# COMS3261:
# Computer Science Theory

Spring 2013

Mihalis Yannakakis

Lecture 5, 9/18/13

## Regular expressions

| | Expression | Language |
|---|---|---|
| Basis: | $\emptyset$ | $\emptyset$ |
| | $\varepsilon$ | $\{ \varepsilon \}$ |
| | a, $\forall a \in \Sigma$ | $\{ a \}$ |
| Induction: | ( E ) | L(E) |
| (Operations) | | |
| Union | E+F | $L(E) \cup L(F)$ |
| Concatenation | E.F or EF | L(E).L(F) |
| Kleene * | E* | (L(E))* |

## Examples

- 0100 : the singleton set {0100}
- 0*: all strings of 0's (including the empty string)
- (0+1)*: all binary strings, including the empty string
- 0*1* : a sequence of 0's (possibly none) followed by a sequence of 1's (possibly none):
  - includes $\varepsilon$, 0, 1, 01, 00 … but not 10
- (0*1*)* = (0+1)* : all binary strings
- 0+10* : {0, 1, 10, 100, 1000 ,…}
- 0(1+0)* : all binary strings that start with 0
- ((0+1) (0+1) )* : all binary strings of even length

## Regular Expressions ⇔ Finite Automata

- Theorem: Languages of regular expressions = Regular languages:

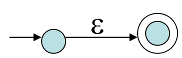
1. From every RE R we can construct an equivalent FA A, i.e. one that accepts the same language: L(A)=L(R)
2. From every FA we can construct an equivalent RE

## Regular Expression → Finite Automaton

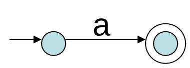
- We will construct an equivalent ε-NFA with one start state and one different accepting state.
- By induction on the construction of the RE R.
- Basis: R = ∅, or ε, or a in Σ

R = ∅

R = ε



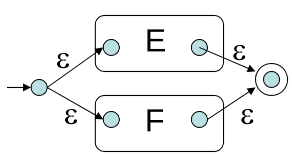

R = a



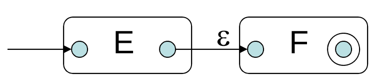

## RE → FA

- Induction step: R= (E), or E+F , or E.F or E*
- Inductively assume that we have ε-NFA with one start state and one accepting state for each subexpression E,F
- R= (E): same NFA as for E

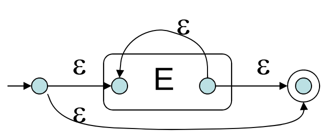
R = E + F




R = E.F

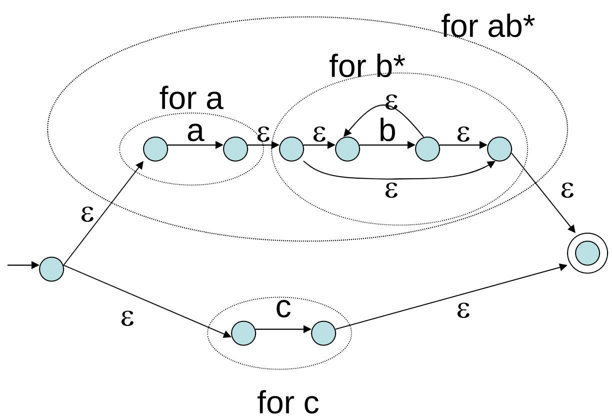



R = E*



## Example

- R = ab* +c = ( a.(b*)) + c




## Complexity of RE → ε-NFA translation

- Complexity of translation and size of the resulting ε-NFA is linear in the size of the RE (#characters in the expression)
- Proof: Easy, by induction.

  -Basis: NFA with 2 states and at most one transition

  - Each operation combines the NFA of the operands and adds a small, fixed # of states (at most 2) and transitions (at most 4).

So, # states of NFA $\leq$ 2 |RE|

and # transitions $\leq$ 4 |RE|

## FA $\rightarrow$ RE Translation

- Various methods:
- Kleene's Method (in HMU)
- State elimination method (in HMU and Sipser)
- Algebraic set of equations & elimination of variables

## Kleene's method (Dynamic programming)

- Input A can be NFA, even with $\varepsilon$-transitions:

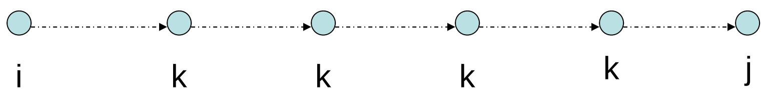
  Labelled directed graph, where every arc has one or more labels in $S \cup \{\varepsilon\}$
- Number the states of A arbitrarily as 1,…,n
- Will compute for each triple of states i,j,k a RE

  $R_{ij}^{(k)}$ whose language = set of labels of $i \rightarrow j$ paths all of whose intermediate nodes are in {1,…,k}
- For k=n, the RE $R_{ij}^{(n)}$ gives the set of labels of all $i \rightarrow j$ paths (no restriction on intermediate nodes)
- If the start state is 1 and the set of accepting states is F then the RE R for the FA A is: $R = \sum_{j \in F} R_{1j}^{(n)}$

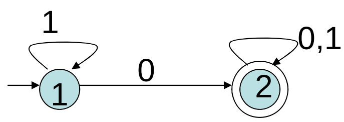
# Kleene's algorithm

- Initialization (k=0):
  [only direct path(edge, no intermediate node]
- i=j: $R_{ii}^{(0)} = \varepsilon + \Sigma$ labels on arc $i \rightarrow i$ (if arc $i \rightarrow i$ present)
- $i \neq j$: $R_{ij}^{(0)} = \begin{cases} \Sigma \text{ labels on arc } i \rightarrow j \text{ , if arc } i \rightarrow i \text{ present} \\ \varnothing \text{ , otherwise (if no arc } i \rightarrow j \text{ )} \end{cases}$
- Loop (Induction Step): For k=1 to n do

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* \ R_{kj}^{(k-1)}$$




# Example




k=0

| | 1 | 2 |
|---|---|---|
| 1 | $\varepsilon$+1 | 0 |
| 2 | $\varnothing$ | $\varepsilon$+0+1 |

## Example ctd.

k=1

| | 1 | 2 |
|---|---|---|
| 1 | $\varepsilon+1+(\varepsilon+1)(\varepsilon+1)^*(\varepsilon+1) \equiv 1^*$ | $0+(\varepsilon+1)(\varepsilon+1)^*0 \equiv 1^*0$ |
| 2 | $\varnothing+\varnothing(\varepsilon+1)^*(\varepsilon+1) \equiv \varnothing$ | $\varepsilon+0+1+\varnothing(\varepsilon+1)^*0 \equiv \varepsilon+0+1$ |

Properties: $(\varepsilon+1)^* \equiv 1^*$

$(\varepsilon+1)1^* \equiv 1^*$ ;  $\varepsilon+1+1^* \equiv 1^*$

For any RE R: $R+\varnothing \equiv \varnothing+R \equiv R$;  $\varnothing R \equiv R\varnothing \equiv \varnothing$

## Example ctd.

k=2

| | 1 | 2 |
|---|---|---|
| 1 | $1+1^*0(\varepsilon+0+1)^*\varnothing \equiv 1^*$ | $1^*0+1^*0(\varepsilon+0+1)^*(\varepsilon+0+1) \equiv 1^*0(0+1)^*$ |
| 2 | $\varnothing+(\varepsilon+0+1)(\varepsilon+0+1)^*\varnothing \equiv \varnothing$ | $\varepsilon+0+1+(\varepsilon+0+1)(\varepsilon+0+1)^*(\varepsilon+0+1) \equiv (0+1)^*$ |

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

## Complexity of FA $\rightarrow$ RE Translation

- Let S(k) = maximum size of an expression in stage k
  (= $\max_{ij} | R_{ij}^{(k)} |$ )
- $S(0) \leq 2 | \Sigma |$

$S(k) \leq 4S(k-1) + 4$

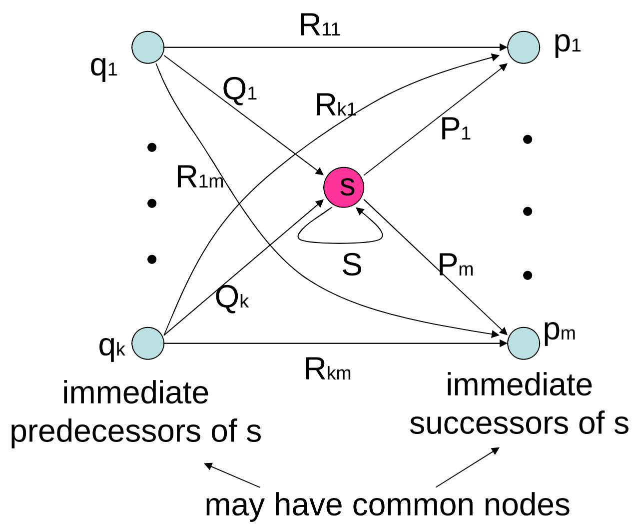
$\Rightarrow$ $S(k) \leq 2 \cdot 4^k \cdot |\Sigma| \Rightarrow S(n) = O(4^n |\Sigma| )$

- Can be exponential: There are DFA for which the constructed RE (even after simplification) is exponential in the #states n of the DFA

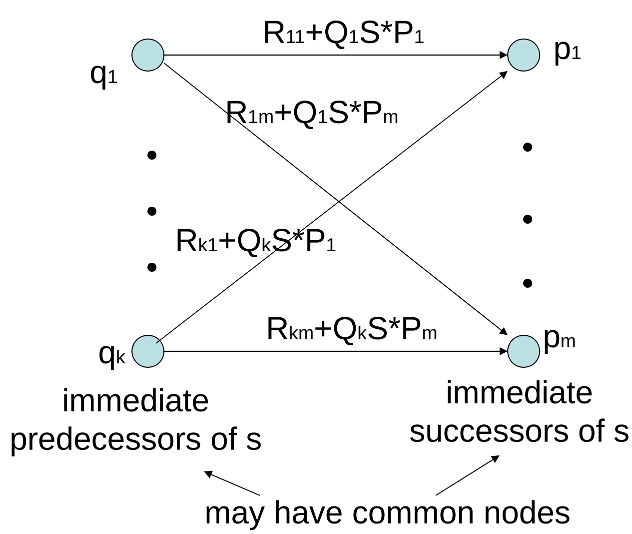
## State elimination method

- Maintain a generalized NFA: edges labeled by regular expressions, not only single symbols and $\varepsilon$
- *Assume one accepting state p* (and one start state $q_0$): if many accepting states, then add new accepting state and $\varepsilon$-transitions from old accepting states
- *Eliminate all the states, except the start state and accepting state*, one by one, updating the edge labels
- *At the end compute a Regular Expression* that captures all accepting paths from the start state $q_0$ to the accepting state p.

## Elimination of a state s

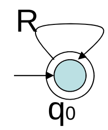



## Elimination of a state s



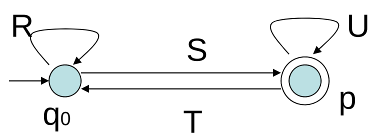
# Final generalized NFA

- Case 1: Accepting state = start state $q_0$



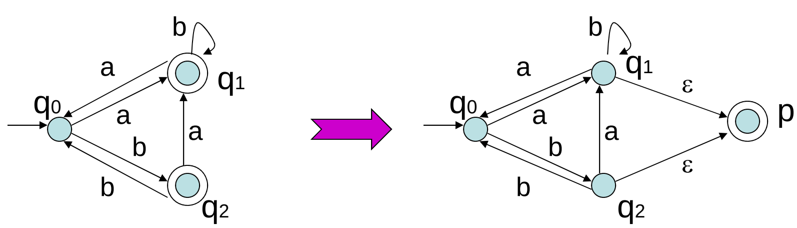

Regular Expression E = R*

- Case 2: Accepting state p ≠ start state $q_0$

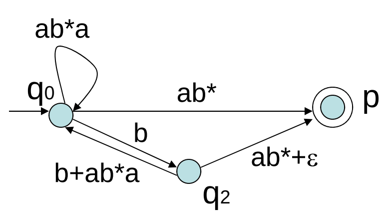



Regular Expression: E = (R+SU*T)*SU*

# Example

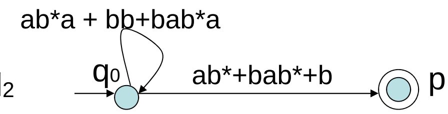



Eliminate $q_1$



# Example ctd

Eliminate $q_2$




Regular expression: (ab*a+bb+bab*a)*(ab*+bab*+b)