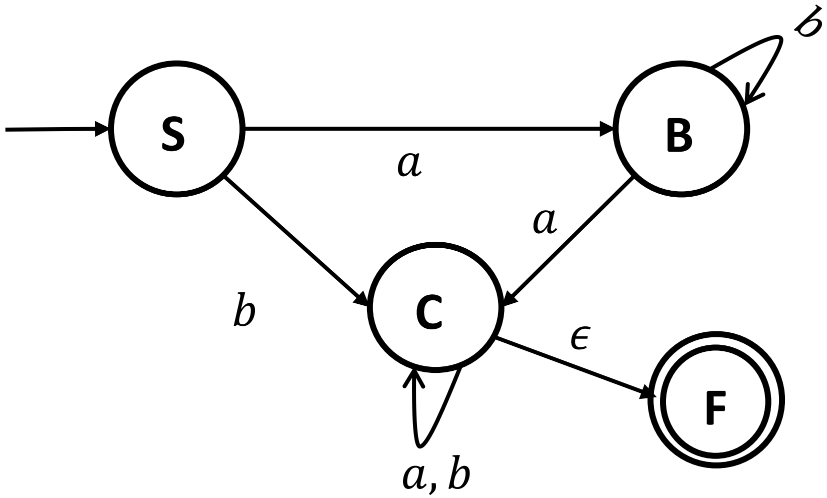
DFA to Regular Expressions: GNFA



Step 2: Eliminate A

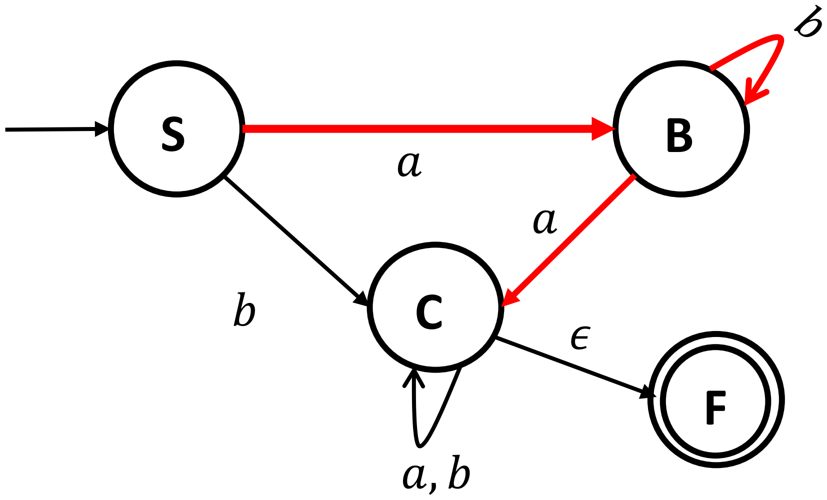


DFA to Regular Expressions: GNFA



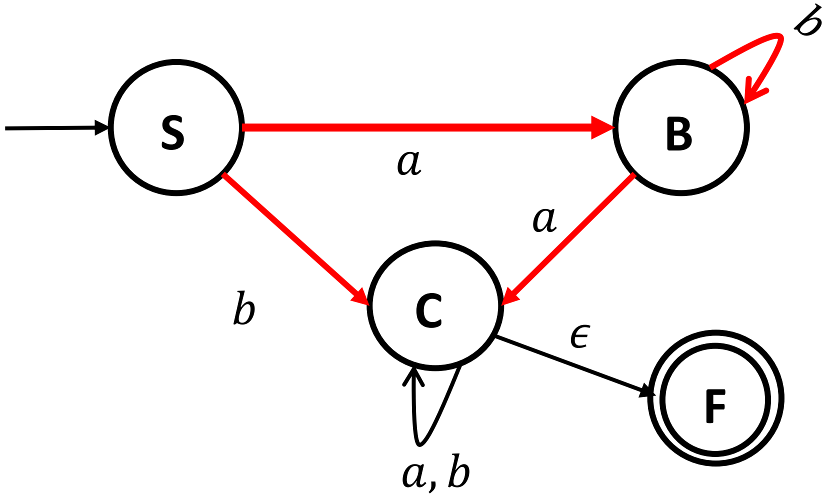
Step 2: Eliminate $\boldsymbol{B}$

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



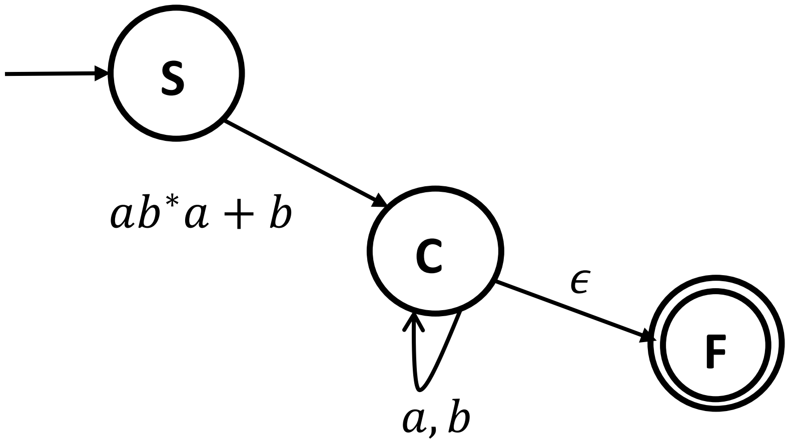
ab^*a

DFA to Regular Expressions: GNFA

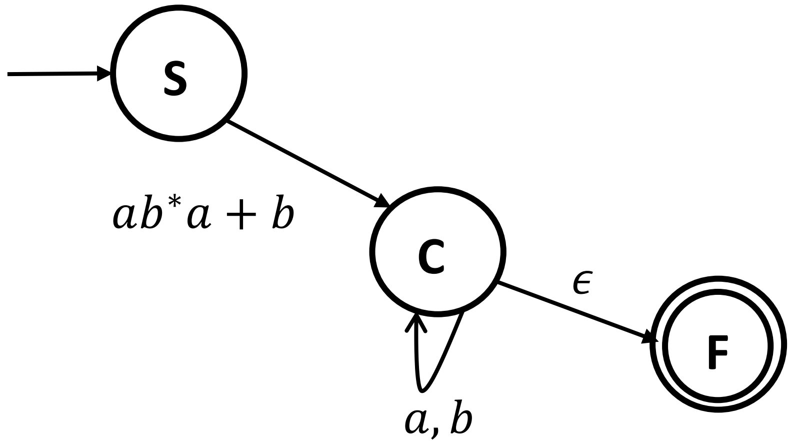


Step 2: Eliminate $\boldsymbol{B}$

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

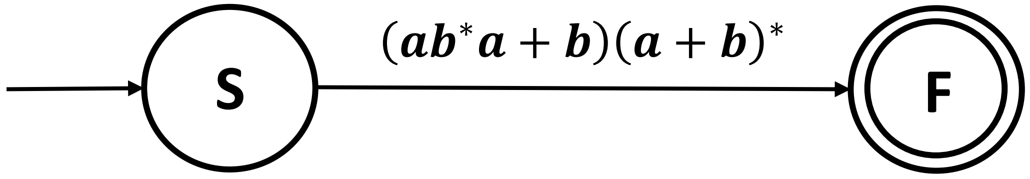


DFA to Regular Expressions: GNFA

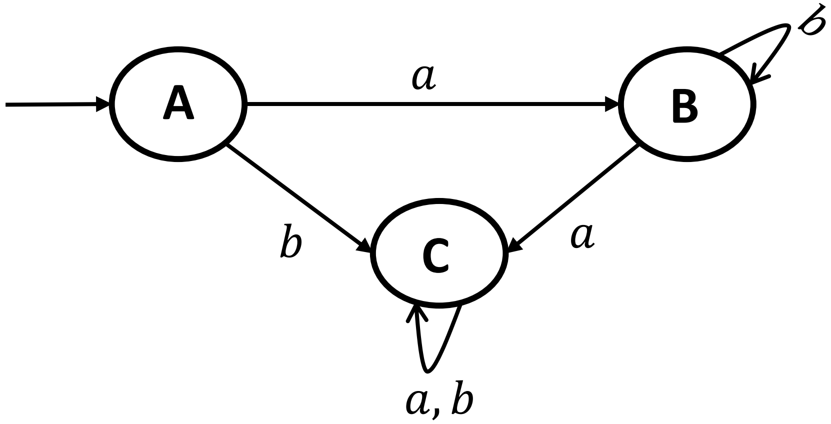


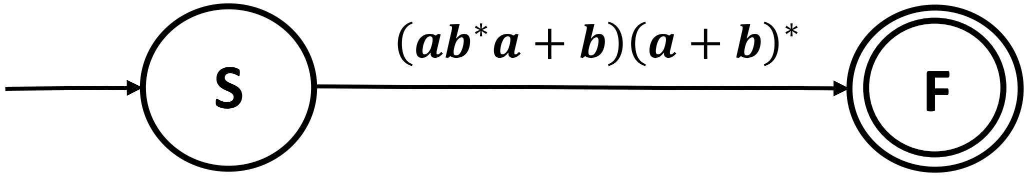
Step 2: Eliminate $\boldsymbol{C}$

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



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 $L(M)$.

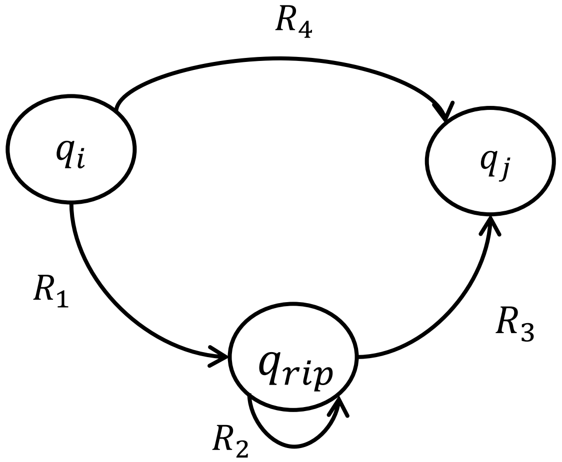
DFA to Regular Expressions: GNFA

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

- Q is a finite set of states.
- Σ is the input alphabet.
- $\delta: Q - \{q_0\} \times Q - \{F\} \mapsto \mathcal{R}$ 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 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 $q_i \in Q' - \{q_0\}$ 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.

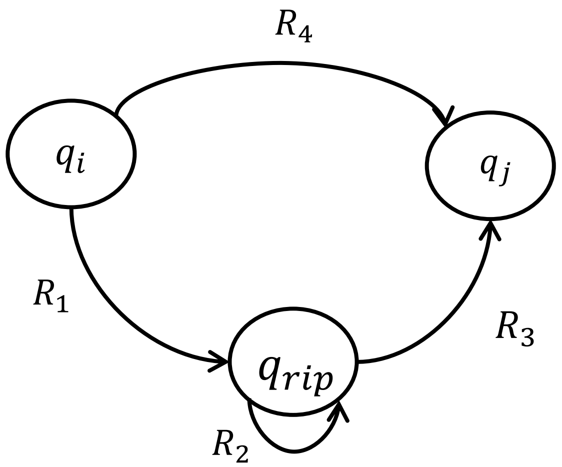
DFA to Regular Expressions: GNFA

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

- Q is a finite set of states.
- Σ is the input alphabet.
- $\delta: Q - \{q_0\} \times Q - \{F\} \mapsto \mathcal{R}$ 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

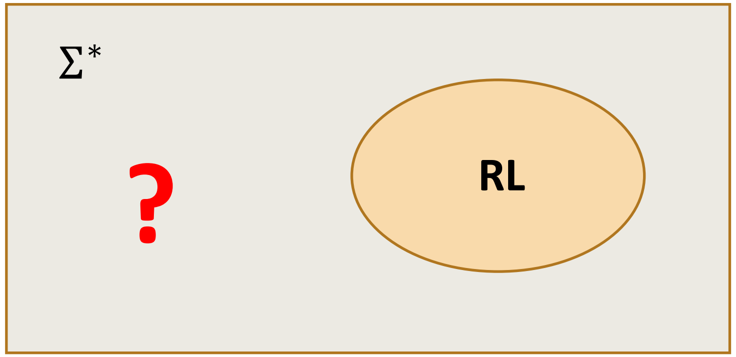
How do Non-regular languages look like?
How can we prove that certain languages are not regular?

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.

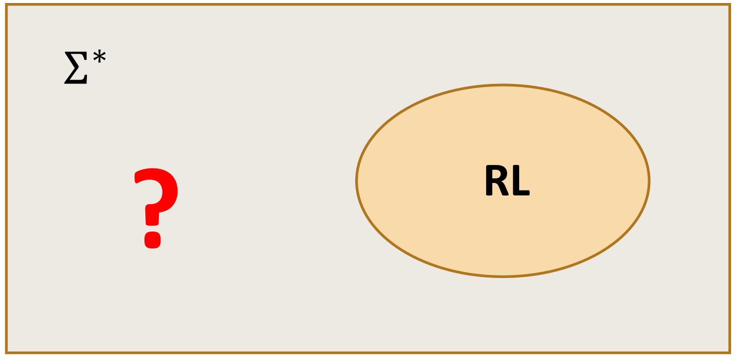


Pumping Lemma

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.

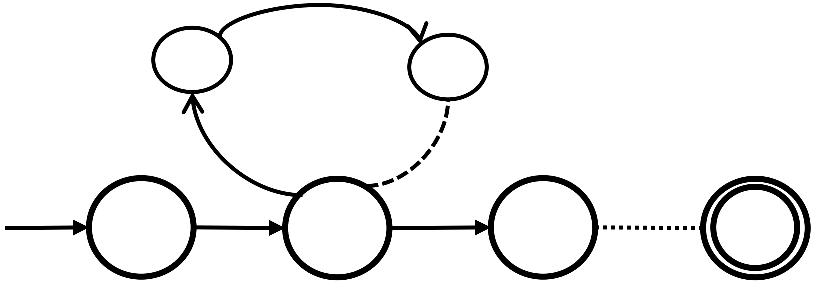
Mil: I have a DFA for L.
Karl: How many states are there?
Mil: n-states (say $n = 10$)
Karl: Then $0^{10}1^{10}$ must be accepted.
By the **pigeonhole principle**, while reading the first $(n = 10)$ symbols, some states need to be revisited. Otherwise $n + 1 = 11$ states would have been present. Hence 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 $L = \{0^n1^n | n \geq 0\}$ and the following conversation between Karl and Mil.

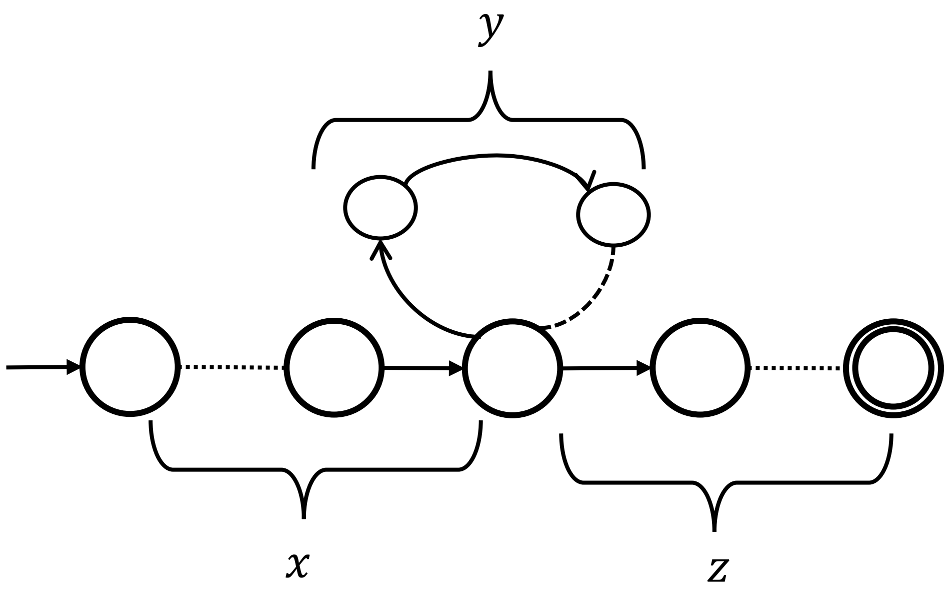
Mil: I have a DFA for L.
Karl: How many states are there?
Mil: n-states (say $n = 10$)
Karl: Then $0^{10}1^{10}$ must be accepted. By the **pigeonhole principle**, while reading the first $(n = 10)$ symbols, some states need to be revisited. Otherwise $n + 1 = 11$ states would have been present. Hence 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^n1^n, it must also accept $0^{n+t}1^n$. This is because, if we take the loop one extra time, we read t more 0's.



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

Pumping Lemma

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

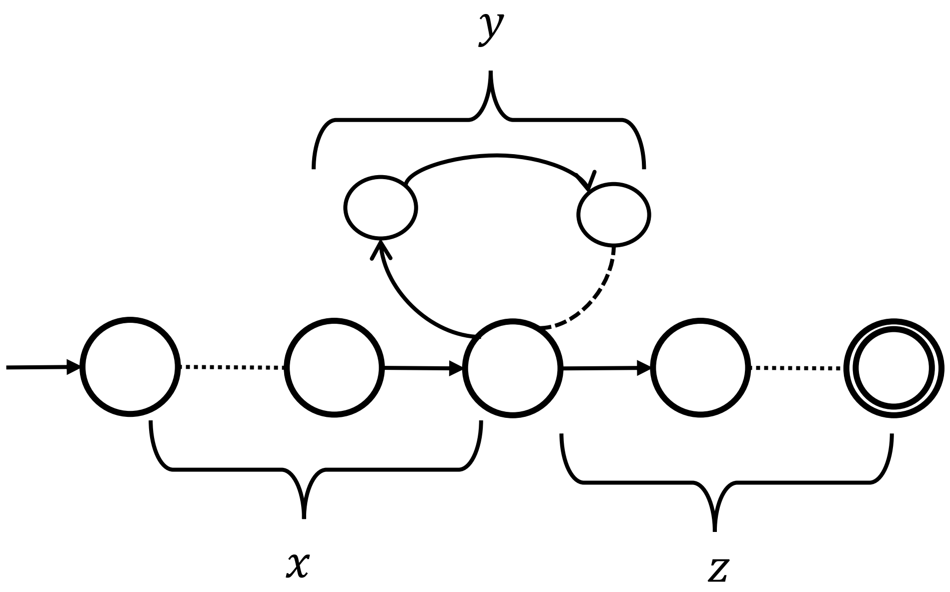


Pumping Lemma

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

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

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



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

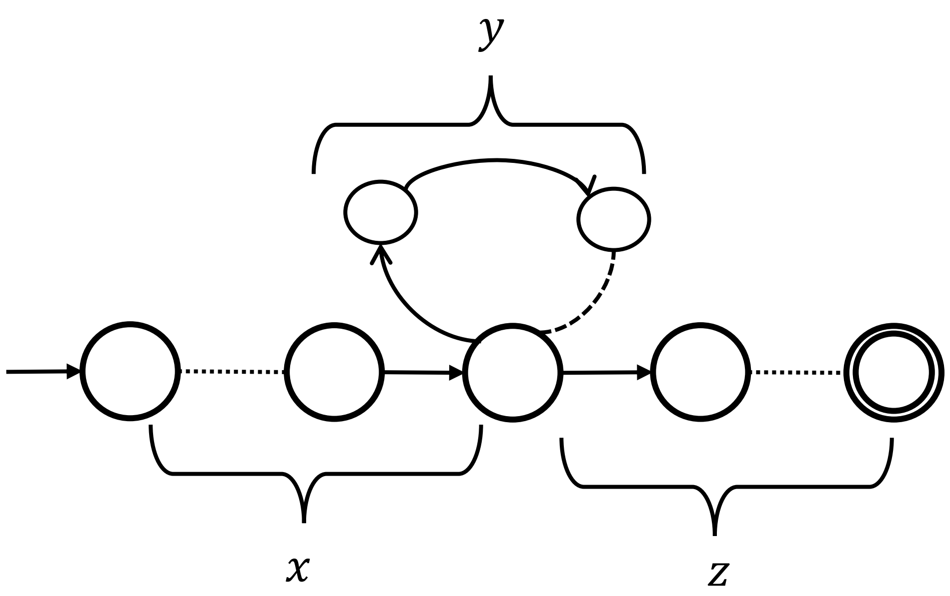
1. $|xy| \leq p$.
2. $|y| \geq 1$
3. $\forall i \geq 0, xy^i z \in L$.

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

If L is regular then, pumping property is satisfied

$\equiv$

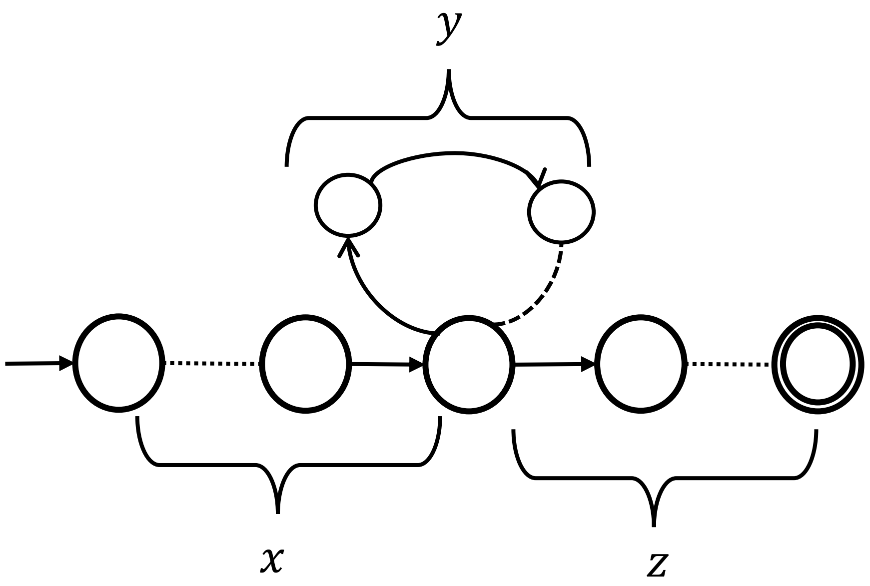
If pumping property is NOT satisfied, then L is NOT regular.



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.



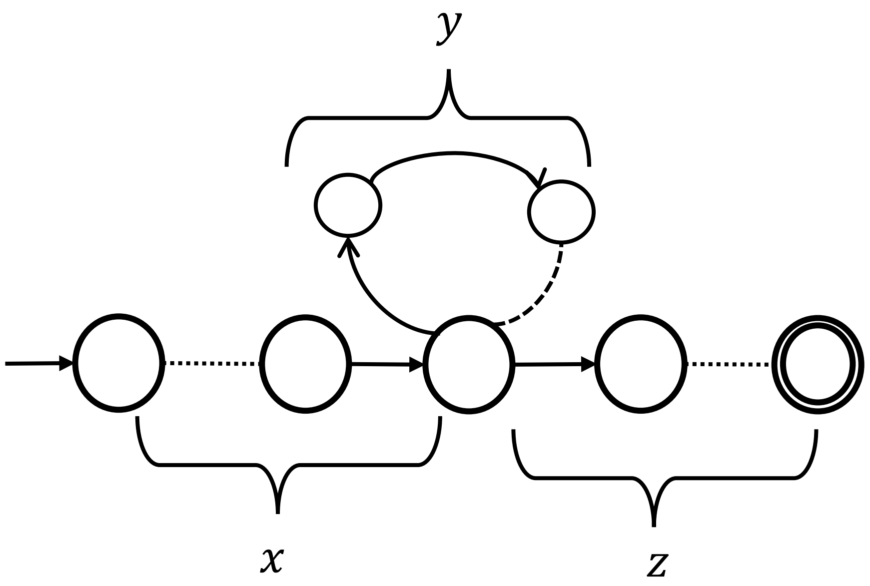
Pumping Lemma

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.

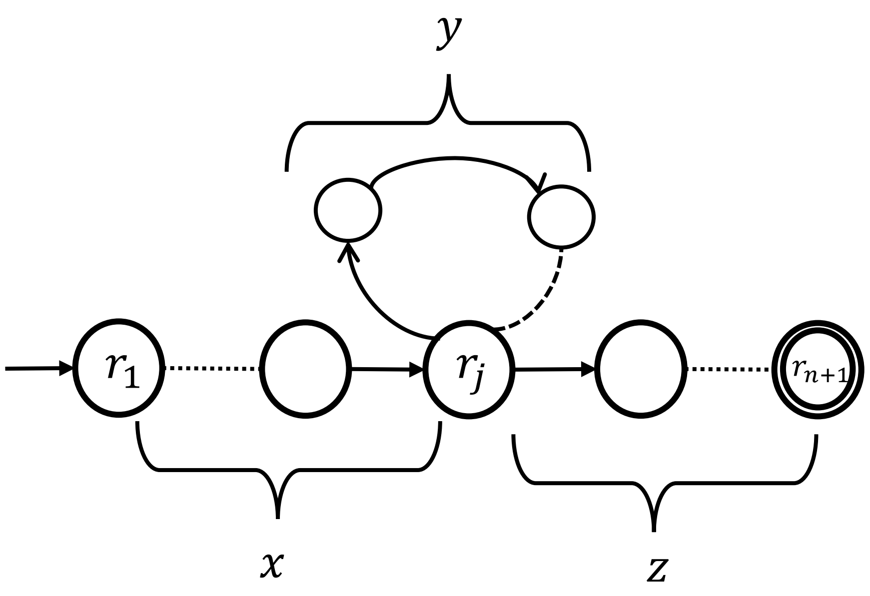
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are p distinct states in the DFA.

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.

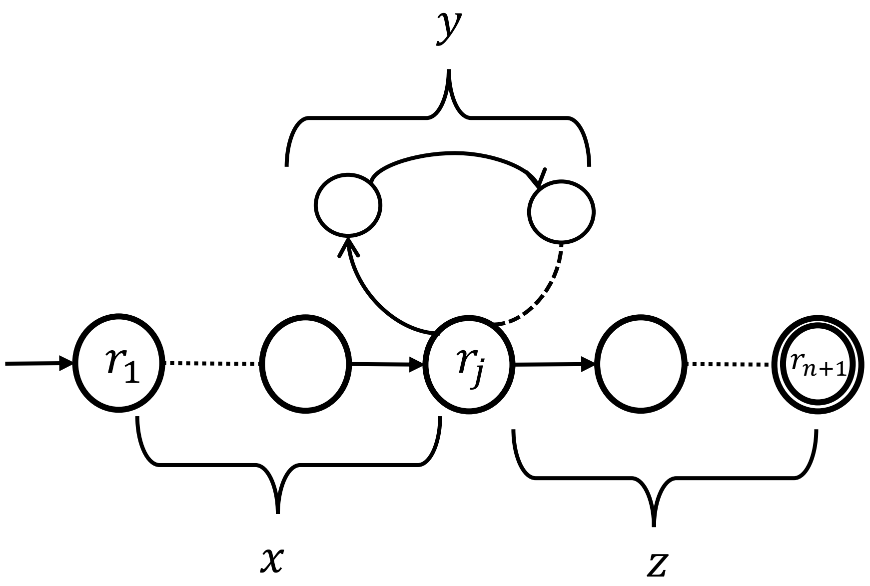
This is because of the ***pigeonhole principle***: any such run would encounter $p+1$ states, but there are p distinct states in the DFA.

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



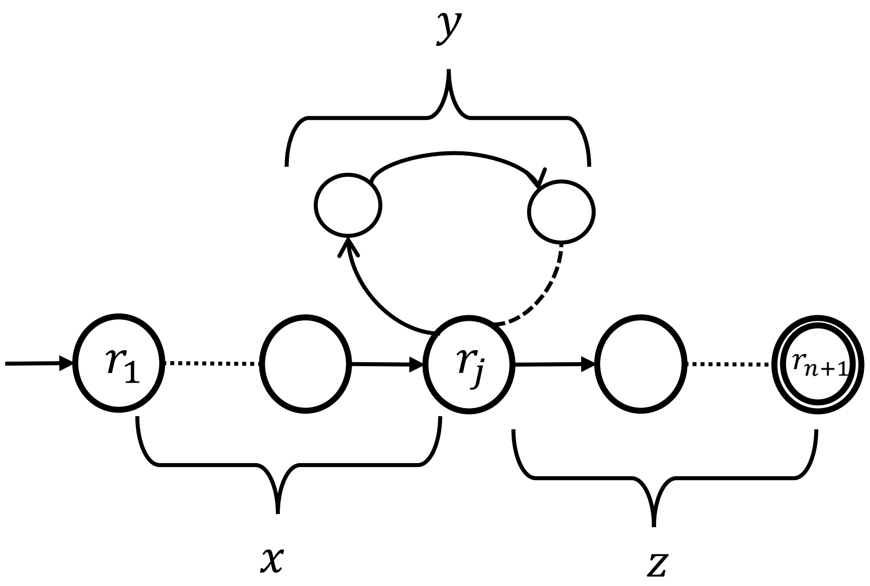
- We can traverse the loop bit any number of times and so $\forall i \geq 0, xy^i z \in L$.

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

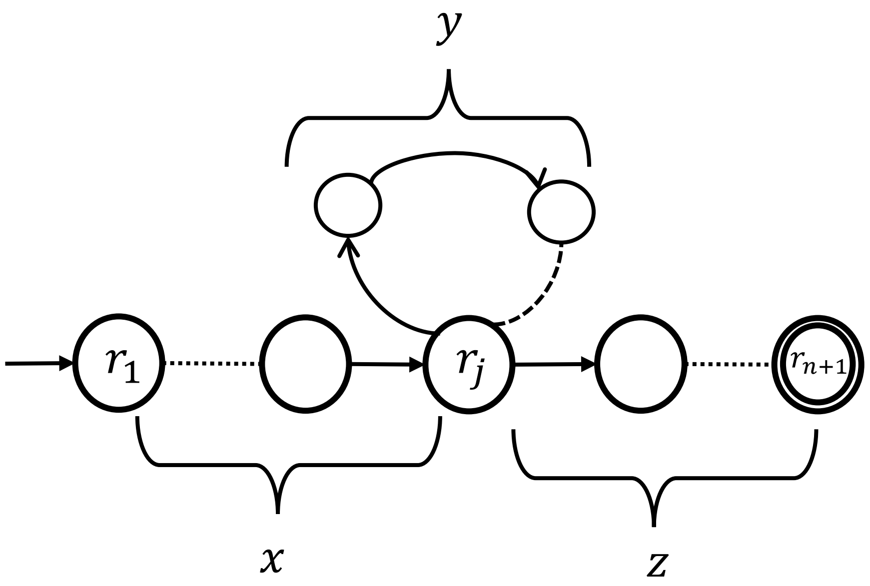
This is because of the ***pigeonhole principle***: any such run would encounter $p + 1$ states, but there are p distinct states in the DFA.

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, $x = s_1 \ldots s_{j-1}$, $y = s_j \ldots . s_{l-1}$, $z = s_l \ldots s_n$. 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 $\forall i \geq 0, xy^iz \in L$.
- Also, as $j \neq l$, $|y| \geq 1$, and
- The DFA reads $|xy|$ by then and so $|xy| \leq p$.

Pumping Lemma

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

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

Examples of languages that are NOT regular:

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

⋮

⋮

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**)

Thank You!