Equivalence of DFAs and REs

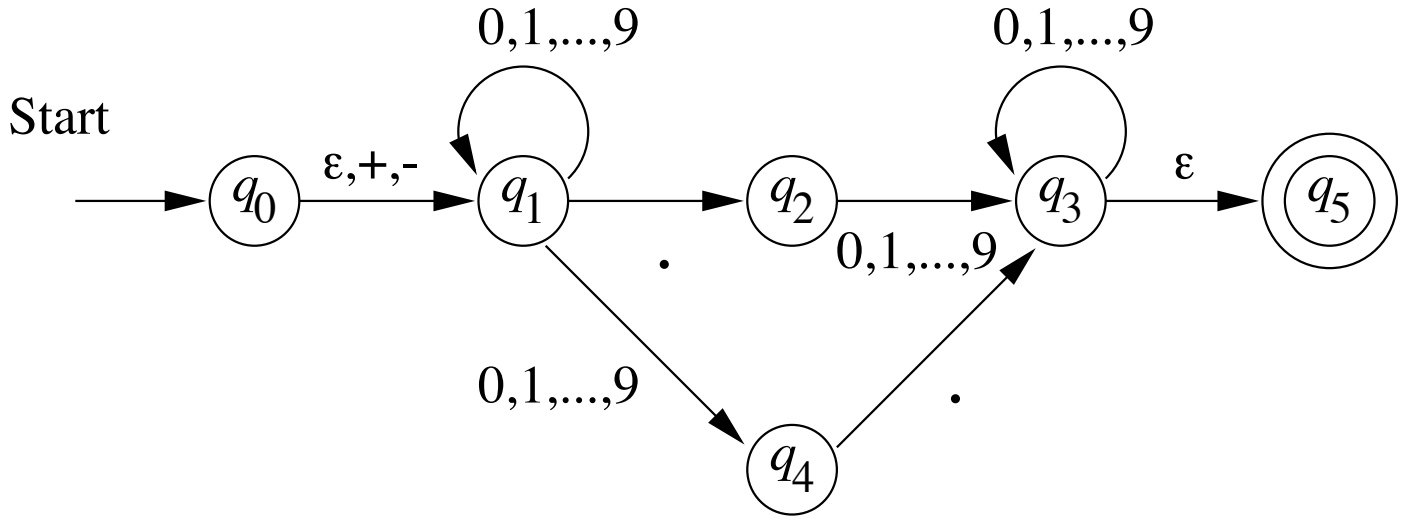
- Theorem: A language is regular if and only if some regular expression describes it.
- We prove this theorem by showing that FAs are equivalent to REs, i.e., by proving the following lemmas:
  - For every DFA A we can construct a RE R, such that $L(R) = L(A)$.
  - For every RE R we can construct ϵ-NFA A, such that $L(A) = L(R)$.

From DFA to RE

For every DFA $A = (Q, \Sigma, \delta, q_0, F)$ there is an RE R, s.t. $L(R) = L(A)$.

Proof idea: we propose a method for constructing a RE that describes strings recognized by A.

Recall from an earlier example how we constructed an RE from the ϵ-NFA below.



From DFA to RE

A method for building an RE that describes strings recognized by a DFA:

- We build REs that describe strings that label paths in the DFA's state transition diagram.
- Induction on the number of states a path can go through
  - **basis**: the paths cannot go through any state.
  - **Induction**: Paths can go through progressively larger subsets of states.
- By applying this induction, we end up generating REs that represent **all possible paths.**
- The final RE is the union of all REs labelling paths that connect **initial and final states.**

From DFA to RE

For every DFA $A = (Q, \Sigma, \delta, q_0, F)$ there is a RE R, s.t. $L(R) = L(A)$.

Proof: Le the states of A be $\{1, 2, 3, \cdots, n\}$, with 1 being the start state.

- Let $R_{ij}^{(k)}$ be a regular expression describing the set of labels of all paths in A from state i to state j that have no intermediate node whose number is greater than k.
- Notice that k is a constraint on which states a path can go through.

Inductively defining $R_{ij}^{(k)}$

$R_{ij}^{(k)}$ will be defined inductively. Note that

$$L\left(\biguplus_{j\in F} {R_{1j}}^{(n)}\right) = L(A), \text{ where } A = (Q, \Sigma, \delta, q_0, F)$$

Basis: $k = 0$, i.e., since states are labeled from 1, then the path has no intermediate states.

- Case 1: $i \neq j$

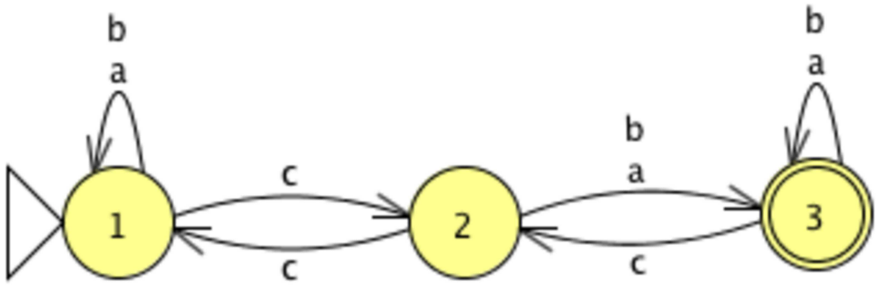
$$R_{ij}^{(0)} = \biguplus_{\{a\in\Sigma:\delta(i,a)=j\}} a$$

- Case 2: $i = j$

$$R_{ij}^{(0)} = \left(\biguplus_{\{a\in\Sigma:\delta(i,a)=j\}} a\right) + \epsilon$$

Example: Obtaining R_{ij}^0

Obtain $R_{ij}^{(0)}$ for the DFA below:



Inductively defining $R_{ij}^{(k)}$ (cnt.)

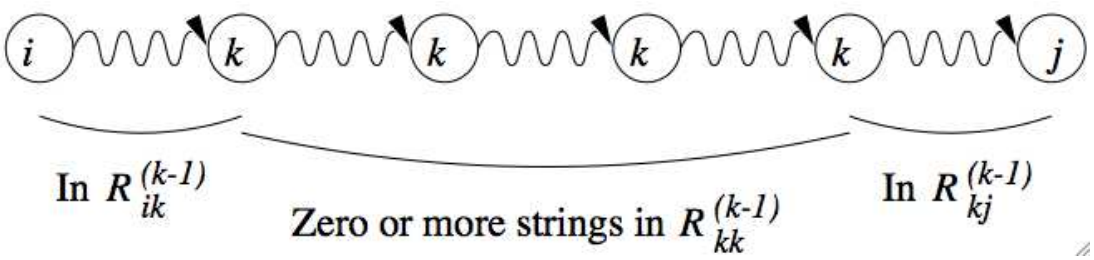
Induction: $k > 0$. Remember: the path goes through no states greater than k. Two cases to consider:

Path does not go through state k:

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} +$$

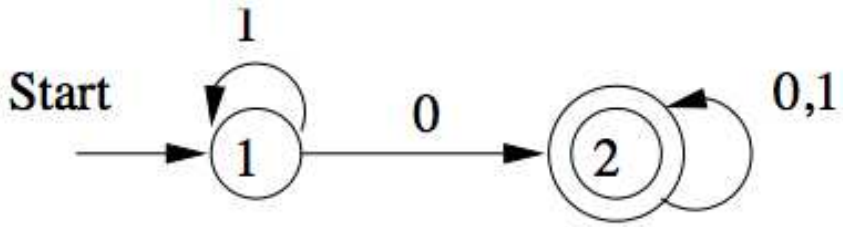
Path goes through k one or more times:

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



Example

Let's find a regular expression R for the following DFA A, where $L(A) = \{x0y : x \in \{1\}^* \text{ and } y \in \{0,1\}^*\}$



$R_{11}^{(0)}$	$\epsilon + 1$
$R_{12}^{(0)}$	0
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\epsilon + 0 + 1$

Example (cnt.)

We will need the following simplification rules:

- $(\epsilon + R)^* = R^*$
- $(\epsilon + R)R^* = R^*(\epsilon + R) = R^*$
- $R + RS^* = RS^*$, analogously $R + S^*R = S^*R$
- $\emptyset R = R\emptyset = \emptyset$ (Annihilation)
- $\emptyset + R = R + \emptyset = R$ (Identity)

Example (cnt.)

$R_{11}^{(0)}$	$\epsilon + 1$
$R_{12}^{(0)}$	$\mathbf{0}$
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\epsilon + 0 + 1$

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} \left(R_{11}^{(0)}\right)^* R_{1j}^{(0)}$$

	By direct substitution	Simplified
$R_{11}^{(1)}$	$\epsilon + 1 + (\epsilon + 1)(\epsilon + 1)^*(\epsilon + 1)$	1^*
$R_{12}^{(1)}$	$0 + (\epsilon + 1)(\epsilon + 1)^*0$	1^*0
$R_{21}^{(1)}$	$\emptyset + \emptyset(\epsilon + 1)^*(\epsilon + 1)$	$\emptyset$
$R_{22}^{(1)}$	$\epsilon + 0 + 1 + \emptyset(\epsilon + 1)^*0$	$\epsilon + 0 + 1$

Example (cnt.)

	Simplified
$R_{11}^{(1)}$	$\mathbf{1^*}$
$R_{12}^{(1)}$	$\mathbf{1^*0}$
$R_{21}^{(1)}$	$\emptyset$
$R_{22}^{(1)}$	$\epsilon + 0 + 1$

$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)} \left(R_{22}^{(1)}\right)^* R_{2j}^{(1)}$$

	By direct substitution
$R_{11}^{(2)}$	$1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset$
$R_{12}^{(2)}$	$1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$
$R_{21}^{(2)}$	$\emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset$
$R_{22}^{(2)}$	$\epsilon + 0 + 1 + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$

Example (cnt.)

	By direct substitution
$R_{11}^{(2)}$	$1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset$
$R_{12}^{(2)}$	$1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$
$R_{21}^{(2)}$	$\emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset$
$R_{22}^{(2)}$	$\epsilon + 0 + 1 + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$

	Simplified
$R_{11}^{(2)}$	1^*
$R_{12}^{(2)}$	$1^*0(0 + 1)^*$
$R_{21}^{(2)}$	$\emptyset$
$R_{22}^{(2)}$	$(0 + 1)^*$

Since $L\left(\biguplus_{j \in F} {R_{1j}}^{(n)}\right)$ denotes the language of the DFA A, then the final regex for A is $R_{12}^{(2)} = 1^*0(0 + 1)^*$

From DFA to RE: Observations

This is definitely an expensive construction method.

- Each inductive step grows the expression 4-fold, which implies $R_{ij}^{(n)}$'s size can be on the order of 4^n symbols.
- There is a more efficient approach: the state elimination technique.

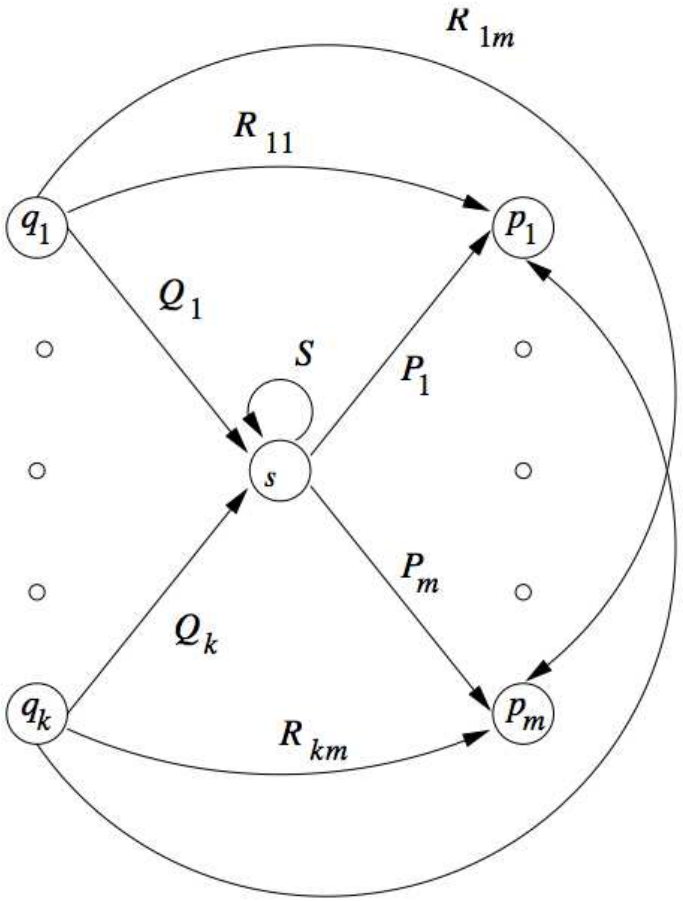
The state elimination technique

Intuition: suppose we have a DFA and want to eliminate a state s in between two states p and q:

- when we eliminate s, all paths that went through s no longer exist in the automaton.
- If the language is not to change, then we must include on an edge that goes from p to q the labels of all paths that went from p to q, through s.
- since there could be an infinite number of such strings, we can use an RE as a label for the path.

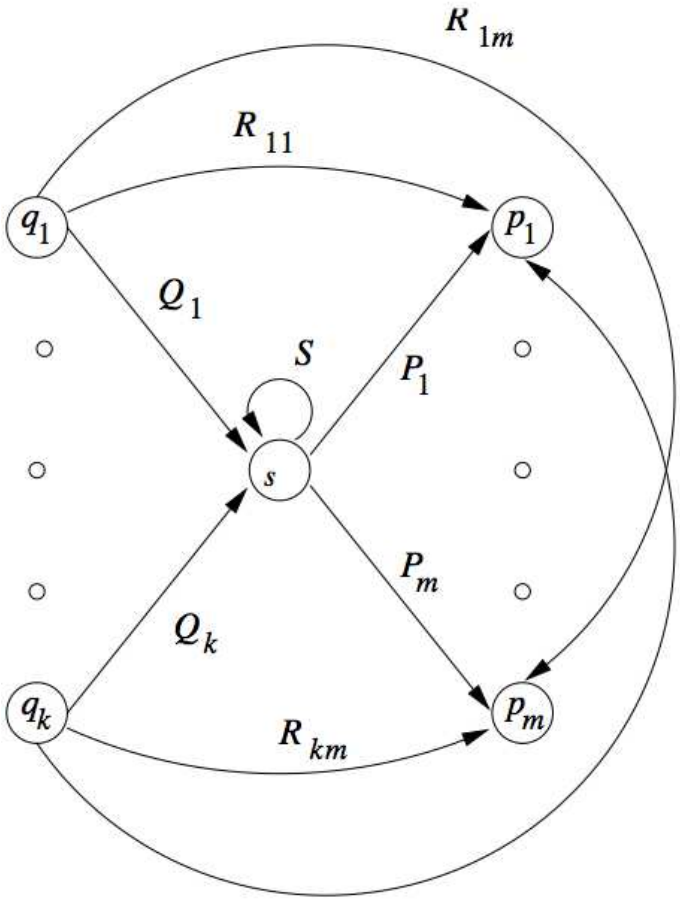
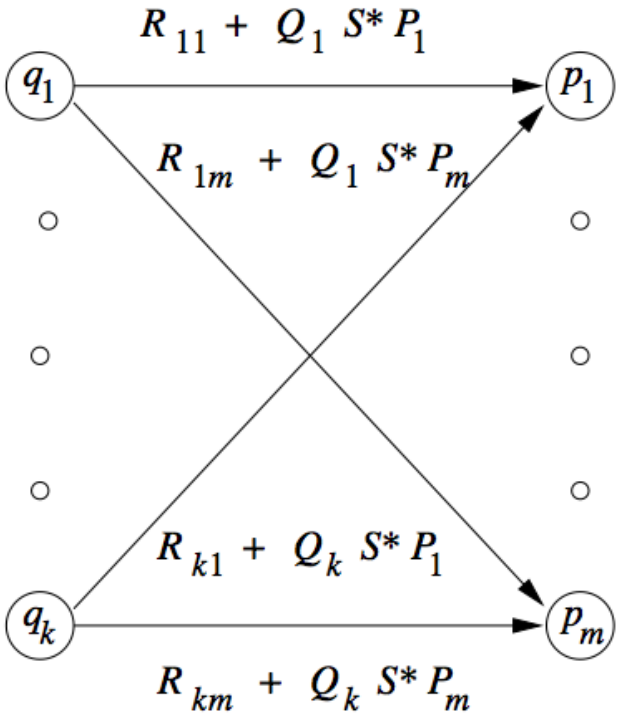
The state elimination technique

Assume in the automaton below the edges are labeled with REs instead of symbols. Suppose we have already eliminated some states and we are about to eliminate state s.



Example (cont.)

Eliminating a non-initial and non-final state:

Before	After
	

From an automaton M to a RE

1. Remove unreachable states from M.
2. If there are no accepting states, return $\emptyset$.
3. If the start state q_0 is part of a loop, create a new state q_0' and make $\delta(q_0', \epsilon) = q_0$.
4. If $|F| > 1$ or there are transitions going out from the final state, then create a new final state f and connect each of the final states in F to f via ϵ-transitions. Remove he old final states from F.
5. If at this point M has only one state, them that state is both initial and the accepting state and M has no transitions. So $L(M) = \{\epsilon\}$.
6. Next slide...

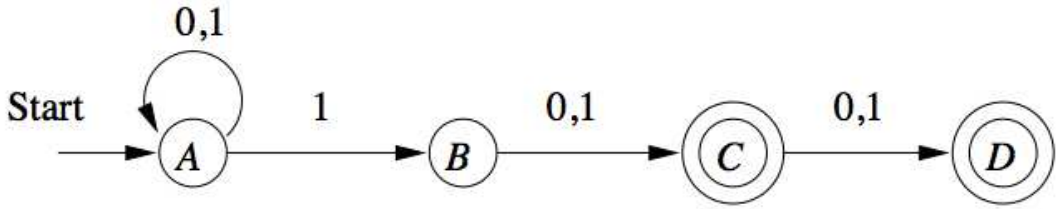
From an automaton M to a RE

6. Until only the start and the accepting state remain do:
   - 6.1 Select some non-final and non-accepting state q of M.
   - 6.2 Remove q from M.
   - 6.3 Modify the transitions rewriting the labels so that M accepts the same strings.
7. Return the one RE that labels the transition that goes from the initial state to the accepting state.

Example

Find a RE from the automaton A whose language is the set of binary strings that have a 1 either two or three positions from the end, *i.e.*,

$$L(A) = \{w : w = x1b \text{ or } w = x1bc, x \in \{0,1\}^*, b \in \{a,1\}, c \in \{0,1\}\}$$

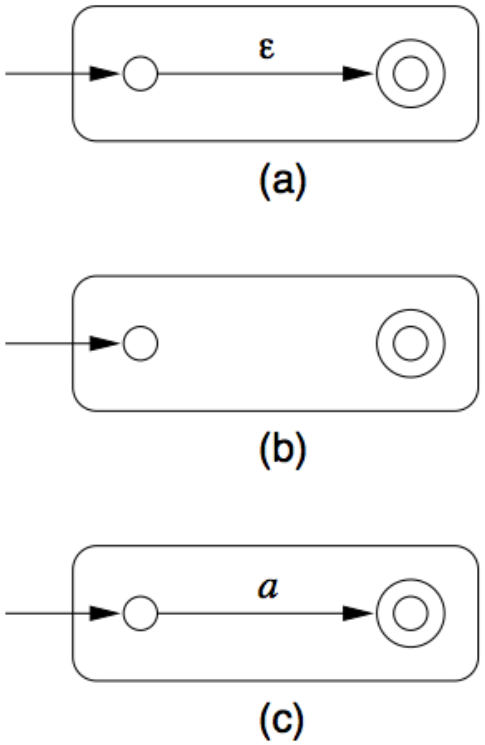


On the board.

From RE to ϵ-NFA

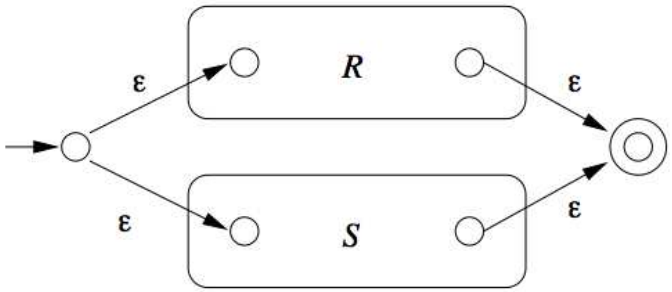
- **Theorem:** For every RE R we can construct and ϵ-NFA A, such that $L(A) = L(R)$.
- **Proof:** By structural induction:

Basis: Automata for ϵ, $\emptyset$, and a.

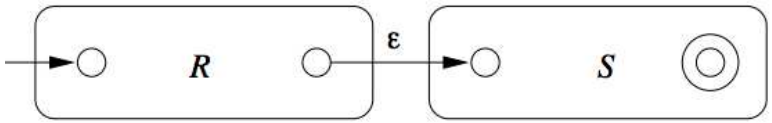


Proof (cont.)

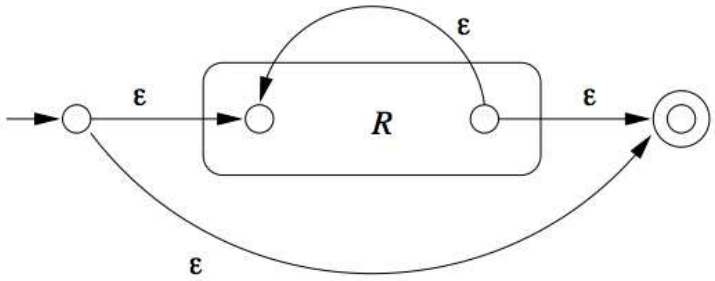
Induction: Automata for $R + S$, RS, and R^*



(a)



(b)



(c)

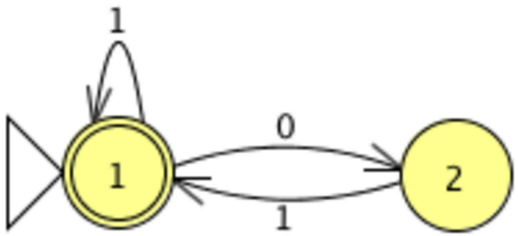
Example

- Convert the following RE to an ϵ-NFA.

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

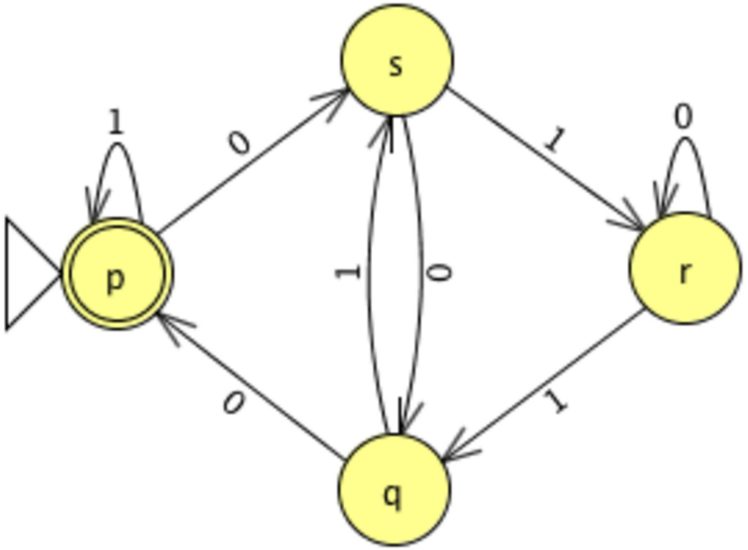
Exercise

- Given the following FA, use the recursive relation $R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1}(R_{kk}^{k-1})R_{kj}^{k-1}$ to compute $R_{ij}^{(0)}$ and $R_{ij}^{(1)}$, for $i = 1, 2, j = 1, 2$.



Exercise

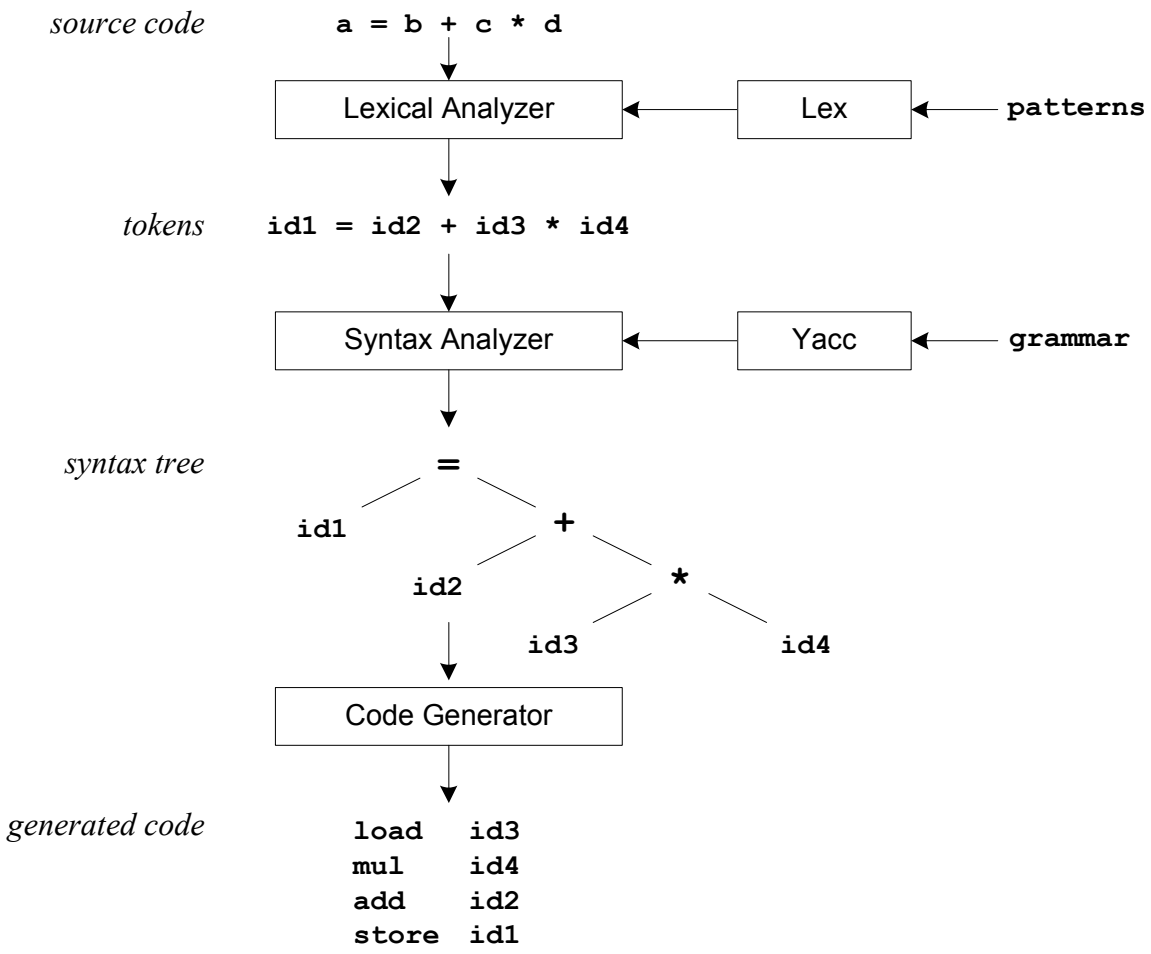
- Convert the following DFA to a regular expression using the state-elimination technique. Eliminate states in the following order: p, r, s, q.



- Do it again eliminating states r, s, q, p, in order, then verify if the resulting RE represents the same language described by the RE you obtained above.

Regular Expressions in the real world

- REs are used in the specification of a lexical analyser, an important component of a compiler.



Specifying patterns in Lex/Flex

Primitives

Metacharacter	Matches
`.`	any character except newline
`\n`	newline
`*`	zero or more copies of the preceding expression
`+`	one or more copies of the preceding expression
`?`	zero or one copy of the preceding expression
`^`	beginning of line
`$`	end of line
`a|b`	`a` or `b`
`(ab)+`	one or more copies of `ab` (grouping)
`"a+b"`	literal "`a+b`" (C escapes still work)
`[]`	character class

Examples

Expression	Matches
`abc`	`abc`
`abc*`	`ab abc abcc abccc ...`
`abc+`	`abc abcc abccc ...`
`a(bc)+`	`abc abcbc abcbcbc ...`
`a(bc)?`	`a abc`
`[abc]`	one of: `a`, `b`, `c`
`[a-z]`	any letter, a-z
`[a\-z]`	one of: `a`, `-`, `z`
`[-az]`	one of: `-`, `a`, `z`
`[A-Za-z0-9]+`	one or more alphanumeric characters
`[ \t\n]+`	whitespace
`[^ab]`	anything except: `a`, `b`
`[a^b]`	one of: `a`, `^`, `b`
`[a|b]`	one of: `a`, `|`, `b`
`a|b`	one of: `a`, `b`

Lex/Flex file structure

```
... definitions ...
%%
... rules ...
%%
... subroutines ...
```

Lex file example

```
digit  [0-9]
letter [A-Za-z]
%{
    int count;
%}
%%    /* match identifier */
{letter}({letter}|{digit})*   count++;
%%
int yywrap(void) {
  return 1;
}
int main(void) {
    yylex();
    printf("# of identigfiers = %d\n", count);
}
```