Regular Expressions

Regular Expression (RE)

Regular expression: An algebraic way to describe regular languages.

Many of today's programming languages use regular expressions to match patterns in strings.

E.g., awk, flex, lex, java, javascript, perl, python

Used for searching texts in UNIX (vi, Perl, Emacs, grep), Microsoft Word (version 6 and beyond), and WordPerfect.

Few Web search engines may allow the use of Regular Expressions

Recursive Definition

Primitive regular expressions: $\emptyset, \quad \lambda, \quad \alpha$

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1 * \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

Examples

A regular expression: $(a + b \cdot c)^* \cdot (c + \emptyset)$

Not a regular expression: $(a + b +)$

Languages of Regular Expressions

$L(r)$: language of regular expression r

Example

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

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

$$
\begin{aligned}
L((a+b)\cdot a^*) &= L((a+b))\,L(a^*) \\
&= L(a+b)\,L(a^*) \\
&= (L(a)\cup L(b))\,(L(a))^* \\
&= (\{a\}\cup\{b\})\,(\{a\})^* \\
&= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}
$$

Regular Expressions

Operator Precedence:

- Highest: Kleene Closure
- Then: Concatenation
- Lowest: Union

Example

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

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

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

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

Regular Expressions

EXAMPLE 2.1 The expression $0(0+1)^*1$ represents the set of all strings that begin with a 0 and end with a 1.

EXAMPLE 2.2 The expression $0+1+0(0+1)^*0+1(0+1)^*1$ represents the set of all nonempty binary strings that begin and end with the same bit. Note the inclusion of the strings 0 and 1 as special cases.

EXAMPLE 2.3 The expressions 0^*, 0^*10^*, and $0^*10^*10^*$ represent the languages consisting of strings that contain no 1, exactly one 1, and exactly two 1's, respectively.

EXAMPLE 2.4 The expressions $(0+1)^*1(0+1)^*1(0+1)^*$, $(0+1)^*10^*1(0+1)^*$, $0^*10^*1(0+1)^*$, and $(0+1)^*10^*10^*$ all represent the same set of strings that contain at least two 1's.

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 }

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

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

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

r_1 and r_2 are equivalent regular expressions

Regular Expression: The IEEE POSIX standard

<table>
<tr><th>Character</th><th>Meaning</th><th>Examples</th></tr>
<tr><td>[ ]</td><td>alternatives</td><td>/[aeiou]/, /m[ae]n/</td></tr>
<tr><td>-</td><td>range</td><td>/[a-z]/</td></tr>
<tr><td>[^]</td><td>not</td><td>/[^pbm]/, /[^ox]s/</td></tr>
<tr><td>?</td><td>optionality</td><td>/Kath?mandu/</td></tr>
<tr><td>*</td><td>zero or more</td><td>/baa*!/</td></tr>
<tr><td>+</td><td>one or more</td><td>/ba+!/</td></tr>
<tr><td>.</td><td>any character</td><td>/cat.[aeiou]/</td></tr>
<tr><td>^, $</td><td>start, end of line</td><td></td></tr>
<tr><td>\</td><td>not special character</td><td>\.\?\^</td></tr>
<tr><td>|</td><td>alternate strings</td><td>/cat|dog/</td></tr>
<tr><td>( )</td><td>substring</td><td>/cit(y|ies)/</td></tr>
</table>

etc.

Regular Expressions

Valid Email Addresses

Valid IP Addresses

Valid Dates

Floating Point Numbers

Variables

Integers

Numeric Values

Naming Regular Expressions

Can assign names to regular expressions

Can use the name of a RE in the definition of another RE

Examples:

```
letter      ::= a | b | ... | z
digit       ::= 0 | 1 | ... | 9
alphanum    ::= letter | digit
```

Grammar-like notation for named RE's: a regular grammar

Can reduce named RE's to plain RE by "macro expansion"

- no recursive definitions allowed,
  unlike full context-free grammars

Specifying Tokens

Identifiers

```
ident        ::= letter (letter | digit)*
```

Integer constants

```
integer      ::= digit+
sign         ::= + | -
signed_int   ::= [sign] integer
```

Real number constants

```
real         ::= signed_int
                 [fraction] [exponent]
fraction     ::= . digit+
exponent     ::= (E|e) signed_int
```

RE specification of initial MiniJava lexical structure

```
Program      ::= (Token | Whitespace)*

Token        ::= ID | Integer | ReservedWord |
                 Operator | Delimiter
ID           ::= Letter (Letter | Digit)*
Letter       ::= a | ... | z | A | ... | Z
Digit        ::= 0 | ... | 9
Integer      ::= Digit+
ReservedWord::= class | public | static |
                 extends | void | int |
                 boolean | if | else |
                 while | return | true | false |
                 this | new | String | main |
                 System.out.println
Operator     ::= + | - | * | / | < | <= | >= |
                 > | == | != | && | !
Delimiter    ::= ; | . | , | = |
                 ( | ) | { | } | [ | ]

Whitespace   ::= <space> | <tab> | <newline>
```

Regular Expressions and Regular Languages

Theorem

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

Theorem (Kleene 1956):

We say that a language $L \subseteq \Sigma^\star$ is regular *if there exists a regular expression r such that $L = L(r)$. In this case, we also say that r* represents *the language L.*

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

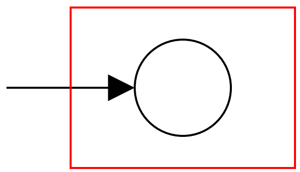
Proof by induction on the size of r

Induction Basis

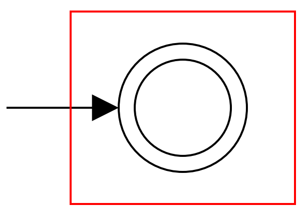
Primitive Regular Expressions: $\emptyset, \quad \lambda, \quad \alpha$

Corresponding

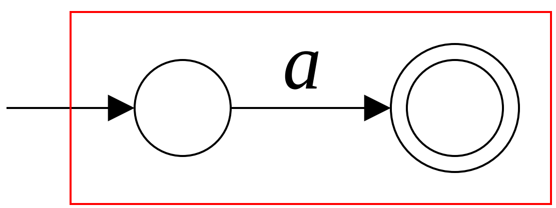
NFAs



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



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



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

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:

$$L(r_1 + r_2)$$

$$L(r_1 \cdot r_2)$$

$$L(r_1 *)$$

$$L((r_1))$$

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:

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

Are regular languages

$$L((r_1)) = L(r_1)$$

is trivially a regular language (by induction hypothesis)

End of Proof-Part 1

Proof - Part 2

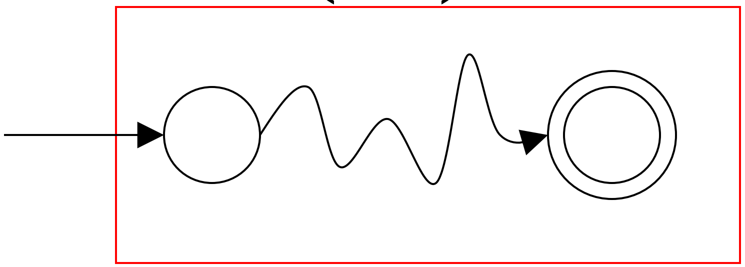
$$\left\{\begin{array}{l}\text{Languages}\\\text{Generated by}\\\text{Regular Expressions}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\\text{Languages}\end{array}\right\}$$

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 final state

From M construct the equivalent

Generalized Transition Graph

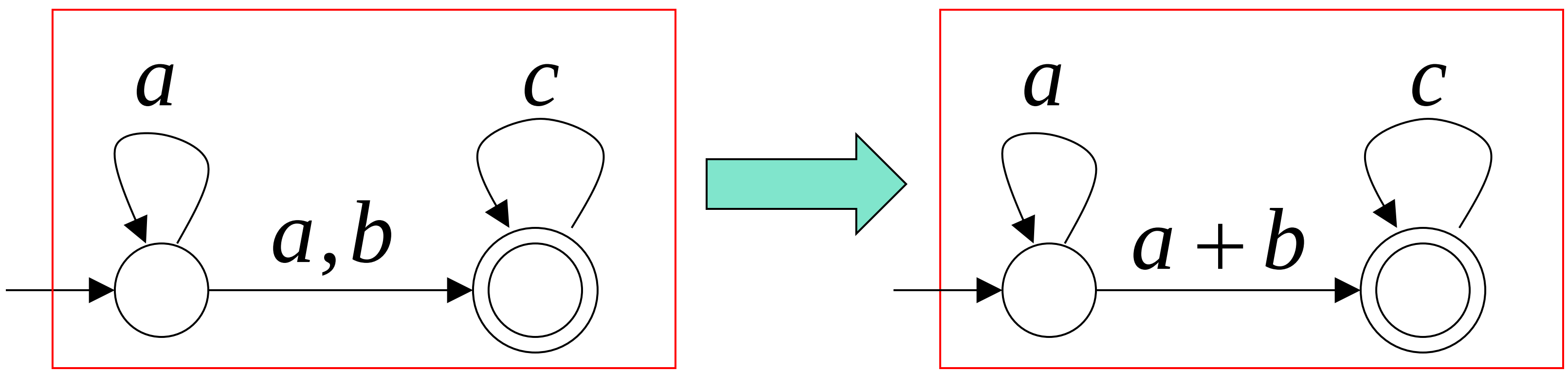
in which transition labels are regular expressions

Example:

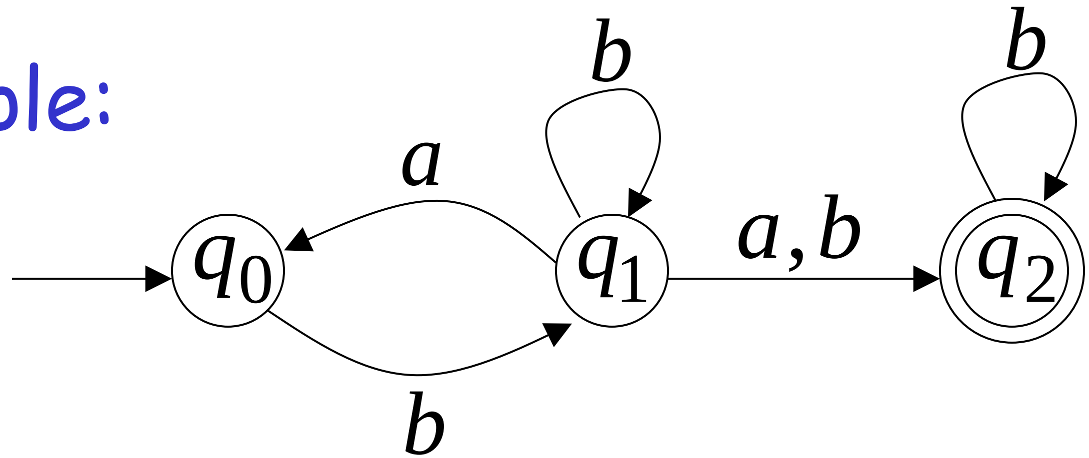
M

Corresponding

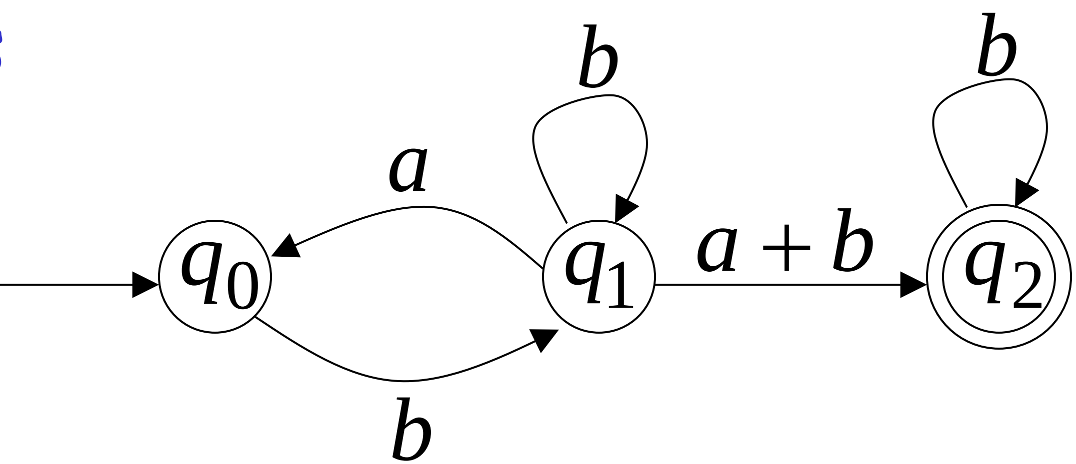
Generalized transition graph



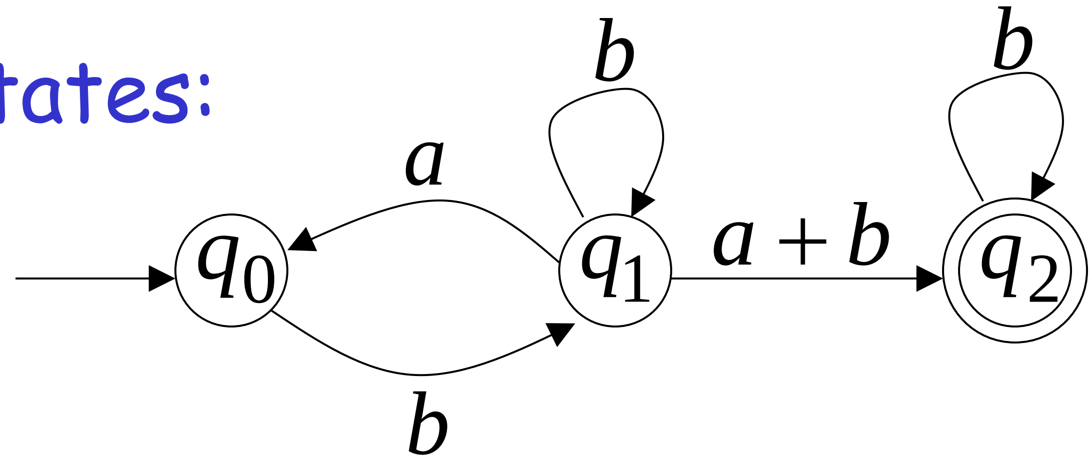
Another Example:



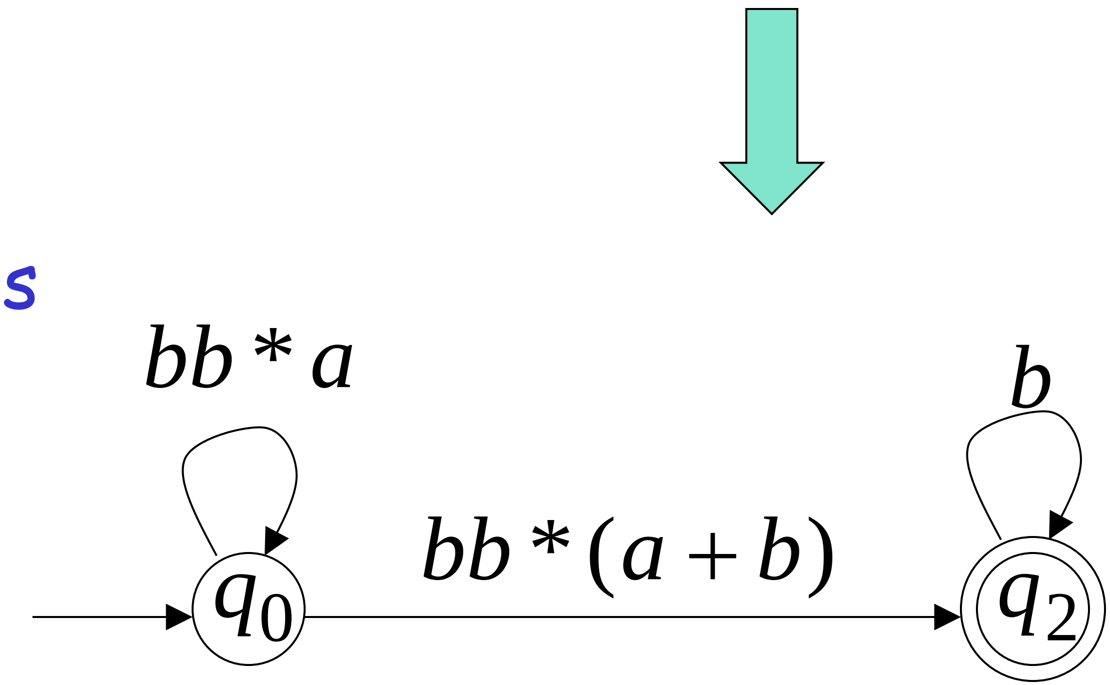
Transition labels are regular expressions



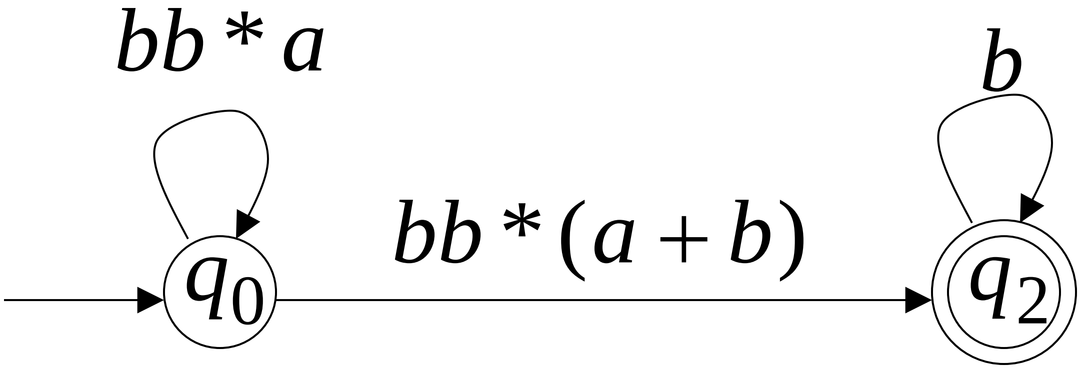
Reducing the states:



Transition labels are regular expressions



Resulting Regular Expression:

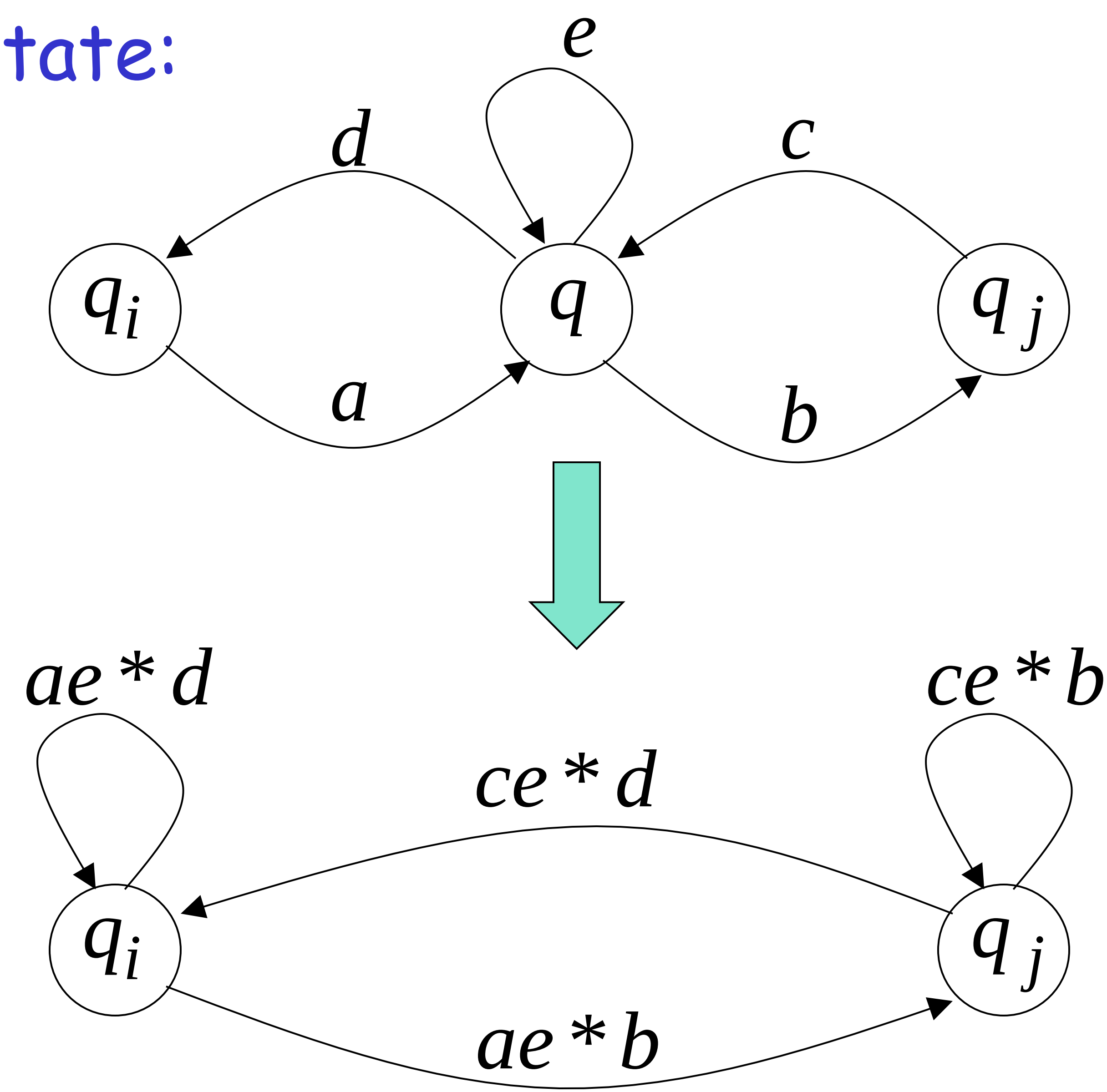


$$r = (bb^*a)^*bb^*(a+b)b^*$$

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

In General

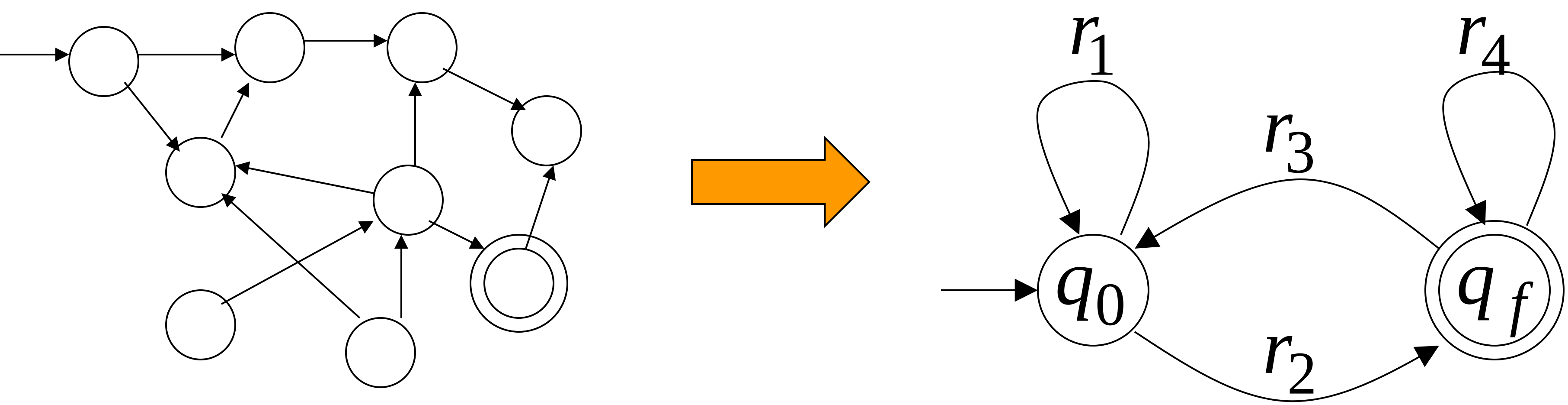
Removing a state:



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

Initial graph

Resulting graph



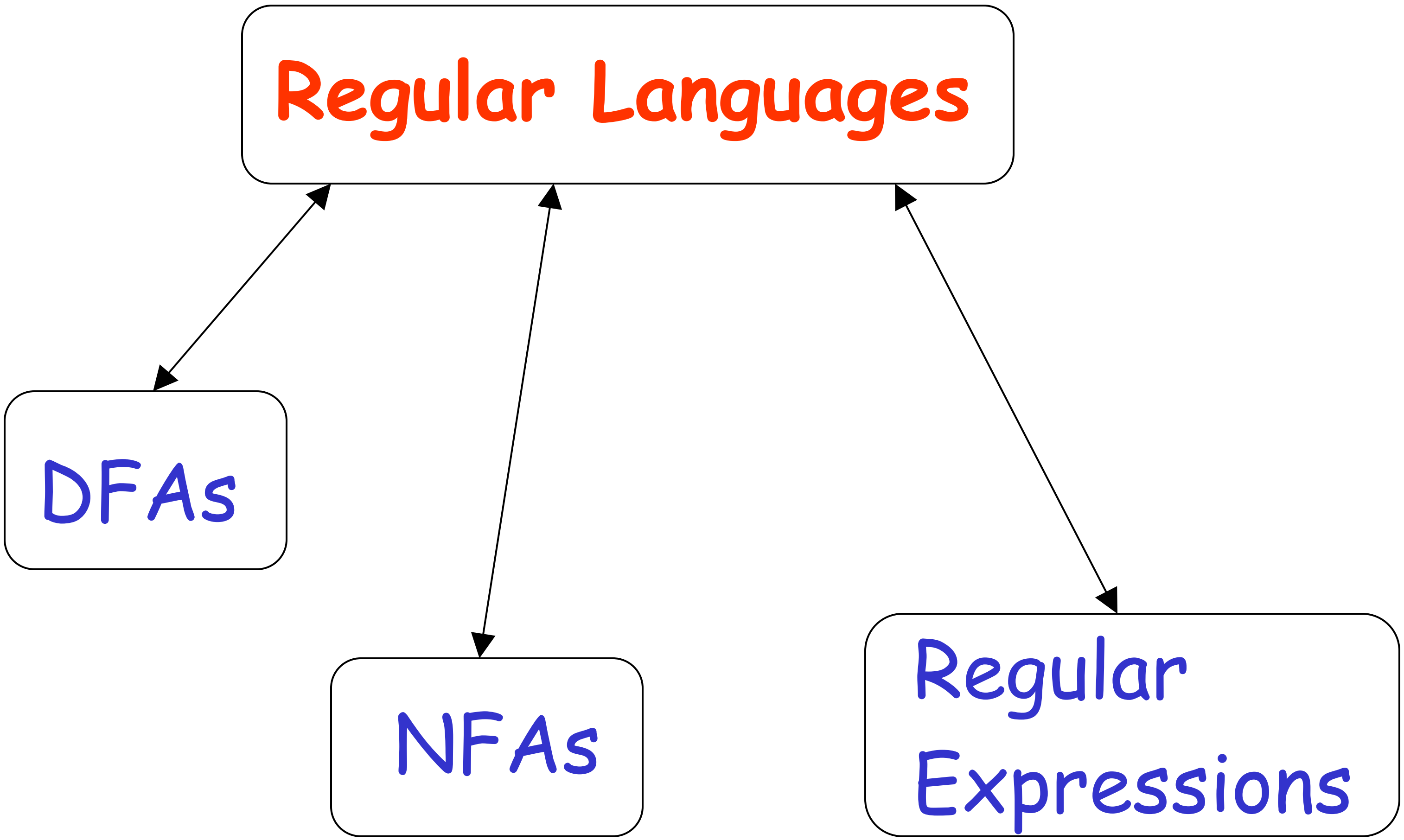
The resulting regular expression:

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

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

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)