Finite Automata and Formal Languages

TMV026/DIT321– LP4 2012

Lecture 7
Ana Bove

March 27th 2012

Overview of today's lecture:

- Regular Expressions
- From FA to RE

Regular Expressions

Regular expressions (RE) are an "algebraic" way to denote languages.
Given a RE R, it defines the language $\mathcal{L}(R)$.

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).
(**Note:** UNIX regular expressions are not exactly as the RE we will study in the course.)

Inductive Definition of Regular Expressions

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

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, 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

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

Question: Can you guess their meaning?

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.

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 3)

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 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/accepted by R.

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

Example of Regular Expressions

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

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

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

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: $R + R = R$;
- $\emptyset^* = \epsilon^* = \epsilon$;
- $R? = \epsilon + R$;
- $R^+ = RR^* = R^*R$;
- $R^* = (R^*)^* = R^*R^* = \epsilon + R^+$.

Note: Compare this slide with slide 19 of lecture 3.

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 $a^*b(c + da^*b)^* = (a + bc^*d)^*bc^*$:

$$a^*b(c + da^*b)^* = a^*b(c^*da^*b)^*c^* \quad \text{by denesting } (R = c, S = da^*b)$$
$$a^*b(c^*da^*b)^*c^* = (a^*bc^*d)^*a^*bc^* \quad \text{by shifting } (R = a^*b, S = c^*d)$$
$$(a^*bc^*d)^*a^*bc^* = (a + bc^*d)^*bc^* \quad \text{by denesting } (R = a, S = bc^*d)$$

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

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

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 we are 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.

- Computing $R_{ij}^{(k)}$ (section 3.2.1): too expensive, produces big and complicated regular expressions;
- Eliminating states (section 3.2.2).

We will also see how to do this by solving a *linear equation system* using Arden's Lemma.

From FA to RE: Eliminating States in an Automaton A

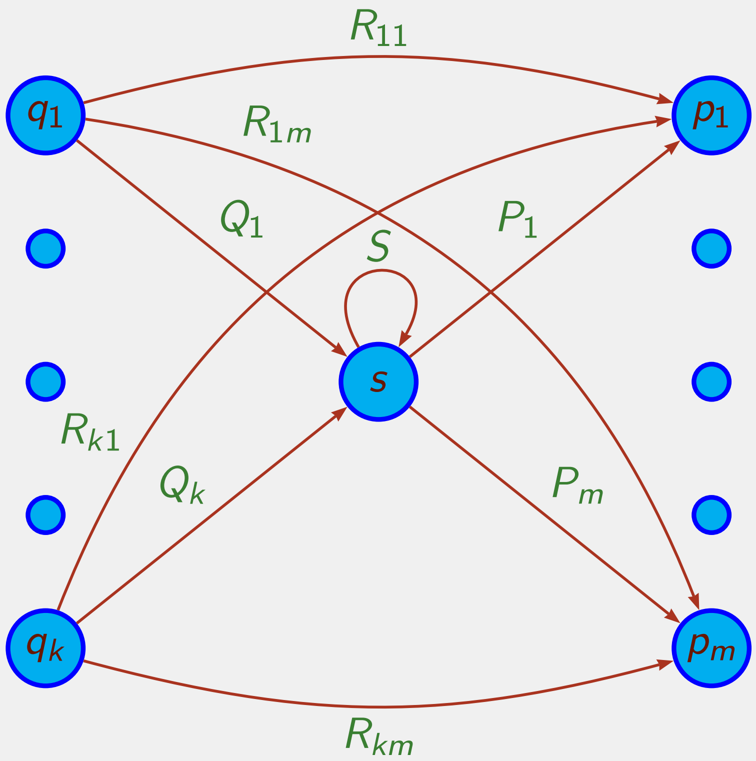
This method of constructing a RE from a FA involves eliminating states.

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

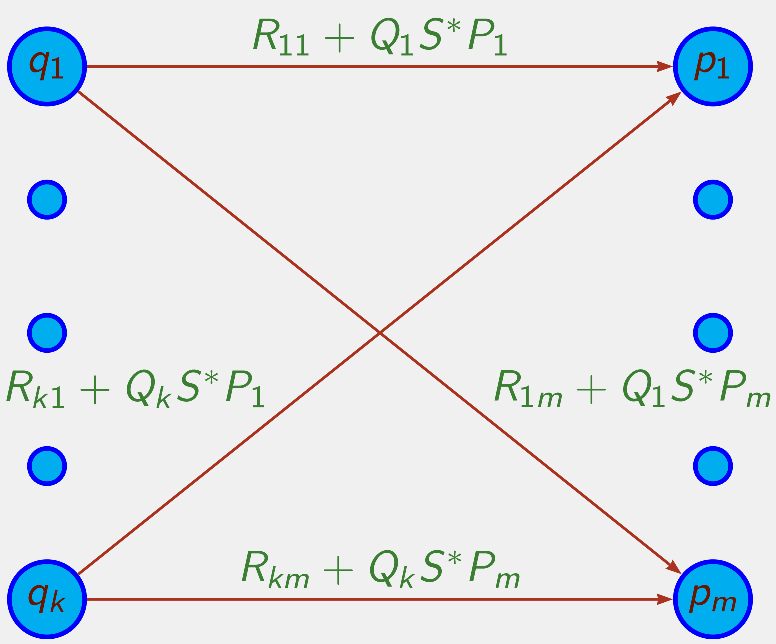
Eliminating State s in A



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.

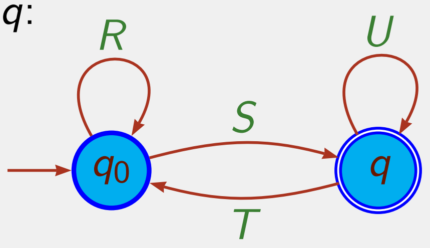
Eliminating State s in A



Eliminating States in A

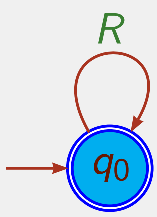
For *each accepting* state q we proceed as before until we have only q_0 and q left. For each accepting state q we have 2 cases: $q_0 \neq q$ or $q_0 = q$.

If $q_0 \neq q$:



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

If $q_0 = q$:

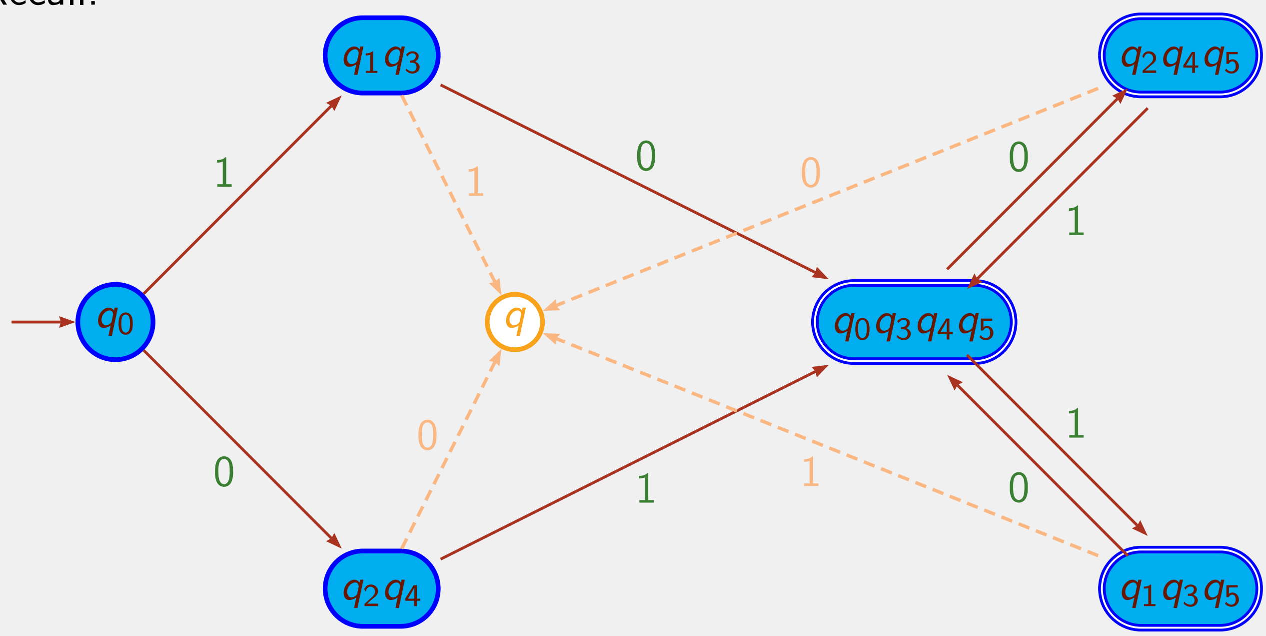


The expression is R^*

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

Example: Regular Expression Representing Gilbreath's Principle

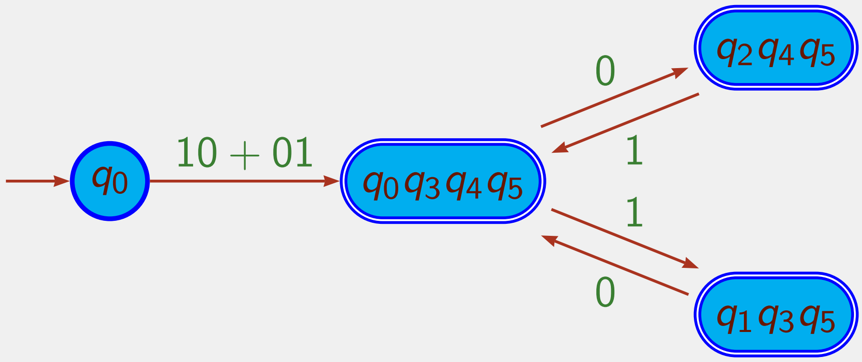
Recall:



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.

$(10+01)(10)^*0(1(10)^*0)^* = (10+01)(10)^*(01(10)^*)^*0$ by shifting

$= (10+01)(10+01)^*0$ by the shifted-denesting rule

$= (10+01)^+0$

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

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

From FA to RE: Linear Equation System

To any automaton we associate a system of equations such that the solution will be regular expressions.

At the end we get a regular expression for the language recognised by the automaton. This works for DFA, NFA and ϵ-NFA.

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).

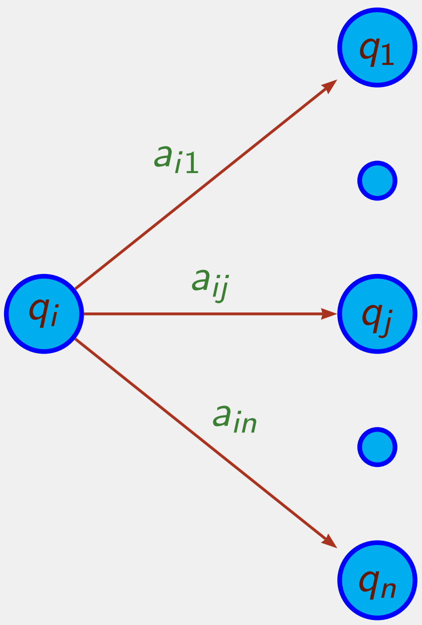
Then E_0 represents the set of words accepted by the FA.

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

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

Constructing the Linear Equation System

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



Then we have the equation

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

If E_i is final then we add ϵ

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

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

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: 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).

Note: 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: Regular Expression Representing Gilbreath's Principle

We obtain the following system of equations (see slide 19):

$$
\begin{aligned}
E_0 &= 1E_{13} + 0E_{24} & E_{0345} &= \epsilon + 0E_{245} + 1E_{135} \\
E_{13} &= 0E_{0345} + 1E_q & E_{245} &= \epsilon + 1E_{0345} \\
E_{24} &= 1E_{0345} + 0E_q & E_{135} &= \epsilon + 0E_{0345} \\
& & E_q &= \emptyset
\end{aligned}
$$

This can be simplified to:

$$
\begin{aligned}
E_0 &= 1E_{13} + 0E_{24} & E_{0345} &= \epsilon + 0E_{245} + 1E_{135} \\
E_{13} &= 0E_{0345} & E_{245} &= \epsilon + 1E_{0345} \\
E_{24} &= 1E_{0345} & E_{135} &= \epsilon + 0E_{0345}
\end{aligned}
$$

Example: Regular Expression Representing Gilbreath's Principle

And further to:

$$E_0 = (10 + 01)E_{0345}$$
$$E_{0345} = (10 + 01)E_{0345} + \epsilon + 0 + 1$$

Then a solution to E_{0345} is

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

and the RE which is the solution to the problem is

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

or

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