Chapter 3 – Regular Languages And Regular Grammar

Outline

- Operations on strings and languages
- Regular Expression and Equivalence to FA
- Regular Grammar
- Language of Regular Grammar

Regular Languages

- A language is **regular** if there exists a finite automata for it.
- The languages which are not accepted by a finite automata are called non-regular languages.
- All languages are either regular or non-regular, not both.
- Examples
  - The languages whose words are in **arithmetic progression and need no comparison** are regular: $L=\{a^{2n}, n>=1\}$ words are aa, aaaa, aaaaaa, ...
  - The languages whose words need some sort of **comparison** are not regular: $L=\{a^n b^n, n>=0\}$ here number

Operations on strings

- Given two strings $s = a_1 \ldots a_n$ and $t = b_1 \ldots b_m$, we define their <u>concatenation</u> $st = a_1 \ldots a_n b_1 \ldots b_m$

  s = abb, t = cba st = abbcba

- We define s^n as the concatenation ss...s *n* times

  $s = 011$ $s^3 = 011011011$

Operations on languages

- The concatenation of languages L_1 and L_2 is

$$L_1L_2 = \{\text{st: s} \in L_1, \text{t} \in L_2\}$$

- Similarly, we write L^n for $LL\ldots L$ (n times)
- The union of languages $L_1 \cup L_2$ is the set of all strings that are in L_1 or in L_2

- Example: $L_1 = \{01, 0\}$, $L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$.
  What is L_1L_2 and $L_1 \cup L_2$?

Operations on languages

- The star (Kleene closure) of L are all strings made up of zero or more chunks from L:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$$

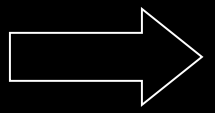
  - This is always infinite, and always contains ε

- Example: $L_1 = \{01, 0\}$, $L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$.
  What is $L_1{}^*$ and $L_2{}^*$?

Constructing languages with operations

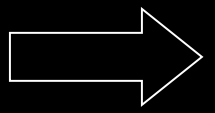
- Let’s fix an alphabet, say $\Sigma = \{0, 1\}$
- We can construct languages by starting with simple ones, like $\{0\}$, $\{1\}$ and combining them

$\{0\}(\{0\}\cup\{1\})^*$
all strings that start with 0



$0(0+1)^*$

$(\{0\}\{1\}^*)\cup(\{1\}\{0\}^*)$



01^*+10^*

Regular expressions

- *Regular expressions* are an algebraic way to describe languages.
- They describe exactly the regular languages.
- If E is a regular expression, then L(E) is the language it defines.
- We'll describe RE's and their languages recursively.

A language is regular if it is represented by a regular expression

RE's: Definition

- Basis 1: If *a* is any symbol, then **a** is a RE, and L(**a**) = {a}.
  - Note: {a} is the language containing one string, and that string is of length 1.
- Basis 2: ε is a RE, and L(ε) = {ε} => The language that contains just the empty string
- Basis 3: ∅ is a RE, and L(∅) = ∅ => The language that contains no strings

RE's: Definition – (2)

- Induction 1: If E_1 and E_2 are regular expressions, then E_1+E_2 is a regular expression, and

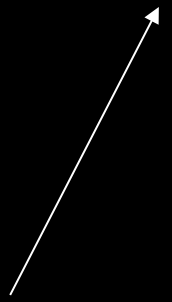
  $$L(E_1+E_2) = L(E_1) \cup L(E_2).$$

- Induction 2: If E_1 and E_2 are regular expressions, then E_1E_2 is a regular expression, and $L(E_1E_2) = L(E_1)L(E_2)$.

Concatenation : the set of strings wx such that w is in $L(E_1)$ and x is in $L(E_2)$.

RE's: Definition – (3)

- Induction 3: If E is a RE, then E* is a RE, and
  L(E*) = (L(E))*.



Closure, or "Kleene closure" = set of strings $w_1w_2...w_n$, for some $n \geq 0$, where each w_i is in L(E).
Note: when n=0, the string is ε.

Precedence of Operators

- Parentheses may be used wherever needed to influence the grouping of operators.
- Order of precedence is:
  - * (Kleene closure)….[highest priority],
  - then concatenation,
  - then + (Union) ….[lowest].

Example of RE and corresponding RL

- **(a + b)²** corresponds to the language {aa, ab, ba, bb}, that is the set of strings of length 2 over the alphabet {a, b}.
- In general **(a + b)^k** corresponds to the set of strings of length k over the alphabet {a, b}.
- **(a + b)*** corresponds to the set of all strings over the alphabet {a, b}.
- **a*b*** corresponds to the set of strings consisting of zero or more a's followed by zero or more b's.
- **a*b$^+$a*** corresponds to the set of strings consisting of zero or more a's followed by one or more b's followed by zero or more a's.

Example of RE and corresponding RL

- **(ab)**$^+$ corresponds to the language {ab, abab, ababab, ... }, that is, the set of strings of repeated ab's.

- **Note:** A regular expression is not unique for a language. That is, a regular language, in general, corresponds to more than one regular expressions.

- For example **(a + b)*** and **(a*****b*** **)*** correspond to the set of all strings over the alphabet {a, b}. (Reading: CHECK!)

Examples: RE's

- (**01**) = {01}.
- (**01**+**0**) = {01, 0}.
- (**0**(**1**+**0**)) = {01, 00}.
  - Note order of precedence of operators.
- (**0***) = {ε, 0, 00, 000,... }.
- ((**0**+**10**)***1**) = all strings of 0's and 1's without two consecutive 1's.
- 01* = {0, 01, 011, 0111, ...}
- (01*)(01) = {001, 0101, 01101, 011101, ...}

Examples

- Construct a RE over $\Sigma = \{0,1\}$ that represents
  - All strings that have two consecutive 0s.

    (0+1)*00(0+1)*
  - All strings except those with two consecutive 0s.

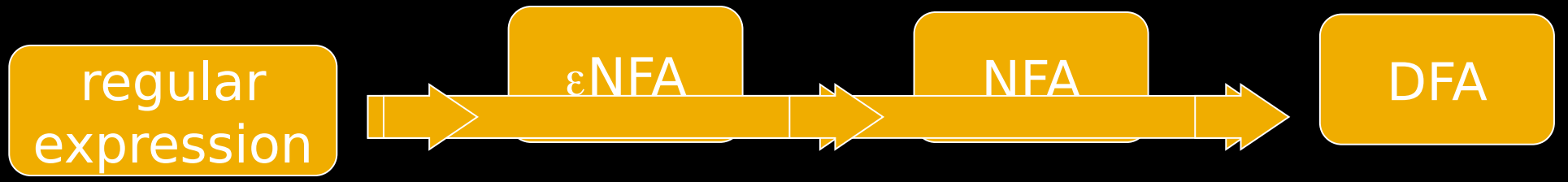
    (1*01*01*)*1*+(1*01*01*)*1*0
  - All strings with an even number of 0s.

Examples

- The set of strings over {0,1} that end in 3 consecutive 1's.
  (0 + 1)* 111
- The set of strings over {0,1} that have at least one 1.
  0* 1 (0 + 1)*
- The set of strings over {0,1} that have at most one 1.
  0* + 0* 1 0*
- All strings over {0,1} that start and end with the same digit
  0(0+1)*0 + 1(0+1)*1 + 0 + 1

RE and DFA

- For every regular expression, we have to give a DFA for the same language



- For every DFA, we give a regular expression for the same language

Constructing a FA from a RE

- If the operand is a character c, then our FA has two states, q0 (the start state) and qf (the final, accepting state), and a transition from q0 to qf with label c.
- If the operand is epsilon, then our FA has two states, q0 (the start state) and qf (the final state), and an epsilon transition from q0 to qf.
- If the operand is null, then our FA has two states, q0 (the start state) and qf (the final state), and no transitions.

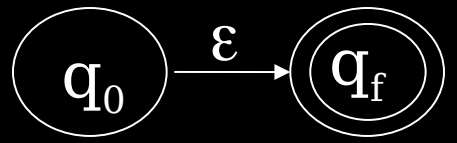
General method

regular expr ⇨ εNFA

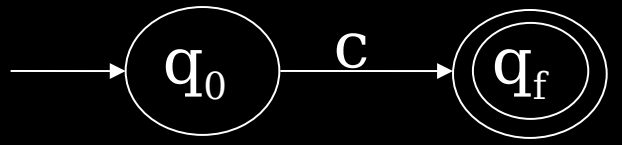
∅



ε



symbol c



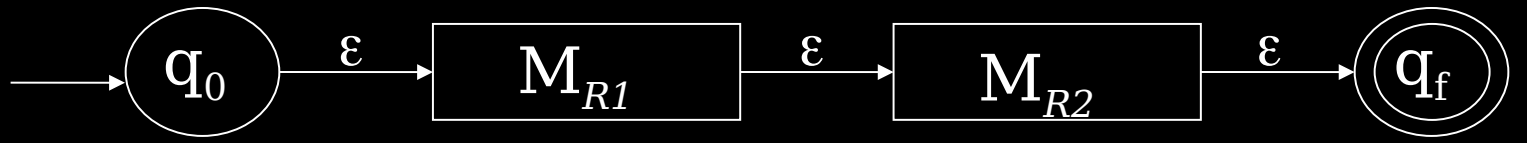
Constructing a FA from a RE

- Given FA for R1 and R2, we now see how to build a FA for R1R2, R1+R2, and R1*.
  - Let A (with start state a0 and final state aF) be the machine accepting L(R1) and
  - B (with start state b0 and final state bF) be the machine accepting L(R2).
- The machine C accepting L(R1R2) includes A and B, with start state a0, final state bF, and an epsilon transition from aF to b0.
- The machine C accepting L(R1+R2) includes A and B, with a new start state c0, a new final state cF, and epsilon transitions from c0 to a0 and b0, and from aF and bF to cF.
- The machine C accepting L(R1*) includes A, with a new start state c0, a new final state cF, and epsilon transitions from c0 to a0 and cF, and from aF to a0, and from aF to cF.

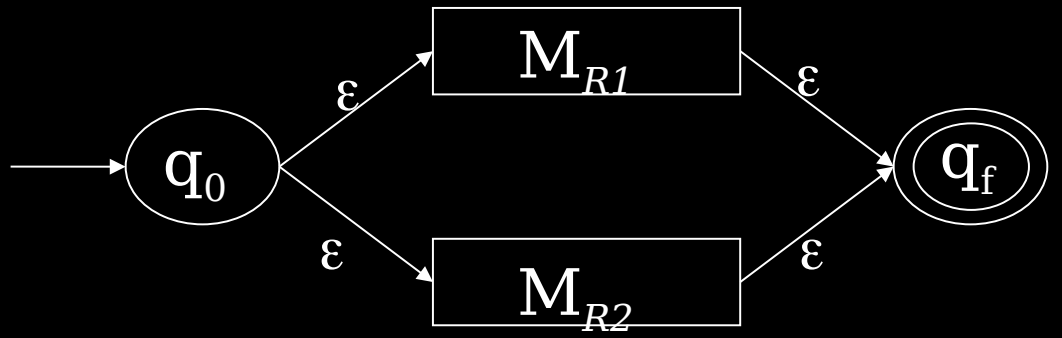
General method continued

regular expr ⇨ εNFA

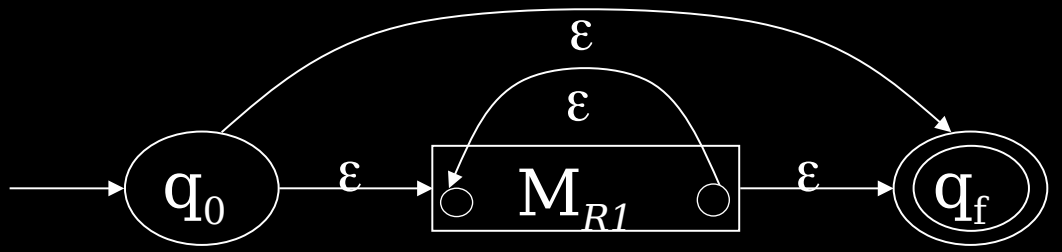
R1R2



R1 + R2

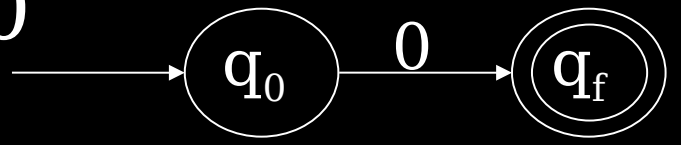


*R1**

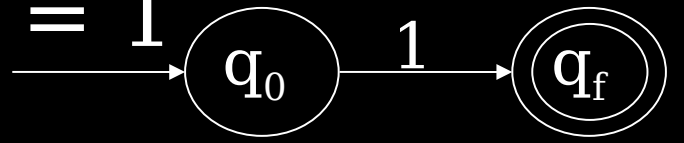


Examples: regular expression → εNFA

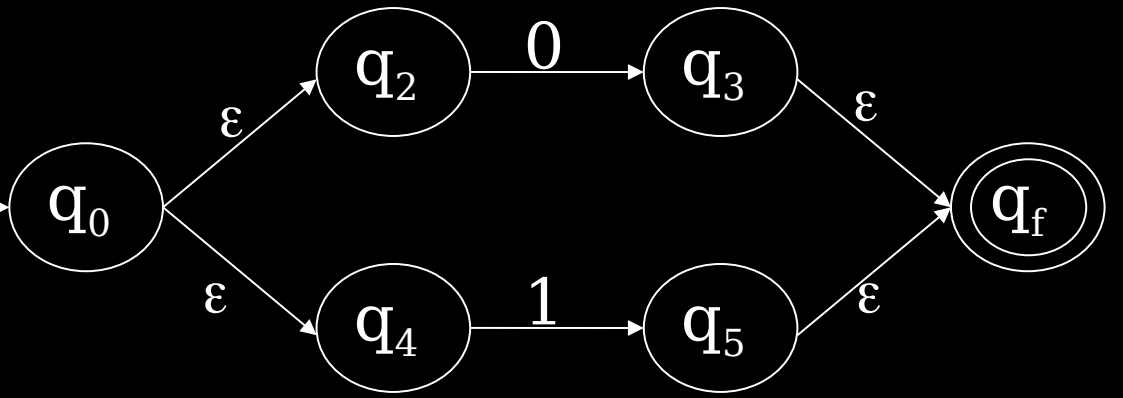
- $R_1 = 0$



$R_2 = 1$

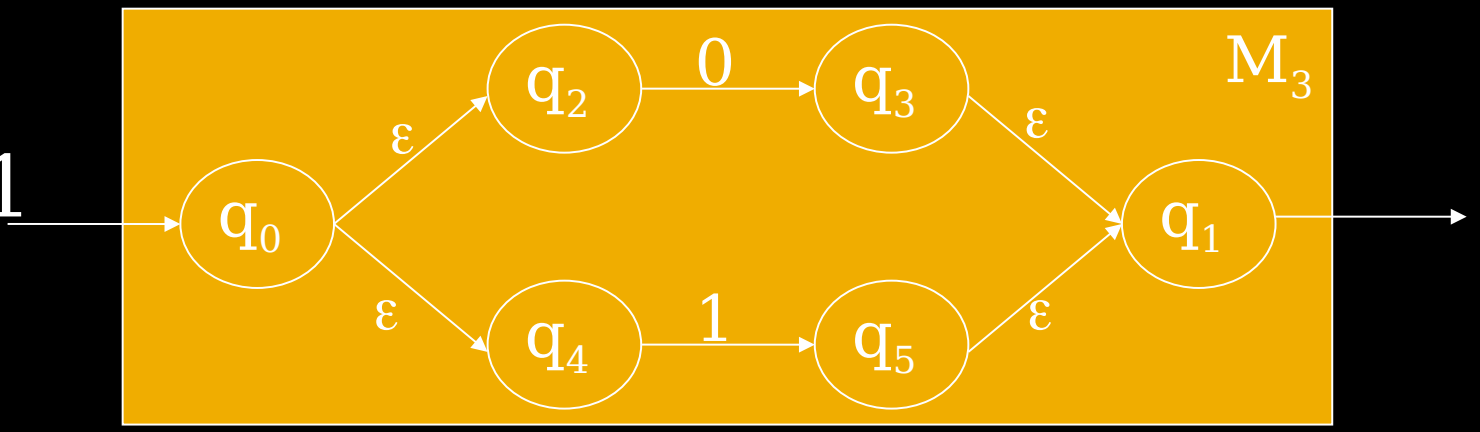


- $R_3 = 0 + 1$
  $= R_1 + R_2$

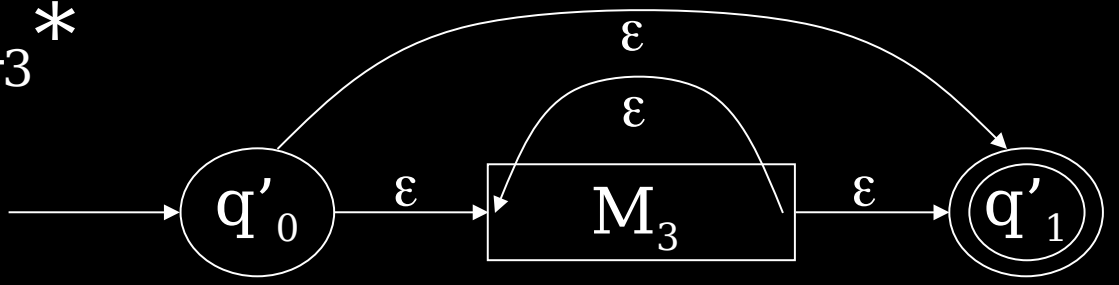


Examples: regular expression → εNFA

- $R_3 = 0 + 1$



- $R_4 = (0 + 1)^*$
  $= R_3^*$



Convention

- When we draw a box around an εNFA:
  - The arrow going in points to the start state
  - The arrow going out represents all transitions going out of accepting states
  - None of the states inside the box are accepting states.
  - The labels of the states inside the box are distinct from all other states in the diagram

Converting FA to RE

- Given a DFA F, we construct a regular expression R such that L(F) == L(R).
- We preprocess the FA, turning the labels on transitions into regular expressions.
- If there is a transition with label {a,b}, then we replace the label with the regular expression a + b.
- For each accepting state qf in F, eliminate all states in F except the start state q0 and qf.

Converting FA to RE

- To eliminate a state qE, consider all pairs of states qA and qB such that there is a transition from qA to qE with label R1, a transition from qE to qE with label R2 (possibly null, meaning no transition), and a transition from qE to qB with label R3. Introduce a transition from qA to qB with label R1R2*R3.

- If there is already a transition from qA to qB with label R4, then replace that label with

Converting FA to RE

- After eliminating all states except q0 and qf:
- If q0 == qf, then the resulting regular expression is R1*, where R1 is the label on the transition from q0 to q0.
- If q0 != qf, then assume the transition from q0 to q0 is labeled R1, the transition from q0 to qf is labeled R2, the transition from qf to qf is labeled R3, and the transition from qf to q0 is labeled R4. The resulting regular

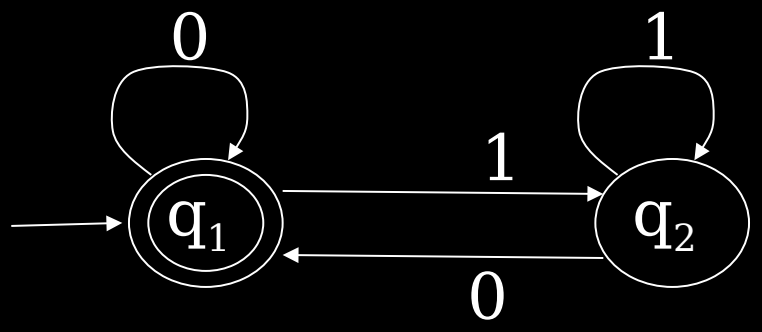
Converting FA to RE

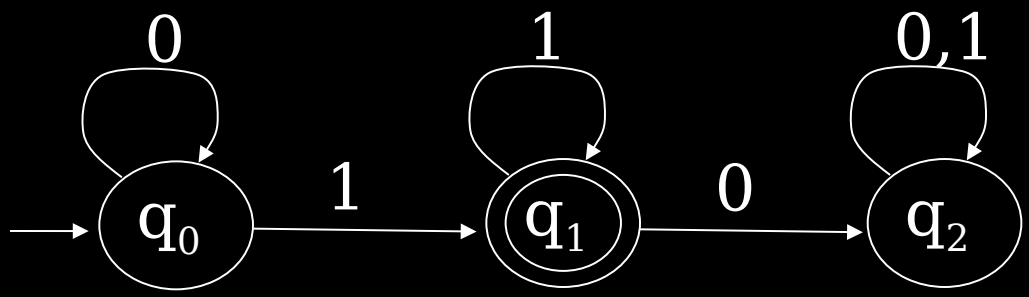
- Let RF_i be the regular expression produced by eliminating all the states except q0 and qfi, and

- If there are n final states in the DFA, then the regular expression that generates the strings accepted by the original DFA is RF1 + RF2 + ... RFn.

Example

- Construct regular expression for these FA:



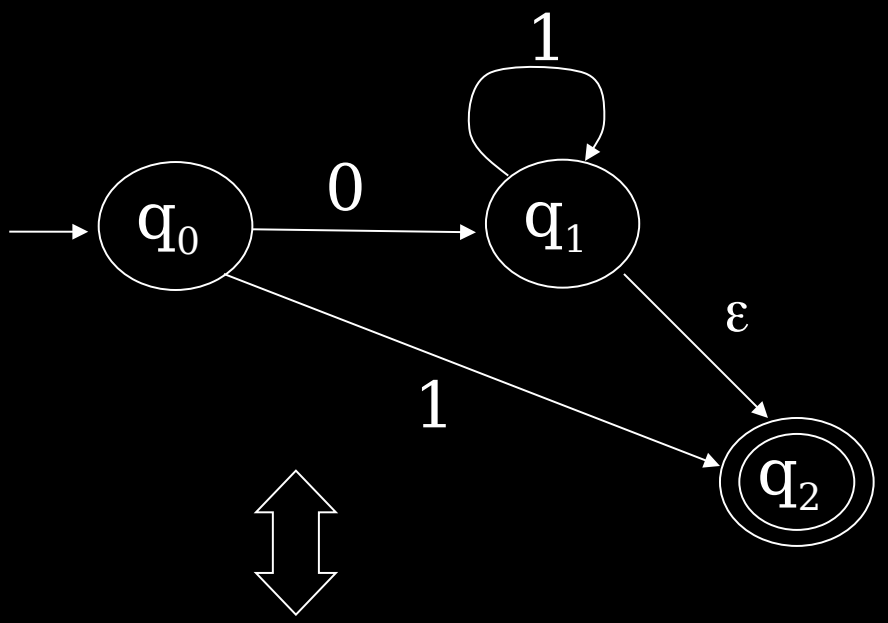


$\varepsilon+0^*+(0^*11^*00^*)^*$

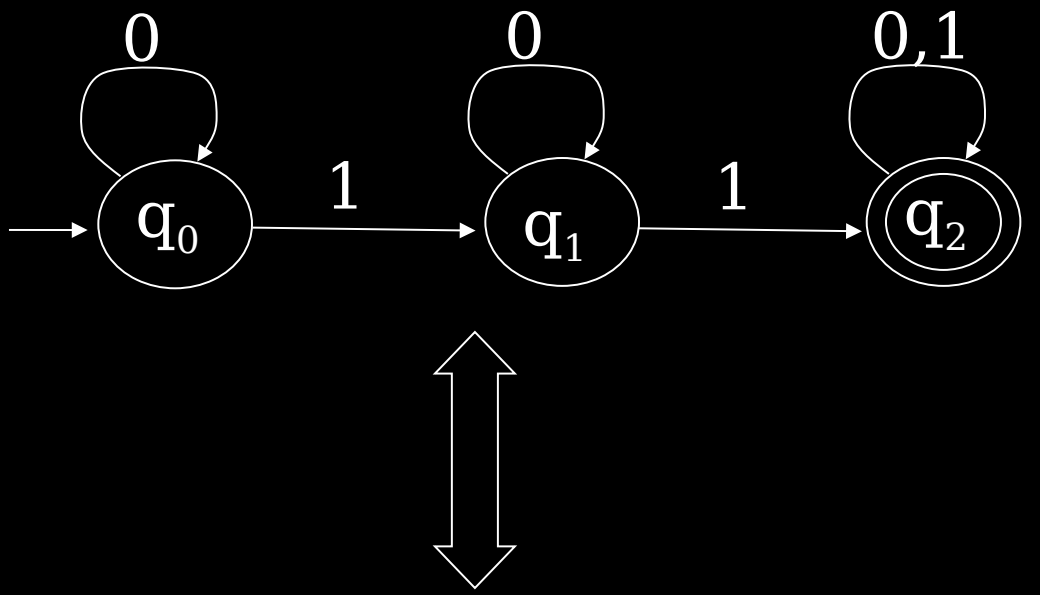
0^*11^*

Example

- Construct regular expression for these FA:



01*+1

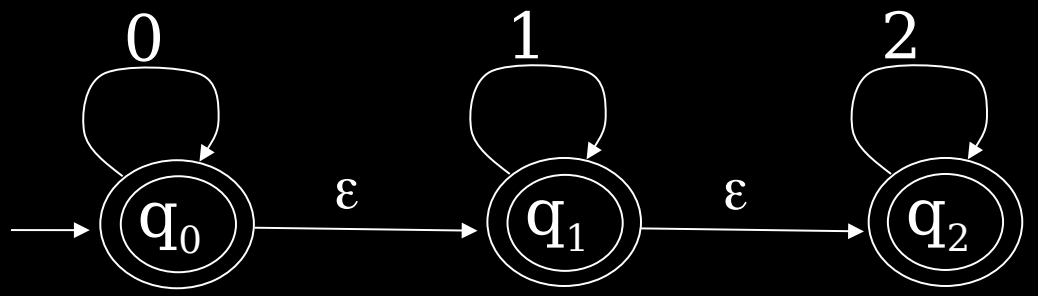


0*10*1(0+1) *

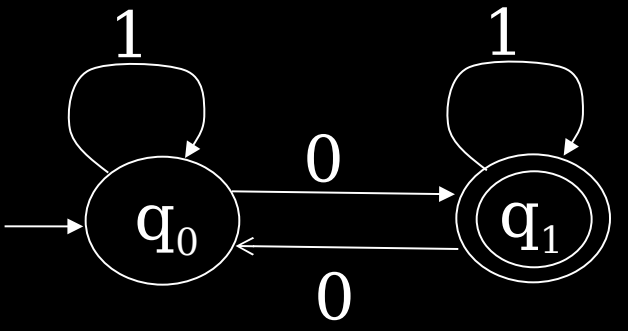
Example

- Construct FA for these regular expressions :

$0^*1^*2^*$

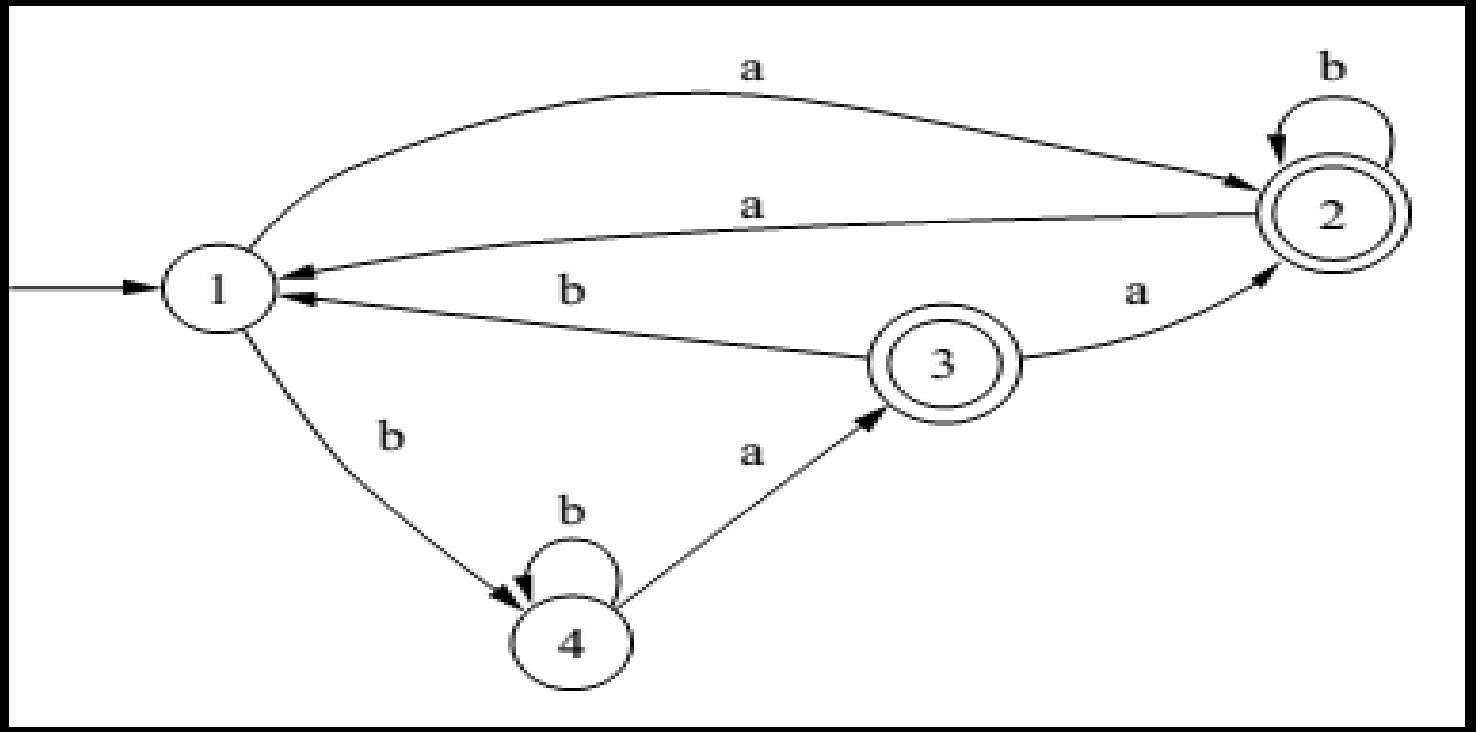


$1^*0(1^*+01^*0)^*$



Homework

Construct RE for the following Finite Automata



Regular Grammars

Grammars

For natural languages, each has its own grammar. An Amharic sentence follows Amharic language grammar.

For programming languages, A C++ program has to obey the C++ programming grammar.

A regular language can be **accepted** by a finite state automaton and **denoted** by a regular expression.

In this section, we shall show that a regular language can be **generated** by a regular grammar.

Grammars

- Grammars express languages
- Example: The English language

$$\langle sentence \rangle \rightarrow \langle noun_phrase \rangle \langle predicat$$

$$\langle noun_phrase \rangle \rightarrow \langle article \rangle \langle noun \rangle$$

$$\langle predicate \rangle \rightarrow \langle verb \rangle$$

- $\langle article \rangle \rightarrow a$
- $\langle article \rangle \rightarrow the$
- $\langle noun \rangle \rightarrow boy$
- $\langle noun \rangle \rightarrow dog$
- $\langle verb \rangle \rightarrow runs$
- $\langle verb \rangle \rightarrow walk$

- A derivation of “the boy walks”:

$$\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun_phrase \rangle \langle predicat \\
&\Rightarrow \langle noun_phrase \rangle \langle verb \rangle \\
&\Rightarrow \langle article \rangle \langle noun \rangle \langle verb \rangle \\
&\Rightarrow the \langle noun \rangle \langle verb \rangle \\
&\Rightarrow the\ boy \langle verb \rangle \\
&\Rightarrow the\ boy\ walks
\end{aligned}$$

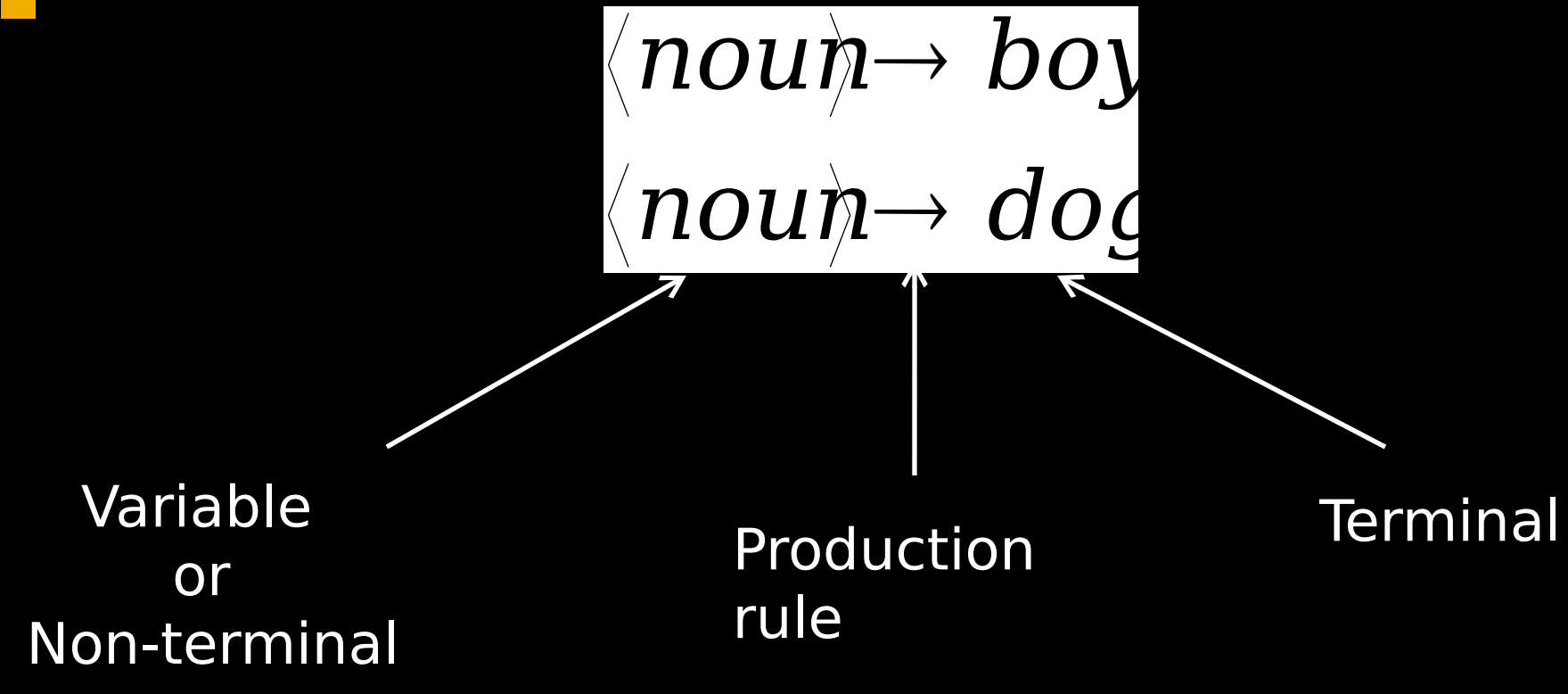
- A derivation of “a dog runs”:

$$\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun_phrase \rangle \langle predicat \\
&\Rightarrow \langle noun_phrase \rangle \langle verb \rangle \\
&\Rightarrow \langle article \rangle \langle noun \rangle \langle verb \rangle \\
&\Rightarrow a \ \langle noun \rangle \langle verb \rangle \\
&\Rightarrow a \ dog \langle verb \rangle \\
&\Rightarrow a \ dog \ runs
\end{aligned}$$

- Language of the grammar:

L = { “a boy runs”,
“a boy walks”,
“the boy runs”,
“the boy walks”,
“a dog runs”,
“a dog walks”,
“the dog runs”,
“the dog walks”
}

Notation

- 
⟨noun⟩→ boy
⟨noun⟩→ dog
Variable or Non-terminal
Production rule
Terminal

Another Example

- Grammar:

$$S \to aSb$$
$$S \to \varepsilon$$

- Derivation of sentence ab :

$$S \Rightarrow aSb \Rightarrow ab$$

$S \to aSb$ $\qquad$ $S \to \varepsilon$

- Grammar:

$$S \rightarrow aSb$$
$$S \rightarrow \varepsilon$$

- Derivation of sentence : $aabb$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

$S \rightarrow aSb$ $\qquad$ $S \rightarrow \varepsilon$

- Other derivations:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$$
$$\Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$

Grammars

Definition: A **grammar** G = (V, T, P, S) is defined as follows.

(1) V is a finite set of variables. S is the start symbol in V.

(2) T is a finite set of terminals, and $T \cap V = \varnothing$.

(3) P is a finite set of productions or rewriting rules. Each production is of the form:

$A \rightarrow aB$, where $A, B \in V$ and $a \in T$,

or $A \rightarrow a$, where $A \in V$ and $a \in T$.

A Non-Linear Grammar

Has the production rules like:

$$S \rightarrow SS$$

$$S \rightarrow \varepsilon$$

$$S \rightarrow aSb$$

$$S \rightarrow bSa$$

Linear Grammars

Grammars with at most one variable at the right side of a production

Examples:

$$S \rightarrow aSb$$

$$S \rightarrow \varepsilon$$

$$S \rightarrow Ab$$

$$A \rightarrow aAb$$

$$A \rightarrow \varepsilon$$

Another Linear Grammar

Grammar:

$$G \qquad S \rightarrow A$$

$$A \rightarrow aB \mid \varepsilon$$

$$B \rightarrow Ab$$

Right-Linear Grammars (RLG)

All productions have the form:

$$A \rightarrow xB \qquad A \rightarrow x$$

Example:

$$S \rightarrow abS$$

$$S \rightarrow a$$

Left-Linear Grammars (LLG)

All productions have the form:

$$A \rightarrow Bx \qquad A \rightarrow x$$

Example:

$$S \rightarrow Aab$$

$$A \rightarrow Aab \mid B$$

$$B \rightarrow a$$

Regular Grammars

A **regular grammar** is any right-linear or left-linear grammar

Examples:

G_1

$S \to abS$

$S \to a$

G_2

$S \to Aab$

$A \to Aab \mid B$

$B \to a$

Language of a Regular Grammar

The language generated by a regular grammar

G = (V, T, P, S) is denoted by L(G).

Example: Find L(G) for G = (V, T, P, S), where V = {S, A, B}, T = {0, 1} and P contains the following productions:

$S \rightarrow 0A \mid 1B \mid 1$

$A \rightarrow 0S \mid 1B \mid 1$

$B \rightarrow 0B \mid 1A \mid 0$

Language of a Regular Grammar

Solution: The set generated by a regular grammar G is Produced by applying the production rules until all the variables are reduced to terminals.

P: **S** → 0A | 1B | 1, **A** → 0S | 1B | 1, **B** → 0B | 1A | 0

S → 0**A**
→ 01**B**
→ 010B
→
0101**A**
→
<u>01011</u>

S → 1B
→ 10B
→ <u>100</u>

S → 0A
→ <u>01</u>

{ω ω has odd number of 1's}

Example

S → aS | bA
A → cA | ε

This grammar produces the language
produced by the regular expression a^*bc^*

S → aS → aaS → … → a…aS →a…abA →a…abcA
→ a…abccA → … → a…abc…c

Reg. Grammars Vs Reg. Languages

Right Linear Grammars are equivalent to εNFA
Two directions:

1. Given a Right Linear grammar, we can construct an εNFA that recognizes the same language with the Right Linear grammar.

2. Given an εNFA, we can construct a Right Linear grammar that describes the same language with the εNFA.

1. RLG→ εNFA

Suppose that we have a right linear grammar (V, Σ, P, S). construct an εNFA (Q, Σ, δ, S, {f}).

- The set of states Q will be the set VU{f}, where f is a new symbol denoting the final state
- Productions in P have three possible forms:
  - A → ε : add the transition δ(A,ε) = f
  - A → a : add the transition δ(A,a) = f
  - A → aB : add the transition δ(A,a) = B

2. εNFA → RLG

Suppose that We have an εNFA (Q, Σ, δ, q_0, F,). construct a right linear grammar (Q, Σ, P, q_0).

- For each transition $\delta(q_i, a) = q_j$, construct the rule $q_i \rightarrow aq_j$ in P.

- Furthermore, for every state q_i in F, add the rule $q_i \rightarrow \varepsilon$ in P.

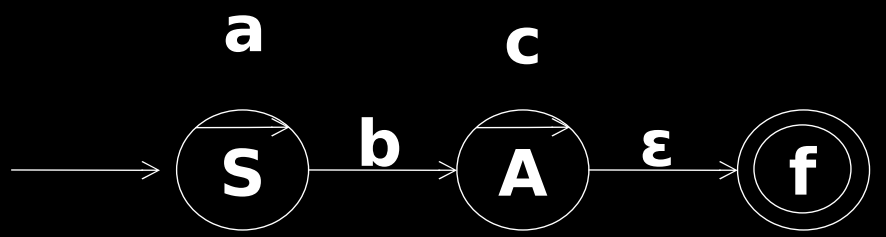
Examples

1) Transform the following Right Linear grammar into an equivalent εNFA.

S → aS |bA

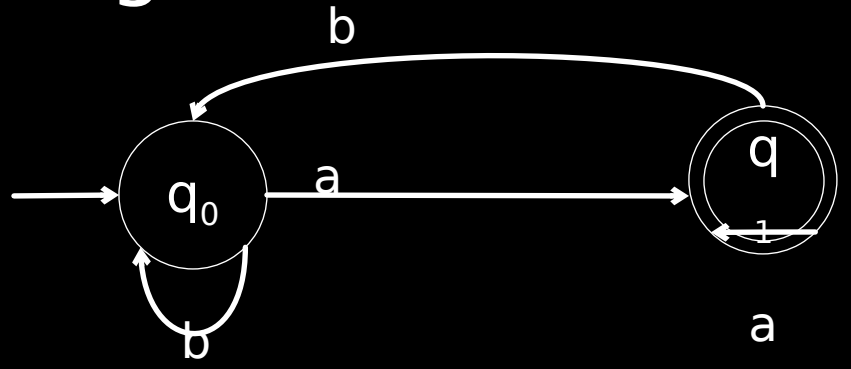
A → cA | ε

Solution:



Examples

2) Transform the following DFA to a right linear grammar



Solution:

$Q_0 \rightarrow aQ_1 \mid bQ_0$

$Q_1 \rightarrow aQ_1 \mid bQ_0 \mid \varepsilon$

Left Linear Grammars

- Left Linear Grammars also produce the Regular Languages but not straightforward.
- Actually, a Left Linear grammar produces the reverse of the language produced by the Right Linear grammar in which we reversed the rules:
- A → Ba to A →aB.

Example

C → Bc
B → Ab
A → a
The derivation of abc is:
  C → Bc → Abc → abc
The Right Regular grammar with the rules of the form A → Ba reversed is
C → cB
B → bA
A → a
and it produces the reverse language.

C → cB → cbA → cba and then take the reverse.

Example (continued)

- So, just create the εNFA for the language produced by the Right Regular grammar and then compute the reverse (change start with final state and reverse the arrows).
- This results in an εNFA for the Left Regular grammar.

Note:

(1)A DFA can **recognize** a regular language.

(2)A regular expression can **represent** a regular language.

(3)A regular grammar can **generate** a regular language.

Exercise

Given the following English grammar.

S →NP VP
VP → V NP
NP → NAME
NP → DET N
NAME → Abebe
V → killed
DET → the
N → lion

Answer:

S ⇒NP VP
⇒ NAME VP
⇒ Abebe VP
⇒ Abebe V NP
⇒ Abebe killed NP
⇒ Abebe killed DET N
⇒ Abebe killed the N
⇒ Abebe killed the lion

Show the generaiton of the sentence
Abebe killed the lion

Home work

1. Create a regular grammar for the language of strings over {a, b} which contains the substring aa.

2. Create NFA's for the following languages and construct the corresponding regular grammars. Use S as the first non-terminal.
   (a) (1+0)*01*

   (b) {w| w ∈ {0,1}* and w doesn't end in 01}

Home work

3. Construct the equivalent DFA for the grammar having the following productions:
   A →0A|1B
   B →0C|1B
   C →0B|1D| ε
   D →0D|1D

4. Construct the RLG and LLG for the regular expression: a(a+b)*ab.

   Hint: for LLG, reverse the regular expression, find its RLG, then change the terminals with variables in the right side of P.

Home work

- State and Prove the Pumping Lemma?
  - Why we use it for?
  - How do we use it?