Regular Expressions

Regular Expressions

Regular expressions
describe regular languages

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

describes the language

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

- Regular expressions are closed-form symbolic way of capturing/describing a "regular" language

字母表：符号的有限集合

- Alphabet is a finite non-empty set of characters – ASCII, {0,1}, …

串：符号有穷序列

- A string is a sequence of characters from an alphabet – \empty, a, ab, abc01,…
- A language L is a set of strings over a finite alphabet

语言：字母表上的个串集

- Given a string s, does s belong to L?

- What is the input language of the program you have been asked to write?

- {add “street names” (1,2)…, add “street names’ , … mod “street name1”…, rm “street name”…, gg…}

- Regular expression matching problem
  - Given a regular expression RE and a string s, decide whether s \in RE
  - When you parse a string, the question being solved is whether the string is in proper format. Whether the parsing is being done as part of a compiler (format is described using a context-free grammar), or in our setting (the format is described using a regular expression), you can reduce it to a matching problem.

Recursion

- Fundamental idea behind recursion is the concept of self-reference or defining things/functions in terms of itself

- In recursion as used to define functions, the structure is something as follows:

- F(type *A* x) = …F(type *A* y)…
- In a meaningful sense, $|y| < |x|$
- The mathematical analogue of recursion is mathematical induction

- A set or language is a collection of strings.

- An empty set is an object that contains no strings.

- An empty string has the following properties:
  - \empty . s = s . \empty = s
  - Prefix p of a string s, is a string, such that \exists another s' wherein s = p.s'

Recursive Definition

Primitive regular expressions: $\underbrace{\emptyset}_{\text{empty set}}, \quad \underbrace{\lambda}_{\text{empty string}}, \quad \underbrace{\alpha}_{\text{character of alphabet}}$

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \quad \text{并集} \\ r_1 \cdot r_2 \\ r_1{}^* = r_1^{\wedge 0} + r_1^{\wedge 1} + r_1^{\wedge 2} + \cdots r_1^{\wedge k} \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

- The language of an empty set is empty. There are no strings in it. ϕ

not the same !

- The language L of the regular expression \empty is a set {\empty} (λ)

- \alpha refers to the characters of the alphabet over which a regular expression is defined.

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

R^2 = R.R = {a,bc} {a,bc} = {aa,abc,bca,bcbc}

Example

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

L(a+b.c) = {a, bc}

R* = R^0 + R^1 + R^2 + R^3 ....R^k + ...

- (a+b.c)
- (a+b).c or does it mean ((a) + (b.c)) = a+b.c

- (a-z)+ lower case of a-z

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

\Sigma = {a,b}, the set of all strings over \Sigma is denoted by \Sigma^* = (a+b)^*

(a,b)(a,b) = (aa, ab, ba, bb)

\Sigma^*.{a,bb} = {\lambda, a, b, aa, ab, ba, bb, aaa...}.{a,bb}

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

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 containing substring 00 }

Example

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

{λ, 1, 01, 101, 011, 11, 0101, ... } (0, λ)

always end with 1

$L(r)$ = { all strings without substring 00 }

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 substring 00 }

11, 10, 01.

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

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

$$L(r_1) = L(r_2) = L \Longrightarrow$$

r_1 and r_2 are equivalent regular expressions

Regular Expressions and Regular Languages

Theorem

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

Proof:

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

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

Proof - Part 1

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

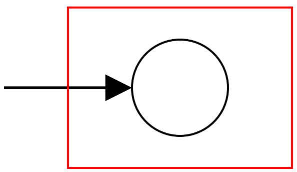
For any regular expression r
the language $L(r)$ is regular

Proof by induction on the structure of r

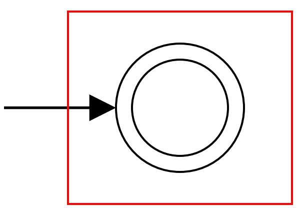
Induction Basis

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

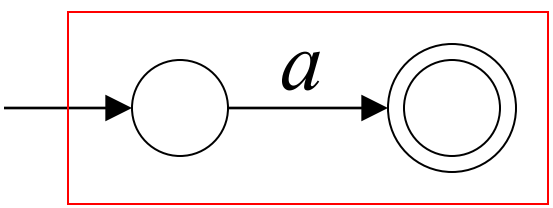
Corresponding NFAs



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



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



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

regular languages

Inductive Hypothesis

Suppose

that for regular expressions r_1 and r_2,

$L(r_1)$ and $L(r_2)$ are regular languages

Inductive Step

We will prove:

$$\left.\begin{array}{l} L(r_1 + r_2) \\ \\ L(r_1 \cdot r_2) \\ \\ L(r_1 *) \\ \\ L((r_1)) \end{array}\right\}$$

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:

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

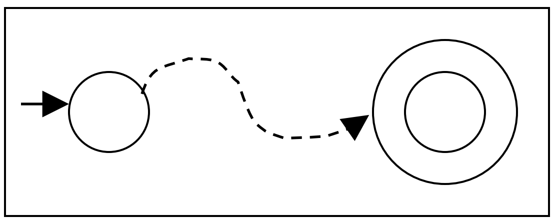
$L((r_1)) = L(r_1)$ is trivially a regular language (by induction hypothesis)

End of Proof-Part 1

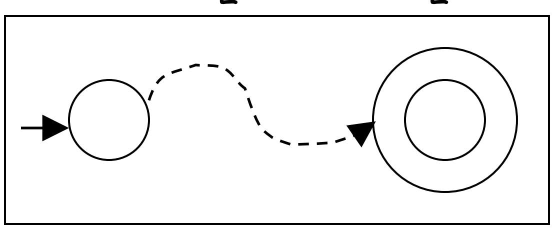
Using the regular closure of these operations, we can construct recursively the NFA M that accepts $L(M) = L(r)$

Example: $r = r_1 + r_2$

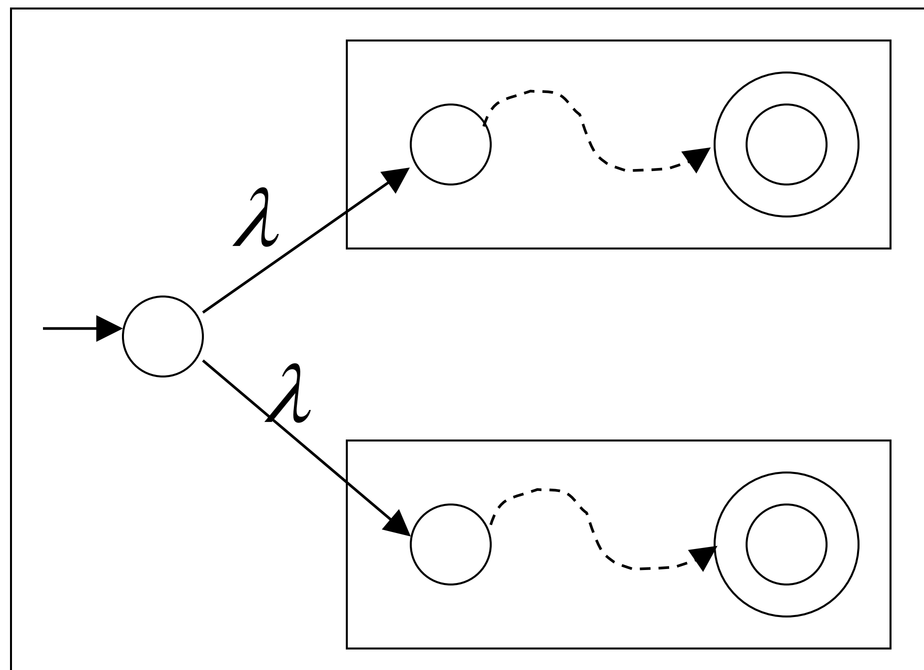
$L(M_1) = L(r_1)$



$L(M_2) = L(r_2)$



$L(M) = L(r)$



Proof - Part 2

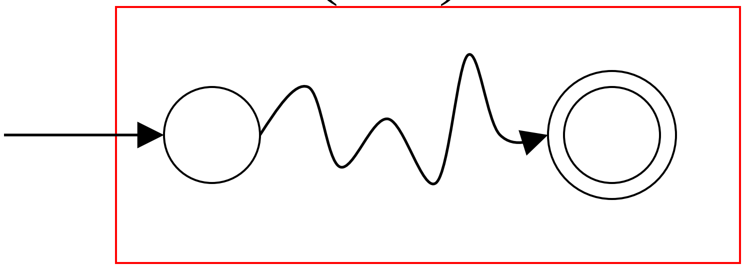
$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

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

We will convert an NFA that accepts L to a regular expression

Since L is regular, there is a NFA M that accepts it

$$L(M) = L$$



Take it with a single accept state

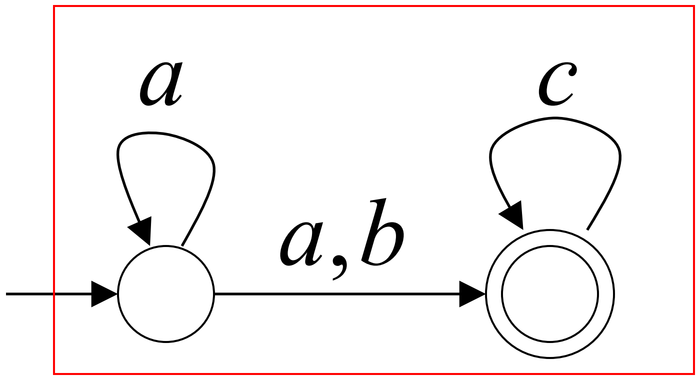
From M construct the equivalent

Generalized Transition Graph

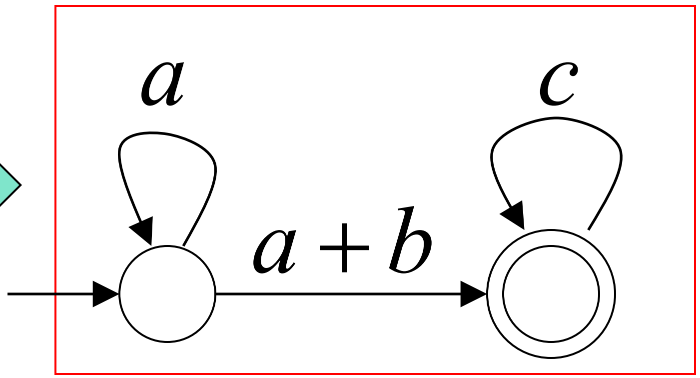
in which transition labels are regular expressions

Example:

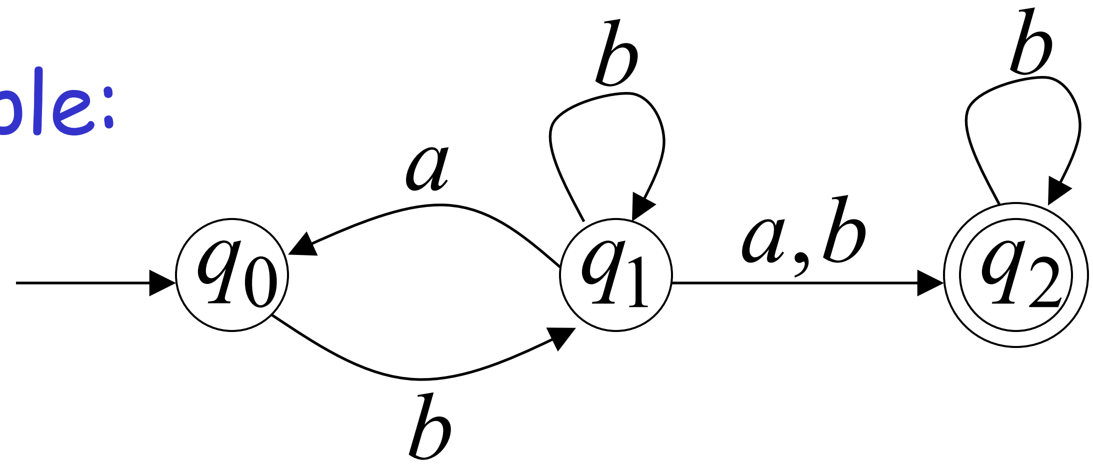
M



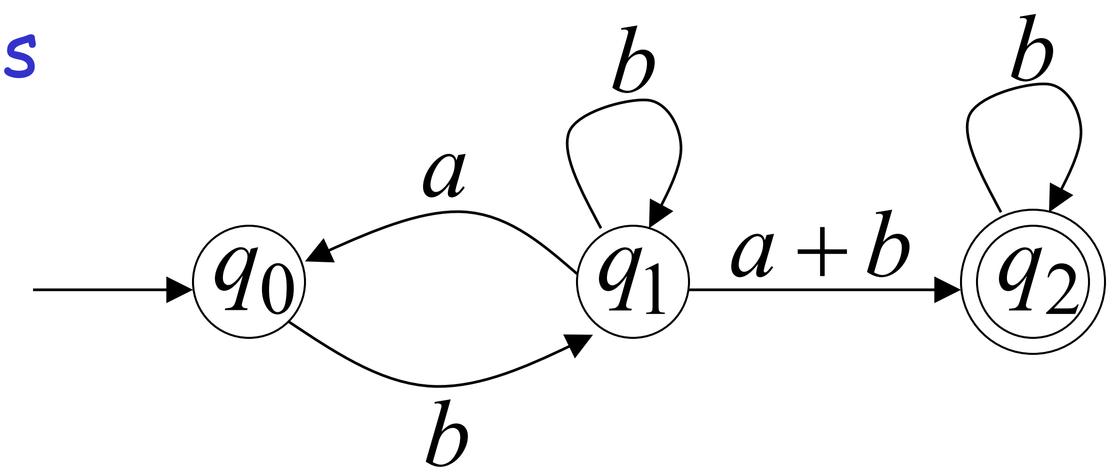
Corresponding
Generalized transition graph



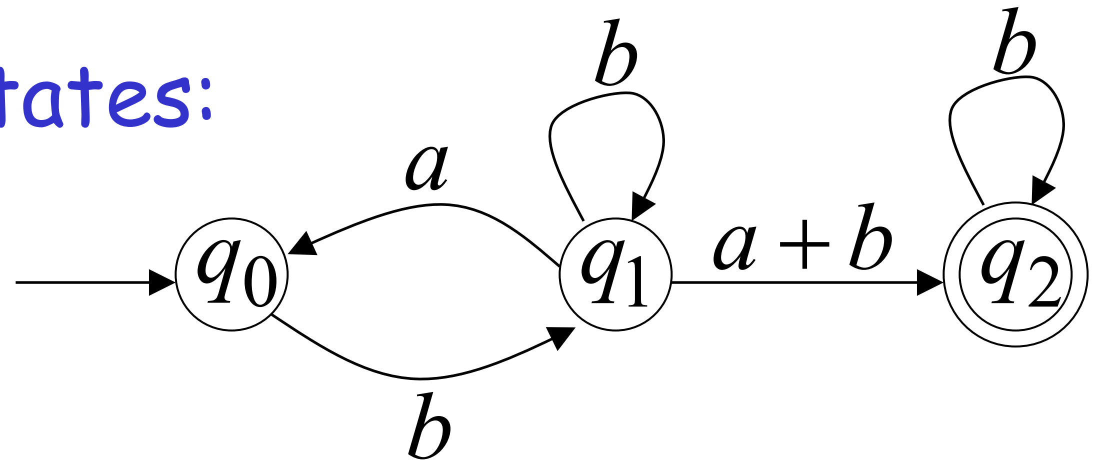
Another Example:



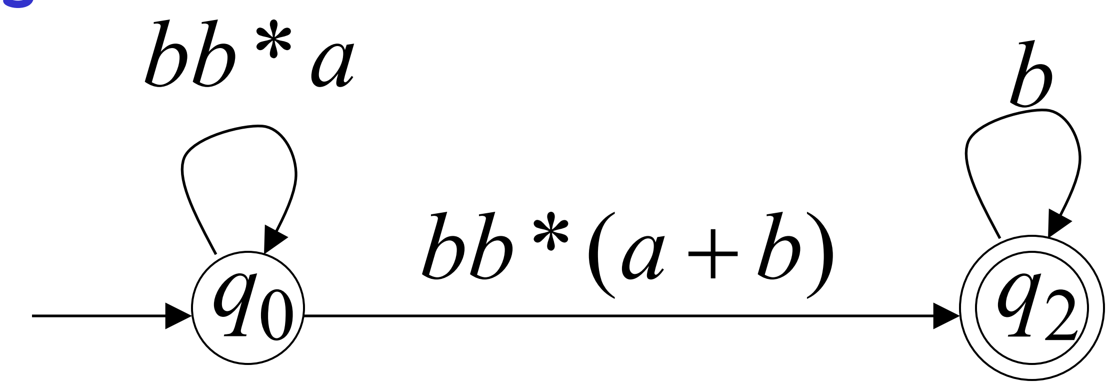
Transition labels are regular expressions



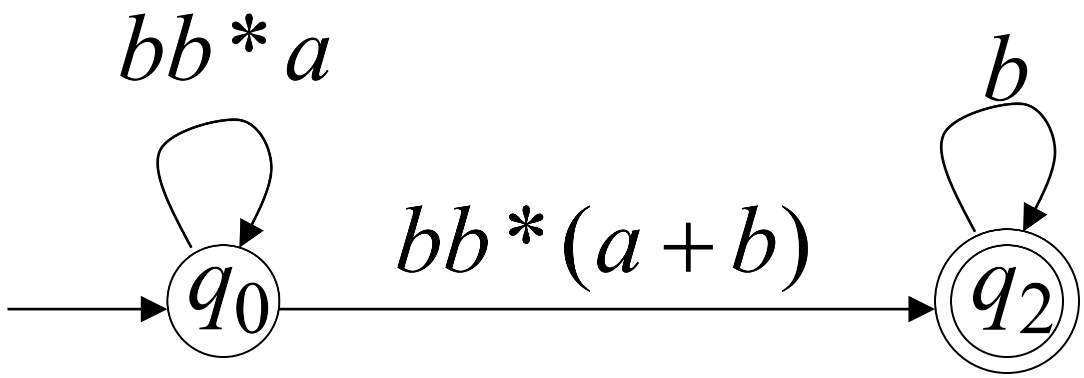
Reducing the states:



Transition labels are regular expressions



Resulting Regular Expression:

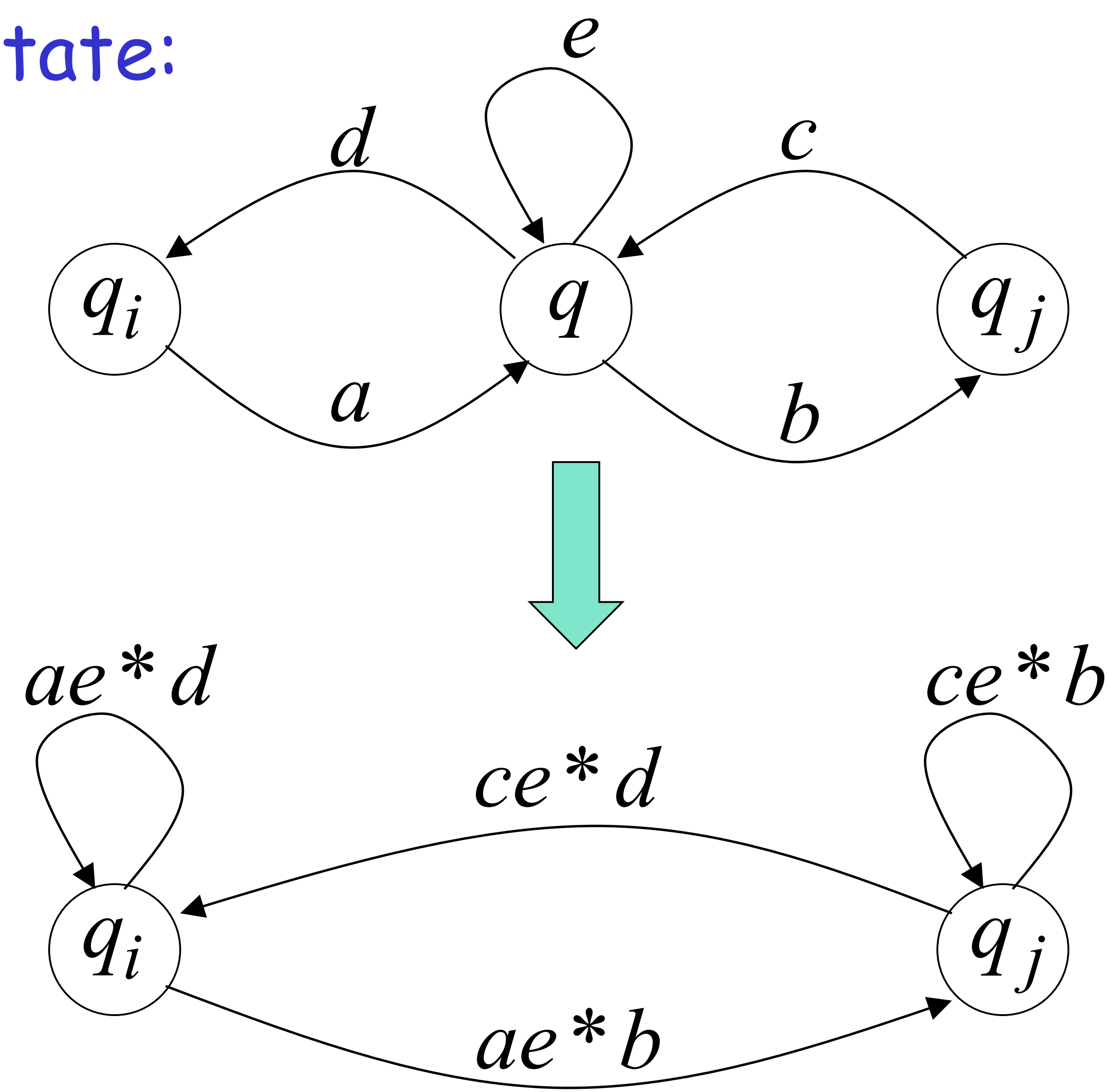


$$r = (bb^*a)^* \cdot bb^*(a+b) \cdot b^*$$

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

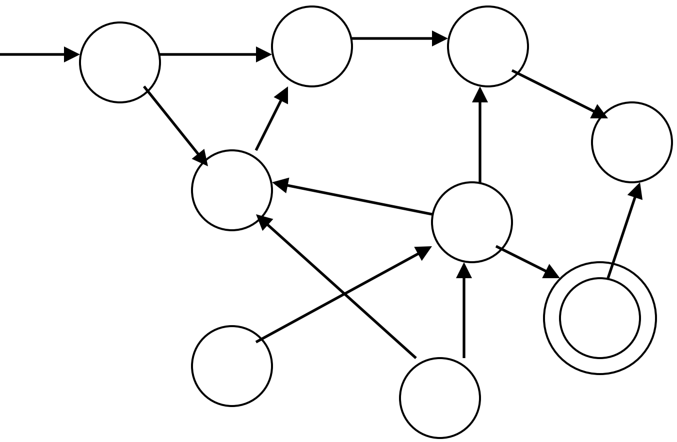
In General

Removing a state:

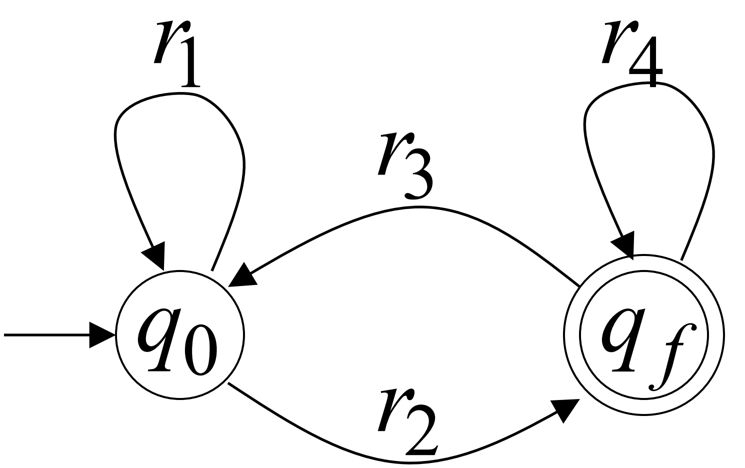


By repeating the process until two states are left, the resulting graph is

Initial graph



Resulting graph



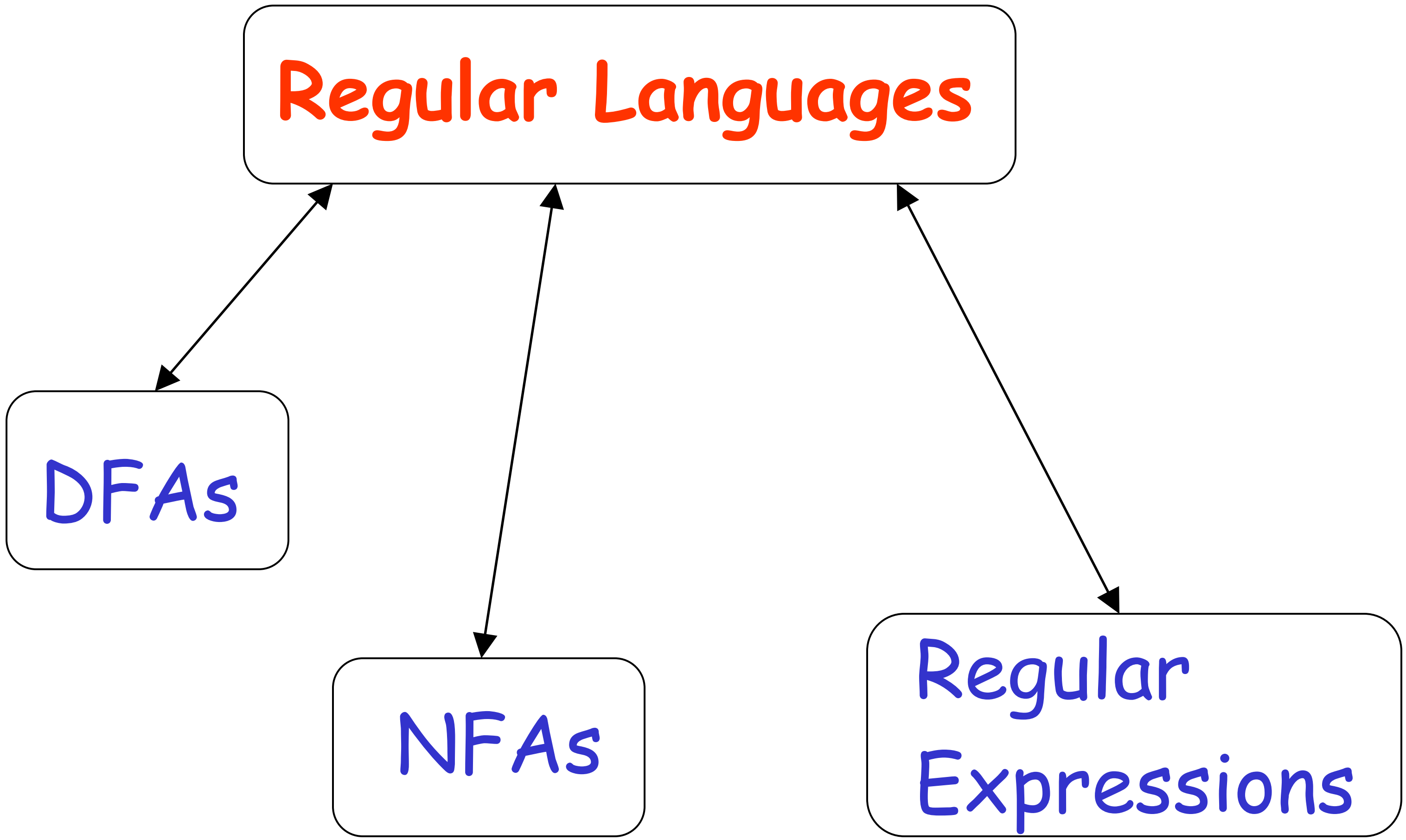
The resulting regular expression:

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

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

End of Proof-Part 2

Standard Representations of Regular Languages



When we say: We are given a Regular Language L

We mean: Language L is in a standard representation

(DFA, NFA, or Regular Expression)