Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2018

Lecture 8

Ana Bove

April 16th 2018

Recap: Non-deterministic Finite Automata (with ϵ-Transitions)

- Product of NFA as for DFA, accepting intersection of languages;
- Union of languages comes naturally, complement not so "immediate";
- By allowing ϵ-transitions we obtain ϵ-NFA:
  - Defined by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$;
  - $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}ow(Q)$;
  - ECLOSE needed for $\hat{\delta}$;
  - Accept set of words x such that $\hat{\delta}(q_0, x) \cap F \neq \emptyset$;
  - Given a ϵ-NFA E we can convert it to a DFA D such that $\mathcal{L}(E) = \mathcal{L}(D)$;
  - Hence, also accept the so called regular language.

Overview of Today's Lecture

- Regular expressions;
- Brief on algebraic laws for regular expressions;
- Equivalence between FA and RE: from FA to RE.

Contributes to the following learning outcome:

- Explain and manipulate the different concepts in automata theory and formal languages;
- Have a clear understanding about the equivalence between (non-)deterministic finite automata and regular expressions;
- Understand the power and the limitations of regular languages and context-free languages;
- Design automata, regular expressions and context-free grammars accepting or generating a certain language;
- Describe the language accepted by an automata or generated by a regular expression or a context-free grammar;
- Determine if a certain word belongs to a language;
- Differentiate and manipulate formal descriptions of languages, automata and grammars.

Regular Expressions

Regular expressions (RE) are an *algebraic* way to denote languages.

RE are a simple way to express the words in a language.

Example: `grep` command in UNIX (K. Thompson) takes a (variation) of a RE as input.

We will show that RE are as expressive as DFA and hence, they define all and only the *regular languages*.

Inductive Definition of Regular Expressions

Definition: Given an alphabet Σ, we inductively define the *regular expressions* over Σ as follows:

Base cases:
- The constants $\emptyset$ and ϵ are RE;
- If $a \in \Sigma$ then a is a RE.

Inductive steps: Given the RE R and S, then
- $R + S$ and RS are RE;
- R^* is RE.

The precedence of the operands is the following:

- The closure operator * has the highest precedence;
- Next comes concatenation;
- Finally, comes the operator $+$;
- We use parentheses (,) to change the precedence.

(Compare with exponentiation, multiplication and addition on numbers.)

Another Way to Define the Regular Expressions

Another way to define the regular expressions is by giving the following BNF (Backus-Naur Form), for $a \in \Sigma$:

$$R ::= \emptyset \mid \epsilon \mid a \mid R + R \mid RR \mid R^*$$

alternatively

$$R, S ::= \emptyset \mid \epsilon \mid a \mid R + S \mid RS \mid R^*$$

Note: BNF is a way to declare the syntax of a language.

It is very useful when describing *context-free grammars* and in particular the syntax of (big parts of) most programming languages.

Functional Representation of Regular Expressions

```
data RExp a = Empty | Epsilon | Atom a |
              Plus (RExp a) (RExp a) |
              Concat (RExp a) (RExp a) |
              Star (RExp a)
```

For example the expression $b + (bc)^*$ is given as

```
Plus (Atom "b") (Star (Concat (Atom "b") (Atom "c")))
```


Language Defined by the Regular Expressions

Definition: Given a RE R, the *language* $\mathcal{L}(R)$ generated/defined by it is defined by recursion on the expression:

Base cases:
- $\mathcal{L}(\emptyset) = \emptyset$;
- $\mathcal{L}(\epsilon) = \{\epsilon\}$;
- Given $a \in \Sigma$, $\mathcal{L}(a) = \{a\}$.

Recursive cases:
- $\mathcal{L}(R + S) = \mathcal{L}(R) \cup \mathcal{L}(S)$;
- $\mathcal{L}(RS) = \mathcal{L}(R)\mathcal{L}(S)$;
- $\mathcal{L}(R^*) = \mathcal{L}(R)^*$.

Note: $x \in \mathcal{L}(R)$ iff x is generated by R.

Notation: We write $x \in \mathcal{L}(R)$ or $x \in R$ indistinctly.

Example of Regular Expressions

Let $\Sigma = \{0, 1\}$:

- $0^* + 1^* = \{\epsilon, 0, 00, 000, \ldots\} \cup \{\epsilon, 1, 11, 111, \ldots\}$
- $(0 + 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots\}$
- $(01)^* = \{\epsilon, 01, 0101, 010101, \ldots\}$
- $(000)^* = \{\epsilon, 000, 000000, 000000000, \ldots\}$
- $01^* + 1 = \{0, 01, 011, 0111, \ldots\} \cup \{1\}$
- $((0(1^*)) + 1) = \{0, 01, 011, 0111, \ldots\} \cup \{1\}$
- $(01)^* + 1 = \{\epsilon, 01, 0101, 010101, \ldots\} \cup \{1\}$
- $(\epsilon + 1)(01)^*(\epsilon + 0) = (01)^* + 1(01)^* + (01)^*0 + 1(01)^*0$
- $(01)^* + 1(01)^* + (01)^*0 + 1(01)^*0 \ = \ldots$

What do they mean? Are there expressions that are equivalent?

Algebraic Laws for Regular Expressions (more on this next lecture)

The following equalities hold for any RE R, S and T:

Idempotent:	$R + R = R$	
Commutative:	$R + S = S + R$	In general, $RS \neq SR$
Associative:	$R + (S + T) = (R + S) + T$	$R(ST) = (RS)T$
Distributive:	$R(S + T) = RS + RT$	$(S + T)R = SR + TR$
Identity:	$R + \emptyset = \emptyset + R = R$	$R\epsilon = \epsilon R = R$
Annihilator:	$R\emptyset = \emptyset R = \emptyset$	
	$\emptyset^* = \epsilon^* = \epsilon$	
	$R^+ = RR^* = R^*R$	
	$R^* = (R^*)^* = R^*R^* = \epsilon + R^+$	

Note: Compare (some of) these laws with those for sets on slide 14 lecture 2.

More Algebraic Laws for Regular Expressions (more on this next lecture)

Other useful laws to simplify regular expressions are:

- *Shifting rule:* $R(SR)^* = (RS)^*R$
- *Denesting rule:* $(R^*S)^*R^* = (R+S)^*$

  Note: By the shifting rule we also get $R^*(SR^*)^* = (R+S)^*$
- Variation of the denesting rule: $(R^*S)^* = \epsilon + (R+S)^*S$

Note: These rules are not always trivial to apply ... :-)

Regular Languages and Regular Expressions

Theorem: *If $\mathcal{L}$ is a regular language then there exists a RE R such that $\mathcal{L} = \mathcal{L}(R)$.*

Proof: Recall that each regular language has a FA that recognises it.

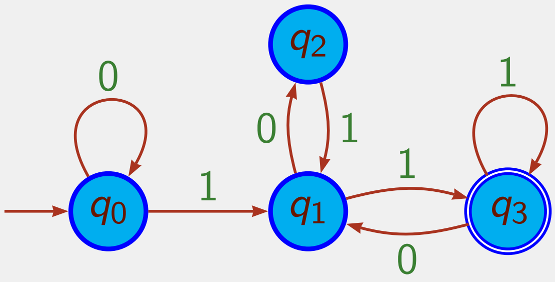
We shall construct a RE from such automaton.

We shall see 2 ways of constructing a RE from a FA:

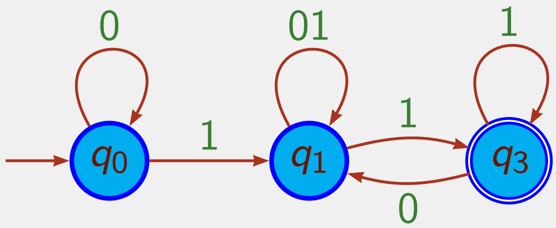
- Eliminating states (section 3.2.2);
- By solving a *linear equation system* using Arden's Lemma.

  (**OBS:** not in the book!)

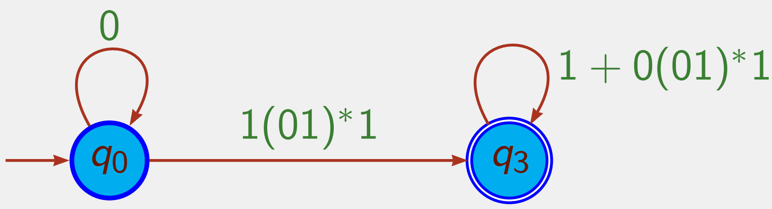
Example: From FA to RE by Eliminating States



If we remove q_2
we should keep all paths
going through it



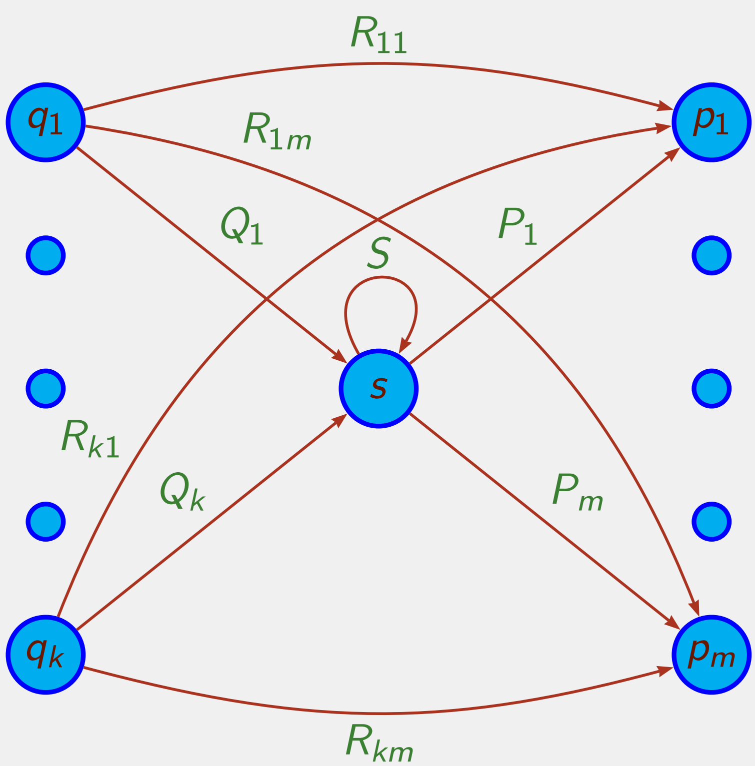
If we remove q_1
we should keep all paths
going through it



Final RE: $0^*1(01)^*1(1 + 0(01)^*1)^*$.

From FA to RE: Eliminating States in an Automaton A

Let the FA A be:



If an arc does not exist in A, then it is labelled $\emptyset$ here.

For simplification, we assume the q's are different from the p's.

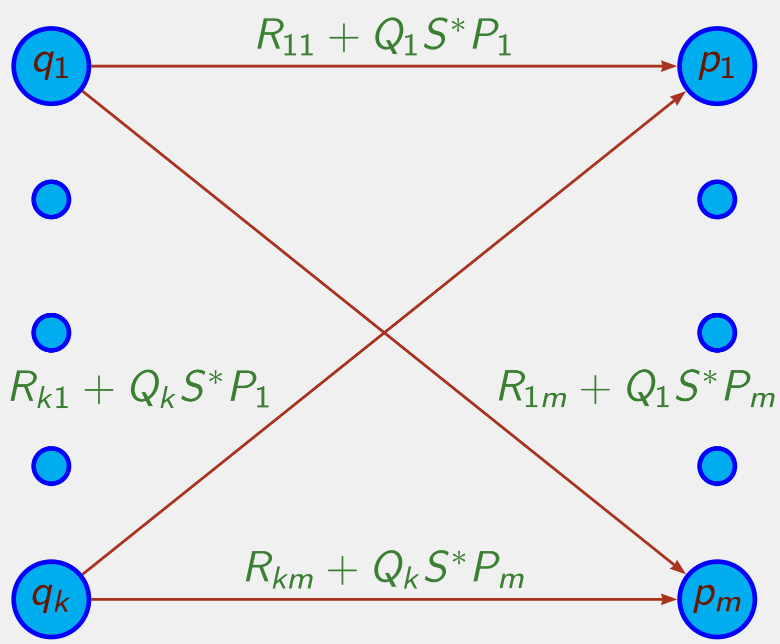
From FA to RE: Eliminating State s in A

When we eliminate the state s, all the paths that went through s do not longer exists!

To preserve the language of the automaton we must include, on an arc that goes directly from q to p, the labels of the paths that went from q to p passing through s.

Labels now are not just symbols but (possible an infinite number of) strings: hence we will use RE as labels.

From FA to RE: Eliminating State s in A

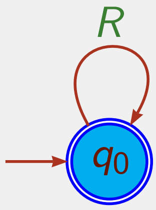


From FA to RE: Eliminating States in A

For *each accepting* state q we eliminate states until we have q_0 and q left.

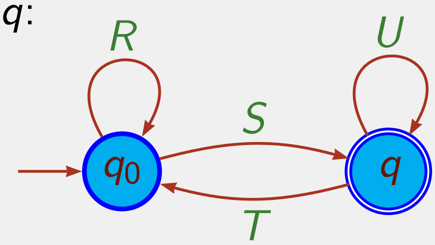
For each accepting state q we have 2 cases: $q_0 = q$ or $q_0 \neq q$.

If $q_0 = q$:



The expression is R^*.

If $q_0 \neq q$:

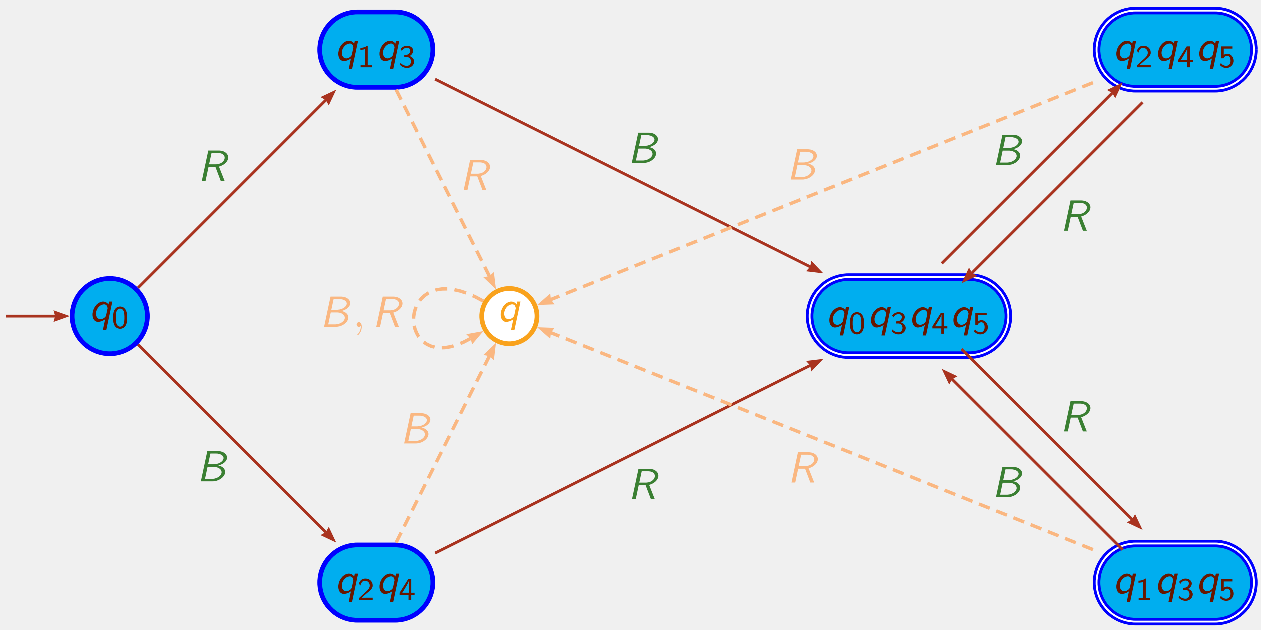


The expression is $(R + SU^*T)^*SU^*$.

The final RE is the *sum of the expressions derived for each final state*.

Example: RE Representing Gilbreath's Principle

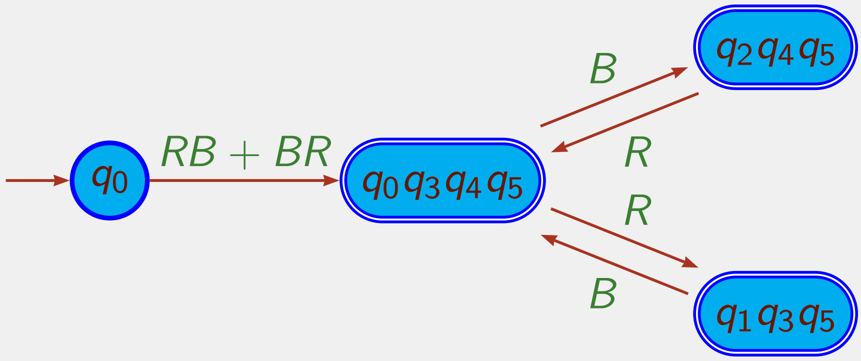
Recall:



Observe: Eliminating q is trivial. Eliminating q_1q_3 and q_2q_4 is also easy.

Example: RE Representing Gilbreath's Principle

After eliminating q, $q_1 q_3$ and $q_2 q_4$ we get:



- RE when final state is $q_0 q_3 q_4 q_5$:
$$(RB + BR)(RB + BR)^* = (RB + BR)^+$$
- RE when final state is $q_2 q_4 q_5$: $(RB + BR)(RB)^* B(R(RB)^* B)^*$
- RE when final state is $q_1 q_3 q_5$: $(RB + BR)(BR)^* R(B(BR)^* R)^*$

Example: RE Representing Gilbreath's Principle

The final RE is the sum of the 3 previous expressions.

Let us first do some simplifications.

$$(RB + BR)(RB)^* B(R(RB)^* B)^* = (RB + BR)(RB)^*(BR(RB)^*)^* B \quad \text{by shifting}$$
$$= (RB + BR)(RB + BR)^* B \quad \text{by the shifted-denesting rule}$$
$$= (RB + BR)^+ B$$

Similarly $(RB + BR)(BR)^* R(B(BR)^* R)^* = (RB + BR)^+ R$.

Hence the final RE is

$$(RB + BR)^+ + (RB + BR)^+ B + (RB + BR)^+ R$$

which is equivalent to

$$(RB + BR)^+(\epsilon + B + R)$$

From FA to RE: Linear Equation System

To any FA we associate a system of equations with REs as solution.

To every state q_i we associate a variable E_i.

Each E_i represents the set $\{x \in \Sigma^* \mid \hat{\delta}(q_i, x) \in F\}$ (for DFA).

Then E_0 represents the set of words accepted by the FA.

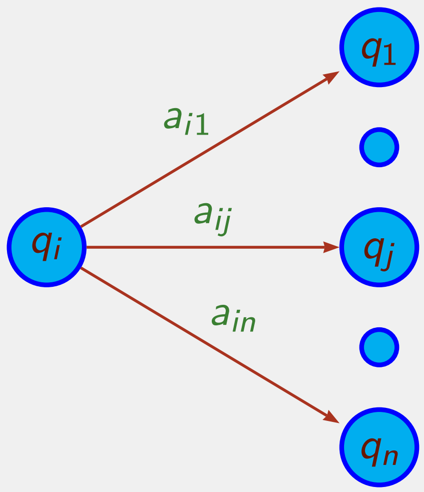
The solution to the linear system of equations associates a RE to each variable E_i.

The solution for E_0 is the RE generating the same language that is accepted by the FA.

From FA to RE: Constructing the Linear Equation System

Consider a state q_i and all the transactions coming out of it:

If there is no arrow coming out of q_i
then $E_i = \emptyset$ if q_i is not final
or $E_i = \epsilon$ if q_i is final



Here we have the equation

$$E_i = a_{i1}E_1 + \ldots + a_{ij}E_j + \ldots + a_{in}E_n$$

If q_i is final then we add ϵ

$$E_i = \epsilon + a_{i1}E_1 + \ldots + a_{ij}E_j + \ldots + a_{in}E_n$$

From FA to RE: Solving the Linear Equation System

Lemma: *(Arden) A solution to $X = RX + S$ is $X = R^*S$. Furthermore, if $\epsilon \notin \mathcal{L}(R)$ then this is the only solution to the equation $X = RX + S$.*

Proof: (sketch) We have that $R^* = RR^* + \epsilon$.

Hence $R^*S = RR^*S + S$ and then $X = R^*S$ is a solution to $X = RX + S$.

One should also prove that:

- Any solution to $X = RX + S$ contains at least R^*S;
- If $\epsilon \notin \mathcal{L}(R)$ then R^*S is the only solution to the equation $X = RX + S$ (that is, no solution is "bigger" than R^*S).

See for example Theorem 6.1, pages 185–186 of *Theory of Finite Automata, with an introduction to formal languages* by John Carroll and Darrell Long, Prentice-Hall International Editions.

Example: RE Representing Automaton in Slide 12

$$E_0 = 0E_0 + 1E_1 \quad E_1 = 0E_2 + 1E_3$$
$$E_2 = 0E_x + 1E_1 \quad E_3 = 0E_1 + 1E_3 + \epsilon \quad E_x = (0+1)E_x$$

We solve E_x: $E_x = (0+1)^*\emptyset = \emptyset$

We eliminate E_x and E_2: $E_0 = 0E_0 + 1E_1 \qquad E_1 = 01E_1 + 1E_3$
$E_3 = 0E_1 + 1E_3 + \epsilon$

We solve E_1: $E_1 = (01)^*1E_3$

We eliminate E_1: $E_0 = 0E_0 + 1(01)^*1E_3 \qquad E_3 = 0(01)^*1E_3 + 1E_3 + \epsilon$

We solve E_3:
$E_3 = (0(01)^*1 + 1)E_3 + \epsilon \Rightarrow E_3 = (0(01)^*1 + 1)^*\epsilon = (0(01)^*1 + 1)^*$

We eliminate E_3: $E_0 = 0E_0 + 1(01)^*1(0(01)^*1 + 1)^*$

We solve E_0: $E_0 = 0^*1(01)^*1(0(01)^*1 + 1)^*$

Example: RE Representing Gilbreath's Principle

We obtain the following system of equations (see slide 17):

$$\begin{aligned} E_0 &= RE_{13} + BE_{24} & E_{0345} &= \epsilon + BE_{245} + RE_{135} \\ E_{13} &= BE_{0345} + RE_q & E_{245} &= \epsilon + RE_{0345} + BE_q \\ E_{24} &= RE_{0345} + BE_q & E_{135} &= \epsilon + BE_{0345} + RE_q \\ & & E_q &= (B + R)E_q \end{aligned}$$

Since $E_q = (B + R)^*\emptyset = \emptyset$, this can be simplified to:

$$\begin{aligned} E_0 &= RE_{13} + BE_{24} & E_{0345} &= \epsilon + BE_{245} + RE_{135} \\ E_{13} &= BE_{0345} & E_{245} &= \epsilon + RE_{0345} \\ E_{24} &= RE_{0345} & E_{135} &= \epsilon + BE_{0345} \end{aligned}$$

Example: RE Representing Gilbreath's Principle

And further to:

$$\begin{aligned} E_0 &= (RB + BR)E_{0345} \\ E_{0345} &= (RB + BR)E_{0345} + \epsilon + B + R \end{aligned}$$

Then a solution to E_{0345} is

$$(RB + BR)^*(\epsilon + B + R)$$

and the RE which is the solution to the problem is

$$(RB + BR)(RB + BR)^*(\epsilon + B + R)$$

or

$$(RB + BR)^+(\epsilon + B + R)$$

Overview of Next Lecture

Sections 3.2.3, 3.4, 4–4.2.1, and notes on *Pumping lemma*:

- Equivalence between FA and RE: from RE to FA;
- More on algebraic laws for regular expressions;
- Pumping Lemma for RL;
- Closure properties of RL.