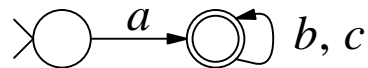


# REGULAR EXPRESSIONS: AN ALTERNATIVE DESCRIPTION OF REGULAR LANGUAGES



$L = \{x: x \text{ equals } a \text{ followed by arbitrary combination of } b \text{ and } c\}$   
 $= a \cdot \{b, c\}^*$ , in algebraic (set theoretic) notation  
 $= a(b + c)^*$ , in regular expression notation

## Notations:

'.' = concatenation (often omitted)

'+' = set union (i.e., alternative choices)

'\*' = Kleene's closure

## Kleene's Closure:

- $L^* = \{x_1 x_2 \cdots x_k: \text{each } x_i \in L \text{ and } k \geq 0\} \supseteq L$ .
- $\emptyset^* = \{\lambda\}$ .

## Regular Expression vs. FSA:

- Does not involve the notion of state and transitions of an FSA (both are extraneous to the language defined by FSA).
- A finite *combinatorial* (intrinsic) description of the structure of strings in the language.
  - Uses concatenation, union, and Kleene's closure. (Kleene's closure is needed only for infinite languages.)
  - Does not use complementation and intersection.

## REGULAR EXPRESSIONS AND THEIR ASSOCIATED LANGUAGES

### Inductive Definition:

- *Base cases.*

(1)	$\emptyset$	$L(\emptyset) = \emptyset$ , empty language
(2)	$a, a \in \Sigma$	$L(a) = \{a\}$

- *Induction step.*

(3)	$\alpha + \beta$	$L(\alpha + \beta) = L(\alpha) \cup L(\beta)$ ; (the new language is larger)
(4)	$\alpha\beta$	$L(\alpha\beta) = L(\alpha) \cdot L(\beta)$ (the new language has longer strings)
(5)	$\alpha^*$	$L(\alpha^*) = [L(\alpha)]^*$ (the new language has infinitely many strings)

### Examples.

$$L((b + c)^*) = [L(b + c)]^* = \{b, c\}^* = \{\lambda, b, c, bb, bc, cb, cc, \dots\}$$

$$L(a(b + c)^*) = L(a)L((b + c)^*) = \{a\}\{b, c\}^*$$

$$= \{a, ab, ac, abb, abc, acb, acc, \dots\}$$

### Some Useful Notations:

- |   |  |
|---|--|
| (1) $\lambda$ for $\emptyset^*$                               | [because $L(\emptyset^*) = [L(\emptyset)]^* = \emptyset^* = \{\lambda\}$ ]   |
| (2) $\alpha^+$ for $\alpha\alpha^*$                           | [because $L(\alpha\alpha^*) = L(\alpha) \cdot [L(\alpha)]^* = \{x_1 x_2 \dots x_k : k \geq 1, x_j \in L(\alpha)\}$ ]                     |
| (3) $\alpha + \beta + \gamma$ for $(\alpha + \beta) + \gamma$ | [because both $(\alpha + \beta) + \gamma$ and $\alpha + (\beta + \gamma)$ denote the language $L(\alpha) \cup L(\beta) \cup L(\gamma)$ ] |
| (4) $x \in \alpha$ for $x \in L(\alpha)$                      |  |

## ADVANTAGES AND DISADVANTAGES OF REGULAR EXPRESSION

### Advantages:

- Does not use extraneous notions of states and transitions of an FSA.
- Easy to construct an FSA from a regular expression and vice-versa (via algorithms).

This means if the languages  $L_1$  and  $L_2$  are described by regular-expressions, then we can answer the following questions (by converting each regular-expression to an FSA).

The membership-question:  $x \in L_1?$

The subset-question:  $L_1 \subseteq L_2?$

The equality-question:  $L_1 = L_2?$

### Disadvantages:

- Many different regular expressions may give the same language. (In general, there is no unique "minimal" regular expression for a regular language  $L$ .)

(1) Both  $aa^+$  and  $a^+a$  give the language  $\{aa, aaa, \dots\}$

(2) Both  $a^*$  and  $(a^*)^*$  give the language  $\{\lambda, a, aa, aaa, \dots\}$ .

(3) Both  $a^*(a + b)^*$  and  $(a + b)^*$  give the language  $\{\lambda, a, b, aa, ab, ba, \dots\}$ .

- A string  $x \in L(\alpha)$  may have more than one *interpretation*.

$x = abbc$  has 3 interpretations w.r.t  $(a + b)^*(b + c)^*$ :

$a.bbc$	$a \in L[(a + b)^*], bbc \in L[(b + c)^*]$
$ab.bc$	$ab \in L[(a + b)^*], bc \in L[(b + c)^*]$
$abb.c$	$abb \in L[(a + b)^*], c \in L[(b + c)^*]$

**EXERCISE**

1. Let  $\beta = (a + b)^*(b + c)^*$ . Find an alternate regular expression  $\beta'$  such that  $L(\beta) = L(\beta')$  and each string  $x \in L(\beta)$  has exactly one interpretation. Show the unique interpretation of  $x = abcbcb$  w.r.t  $\beta'$ . (Hint: use the first occurrence of  $c$  in  $x$ , if any, as a basis for the new regular expression.)

## FINDING A REGULAR EXPRESSION FROM AN FSA

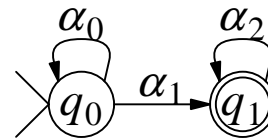
### Transition-Diagram:

- An FSA-like state diagram, except that the labels on the transitions may be a regular expression.

**Method:** Reduce the FSA to a transition-diagram having one of the following basic forms (the loops may not be present):



(i) One state;  
regular expr:  $\alpha^*$ .



(ii) Two states and no transition from the final-state to the start-state;  
regular expr:  $\alpha_0^* \alpha_1 \alpha_2^*$ .

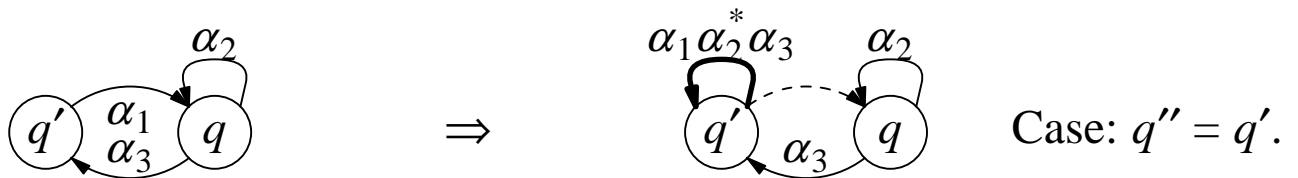
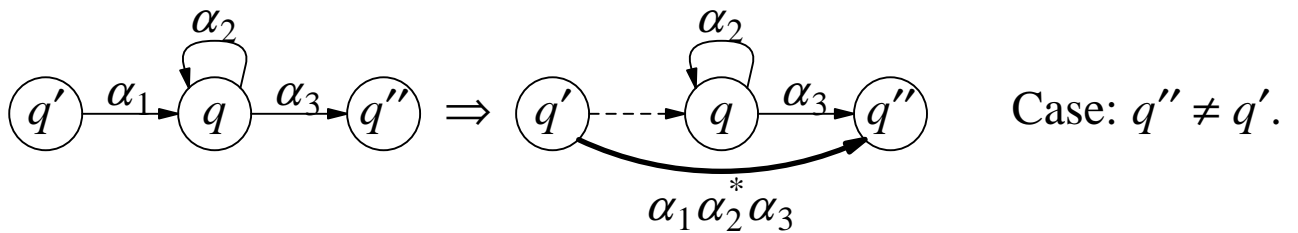
### Regular Expression Algorithm for FSA $M$ :

- Simplify  $M$  by eliminating unreachable states and dead-states (from which no final-states can be reached).
- If there are no states left, then the regular expression is  $\emptyset$ .
- For each final-state  $q_i$  in  $M$ , let  $M_i$  be the FSA which is same as  $M$  except that  $q_i$  is the only final-state. Simplify  $M_i$  by eliminating unreachable states, if any, and then reduce  $M_i$  to a transition diagram of the form (i) or (ii) above. Let  $\beta_i$  be the corresponding regular expression.
- The final regular expression for  $M$  is  $\beta_1 + \beta_2 + \dots + \beta_n$ , where  $n = \#(\text{final-states in } M)$ .

## REDUCING AN FSA TO A TRANSITION DIAGRAM WITH JUST ONE OR TWO STATES

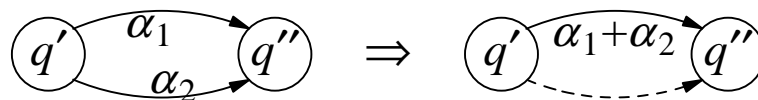
**Three Key Steps** (start with an FSA with just one final-state):

- For each non-final state  $q$  and for each transition  $(q', q)$  with  $q' \neq q$ , remove that transition by adding a new transition from  $q'$  as illustrated below and shown in bold line. There is one new transition  $(q, q'')$  from  $q$  for each  $q'' \neq q$ .



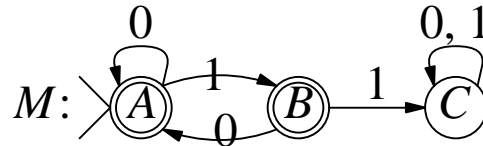
Replace  $\alpha_1 \alpha_2^* \alpha_3$  by  $\alpha_1 \alpha_3$   
if there is no  $\alpha_2$ -loop.

- Merge parallel transitions.

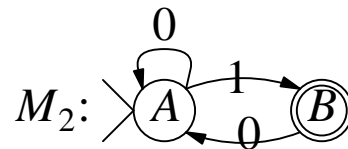
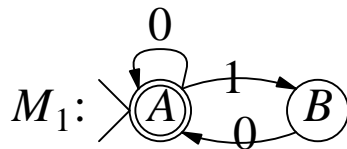


- When there are no transitions to  $q$ , remove  $q$  if  $q \neq$  start-state.

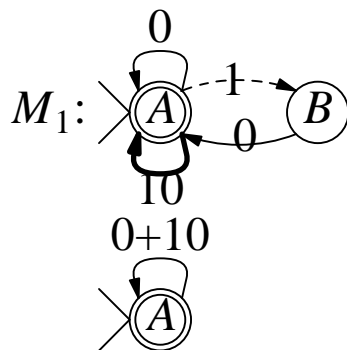
## ILLUSTRATION OF REGULAR EXPRESSION GENERATION FOR AN FSA



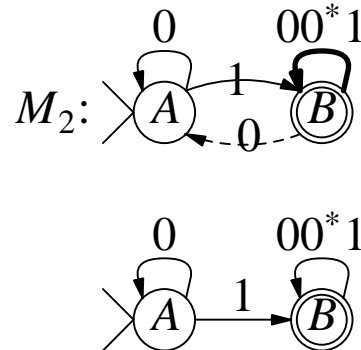
- After removing the unreachable state  $C$ ,  $M$  is replaced by two FSAs  $M_1$  and  $M_2$ , each with one final-state.  $L(M) = L(M_1) \cup L(M_2)$ .



- Now convert each of  $M_1$  and  $M_2$  to the special form by elimination of transitions and states.



Regular Expr:  $(0 + 10)^*$ .



Regular Expr:  $0^*1(0^+1)^*$ .

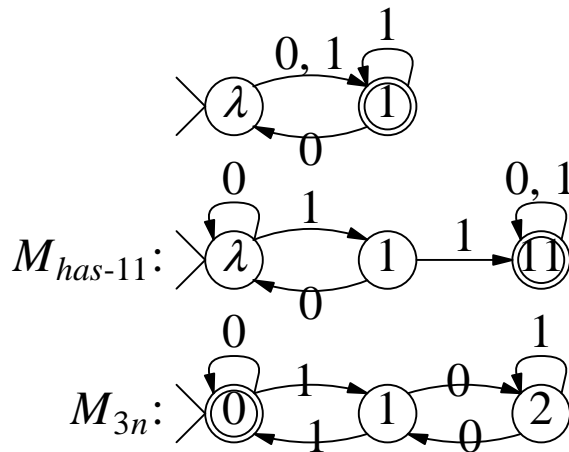
- Final regular expr:  $(0 + 10)^* + 0^*1(0^+1)^* = (0 + 10)^*(\lambda + 1)$ .  
The two parts correspond to strings ending in 0 and 1, respectively.

**Question:** Why strings in  $(0 + 10)^* + 0^*1(0^+1)^*$  has no "11"?

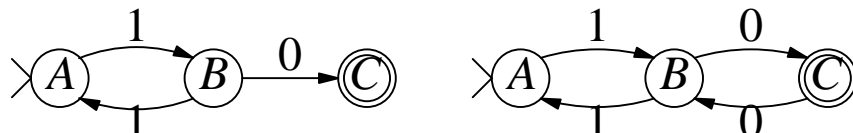
Final reg. expr. may depend on the order in which transitions are eliminated.

### EXERCISE

- Argue that the regular expression  $\beta$  obtained by the above method for an FSA  $M$  has the property that it allows an unambiguous interpretation of each string in  $L(M)$ . That is, if  $\beta = \beta_1 + \beta_2$ , then  $L(\beta_1) \cap L(\beta_2) = \emptyset$ , and if  $\beta = \beta_1\beta_2$ , then each string  $x \in L(\beta)$  has a unique decomposition as  $x = yz$ , where  $y \in L(\beta_1)$  and  $z \in L(\beta_2)$ .
- Find a regular expression for each FSA below; show all details of the reduction process.



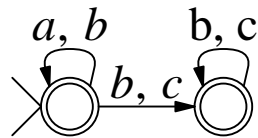
- Follow the method given here to find a regular expression for each of the FSAs below and determine which of the English descriptions correspond to the associated languages, if any.
  - Odd number of 1's followed by 0.
  - Begins with odd number of 1's, end with odd number of 0's, and between them each run of 0's and each run of 1's, which alternate, has even length.





## APPLICATION TO NON-DETERMINISTIC FSA

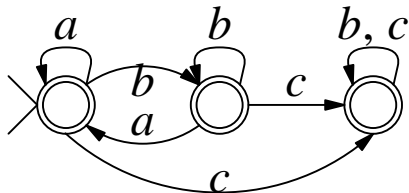
- The method to obtain a regular expression for an FSA applies equally well for an NFSA as illustrated below.



This gives the regular expression (one term for each final-state):

$$\begin{aligned}
 & (a + b)^* + (a + b)^* (b + c)(b + c)^* \\
 &= (a + b)^* + (a + b)^* (b + c)^+ = (a + b)^* [\lambda + (b + c)^+] \\
 &= (a + b)^* (b + c)^*.
 \end{aligned}$$


- The deterministic form of the above NFSA:

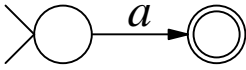


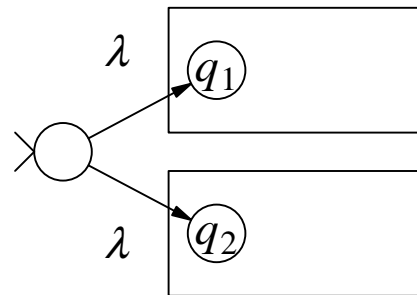
This gives the reg-expr (one term for each final-state):

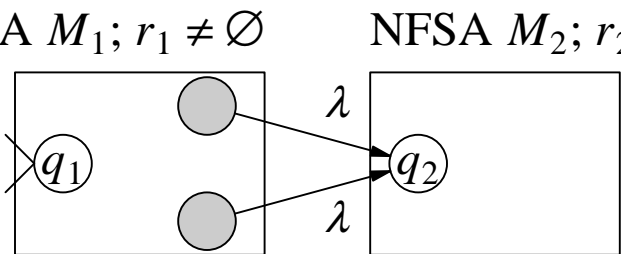
$$\begin{aligned}
 & (a + b^+ a)^* + a^* b (b + a^+ b)^* + (a + b^+ a)^* (b^+ c + c)(b + c)^* \\
 &= (b^* a)^* + a^* b (a^* b)^* + (b^* a)^* (b^* c)(b + c)^* \\
 &= (\lambda \text{ or strings in } \{a, b\} \text{ ending in } a) + (\text{strings in } \{a, b\} \text{ ending in } b) \\
 &\quad + (b^* a)^* b^* . c (b + c)^* \\
 &= (a + b)^* + (a + b)^* c (b + c)^*, \text{ because } (b^* a)^* b^* = (a + b)^*.
 \end{aligned}$$

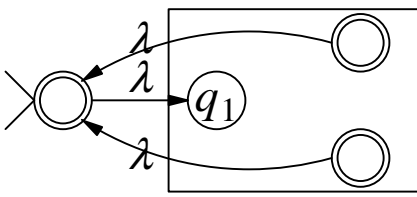
# FROM REGULAR EXPRESSION TO NFSA

$r = \emptyset$ :  no final-state

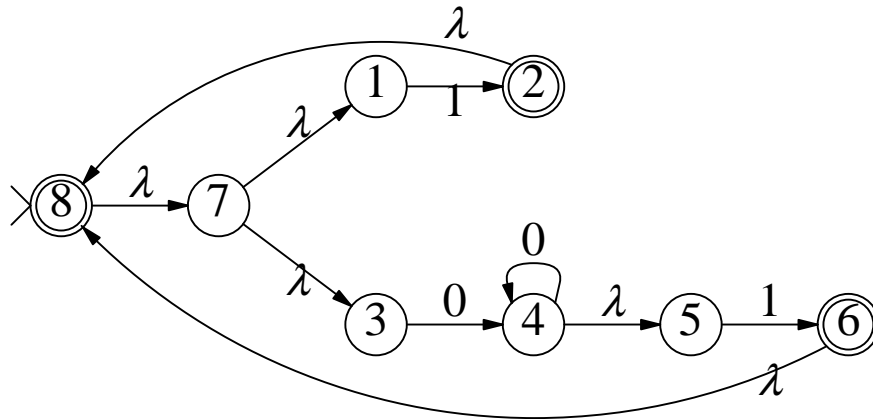
$r = a$ : 

$r = r_1 + r_2$ :  NFSA  $M_1$  for  $r_1$ ;  
 $q_1 = \text{start-state of } M_1$   
NFSA  $M_2$  for  $r_2$ ;  
 $q_2 = \text{start-state of } M_2$

$r = r_1 r_2$ :  NFSA  $M_1$ ;  $r_1 \neq \emptyset$       NFSA  $M_2$ ;  $r_2 \neq \emptyset$   
Add a  $\lambda$ -transition from *each* final-state of  $M_1$  to the start-state of  $M_2$ , the start-state of  $M_1$  is the new start-state, and the new final-states are those of  $M_2$ .

$r = r_1^*$ :  NFSA  $M_1$   
for  $r_1 \neq \emptyset$   
Add a  $\lambda$ -transition from *each* final-state of  $M_1$  to the new start-state (which is also a final-state now), the final-states of  $M_1$  remains final-states, and add a  $\lambda$ -transition from the new start-state to  $q_1$ .

## NON-DET FSA FOR $(1 + 0^+1)^*$ STARTING FROM FSAS FOR 1 and $0^+$



### EXERCISE

1. Apply the algorithm to the regular expression  $(0 + 10)^* 11(0 + 1)^*$  and show the NFSA obtained; keep all intermediate states and transitions introduced and start from FSAs for  $\{0\}$  and for  $\{1\}$ .
2. Show the NFSA after elimination of all  $\lambda$ -transitions in the NFSA obtained in Problem 1.
3. Show the deterministic FSA for the NFSA obtained in Problem 2.
4. Show the minimum form of the FSA in Problem 3.
5. How is  $L(M)$  affected if we just add a  $\lambda$ -transition from each final-state of  $M$  to the start-state?