Finite Automata and Formal Languages

TMV026/DIT321 – LP4 2010

Lecture 6

April 15th 2010

Overview of today's lecture:

- **Regular Expressions**
- **From Finite Automata to Regular Expressions**

Regular Expressions

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

Given a RE R, it defines the language $\mathcal{L}(R)$.

Actually, they can be seen as a simpler representation of ϵ-NFA.

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

RE can also be seen as a declarative way to express the strings we want to accept and serve as input language for certain systems.

Example: `grep` command in UNIX (K. Thompson).

Inductive Definition of Regular Expressions

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

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

Inductive cases: Given the RE R and S, we define the following RE:
- $R + S$ 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 precedences

Another Way to Define the Regular Expressions

A nicer 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 most programming languages.

Example of Regular Expressions

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

- $(01)^*$
- $0^* + 1^*$
- $(0 + 1)^*$
- $(000)^*$
- $01^* + 1$
- $((0(1^*)) + 1)$
- $(01)^* + 1$
- $(\epsilon + 1)(01)^*(\epsilon + 0)$
- $(01)^* + 1(01)^* + (01)^*0 + 1(01)^*0$

Can you guess their meaning? Are there expressions that are equivalent?

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")))
```


Recall: Some Operations on Languages (Lecture 2)

Definition: Given $\mathcal{L}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ languages then we define the following languages:

Union: $\mathcal{L}_1 \cup \mathcal{L}_2 = \{x \mid x \in \mathcal{L}_1 \text{ or } x \in \mathcal{L}_2\}$

Intersection: $\mathcal{L}_1 \cap \mathcal{L}_2 = \{x \mid x \in \mathcal{L}_1 \text{ and } x \in \mathcal{L}_2\}$

Concatenation: $\mathcal{L}_1\mathcal{L}_2 = \{x_1x_2 \mid x_1 \in \mathcal{L}_1,\ x_2 \in \mathcal{L}_2\}$

Closure: $\mathcal{L}^* = \bigcup_{n\in\mathbb{N}} \mathcal{L}^n$
where $\mathcal{L}^0 = \{\epsilon\}$, $\mathcal{L}^{n+1} = \mathcal{L}^n\mathcal{L}$.

Note: We have then that $\emptyset^* = \{\epsilon\}$ and
$\mathcal{L}^* = \mathcal{L}^0 \cup \mathcal{L}^1 \cup \mathcal{L}^2 \cup \ldots = \{\epsilon\} \cup \{x_1 \ldots x_n \mid n > 0, x_i \in \mathcal{L}\}$

Notation: $\mathcal{L}^+ = \mathcal{L}^1 \cup \mathcal{L}^2 \cup \mathcal{L}^3 \cup \ldots$ and $\mathcal{L}? = \mathcal{L} \cup \{\epsilon\}$.

Language Defined by the Regular Expressions

Definition: The ***language*** defined by a regular expression is defined by induction on the expression:

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\epsilon) = \{\epsilon\}$
- Given $a \in \Sigma$, $\mathcal{L}(a) = \{a\}$
- $\mathcal{L}(R + S) = \mathcal{L}(R) \cup \mathcal{L}(S)$
- $\mathcal{L}(RS) = \mathcal{L}(R)\mathcal{L}(S)$
- $\mathcal{L}(R^*) = \mathcal{L}(R)^*$

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

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

Algebraic Laws for Regular Expressions

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

- Associativity: $R + (S + T) = (R + S) + T$ and $R(ST) = (RS)T$
- Commutativity: $R + S = S + R$
- In general, $RS \neq SR$
- Distributivity: $R(S + T) = RS + RT$ and $(S + T)R = SR + TR$
- Identity: $R + \emptyset = \emptyset + R = R$ and $R\epsilon = \epsilon R = R$
- Annihilator: $R\emptyset = \emptyset R = \emptyset$
- Idempotent: $R + R = R$
- $\emptyset^* = \epsilon^* = \epsilon$
- $R? = \epsilon + R$
- $R^+ = RR^* = R^*R$
- $R^* = (R^*)^* = R^*R^* = \epsilon + R^+$

Algebraic Laws for Regular Expressions

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: $(R^*S)^* = \epsilon + (R + S)^*S$

Example: Proving Equalities Using the Algebraic Laws

Example: A proof that $a^*b(c + da^*b)^* = (a + bc^*d)^*bc^*$:

$$
\begin{array}{ll}
a^*b(c + da^*b)^* = a^*b(c^*da^*b)^*c^* & \text{by denesting } (R = c, S = da^*b) \\
a^*b(c^*da^*b)^*c^* = (a^*bc^*d)^*a^*bc^* & \text{by shifting } (R = a^*b, S = c^*d) \\
(a^*bc^*d)^*a^*bc^* = (a + bc^*d)^*bc^* & \text{by denesting } (R = a, S = bc^*d)
\end{array}
$$

Example: The set of all words with no substring of more than two adjacent 0's is $(1 + 01 + 001)^*(\epsilon + 0 + 00)$. Now,

$$
\begin{aligned}
(1 + 01 + 001)^*(\epsilon + 0 + 00) &= ((\epsilon + 0)(\epsilon + 0)1)^*(\epsilon + 0)(\epsilon + 0) \\
&= (\epsilon + 0)(\epsilon + 0)(1(\epsilon + 0)(\epsilon + 0))^* \qquad \text{by shifting} \\
&= (\epsilon + 0 + 00)(1 + 10 + 100)^*
\end{aligned}
$$

Then $(1 + 01 + 001)^*(\epsilon + 0 + 00) = (\epsilon + 0 + 00)(1 + 10 + 100)^*$

Equality of Regular Expressions

Remember that RE are a way to denote languages.

Then, for RE R and S, $R = S$ actually means $\mathcal{L}(R) = \mathcal{L}(S)$.

Hence we can prove the equality of RE in the same way we can prove the equality of languages.

Example: Let us prove that $R^* = R^*R^*$. Let $\mathcal{L} = \mathcal{L}(R^*)$.

$\mathcal{L}^* \subseteq \mathcal{L}^*\mathcal{L}^*$ since $\epsilon \in \mathcal{L}^*$.

Conversely, if $\mathcal{L}^*\mathcal{L}^* \subseteq \mathcal{L}^*$ then $x = x_1x_2$ with $x_1 \in \mathcal{L}^*$ and $x_2 \in \mathcal{L}^*$.

If $x_1 = \epsilon$ or $x_2 = \epsilon$ then it is clear that $x \in \mathcal{L}^*$.

Otherwise $x_1 = u_1u_2 \ldots u_n$ with $u_i \in \mathcal{L}$ and $x_2 = v_1v_2 \ldots v_m$ with $v_j \in \mathcal{L}$.

Then $x = x_1x_2 = u_1u_2 \ldots u_nv_1v_2 \ldots v_m$ is in $\mathcal{L}^*$.

Proving Algebraic Laws for Regular Expressions

Given the RE R and S we can prove the law $R = S$ as follows:

1. Convert R and S into *concrete* regular expressions C and D, respectively, by replacing each variable in the RE R and S by (different) concrete symbols.
   Example: $R(SR)^* = (RS)^*R$ can be converted into $a(ba)^* = (ab)^*a$.
2. Prove or disprove whether $\mathcal{L}(C) = \mathcal{L}(D)$. If $\mathcal{L}(C) = \mathcal{L}(D)$ then $R = S$ is a true law, otherwise it is not.

Theorem: *The above procedure correctly identifies the true laws for RE.*

Proof: See theorems 3.14 and 3.13 in pages 121 and 120 respectively.

Example: Proving the shifting law was (somehow) one of the exercises in assignment 1: prove that for all n, $a(ba)^n = (ab)^n a$.

Example: Proving the Denesting Rule

We can state $(R^*S)^*R^* = (R+S)^*$ by proving $\mathcal{L}((a^*b)^*a^*) = \mathcal{L}((a+b)^*)$:

$\subseteq$: Let $x \in (a^*b)^*a^*$, then $x = vw$ with $v \in (a^*b)^*$ and $w \in a^*$.

By induction on v.

If $v = \epsilon$ we are done. Otherwise $v = av'$ or $v = bv'$.

Observe that in both cases $v' \in (a^*b)^*$ hence by IH $v'w \in (a+b)^*$ and so is vw.

$\supseteq$: Let $x \in (a+b)^*$. By induction on x. If $x = \epsilon$ then done.

Otherwise $x = x'a$ or $x = x'b$ and $x' \in (a+b)^*$.

By IH $x' \in (a^*b)^*a^*$ and then $x' = vw$ with $v \in (a^*b)^*$ and $w \in a^*$.

If $x'a = v(wa) \in (a^*b)^*a^*$ since $v \in (a^*b)^*$ and $(wa) \in a^*$.

If $x'b = (v(wb))\epsilon \in (a^*b)^*a^*$ since $v(wb) \in (a^*b)^*$ and $\epsilon \in a^*$.

Regular Languages and Regular Expressions

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

Proof: Recall that each regular language has an automata that recognises it.

We shall construct a regular expression from such automata.

The book shows 2 ways of constructing a regular expression from an automata (sections 3.2.1 –computing $R_{ij}^{(k)}$– and 3.2.2. –eliminating states–).

From FA to RE: Computing $R_{ij}^{(k)}$ from an Automaton A

Let $Q_A = \{1, 2, \ldots, n\}$ with 1 being the initial state.

We construct a collection of RE that progressively describe the paths in the transition diagram of A:

> Let $R_{ij}^{(k)}$ be the RE whose language is the set of strings w which label a path from state i to state j in A without passing by an intermediate state bigger than k.
> Note that neither i nor j are intermediate states!

We define $R_{ij}^{(k)}$ by induction on k.

If $F_A = \{f_1, \ldots, f_r\}$ then our final regular expression is

$$R_{1f_1}^{(n)} + \ldots + R_{1f_r}^{(n)}$$

Base Case: $R_{ij}^{(0)}$

We have no intermediate states here! We have the following scenarios:

- Arcs from state i to j?:
  - If there are no arc then $R_{ij}^{(0)} = \emptyset$
  - If there is one arc labelled a then $R_{ij}^{(0)} = a$
  - If there are m arcs labelled $a_1, \ldots, a_m$ then $R_{ij}^{(0)} = a_1 + \ldots + a_m$

  Note: If $i = j$ then we must consider the loops from i to itself.

- We have a path of length 0 from i to itself.

  In a ϵ-NFA we can also have paths of length 0 between i and j.

  Such a path is represented as an ϵ-transition in the automaton and as the RE ϵ.

  Then we need to add ϵ to the corresponding case above, obtaining then $R_{ij}^{(0)} = \epsilon$, $R_{ij}^{(0)} = \epsilon + a$ or $R_{ij}^{(0)} = \epsilon + a_1 + \ldots + a_m$ respectively.

Inductive Step: from $R_{ij}^{(k)}$ to $R_{ij}^{(k+1)}$

Given a path from state i to state j without passing by an intermediate state bigger than $(k+1)$, we have 2 possible cases:

- The path does not actually pass by state $(k+1)$.

  Hence the label of the path is in the language of the RE $R_{ij}^{(k)}$.

- The path goes through $(k+1)$ at least once.

  We can break the path into pieces that do not pass through $k+1$: first from i to $(k+1)$, one or more from $(k+1)$ to $(k+1)$, last from $(k+1)$ to j.

  The label for this path is represented by the RE $R_{i(k+1)}^{(k)}(R_{(k+1)(k+1)}^{(k)})^* R_{(k+1)j}^{(k)}$.

The resulting RE is $R_{ij}^{(k+1)} = R_{ij}^{(k)} + R_{i(k+1)}^{(k)}(R_{(k+1)(k+1)}^{(k)})^* R_{(k+1)j}^{(k)}$

Remarks on the Method for Computing $R_{ij}^{(k)}$

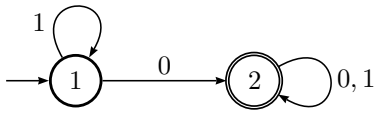
- Works for any kind of FA (DFA, NFA and ϵ-NFA).
- The method is similar to Floyd-Warshall algorithm (graph analysis algorithm for finding shortest paths in a weighted, directed graph).

  See Wikipedia.
- It is expensive: we need to compute n^2 RE!

  It also produces very big and complicated expressions!

  The (intermediate) RE can usually be simplified. Still not trivial!

  Example: See example 3.5 in the book (pages 95-97).



From FA to RE: Eliminating States in an Automaton A

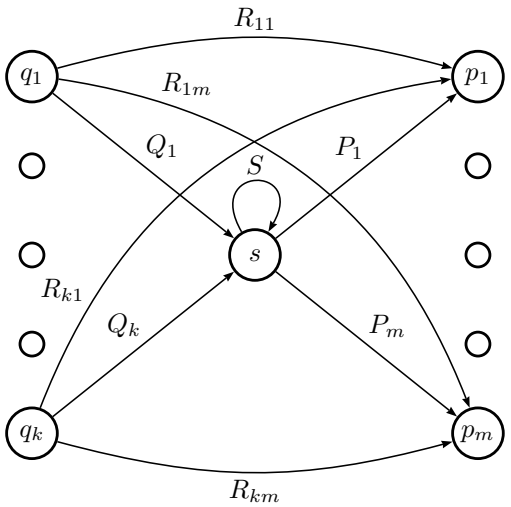
This method of constructing a RE from a FA involves eliminating states.

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 longer symbols but (possible an infinite number of) strings: hence we will use RE as labels.

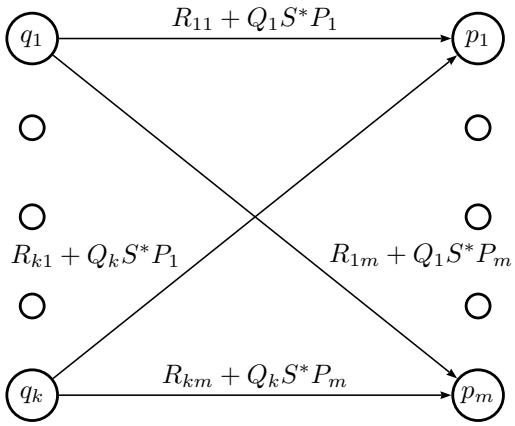
Eliminating State s in A



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

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

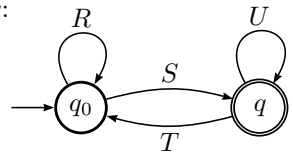
Eliminating State s in A



Eliminating States in A

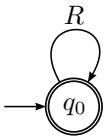
For each accepting state q we proceed as before until we have only q_0 and q left. For each q we have 2 cases: $q_0 \neq q$ or $q_0 = q$.

If $q_0 \neq q$:



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

If $q_0 = q$:

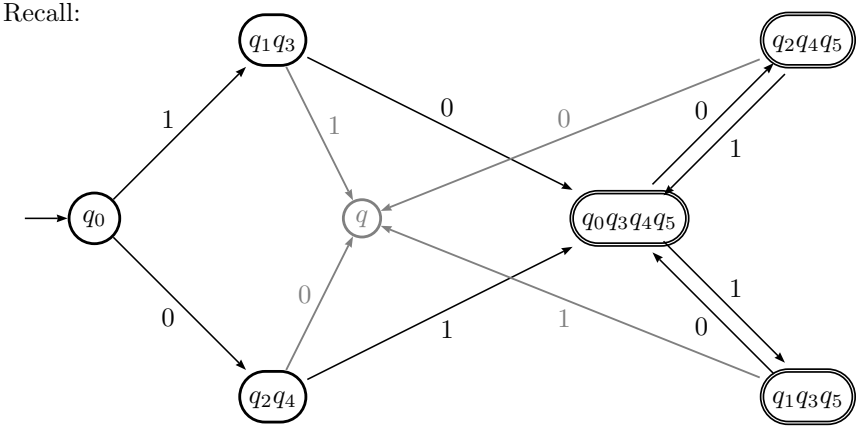


The expression is R^*

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

Example: Regular Expression Representing Gilbreath's Principle

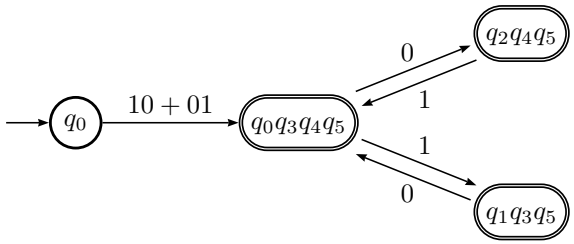
Recall:



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

Example: Regular Expression Representing Gilbreath's Principle

After eliminating q, q_1q_3 and q_2q_4 we get:



- RE when final state is $q_0q_3q_4q_5$: $(10+01)(10+01)^* = (10+01)^+$
- RE when final state is $q_2q_4q_5$: $(10+01)(10)^*0(1(10)^*0)^*$
- RE when final state is $q_1q_3q_5$: $(10+01)(01)^*1(0(01)^*1)^*$

Example: Regular Expression Representing Gilbreath's Principle

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

Let us first do some simplifications.

$$\begin{aligned}(10+01)(10)^*0(1(10)^*0)^* &= (10+01)(10)^*(01(10)^*)^*0 \quad \text{by shifting}\\ &= (10+01)(10+01)^*0 \quad \text{by the shifted-denesting rule}\\ &= (10+01)^+0\end{aligned}$$

Similarly $(10+01)(01)^*1(0(01)^*1)^* = (10+01)^+1$.

Hence the final RE is

$$(10+01)^+ + (10+01)^+0 + (10+01)^+1$$

which is equivalent to

$$(10+01)^+(\epsilon+0+1)$$