Department of Computer & Information Sciences
Pakistan Institute of Engineering and Applied Sciences

Regular Expressions

Umar Faiz
http://www.pieas.edu.pk/umarfaiz/cis317

Regular Expressions

Regular expressions describe regular languages

Example:

$$(a+b\cdot c)^*$$

describes the language

$$\{a,bc\}^* = \{\lambda, a, bc, aa, abc, bca, ...\}$$

Regular Expressions

The regular expressions over finite Σ are the strings over the alphabet Σ such that:

1. { } (empty set) is a regular expression for the empty set
2. ε is a regular expression denoting { ε }
3. *a* is a regular expression denoting set { *a* } for any *a* in Σ

Recursive Definition

Primitive regular expressions: $\varnothing, \quad \lambda, \quad \alpha$

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1^* \\ (r_1) \end{array}\right\} \text{are regular expressions}$$

Examples

A regular expression: $(a+b\cdot c)^*\cdot(c+\varnothing)$

Not a regular expression: $(a+b+)$

Languages of Regular Expressions

$L(r)$: language of regular expression r

Example:

$$L((a+b\cdot c)^*) = \{\lambda, a, bc, aa, abc, bca, ...\}$$

Definition

For primitive regular expressions:

$$L(\varnothing)=\varnothing$$
$$L(\lambda)=\{\lambda\}$$
$$L(a)=\{a\}$$

Definition (continued)

For regular expressions r_1 and r_2

$$L(r_1+r_2)=L(r_1)\cup L(r_2)$$
$$L(r_1\cdot r_2)=L(r_1)\,L(r_2)$$
$$L(r_1*)=(L(r_1))*$$
$$L((r_1))=L(r_1)$$

Example

Regular expression: $(a+b)\cdot a*$

$$\begin{aligned}L((a+b)\cdot a*) &= L((a+b))\,L(a*)\\ &= L(a+b)\,L(a*)\\ &= (L(a)\cup L(b))\,(L(a))*\\ &= (\{a\}\cup\{b\})\,(\{a\})*\\ &= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\}\\ &= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}\end{aligned}$$

Example

Regular expression $r=(a+b)*(a+bb)$

$$L(r)=\{a,bb,aa,abb,ba,bbb,\ldots\}$$

Example

Regular expression $r=(aa)*(bb)*b$

$$L(r)=\{a^{2n}b^{2m}b:\quad n,m\geq 0\}$$

Example

Regular expression $r=(0+1)*00\,(0+1)*$

$L(r)$ = { all strings with at least two consecutive 0 }

Example

Regular expression $r = (1+01)^*(0+\lambda)$

$L(r)$ = { all strings without two consecutive 0 }

Equivalent Regular Expressions

Definition:

Regular expressions r_1 and r_2

are equivalent if $L(r_1) = L(r_2)$

Example

L = { all strings without two consecutive 0 }

$$r_1 = (1+01)^*(0+\lambda)$$

$$r_2 = (1^*011^*)^*(0+\lambda)+1^*(0+\lambda)$$

$L(r_1) = L(r_2) = L$ $\Rightarrow$ r_1 and r_2 are equivalent regular expr.

Algebraic Laws for REs

Just like we have an algebra for arithmetic, we also have an algebra for regular expressions.

- Commutative Law for Union
  - L + M = M + L
- Associative Law for Union
  - (L + M) + N = L + (M + N)
- Associative Law for Concatenation
  - (LM)N = L (MN)
- There is no commutative law for concatenation
  - LM $\neq$ ML

Algebraic Laws for REs

- The identity for union is:
  - L + ϕ = ϕ + L = L
- The identity for concatenation is:
  - L ε = ε + L
- The annihilator for concatenation is:
  - ϕ L = Lϕ = L

Algebraic Laws for REs

- Left Distributive law:
  - L(M + N) = LM + LN
- Right Distributive law:
  - (M + N)L = LM + LN
- Idempotent law:
  - L + L = L

Laws involving Closures

- $(L^*)^* = L^*$
  - i.e., taking the closure of a regular expression under closure does not change the language
- $\phi^* = \varepsilon$
- $\varepsilon^* = \varepsilon$
- $L^+ = LL^* = L^*L$
- $L^* = L^+ + \varepsilon$
- $L? = \varepsilon + L$

19

Checking a Law

Suppose we are told that the law

$(R + S)^* = (R^*S^*)^*$

holds for regular expressions. How would we check that this claim is true?

20

Checking a Law

1. Convert the RE's to DFA's and minimize the DFA's to see if they are equivalent
2. We can use the concretization test:
   - Think of R and S as if they were single symbols, rather than placeholders for languages, i.e, R = {0} and S = {1}
   - Test whether the law holds under the concrete symbols. If so, then the law is true, and if not, the law is false

21

Concretization Test

For the example

$(R + S)^* = (R^*S^*)^*$

We can substitute 0 for R and 1 for S. The left side is clearly any sequence of 0's and 1's. The right side also denotes any string of 0's and 1's, since 0 and 1 are each in $L(0^*1^*)$

22

Concretization Test

- NOTE: extensions of the test beyond regular expressions may fail.
- Consider the "law" $L \cap M \cap N = L \cap M$.
- This is clearly false
  - Let L=M={a} and N=Ø. {a} ≠ Ø.
  - But if L={a} and M = {b} and N={c} then
  - L∩M does equal L ∩ M ∩ N which is empty.
  - The test would say this law is true, but it is not because we are applying the test beyond regular expressions.
- We'll see soon various languages that do not have corresponding regular expressions.

23

Theorem

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} = \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Costas Busch - RPI

24

Theorem - Part 1

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

1. For any regular expression r the language $L(r)$ is regular

Theorem - Part 2

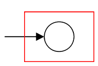
$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

2. For any regular language L there is a regular expression r with $L(r) = L$

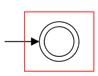
Induction Basis

Primitive Regular Expressions: $\emptyset, \ \lambda, \ \alpha$

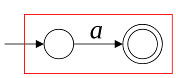
NFAs



$L(M_1) = \emptyset = L(\emptyset)$



$L(M_2) = \{\lambda\} = L(\lambda)$



$L(M_3) = \{a\} = L(a)$

regular languages

Inductive Hypothesis

Assume for regular expressions r_1 and r_2 that $L(r_1)$ and $L(r_2)$ are regular languages

Proof - Part 1

1. For any regular expression r the language $L(r)$ is regular

Proof by induction on the size of r

Inductive Step

We will prove:

$L(r_1 + r_2)$

$L(r_1 \cdot r_2)$

$L(r_1 *)$

$L((r_1))$

Are regular Languages

By definition of regular expressions:

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$
$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$
$$L(r_1 *) = (L(r_1))*$$
$$L((r_1)) = L(r_1)$$

By inductive hypothesis we know:
$L(r_1)$ and $L(r_2)$ are regular languages

We also know:
Regular languages are closed under:

Union	$L(r_1) \cup L(r_2)$
Concatenation	$L(r_1)\, L(r_2)$
Star	$(L(r_1))*$

Therefore:

$$\left.\begin{array}{l} L(r_1 + r_2) = L(r_1) \cup L(r_2) \\ \\ L(r_1 \cdot r_2) = L(r_1)\, L(r_2) \\ \\ L(r_1 *) = (L(r_1))* \end{array}\right\}$$ Are regular languages

And trivially:

$L((r_1))$ is a regular language

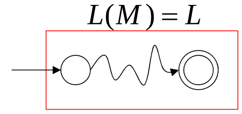
Proof – Part 2

2. For any regular language L there is a regular expression r with $L(r) = L$

Proof by construction of regular expression

Since L is regular take the NFA M that accepts it

$L(M) = L$

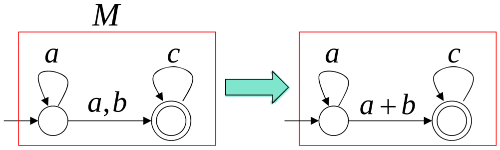


Single final state

From M construct the equivalent **Generalized Transition Graph** in which transition labels are regular expressions

Example:

M



Equivalence of RE and Finite Automata

Finite Automata and Regular Expressions are equivalent.

1. There is an algorithm for converting any RE into an NFA.
2. There is an algorithm for converting any NFA to a DFA.
3. There is an algorithm for converting any DFA to a RE.

These facts tell us that REs, NFAs and DFAs have equivalent expressive power. All three describe the class of regular languages.

Converting Regular Expressions to NFAs

RE to ε-NFAs

We can convert a Regular Expression to a finite automaton.

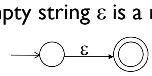
We can do this easiest by converting a RE to epsilon-NFA

Converting RE to ε-NFAs

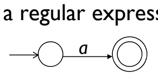
The regular expressions over finite Σ are the strings over the alphabet Σ such that:

- { } (empty set) is a regular expression for the empty set

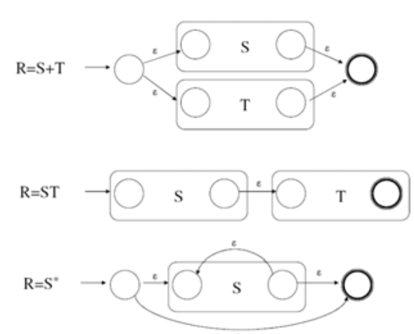
- Empty string ε is a regular expression denoting { ε }



- *a* is a regular expression denoting {*a* } for any *a* in Σ



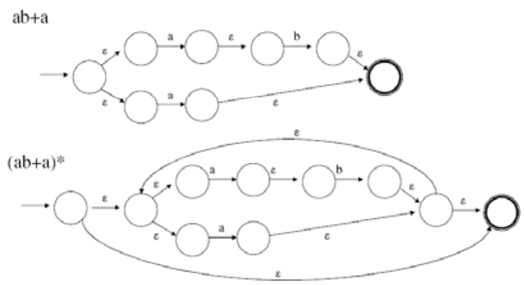
RE to ε-NFAs



RE to ε-NFAs

Example 1:

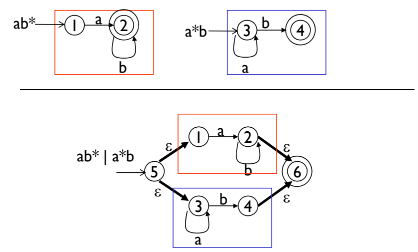
Convert ab+a and (ab+a)* into an NFA



RE to ε-NFAs

Example 2:

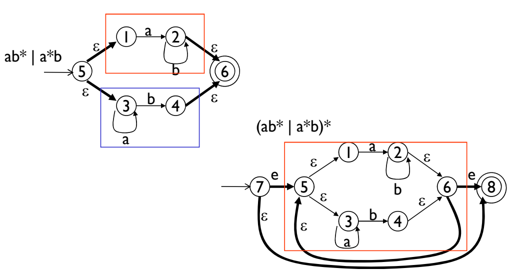
Convert (ab* | a*b)* into an NFA



RE to ε-NFAs

Example 2:

Convert (ab* | a*b)* into an NFA



Converting DFAs to Regular Expressions

Converting DFAs to REs

There are two FA to RE Construction Algorithms

- State Elimination
- Direct Substitution Method

Converting DFAs to REs

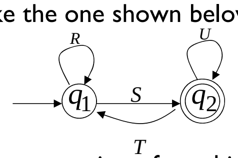
State Elimination Method:

1. Starting with intermediate states and then moving to accepting states, apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges.
   - The result will be one or two state automaton with a **Start** state and an **Accept** state.

DFA to RE: State Elimination

State Elimination Method:

2. If the two sates are different, we get an automaton like the one shown below:

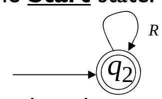


The regular expression for this automaton is (R+SU*T)*SU*

DFA to RE: State Elimination

State Elimination Method:

3. If the **Start** state is also an accepting state, then we must also perform a state elimination from the original automaton that gets rid of every state but the **Start** state. This leads us to:



We can describe this automaton as a regular expression as R*

DFA to RE: State Elimination

State Elimination Method:

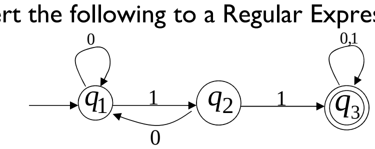
4. If there are n **Accept** states, the repeat steps 1-3 for each Accept state to get n different regular expressions R1, R2, …….Rn. For each repeat we turn any other **Accept** state to **non-Accept** state.

The final regular expression for the automaton is then the union of each of the n regular expressions

DFA to RE: Example 1

State Elimination:

Convert the following to a Regular Expression

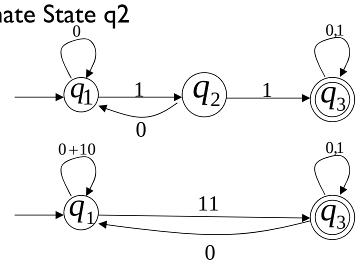


52

DFA to RE: Example 1

State Elimination:

- Eliminate State q2



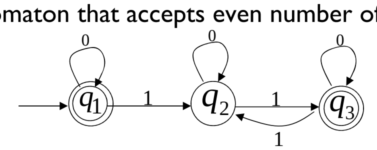
- The regular expression is (0+10)*11(0+1)*

53

DFA to RE: Example 2

State Elimination:

- Automaton that accepts even number of 1's

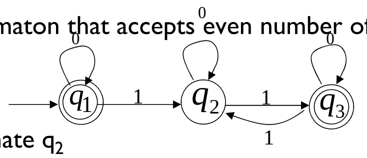


54

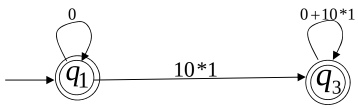
DFA to RE: Example 2

State Elimination:

- Automaton that accepts even number of 1's



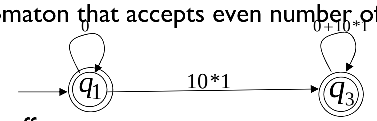
- Eliminate q_2



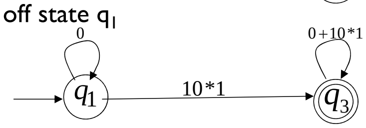
DFA to RE: Example 2

State Elimination:

- Automaton that accepts even number of 1's



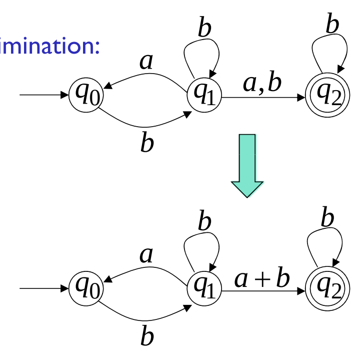
- Turn off state q_1



- The regular expression is 0*+0*10*1(0+10*1)*

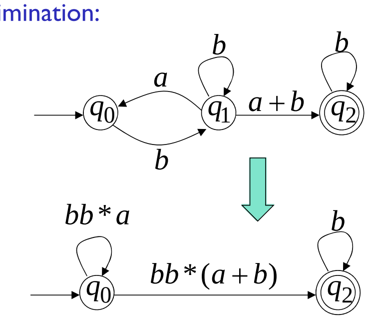
DFA to RE: Example 3

State Elimination:



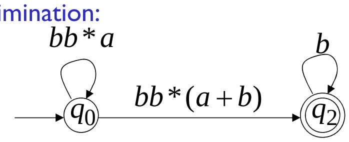
DFA to RE: Example 3

State Elimination:



DFA to RE: Example 3

State Elimination:

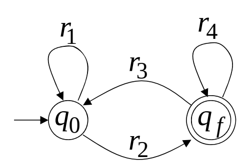


$$r = (bb*a)*bb*(a+b)b*$$

$$L(r) = L(M) = L$$

DFA to RE: Example 4

State Elimination:



The resulting regular expression:

$$r = r_1 * r_2 (r_4 + r_3 r_1 * r_2)*$$
$$L(r) = L(M) = L$$

DFA to RE

Inductive Construction:

Let A be a FA with states 1,2,…,n. Let $R_{ij}^{(k)}$ be a regular expression whose language is the set of labels of paths that go from state I to state j without passing through any state numbered above k.

DFA to RE

Inductive Construction:

Basis:

k = 0; Path can not go through any states

Thus, path is either an arc or the null path (a single node).

- If i ≠ j, then $R_{ij}^{(0)}$ is the sum of all symbols a such that A has a transition from i to j on symbol a (ϕ if none)
- If i = j, then add ε to above.

DFA to RE

Inductive Construction:

Induction:

- Assuming that correct expressions have been developed for the $R_{ij}^{(k-1)}$'s. Then for the $R_{ij}^{(k)}$'s

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

DFA to RE

Inductive Construction:

Proof: A path from i to j that goes through no state higher than k either:

1. Never goes through k, in which case the path's label is in the language of $R_{ij}^{(k-1)}$ or
2. Goes through k one or more times. In this case:
   - $R_{ik}^{(k-1)}$ contains the portion of the path that goes from i to k for the first time
   - $(R_{kk}^{(k-1)})^{*}$ contains the portion of the path (possibly empty) from the first k visit to the last.
   - $R_{kj}^{(k-1)}$ contains the portion of the path from the last k visit to j.

DFA to RE

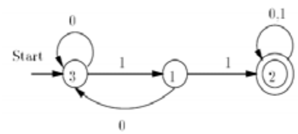
Inductive Construction:

Final Step: The RE for the entire FA is the sum (union) of the RE's R_{ij}^{n} , where i is the start state and j is one of the accepting states

DFA to RE: Example

Inductive Construction:

Consider the automaton given below



$R_{11}^{(0)} = \varepsilon$

$R_{12}^{(0)} = 1$

$R_{12}^{(0)} = \varepsilon + 0 + 1$

$R_{31}^{(0)} = 1$

$R_{32}^{(1)} = R_{32}^{(0)} + R_{31}^{(0)} {R_{11}^{(0)}}^{*} R_{12}^{(0)} = \phi + 1\varepsilon^{*}1 = 11$

$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} {R_{11}^{(0)}}^{*} R_{12}^{(0)} = \varepsilon + 0 + 1 + \phi\varepsilon^{*}1$

$\varepsilon + 0 + 1$

Converting ε-NFAs to DFAs

Converting ε-NFAs to DFAs

As we have seen earlier, each state in the new DFA will correspond to some set of states from the NFA. The DFA will be in state $\{s_0, s_1, \ldots\}$ after input if the NFA could be in *any* of these states for the same input.

Converting ε-NFAs to DFAs

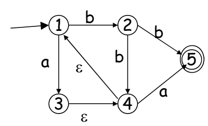
Epsilon Closure (ε-closure())

Epsilon Closure of a state is simply the set of all states we can reach by following the transition function from the given state that are labeled.

Converting ε-NFAs to DFAs

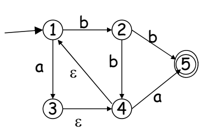
ε-closure(T) = T + all NFA states reachable from any state in T using only ε

Example: Convert the following ε -NFA into a DFA



Converting ε-NFAs to DFAs: Example 1

Subset Construction:

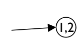


ε -closure({1}) = {1}
ε -closure({4}) = {1,4}
ε -closure({3}) = {1,3,4}

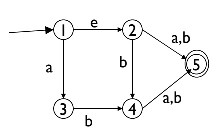
Converting ε-NFAs to DFAs: Example 1

Subset Construction:

DFA



NFA

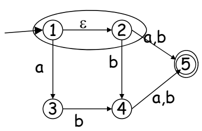


	a	b
{1,2}		

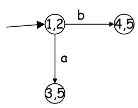
Converting ε-NFAs to DFAs: Example 1

Subset Construction:

NFA



DFA

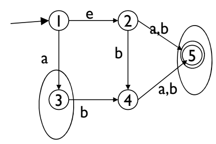


	a	b
{1,2}	{3,5}	{4,5}
{3,5}		
{4,5}		

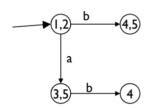
Converting ε-NFAs to DFAs: Example 1

Subset Construction:

NFA



DFA

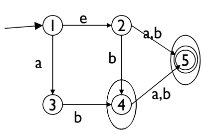


	a	b
{1,2}	{3,5}	{4,5}
{3,5}	-	{4}
{4,5}		
{4}		

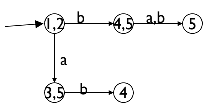
Converting ε-NFAs to DFAs: Example 1

Subset Construction:

NFA



DFA

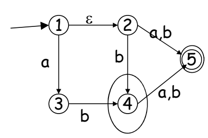


	a	b
{1,2}	{3,5}	{4,5}
{3,5}	-	{4}
{4,5}	{5}	{5}
{4}		
{5}		

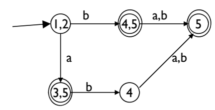
Converting ε-NFAs to DFAs: Example 1

Subset Construction:

NFA



DFA



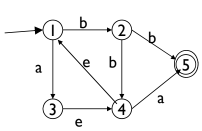
	a	b
{1,2}	{3,5}	{4,5}
{3,5}	-	{4}
{4,5}	{5}	{5}
{4}	{5}	{5}
{5}	-	-

All final states since the NFA final state is included

Converting ε-NFAs to DFAs: Example 2

Subset Construction:

NFA



DFA