Regular Expressions — FA

Notation

language is regular

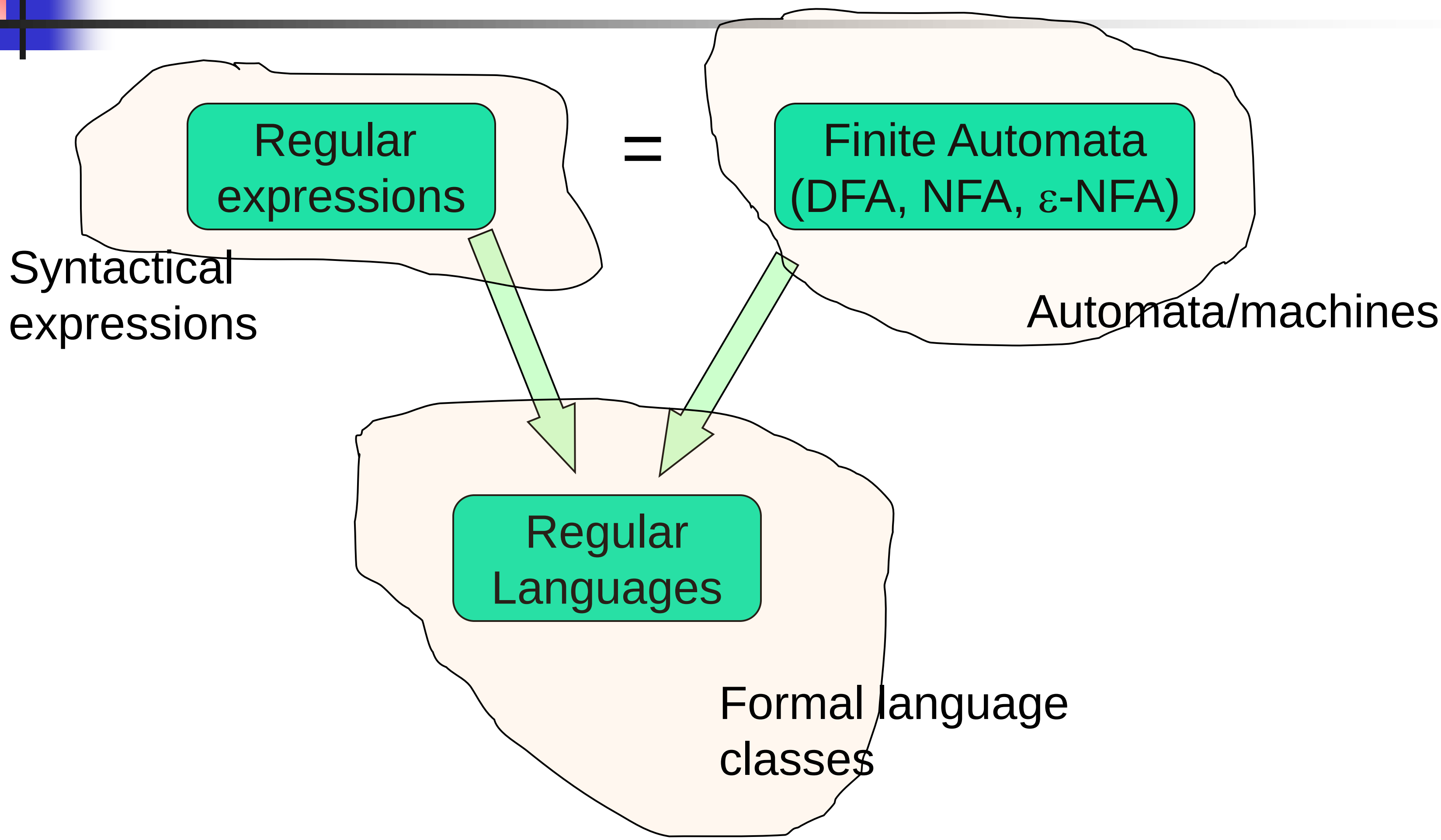
Regular Expressions vs. Finite Automata

- Offers a declarative way to express the pattern of any string we want to accept
  - E.g., 011, 01*, 10*

- Automata => more machine-like

  < input: string , output: [accept/reject] >
- Regular expressions => more program syntax-like
- Unix environments heavily use regular expressions
  - E.g., bash shell, grep, vi & other editors, sed
- Perl scripting – good for string processing
- Lexical analyzers such as Lex or Flex

Regular Expressions



Language Operators

- <u>Union</u> of two languages:
  - **L U M** = all strings that are either in L or M
  - <u>Note:</u> A union of two languages produces a third language

- <u>Concatenation</u> of two languages:
  - **L . M** = all strings that are of the form *xy* s.t., $x \in L$ and $y \in M$
  - The *dot* operator is usually omitted
    - i.e., **LM** is same as L.M

"i" here refers to how many strings to concatenate from the parent language L to produce strings in the language L^i

Kleene Closure (the * operator)

- Kleene Closure of a given language L:
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{w \mid \text{for some } w \in L\}$
  - $L^2 = \{ w_1w_2 \mid w_1 \in L, w_2 \in L \text{ (duplicates allowed)}\}$
  - $L^i = \{ w_1w_2\ldots w_i \mid \text{all } w\text{'s chosen are} \in L \text{ (duplicates allowed)}\}$
  - (Note: the choice of each w_i is independent)
  - $\mathbf{L^*} = \bigcup_{i\geq 0} L^i$ (arbitrary number of concatenations)

Example:

- Let L = { 1, 00}
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{1,00\}$
  - $L^2 = \{11,100,001,0000\}$
  - $L^3 = \{111,1100,1001,10000,000000,00001,00100,0011\}$
  - $\mathbf{L^*} = L^0 \cup L^1 \cup L^2 \cup \ldots$

Building Regular Expressions

- Let E be a regular expression and the language represented by E is L(E)
- Then:
  - (E) = E
  - L(E + F) = L(E) U L(F)
  - L(E F) = L(E) L(F)
  - L(E*) = (L(E))*

Example: how to use these regular expression properties and language operators?

- *L = { w | w is a binary string which does not contain two consecutive 0s or two consecutive 1s anywhere)*
  - E.g., w = 01010101 is in L, while w = 10010 is not in L
- Goal: Build a regular expression for L
- Four cases for w:
  - Case A: w starts with 0 and |w| is even   0101
  - Case B: w starts with 1 and |w| is even   1010
  - Case C: w starts with 0 and |w| is odd   010
  - Case D: w starts with 1 and |w| is odd   101
- Regular expression for the four cases:
  - Case A: (01)*
  - Case B: (10)*
  - Case C: 0(10)*
  - Case D: 1(01)*
- Since L is the union of all 4 cases:
  - Reg Exp for L = (01)* + (10)* + 0(10)* + 1(01)*
-

(01)* | (10)* | 0(10)* | 1(01)*

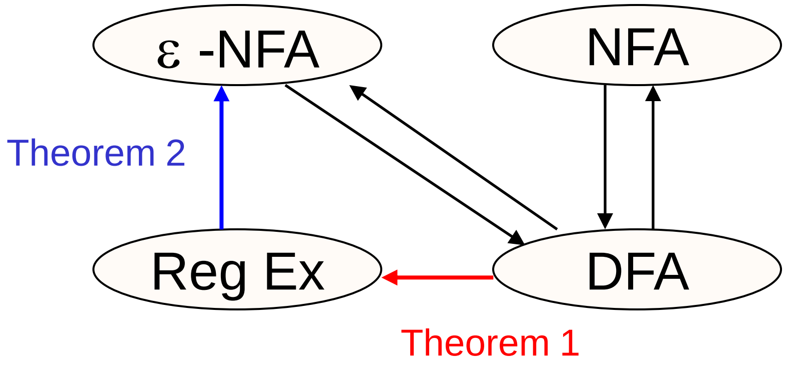
Precedence of Operators in RE

- Highest to lowest
  - * operator (star)
  - .   (concatenation)
  - + operator

- Example:
  - 01* + 1   =   (0 . ((1)*)) + 1

Finite Automata (FA) & Regular Expressions (Reg Ex)

- To show that they are interchangeable, consider the following theorems:
  - *Theorem 1: For every DFA A there exists a regular expression R such that L(R)=L(A)*
  - *Theorem 2: For every regular expression R there exists an ε -NFA E such that L(E)=L(R)*



Kleene Theorem

Question -1

- Write the proof of theorem 1 and theorem 2.

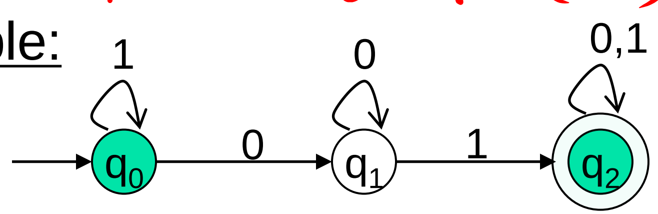
DFA to RE construction

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

Informally, trace all distinct paths (traversing cycles only once) from the start state to *each of the* final states and enumerate all the expressions along the way

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

<u>Example:</u>



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

$\underbrace{(1^*)}_{1^*}\ \underbrace{0\ (0^*)}_{00^*}\ \underbrace{1}_{1}\ \underbrace{(0+1)^*}_{(0+1)^*}$

$1^*00^*1(0+1)^*$ → substring 01

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

Question -2

- Write the language given by the previous example.
- Construct NFA for the expression

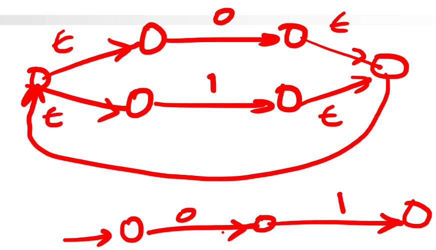
$$(0/1)^*01(0/1)^*$$

Reg Ex $\xrightarrow{\text{Theorem 2}}$ ε -NFA

RE to ε-NFA construction

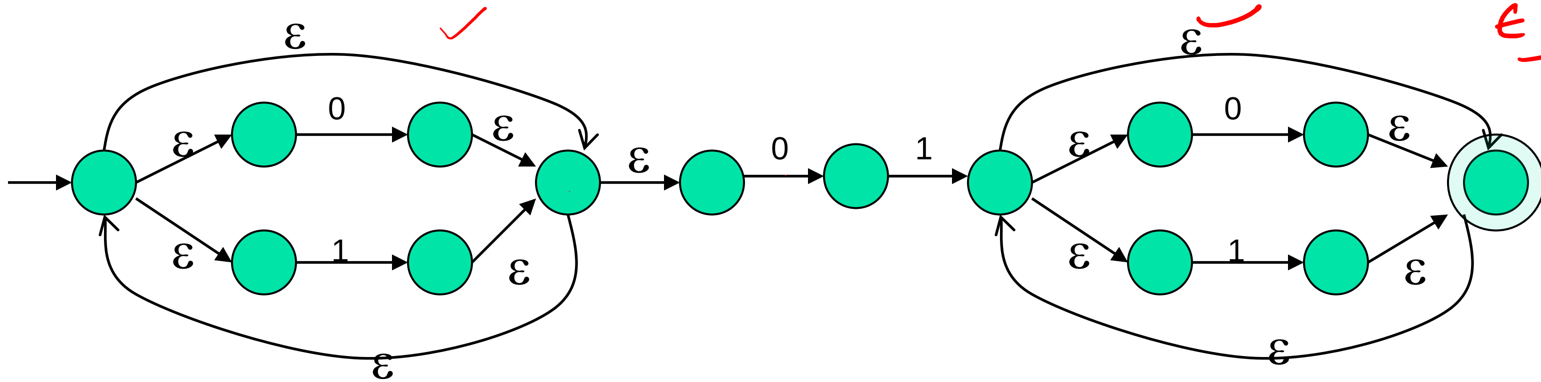
Example: (0/1)*01(0/1)* or
(0+1)*01(0+1)*

01
001
0001
101
01010



(0+1)* 01 (0+1)*

ε NFA



Algebraic Laws of Regular Expressions

- Commutative:
  - $E+F = F+E$
- Associative:
  - $(E+F)+G = E+(F+G)$
  - $(EF)G = E(FG)$
- Identity:
  - $E+\Phi = E$
  - $\varepsilon E = E \varepsilon = E$
- Annihilator:
  - $\Phi E = E\Phi = \Phi$

Algebraic Laws…

- Distributive:
  - E(F+G) = EF + EG
  - (F+G)E = FE+GE
- Idempotent: E + E = E
- Involving Kleene closures:
  - $(E^*)^* = E^*$
  - $\Phi^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $E^+ = EE^*$
  - $E? = \varepsilon + E$

1. $((R^*)^*)^* = R^*$?
2. $(R+S)^* = R^* + S^*$?
3. $(RS + R)^* RS = (RR^*S)^*$

True or False?

Let R and S be two regular expressions. Then:

1. $((R^*)^*)^* = R^*$?
2. $(R+S)^* = R^* + S^*$?
3. $(RS + R)^* RS = (RR^*S)^*$?

RE → ENFA → DFA → minimized DFA

- **Examples:** Let Σ = {0, 1}

RE	Language
(0 + 1)* — ε NFA	All strings of 0's and 1's
01*	0 followed by any number 1's
0(0 + 1)*	All strings of 0's and 1's, beginning with a 0
(0 + 1)*1	All strings of 0's and 1's, ending with a 1
(0 + 1)*0(0 + 1)*	All strings of 0's and 1's containing at least one 0
(0 + 1)*0(0 + 1)*0(0 + 1)*	All strings of 0's and 1's containing at least two 0's
01010 010001 (0 + 1)*01*01*	All strings of 0's and 1's containing at least two 0's
(1 + 01*0)* ~~of 0's~~	All strings of 0's and 1's containing an even number of 0's
1*(01*01*)* ~~of 0's~~	All strings of 0's and 1's containing an even number of 0's
(1*01*0)*1* of 0's	All strings of 0's and 1's containing an even number of 0's
(0+1)* = (0*1*)*	Any string, or (sigma)*, sigma={0, 1} in all cases here

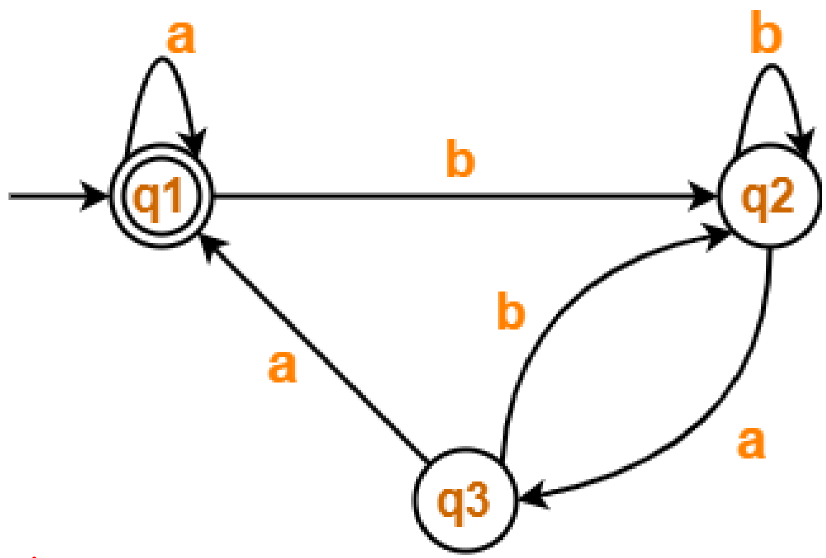
$(0+1)^*00(0+1)^*$ εNFA

$(01)^*10$ εNFA

Summary

- Regular expressions
- Equivalence to finite automata
- DFA to regular expression conversion
- Regular expression to ε-NFA conversion
- Algebraic laws of regular expressions
- Unix regular expressions and Lexical Analyzer

- **Find regular expression for the following DFA**



$a^*bb^*ab^*a$

$(a+b(b+ab)^*ab)^*$

- **Regular Expression for the given DFA = $(a + b(b + ab)^*aa)^*$**

- **Identities:**

1. $\varnothing u = u\varnothing = \varnothing$ Like multiplying by 0
2. $\varepsilon u = u\varepsilon = u$ Like multiplying by 1
3. $\varnothing^* = \varepsilon$ $L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots$
4. $\varepsilon^* = \varepsilon$ $= \{\varepsilon\}$
5. $u+v = v+u$
6. $u + \varnothing = u$
7. $u + u = u$
8. $u^* = (u^*)^*$
9. $u(v+w) = uv+uw$ [*which operation is hidden before parenthesis?*]
10. $(u+v)w = uw+vw$
11. $(uv)^*u = u(vu)^*$ [*note: you have to have a single u, at start or end*]

    [note $(uv)^* =/= u^*v^*$]

12. $(u+v)^* = (u^*+v)^*$

    $= u^*(u+v)^*$

    $= (u+vu^*)^*$

    $= (u^*v^*)^*$

    $= u^*(vu^*)^*$

    $= (u^*v)^*u^*$

Regular Expressions

- Highlights:
  - A regular expression is used to specify a language, and it does so precisely.
  - Regular expressions are very intuitive.
  - Regular expressions are very useful in a variety of contexts.
  - Given a regular expression, an NFA-ε can be constructed from it automatically.
  - Thus, so can an NFA be constructed, and a DFA, and a corresponding program, all automatically!

Two Operations

- Concatenation:
  - x = 010
  - y = 1101
  - xy = <u>010</u> <u>1101</u>

- Language Concatenation: $L_1L_2 = \{xy \mid x \text{ is in } L_1 \text{ and } y \text{ is in } L_2\}$
  - $L_1 = \{01, 00\}$
  - $L_2 = \{11, 010\}$
  - $L_1L_2 = \{\underline{01}\ \underline{11}, \underline{01}\ \underline{010}, \underline{00}\ \underline{11}, \underline{00}\ \underline{010}\}$

- Language Union:
  - $L_1 = \{01, 00\}$
  - $L_2 = \{01, 11, 010\}$
  - $L_1 \cup L_2 = \{01, 00, 11, 010\}$

Operations on Languages

- Let L, L_1, L_2 be subsets of Σ^*
- Concatenation: L_1L_2 = {xy | x is in L_1 and y is in L_2}
- Concatenating a language with itself: $L^0 = \{\varepsilon\}$
  $L^i = LL^{i-1}$, for all i >= 1

Kleene closure

Say, L, or L^1 ={a, abc, ba}, on Σ ={a,b,c}

Then, L^2 = {aa, aabc, aba, abca, abcabc, abcba, baa, baabc, baba}

L^3= {a, abc, ba}. L^2

…..

But, $L^0 = \{\varepsilon\}$

Kleene closure of L, $L^* = \{\varepsilon, L^1, L^2, L^3, \ldots\}$

Operations on Languages

- Let L, L_1, L_2 be subsets of Σ^*
- Concatenation: $L_1L_2 = \{xy \mid x \text{ is in } L_1 \text{ and } y \text{ is in } L_2\}$
- Union is set union of L1 and L2
- Kleene Closure: $L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots$
- Positive Closure: $L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup \ldots$
- Question: Does L^+ contain ε?

Definition of a Regular Expression

- Let Σ be an alphabet. The regular expressions over Σ are:
  - Ø Represents the empty set { }
  - ε Represents the set {ε}
  - a Represents the set {a}, one string of length 1, for any symbol *a* in Σ

  Let *r* and *s* be regular expressions that represent the sets R and S, respectively.

  - r+s Represents the set R U S (precedence 3)
  - rs Represents the set RS (precedence level 2)
  - r^* Represents the set R* (highest precedence, level 1)
  - (r) Represents the set R (not an operator, rather provides precedence)
- If r is a regular expression, then L(r) is used to denote the corresponding language.

Equivalence of Regular Expressions and NFA-εs

- **Note:**

  Throughout the following, keep in mind that a string is accepted by an NFA-ε if there exists ANY path from the start state to any final state.

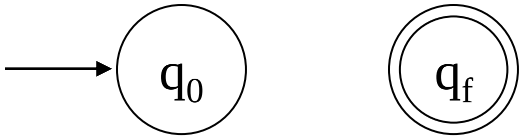
- **Lemma 1:** Let *r* be a regular expression. Then there exists an NFA-ε M such that L(M) = L(r). Furthermore, M has exactly *one* final state with no transitions out of it.

- **Proof:** (by induction on the number of operators, denoted by OP(r), in *r*).

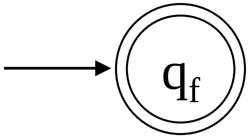
Basis: OP(r) = 0

Then r is either Ø, ε, or **a**, for some symbol **a** in Σ

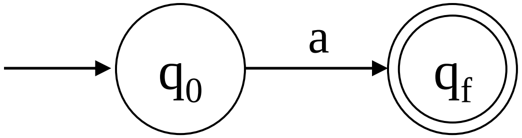
For Ø:



For ε:



For **a**:



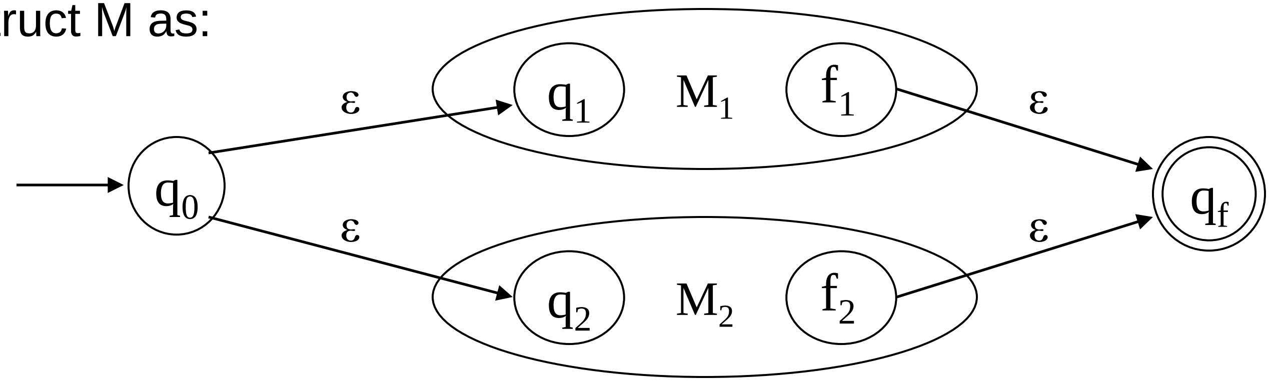
Inductive Hypothesis: Suppose there exists a $k \geq 0$ such that for any regular expression r where $0 \leq OP(r) \leq k$, there exists an NFA-ε such that L(M) = L(r). Furthermore, suppose that *M* has exactly one final state.

Inductive Step: Let r be a regular expression with k + 1 operators (OP(r) = k + 1), where k + 1 >= 1.

Case 1) $r = r_1 + r_2$

Since OP(r) = k +1, it follows that 0<= $OP(r_1)$, $OP(r_2)$ <= k. By the inductive hypothesis there exist NFA-ε machines M_1 and M_2 such that $L(M_1) = L(r_1)$ and $L(M_2) = L(r_2)$. Furthermore, both M_1 and M_2 have exactly one final state.

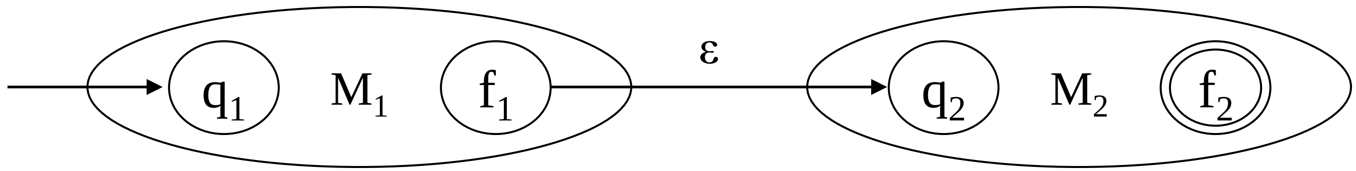
Construct M as:



Case 2) $r = r_1r_2$

Since $OP(r) = k+1$, it follows that $0 <= OP(r_1), OP(r_2) <= k$. By the inductive hypothesis there exist NFA-ε machines M_1 and M_2 such that $L(M_1) = L(r_1)$ and $L(M_2) = L(r_2)$. Furthermore, both M_1 and M_2 have exactly one final state.

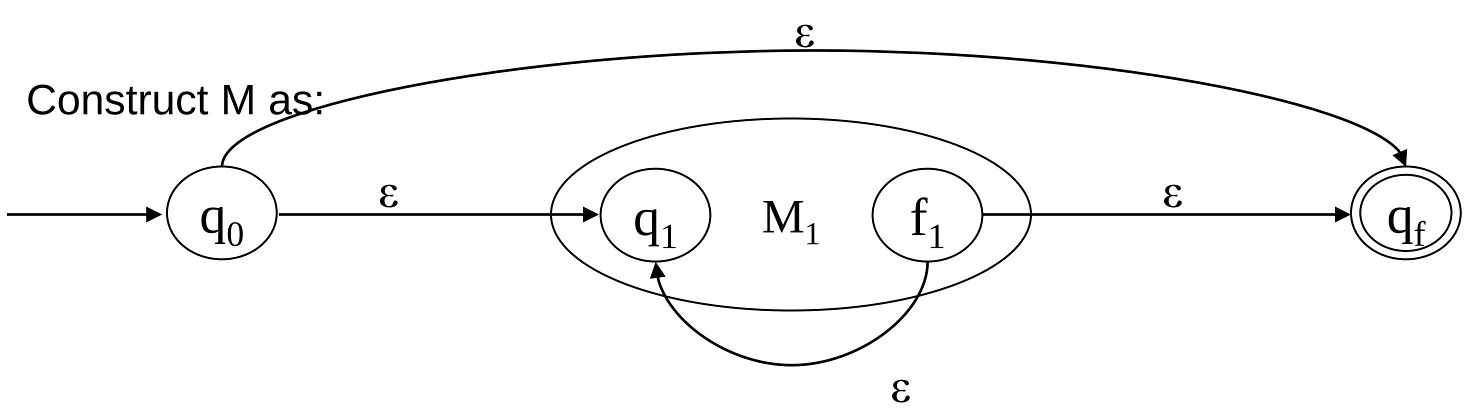
Construct M as:



Case 3) $r = r_1^*$

Since $OP(r) = k+1$, it follows that $0 <= OP(r_1) <= k$. By the inductive hypothesis there exists an NFA-ε machine M_1 such that $L(M_1) = L(r_1)$. Furthermore, M_1 has exactly one final state.

Construct M as:



- **Example:**

$r = 0(0+1)^*$

$r = r_1r_2$

$r_1 = 0$

$r_2 = (0+1)^*$

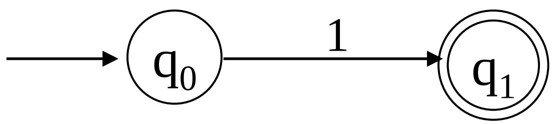
$r_2 = r_3^*$

$r_3 = 0+1$

$r_3 = r_4 + r_5$

$r_4 = 0$

$\mathbf{r_5 = 1}$



- **Example:**

$r = 0(0+1)^*$

$r = r_1 r_2$

$r_1 = 0$

$r_2 = (0+1)^*$

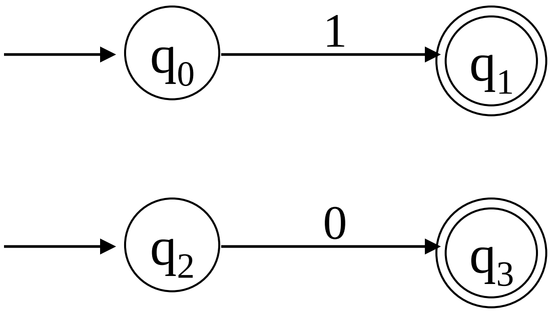
$r_2 = r_3^*$

$r_3 = 0+1$

$r_3 = r_4 + r_5$

$\mathbf{r_4 = 0}$

$r_5 = 1$



- **Example:**

$r = 0(0+1)^*$

$r = r_1 r_2$

$r_1 = 0$

$r_2 = (0+1)^*$

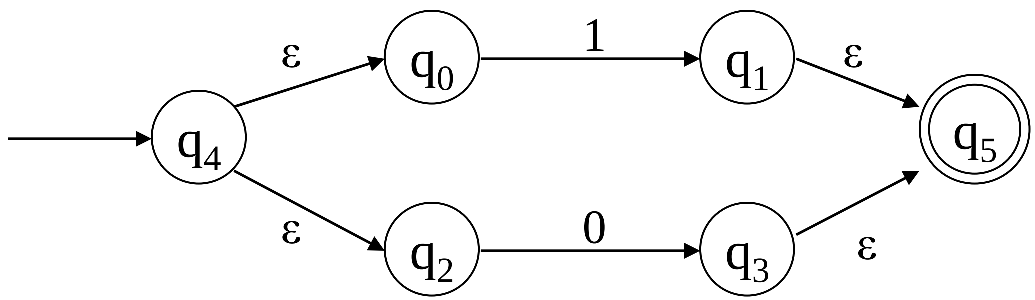
$r_2 = r_3^*$

$r_3 = 0+1$

$\mathbf{r_3 = r_4 + r_5}$

$r_4 = 0$

$r_5 = 1$



- **Example:**

$r = 0(0+1)^*$

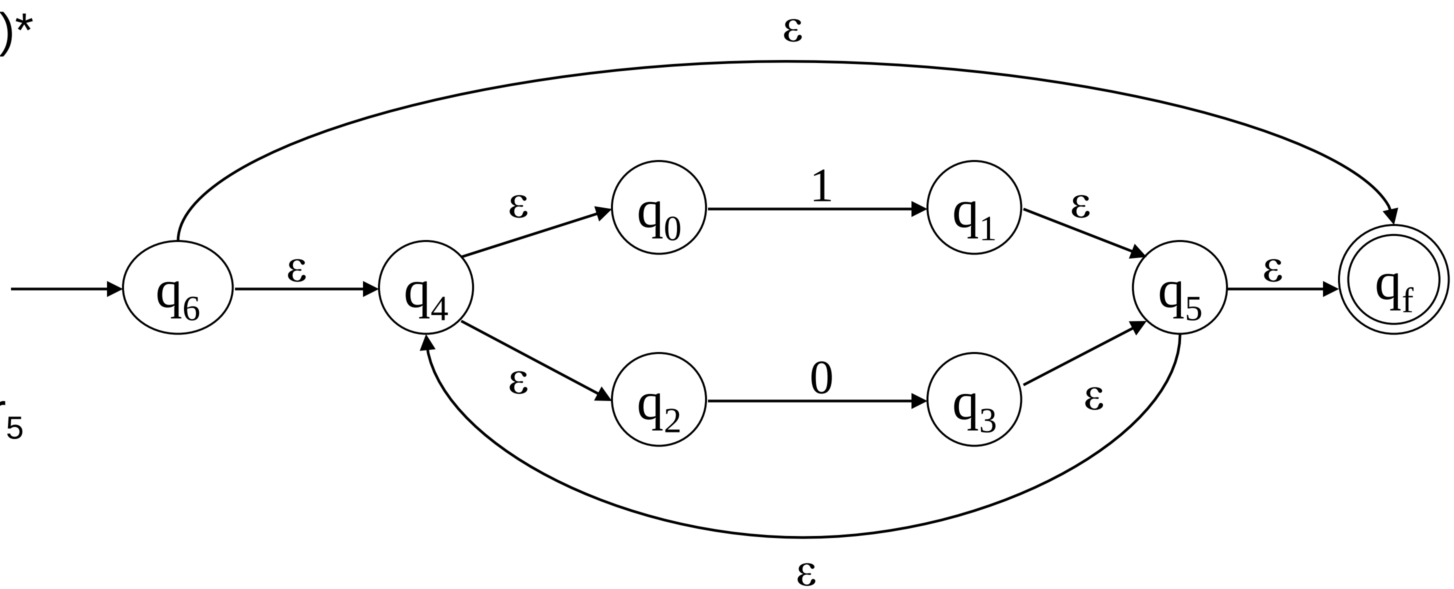
$r = r_1 r_2$

$r_1 = 0$

$r_2 = (0+1)^*$

$\mathbf{r_2 = r_3^*}$

$r_3 = 0+1$



$r_3 = r_4 + r_5$

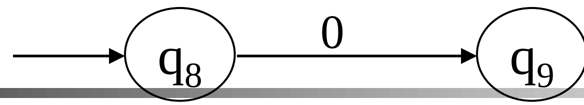
$r_4 = 0$

$r_5 = 1$

- **Example:**

$r = 0(0+1)^*$

$r = r_1 r_2$

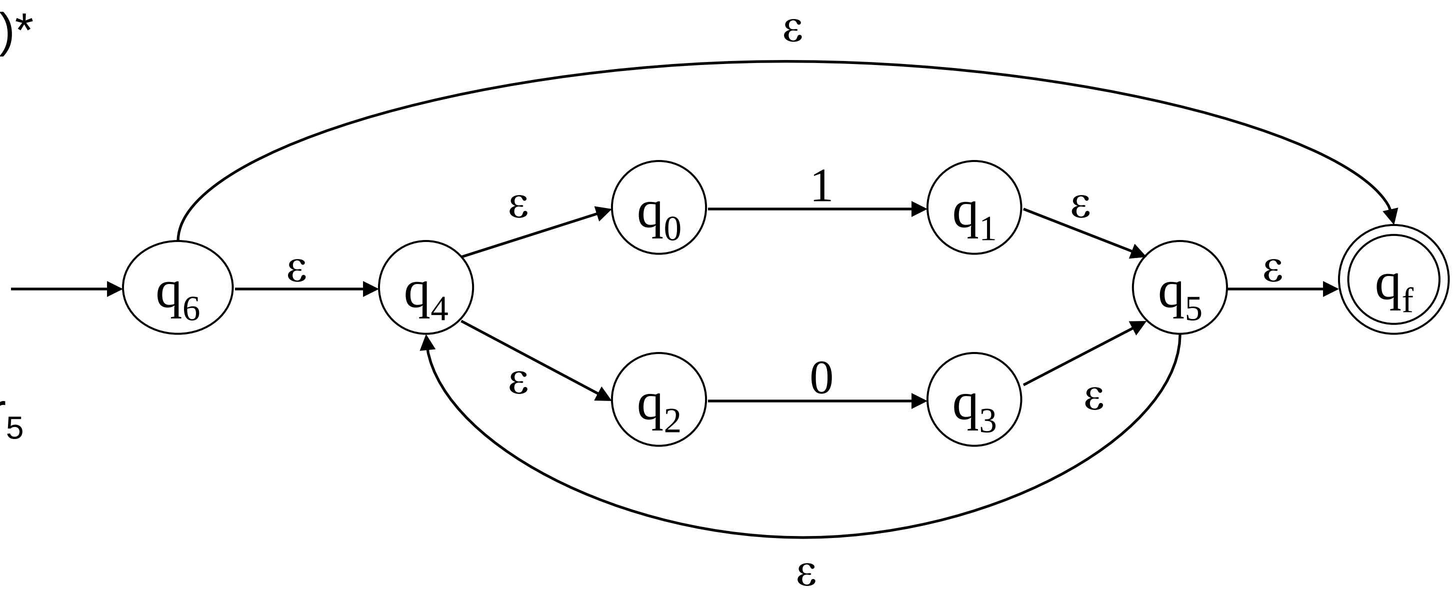


$r_1 = 0$

$r_2 = (0+1)^*$

$r_2 = r_3^*$

$r_3 = 0+1$



$r_3 = r_4 + r_5$

$r_4 = 0$

$r_5 = 1$

- **Example:**

r = 0(0+1)*

$r = r_1 r_2$

$r_1 = 0$

$r_2 = (0+1)^*$

$r_2 = r_3^*$

$r_3 = 0+1$

$r_3 = r_4 + r_5$

$r_4 = 0$

$r_5 = 1$