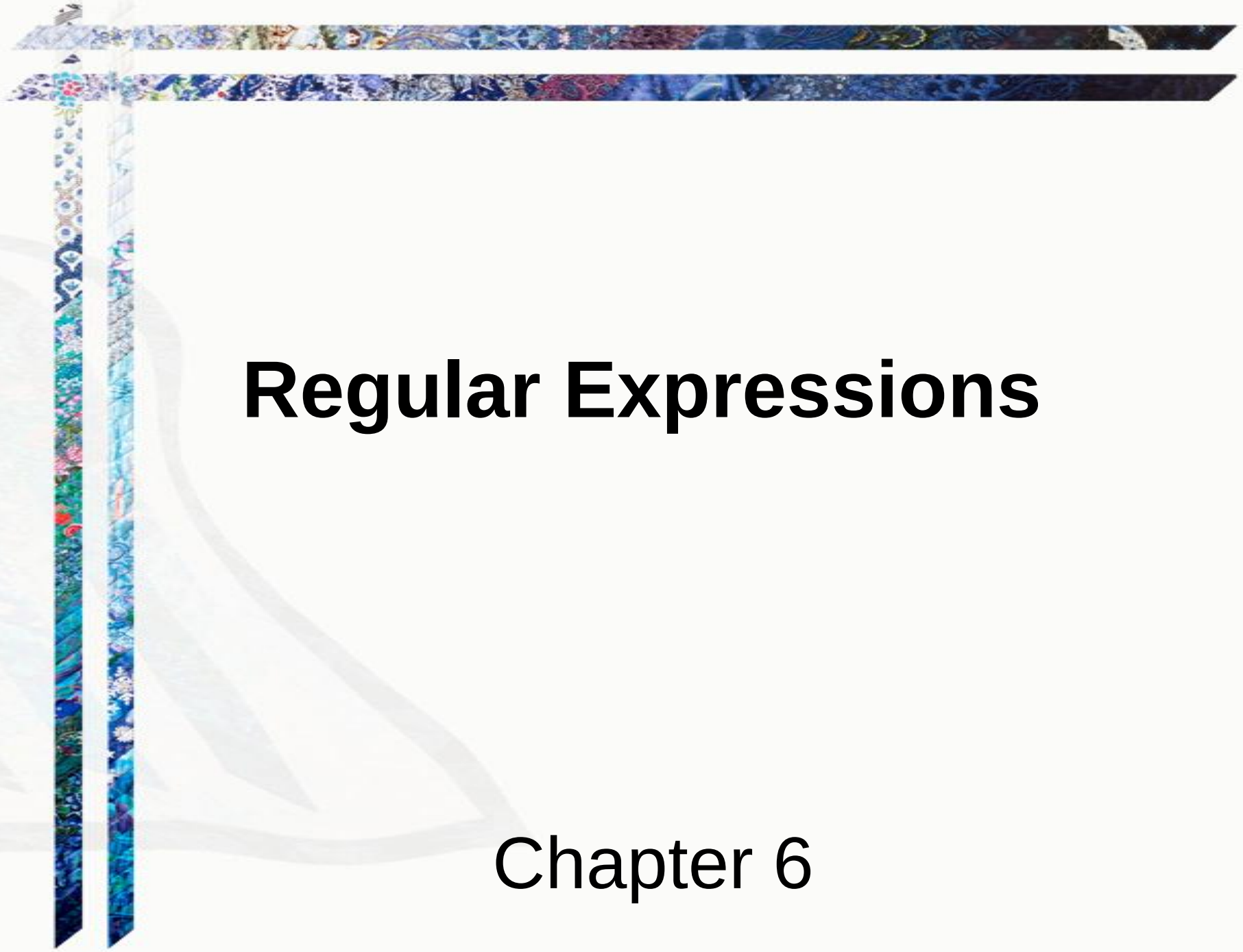
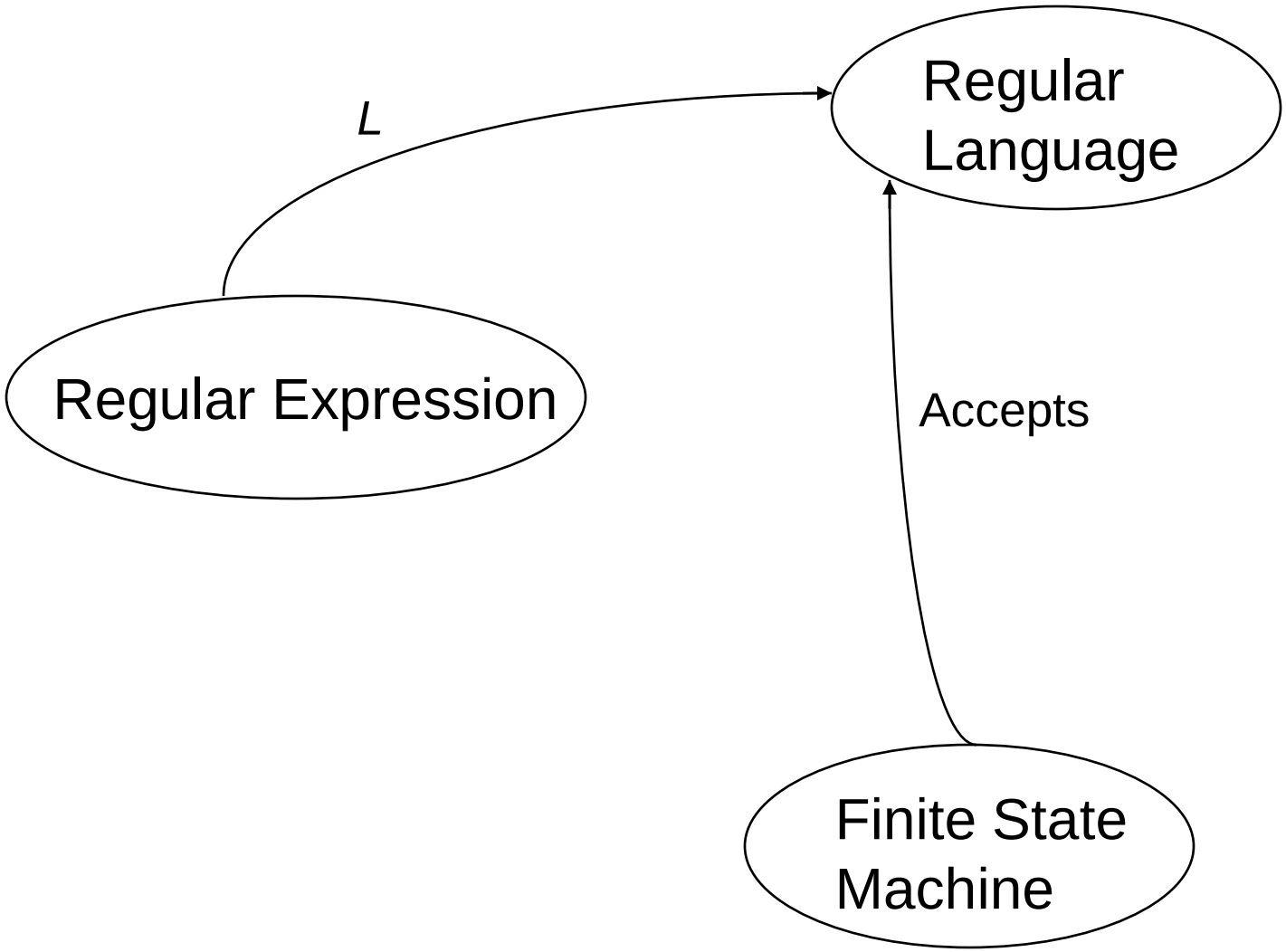


Regular Expressions

Chapter 6

Regular Languages



Regular Expressions

The regular expressions over an alphabet Σ are all and only the strings that can be obtained as follows:

1. $\varnothing$ is a regular expression.
2. ε is a regular expression.
3. Every element of Σ is a regular expression.
4. If α , β are regular expressions, then so is $\alpha\beta$.
5. If α , β are regular expressions, then so is $\alpha \cup \beta$.
6. If α is a regular expression, then so is α^*.
7. α is a regular expression, then so is α^+.
8. If α is a regular expression, then so is (α).

Regular Expression Examples

If $\Sigma = \{a, b\}$, the following are regular expressions:

$\emptyset$

ε

a

$(a \cup b)^*$

$abba \cup \varepsilon$

Regular Expressions Define Languages

Define L, a **semantic interpretation function** for regular expressions:

1. $L(\varnothing) = \varnothing$.
2. $L(\varepsilon) = \{\varepsilon\}$.
3. $L(c)$, where $c \in \Sigma = \{c\}$.
4. $L(\alpha\beta) = L(\alpha)\ L(\beta)$.
5. $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$.
6. $L(\alpha^*) = (L(\alpha))^*$.
7. $L(\alpha^+) = L(\alpha\alpha^*) = L(\alpha)\ (L(\alpha))^*$. If $L(\alpha)$ is equal to $\varnothing$, then $L(\alpha^+)$ is also equal to $\varnothing$. Otherwise $L(\alpha^+)$ is the language that is formed by concatenating together one or more strings drawn from $L(\alpha)$.
8. $L((\alpha)) = L(\alpha)$.

The Role of the Rules

- Rules 1, 3, 4, 5, and 6 give the language its power to define sets.
- Rule 8 has as its only role grouping other operators.
- Rules 2 and 7 appear to add functionality to the regular expression language, but they don't.

2. ε is a regular expression.

7. α is a regular expression, then so is α^+.

Analyzing a Regular Expression

$$L((a \cup b)^*b) = L((a \cup b)^*)\ L(b)$$

$$= (L((a \cup b)))^*\ L(b)$$

$$= (L(a) \cup L(b))^*\ L(b)$$

$$= (\{a\} \cup \{b\})^*\ \{b\}$$

$$= \{a, b\}^*\ \{b\}.$$

Examples

$L(\ a^*b^*\ ) =$

$L(\ (a \cup b)^*\ ) =$

$L(\ (a \cup b)^*a^*b^*\ ) =$

$L(\ (a \cup b)^*abba(a \cup b)^*\ ) =$

Going the Other Way

$L = \{w \in \{\texttt{a}, \texttt{b}\}^* : |w| \text{ is even}\}$

Going the Other Way

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : |w| \text{ is even}\}$

$((\mathtt{a} \cup \mathtt{b})\ (\mathtt{a} \cup \mathtt{b}))^*$

$(\mathtt{aa} \cup \mathtt{ab} \cup \mathtt{ba} \cup \mathtt{bb})^*$

Going the Other Way

$L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$

$(a \cup b)\ (a \cup b))^*$

$(aa \cup ab \cup ba \cup bb)^*$

$L = \{w \in \{a, b\}^* : w \text{ contains an odd number of } a\text{'s}\}$

Going the Other Way

$L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$

$(a \cup b)\ (a \cup b))^*$

$(aa \cup ab \cup ba \cup bb)^*$

$L = \{w \in \{a, b\}^* : w \text{ contains an odd number of } a\text{'s}\}$

$b^*\ (ab^*ab^*)^*\ a\ b^*$

$b^*\ a\ b^*\ (ab^*ab^*)^*$

More Regular Expression Examples

L ((aa*) $\cup$ ε) =

L ((a $\cup$ ε)*) =

L = {$w \in$ {a, b}*: there is no more than one b in w}

L = {$w \in$ {a, b}* : no two consecutive letters in w are the same}

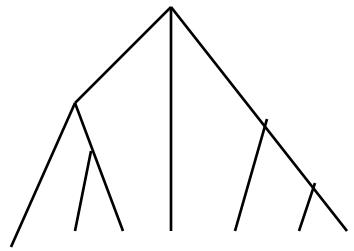
Common Idioms

$(\alpha \cup \varepsilon)$

$(a \cup b)^*$

Operator Precedence in Regular Expressions

	Regular Expressions	Arithmetic Expressions
Highest	Kleene star	exponentiation
	concatenation	multiplication
Lowest	union	addition



$x\,y^2 + i\,j^2$

The Details Matter

$a^* \cup b^* \neq (a \cup b)^*$

$(ab)^* \neq a^*b^*$

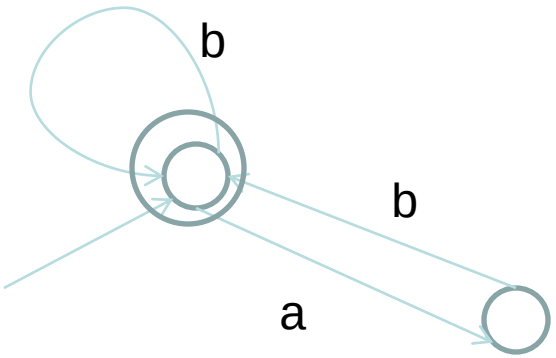
The Details Matter

L_1 = {$w \in$ {a, b}* : every a is immediately followed a b}

A regular expression for L_1:

$$(b \cup ab)^*$$

A FSM for L_1:



Kleene's Theorem

Finite state machines and regular expressions define the same class of languages. To prove this, we must show:

Theorem*:* Any language that can be defined with a regular expression can be accepted by some FSM and so is regular.

Theorem: Every regular language (i.e., every language that can be accepted by some DFSM) can be defined with a regular expression.

For Every Regular Expression There is a Corresponding FSM

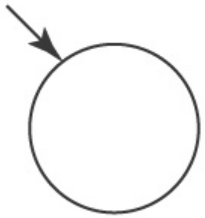
We'll show this by construction. An FSM for:

∅:

For Every Regular Expression There is a Corresponding FSM

We'll show this by construction. An FSM for:

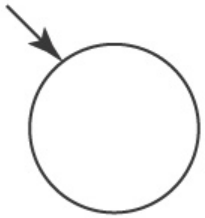
∅:



For Every Regular Expression There is a Corresponding FSM

We'll show this by construction. An FSM for:

$\varnothing$:

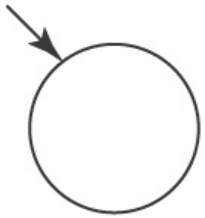


A single element of Σ:

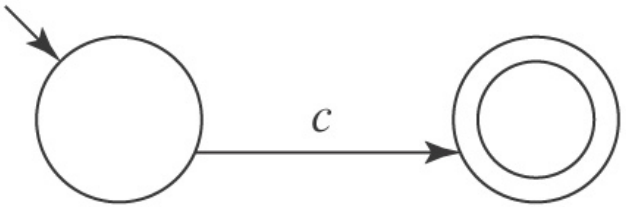
For Every Regular Expression There is a Corresponding FSM

We'll show this by construction. An FSM for:

$\varnothing$:



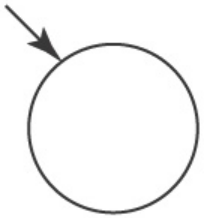
A single element of Σ:



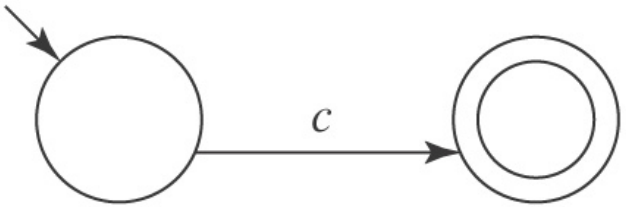
For Every Regular Expression There is a Corresponding FSM

We'll show this by construction. An FSM for:

∅:



A single element of Σ:

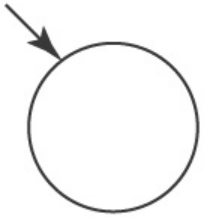


ε (∅*):

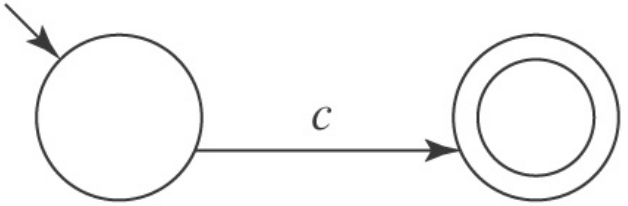
For Every Regular Expression There is a Corresponding FSM

We'll show this by construction. An FSM for:

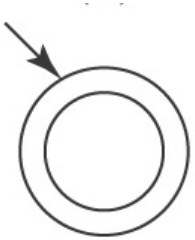
∅:



A single element of Σ:



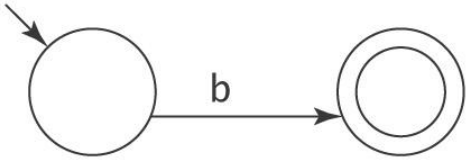
ε (∅*):



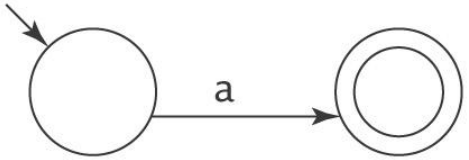
An Example

(b ∪ ab)*

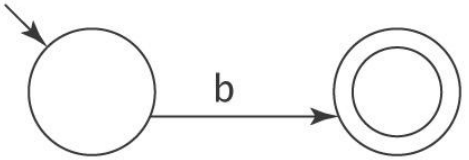
An FSM for b



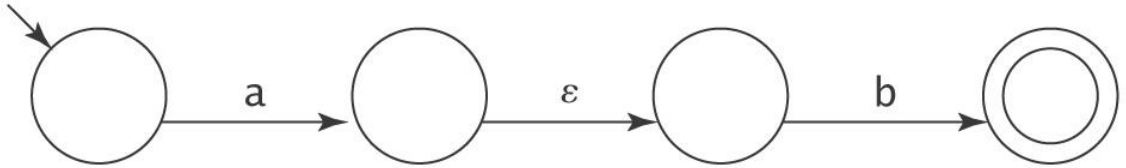
An FSM for a



An FSM for b



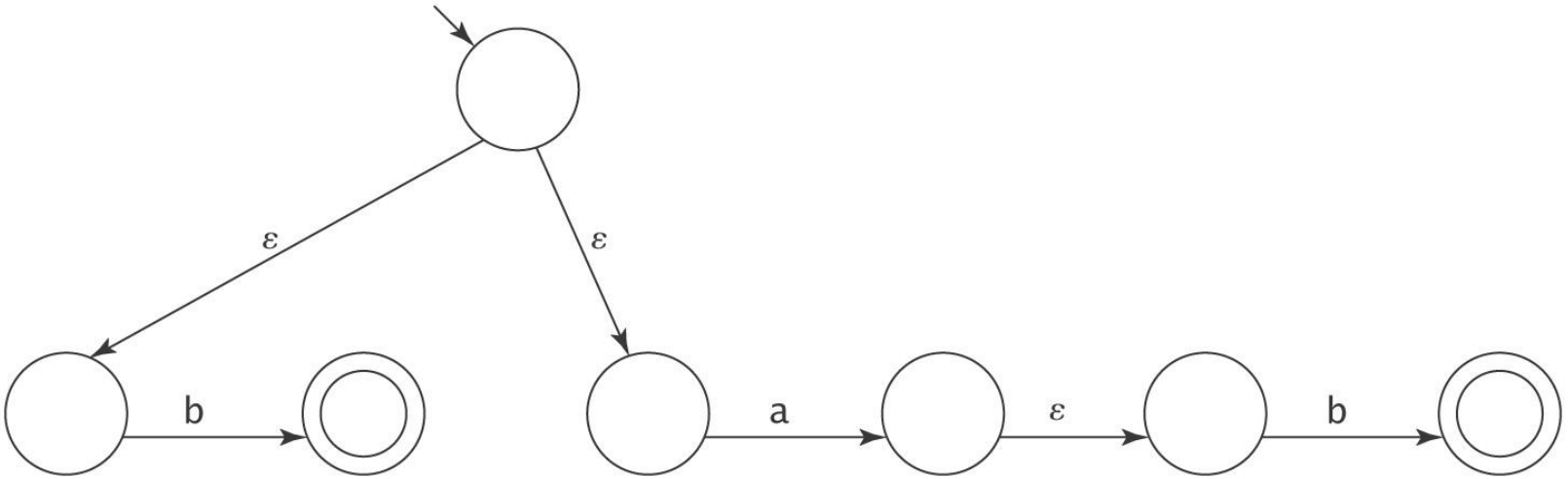
An FSM for ab:



An Example

(b ∪ ab)*

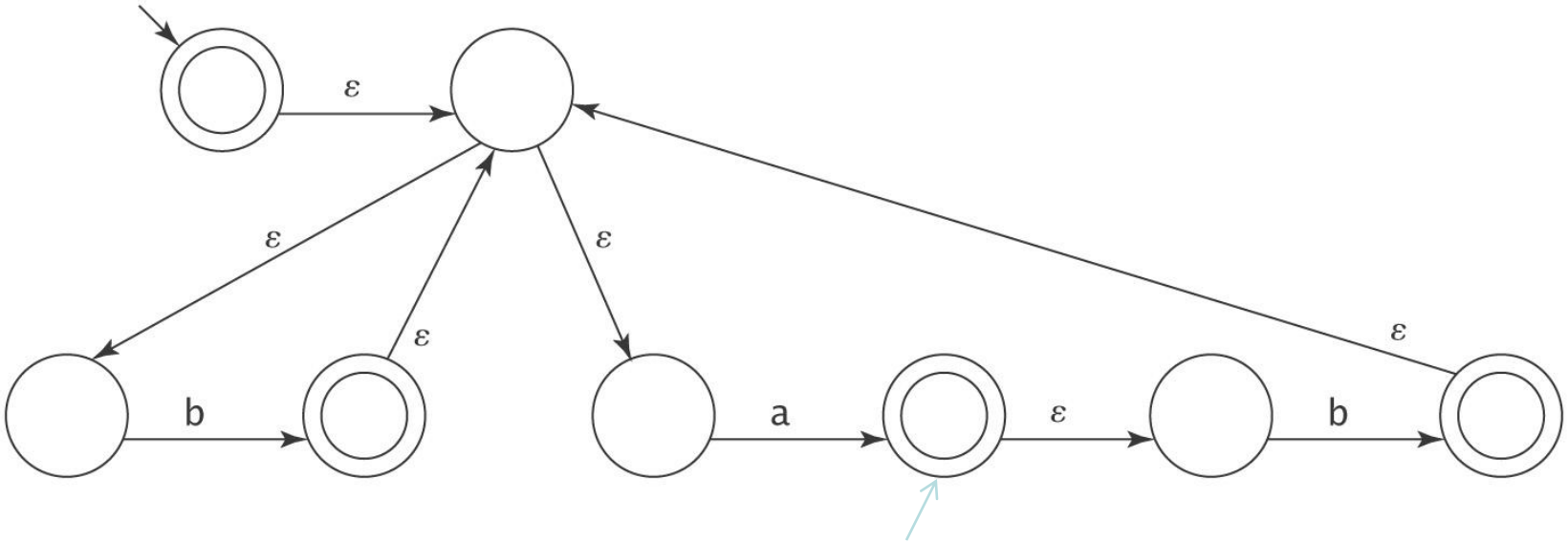
An FSM for (b ∪ ab):



An Example

(b ∪ ab)*

An FSM for (b ∪ ab)*:



Error in Book, Not an Accept State

The Algorithm *regextofsm*

regextofsm(α: regular expression) =

Beginning with the primitive subexpressions of α and working outwards until an FSM for all of α has been built do:

Construct an FSM as described above.

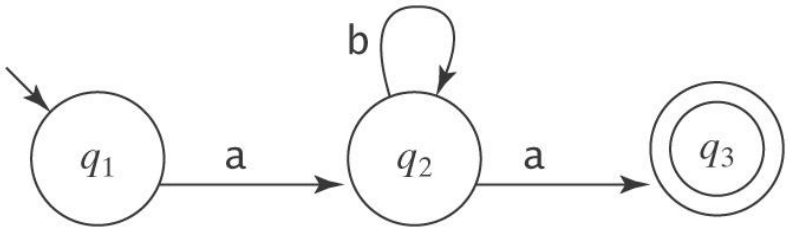
For Every FSM There is a Corresponding Regular Expression

We'll show this by construction.

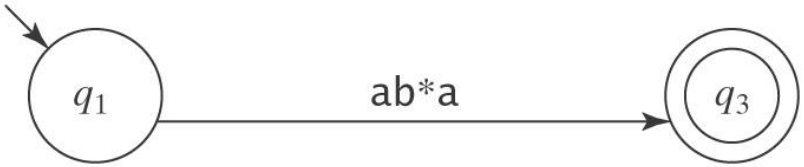
The key idea is that we'll allow arbitrary regular expressions to label the transitions of an FSM.

A Simple Example

Let *M* be:



Suppose we rip out state 2:

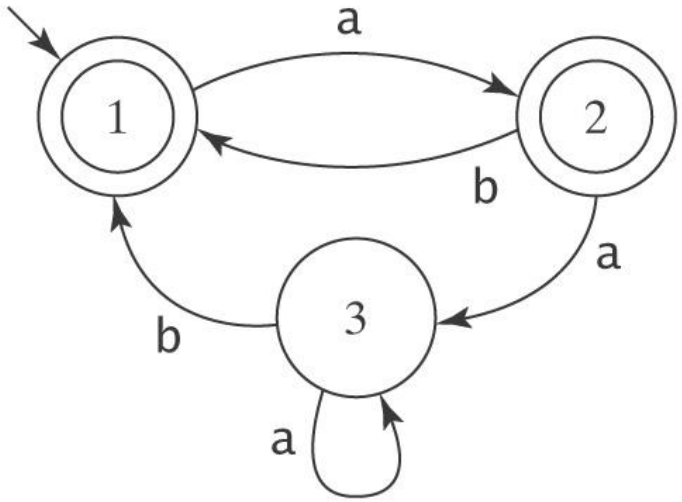


The Algorithm *fsmtoregexheuristic*

fsmtoregexheuristic(*M*: FSM) =

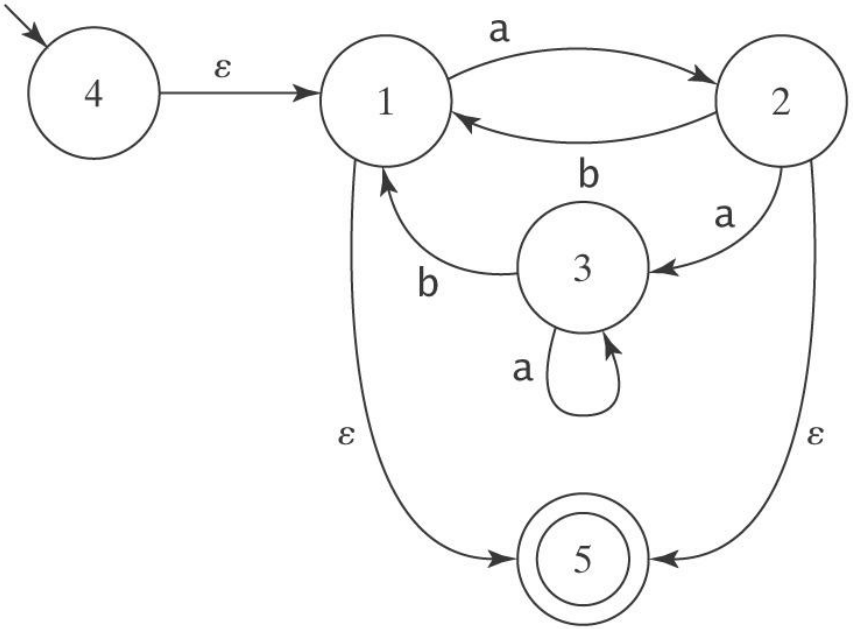
1. Remove unreachable states from *M*.
2. If *M* has no accepting states then return ∅.
3. If the start state of *M* is part of a loop, create a new start state *s* and connect *s* to *M*'s start state via an ε-transition.
4. If there is more than one accepting state of *M* or there are any transitions out of any of them, create a new accepting state and connect each of *M*'s accepting states to it via an ε-transition. The old accepting states no longer accept.
5. If *M* has only one state then return ε.
6. Until only the start state and the accepting state remain do:
    - 6.1 Select *rip* (not *s* or an accepting state).
    - 6.2 Remove *rip* from *M*.
    - 6.3 *Modify the transitions among the remaining states so *M* accepts the same strings.
7. Return the regular expression that labels the one remaining transition from the start state to the accepting state.

An Example



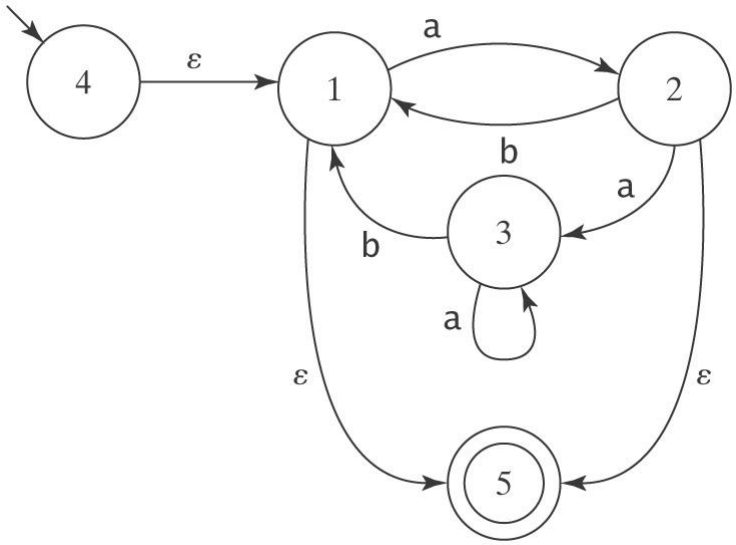
1. Create a new initial state and a new, unique accepting state, neither of which is part of a loop.

An Example, Continued

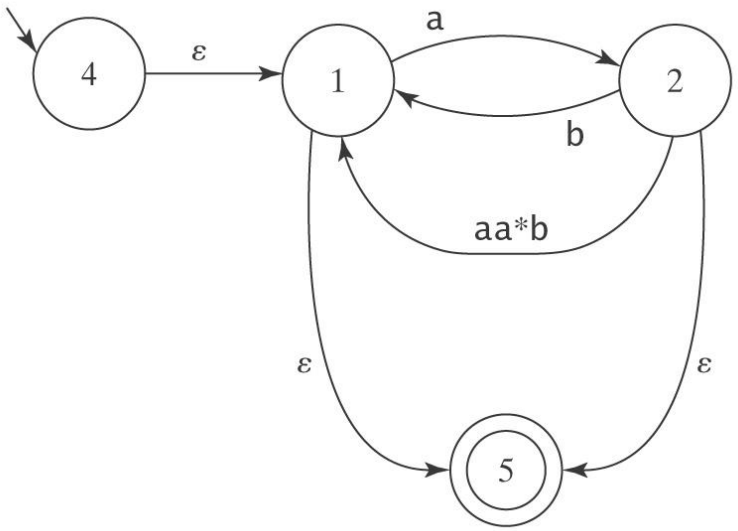


2. Remove states and arcs and replace with arcs labelled with larger and larger regular expressions.

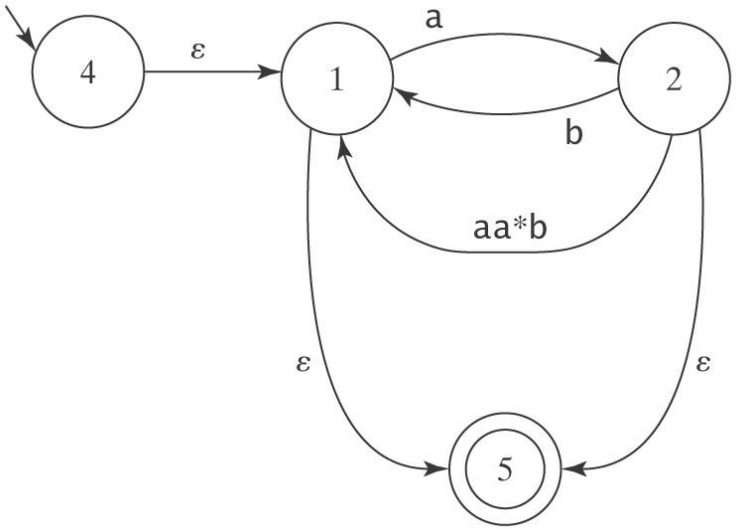
An Example, Continued



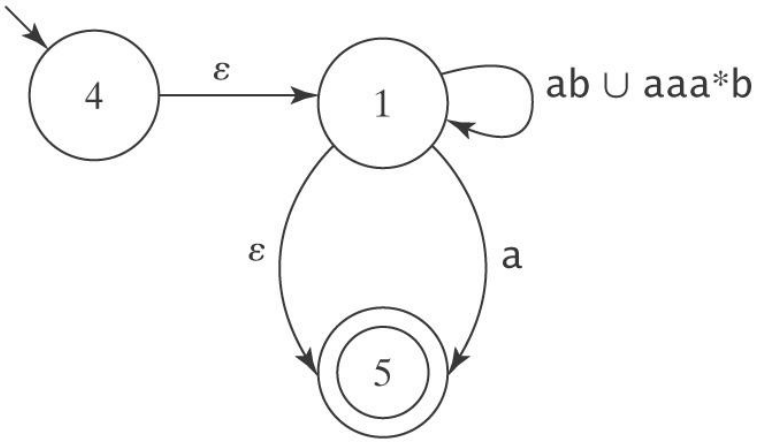
Remove state 3:



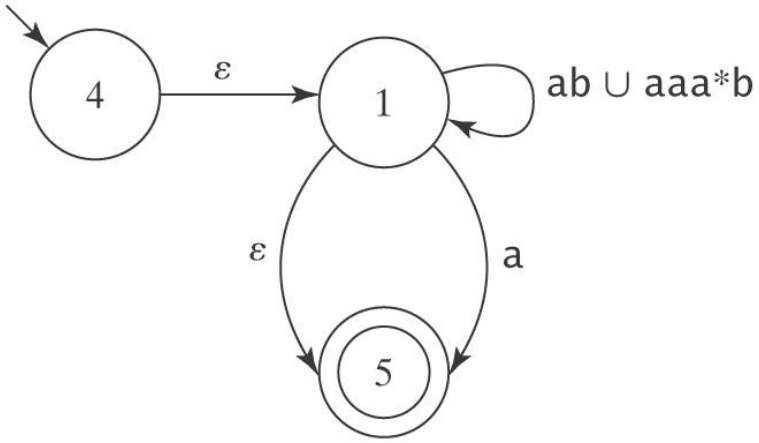
An Example, Continued



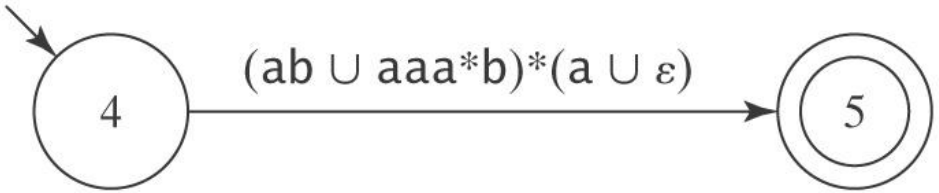
Remove state 2:



An Example, Continued

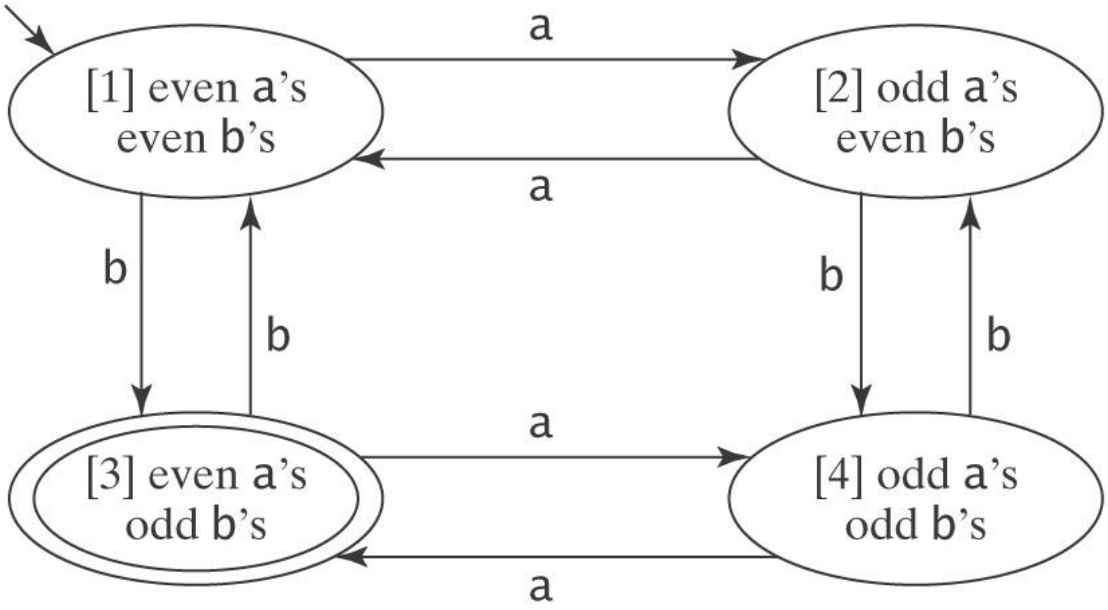


Remove state 1:



When It's Hard

M =



When It's Hard

A regular expression for *M*:

$$a\,(aa)^*$$

$$\cup\,(aa)^*\,b(b(aa)^*b)^*\,ba(aa)^*$$

$$\cup\,[a(aa)^*\,b \cup (b \cup\ a(aa)^*\,b)\ \ (b(aa)^*\,b)^*\,(a \cup ba(aa)^*b)]$$

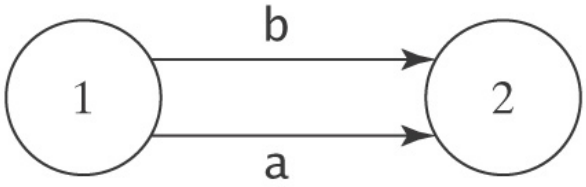
$$[b(aa)^*\,b \cup (a \cup b(aa)^*\,ab)\,(b(aa)^*\,b)^*\,(a \cup ba(aa)^*b)]^*$$

$$[b(aa)^*\ \ \cup (a \cup b(aa)^*\,ab)\,(b(aa)^*\,b)^*\,ba(aa)^*]$$

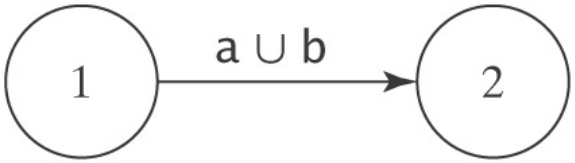
Further Modifications to *M* Before We Start

We require that, from every state other than the accepting state there must be exactly one transition to every state (including itself) except the start state. And into every state other than the start state there must be exactly one transition from every state (including itself) except the accepting state.

1. If there is more than one transition between states *p* and *q*, collapse them into a single transition:

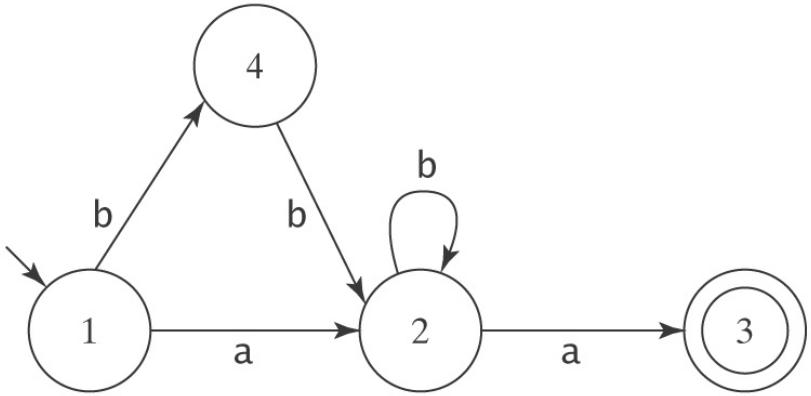


becomes:

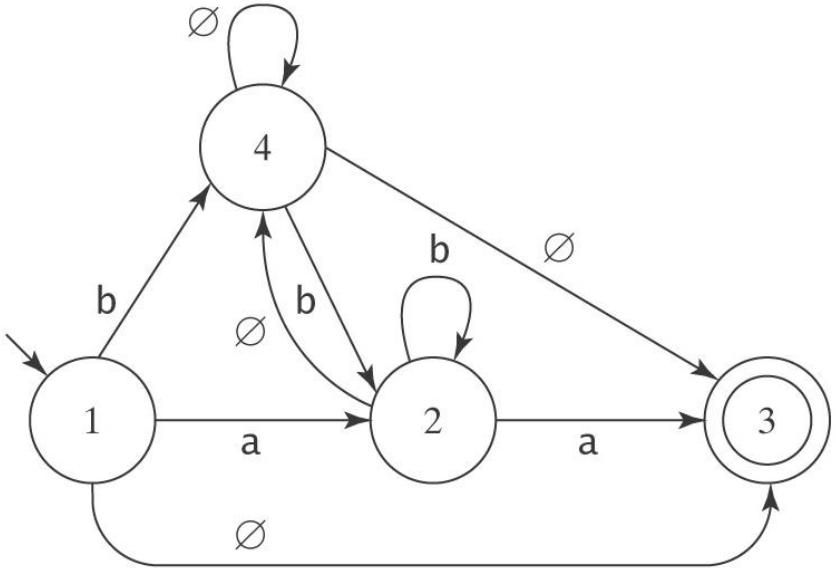


Further Modifications to *M* Before We Start

2. If any of the required transitions are missing, add them:

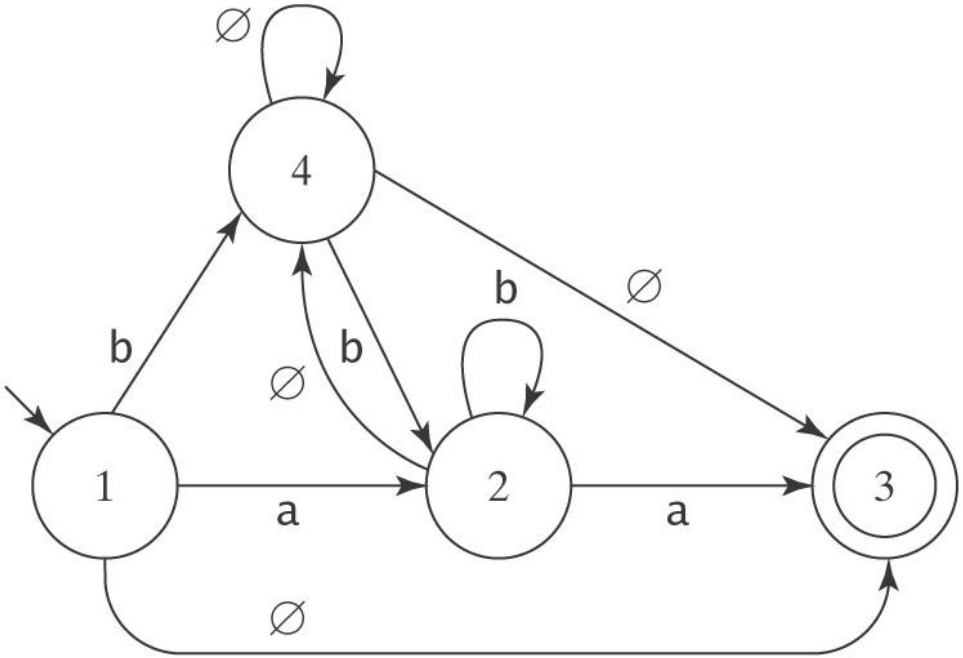


becomes:



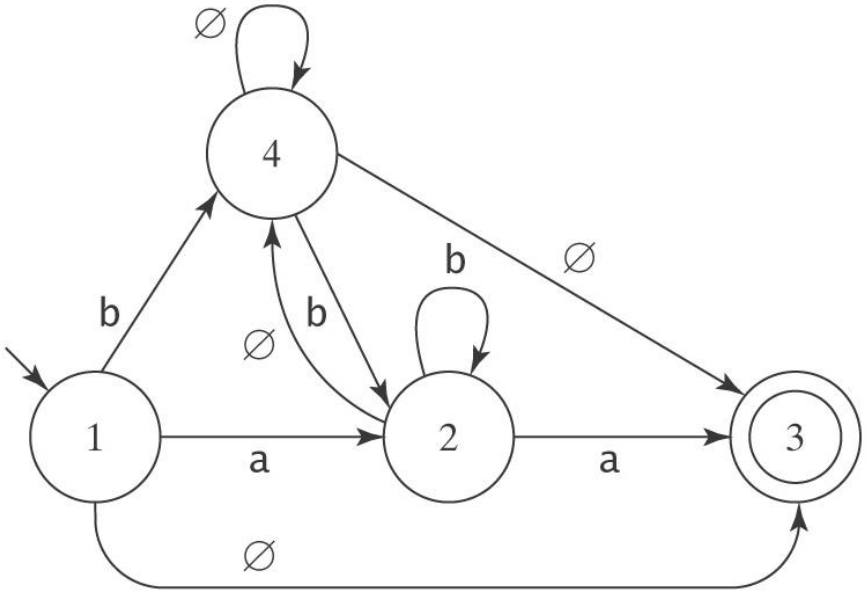
Ripping Out States

3. Choose a state. Rip it out. Restore functionality.



Suppose we rip state 2.

What Happens When We Rip?



Consider any pair of states *p* and *q*. Once we remove *rip*, how can *M* get from *p* to *q*?

- It can still take the transition that went directly from *p* to *q*, or
- It can take the transition from *p* to *rip*. Then, it can take the transition from *rip* back to itself zero or more times. Then it can take the transition from *rip* to *q*.

Defining $R(p, q)$

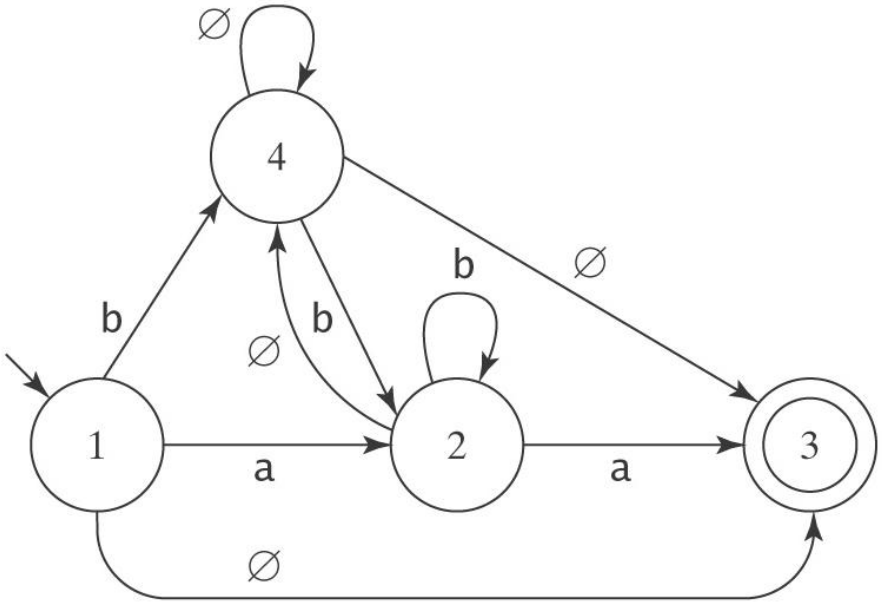
After removing *rip*, the new regular expression that should label the transition from *p* to *q* is:

$R(p, q)$		/* Go directly from *p* to *q*
	$\cup$	/* or
$R(p, rip)$		/* Go from *p* to *rip*, then
$R(rip, rip)^*$		/* Go from *rip* back to itself any number of times, then
$R(rip, q)$		/* Go from *rip* to *q*

Without the comments, we have:

$$R' = R(p, q) \cup R(p, rip)\, R(rip, rip)^*\, R(rip, q)$$

Returning to Our Example



$R' = R(p, q) \cup R(p, rip)\, R(rip, rip)^*\, R(p, rip)$

Let *rip* be state 2. Then:

$R'(1, 3) = R(1, 3) \cup R(1, rip)R(rip, rip)^*R(rip, 3)$

$= R(1, 3) \cup R(1, 2)R(2, 2)^*R(2, 3)$

$= \emptyset \cup \mathtt{a}\ \mathtt{b}^*\ \mathtt{a}$

$= \mathtt{ab}^*\mathtt{a}$

The Algorithm *fsmtoregex*

fsmtoregex(*M*: FSM) =

1. *M′* = *standardize*(*M*: FSM).
2. Return *buildregex*(*M′*).

standardize(*M*: FSM) =

1. Remove unreachable states from *M*.
2. If necessary, create a new start state.
3. If necessary, create a new accepting state.
4. If there is more than one transition between states *p* and *q*, collapse them.
5. If any transitions are missing, create them with label ∅.

The Algorithm *fsmtoregex*

buildregex(*M*: FSM) =

1. If *M* has no accepting states then return ∅.
2. If *M* has only one state, then return ε.
3. Until only the start and accepting states remain do:
    3.1 Select some state *rip* of *M*.
    3.2 For every transition from *p* to *q*, if both *p* and *q* are not *rip* then do
        Compute the new label *R′* for the transition from *p* to *q*:

$$R'(p, q) = R(p, q) \cup R(p, rip)\, R(rip, rip)^*\, R(rip, q)$$

    3.3 Remove *rip* and all transitions into and out of it.
4. Return the regular expression that labels the transition from the start state to the accepting state.

Simplifying Regular Expressions

Regex's describe sets:

- Union is commutative: $\alpha \cup \beta = \beta \cup \alpha$.
- Union is associative: $(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma)$.
- $\varnothing$ is the identity for union: $\alpha \cup \varnothing = \varnothing \cup \alpha = \alpha$.
- Union is idempotent: $\alpha \cup \alpha = \alpha$.

Concatenation:

- Concatenation is associative: $(\alpha\beta)\gamma = \alpha(\beta\gamma)$.
- ε is the identity for concatenation: $\alpha\ \varepsilon = \varepsilon\ \alpha = \alpha$.
- $\varnothing$ is a zero for concatenation: $\alpha\ \varnothing = \varnothing\ \alpha = \varnothing$.

Concatenation distributes over union:

- $(\alpha \cup \beta)\ \gamma = (\alpha\ \gamma) \cup (\beta\ \gamma)$.
- $\gamma\ (\alpha \cup \beta) = (\gamma\ \alpha) \cup (\gamma\ \beta)$.

Kleene star:

- $\varnothing^* = \varepsilon$.
- $\varepsilon^* = \varepsilon$.
- $(\alpha^*)^* = \alpha^*$.
- $\alpha^*\alpha^* = \alpha^*$.
- $(\alpha \cup \beta)^* = (\alpha^*\beta^*)^*$.