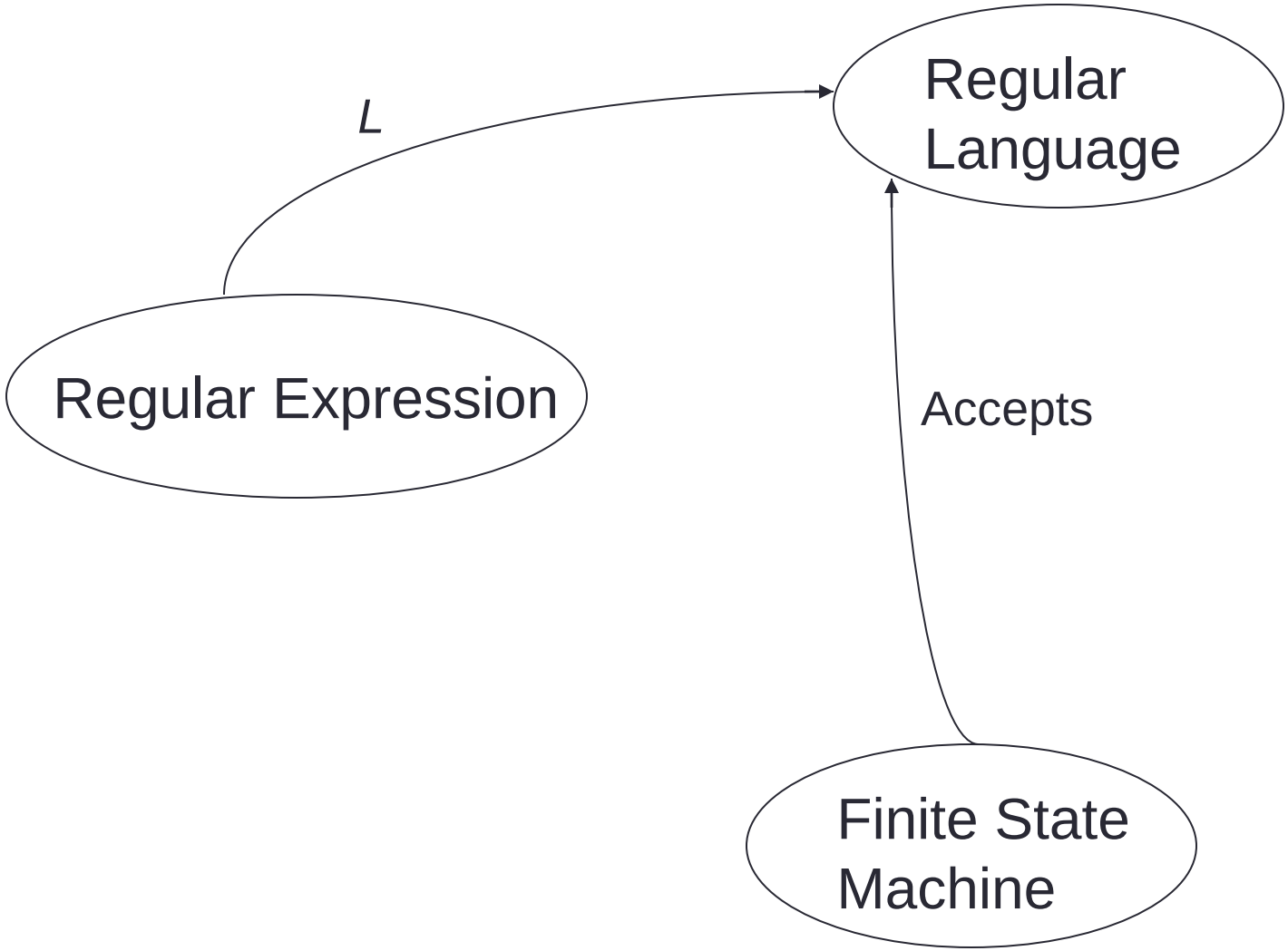
Regular Languages



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 $\alpha \cup \beta$. $(\alpha+\beta)$
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 Expressions

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. $L((\alpha)) = L(\alpha)$.

Regular Sets

- Definition:

1. Basis: ϕ, $\{\varepsilon\}$ and $\{a\}$, for every $a \in \Sigma$ are regular sets over Σ.
2. Recursive step: Let X and Y be regular sets over Σ. Then the sets $X \cup Y$, XY, and X* are regular sets over Σ.
3. Closure: X is a regular set over Σ only if it can be obtained from the basis elements by a finite number of applications of the recursive step

- Examples: Let $\Sigma = \{0, 1\}$
- Following are some regular sets over Σ:
- ϕ, $\{\varepsilon\}$, $\{0\}$, $\{1\}$, $\{00, 11\}$, $\{0\}^*$, $\{0, 1\}^*$

Analyzing a Regular Expression

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

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

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

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

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

Constructing Regular Expression

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

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

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

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

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

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

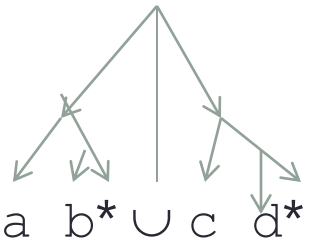
Analyzing a Regular Expression

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

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

Operator Precedence in Regular Expressions

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



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

Kleene's Theorem

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

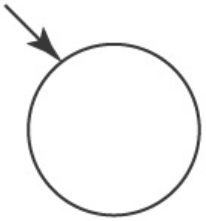
Theorem*:* Any language that can be defined with a regular expression can be accepted by some FSM and so is regular.

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

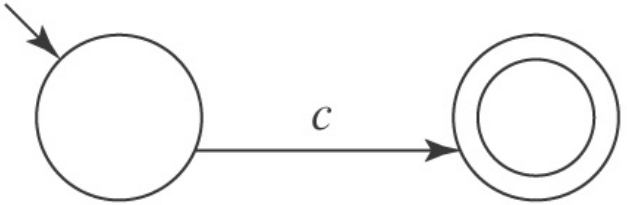
For Every Regular Expression There is a Corresponding FSM

Proof by construction (construct FSM corresponding to re:

∅:



A single element of Σ:



ε:

For Every Regular Expression There is a Corresponding FSM

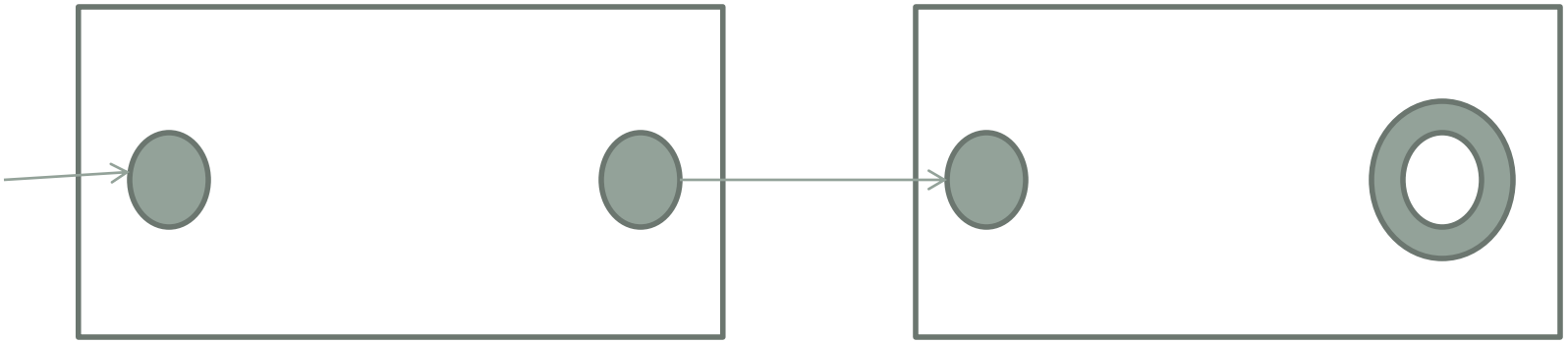
Proof by construction (construct FSM corresponding to re:

If α is the regular expression $\beta \cup \gamma$ and if both $L(\beta)$ and $L(\gamma)$ are regular:

For Every Regular Expression There is a Corresponding FSM

Proof by construction (construct FSM corresponding to re:

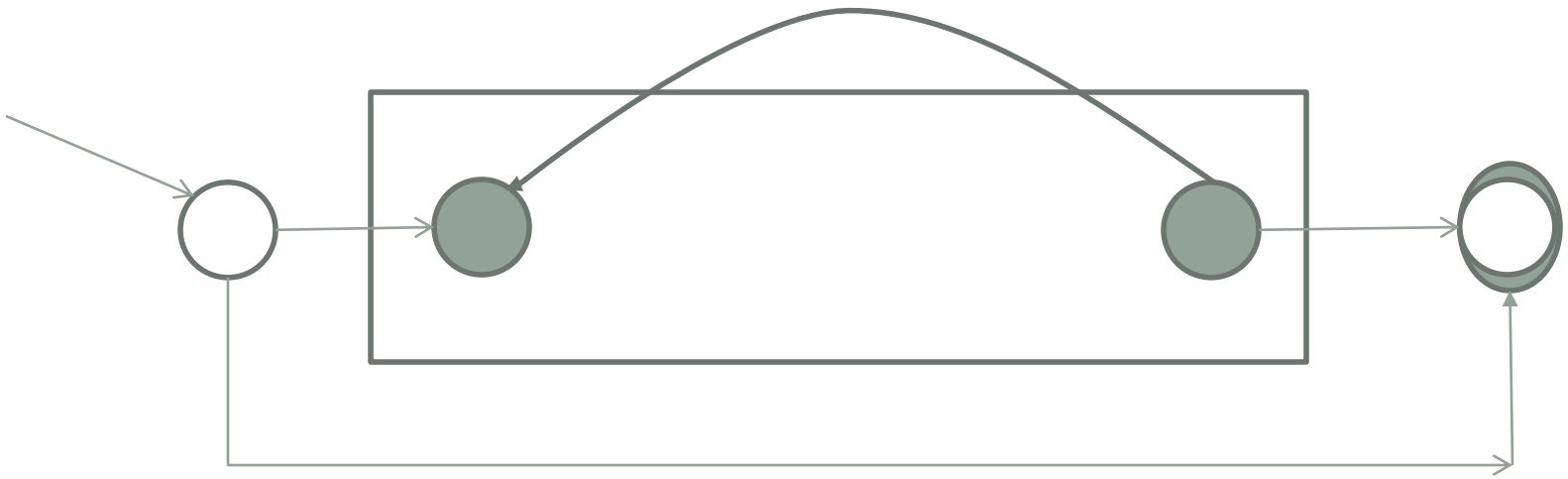
If α is the regular expression $\beta\gamma$ and if both $L(\beta)$ and $L(\gamma)$ are regular:



For Every Regular Expression There is a Corresponding FSM

Proof by construction (construct FSM corresponding to re:

If α is the regular expression β^* and if $L(\beta)$ is regular:

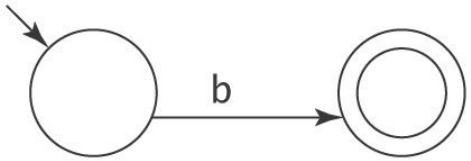


For Every Regular Expression There is a Corresponding FSM

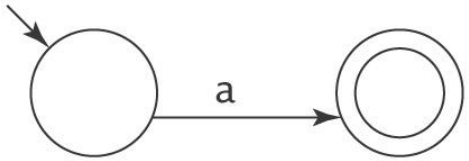
Example:

$(b \cup ab)^*$

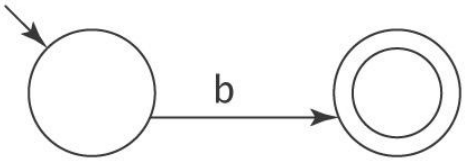
An FSM for b



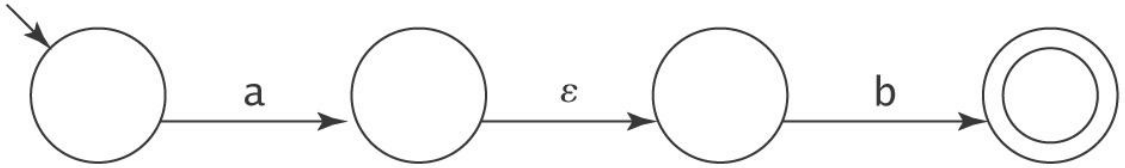
An FSM for a



An FSM for b



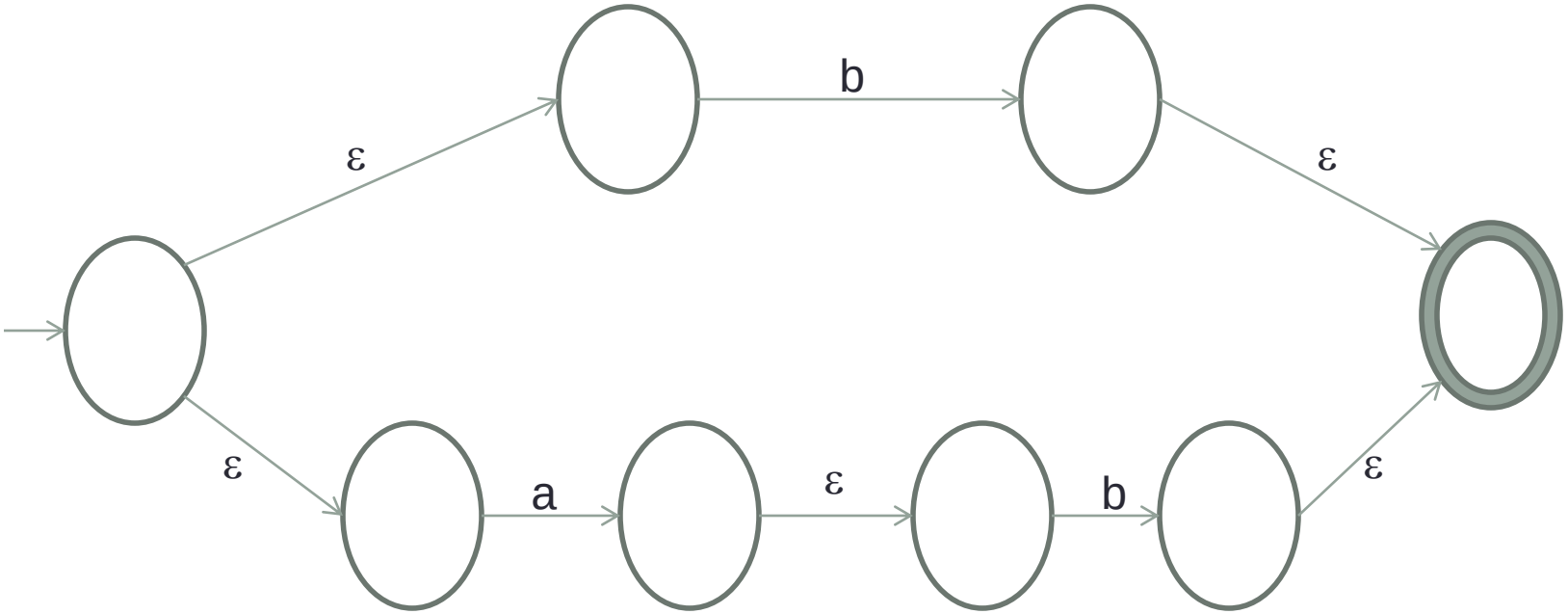
An FSM for ab:



For Every Regular Expression There is a Corresponding FSM

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

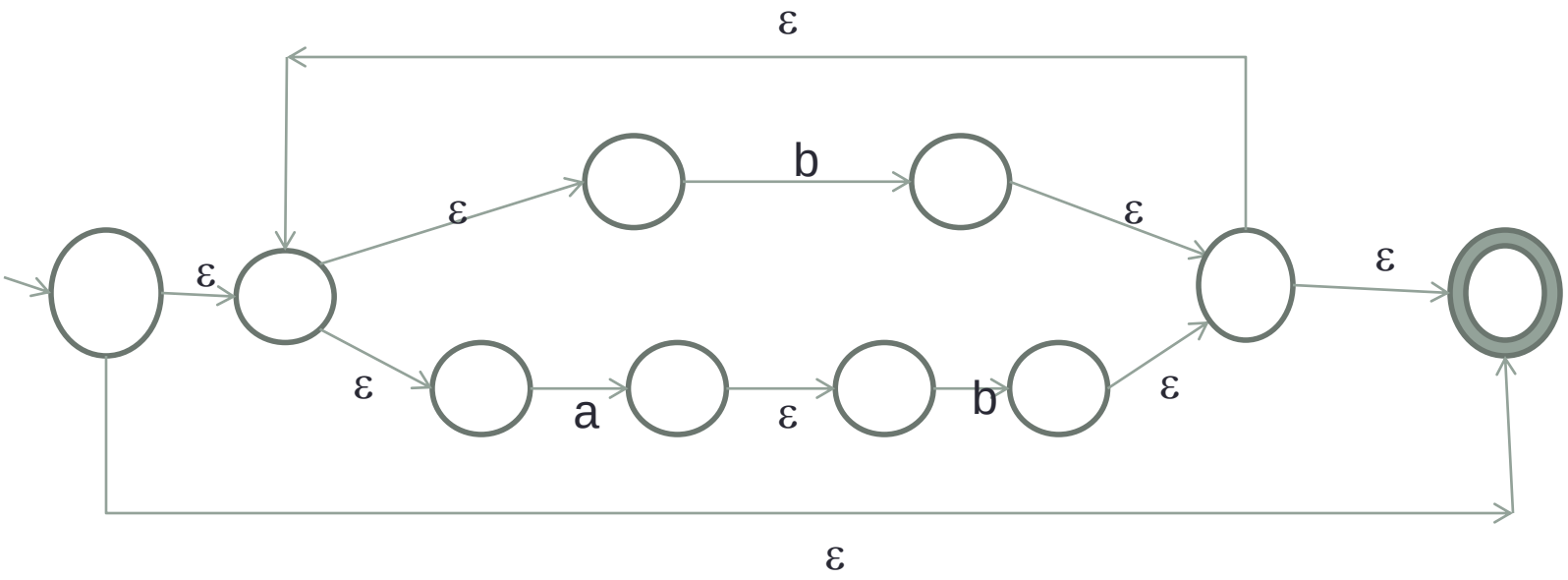
An FSM for $(b \cup ab)$:



For Every Regular Expression There is a Corresponding FSM

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

An FSM for $(b \cup ab)^*$:



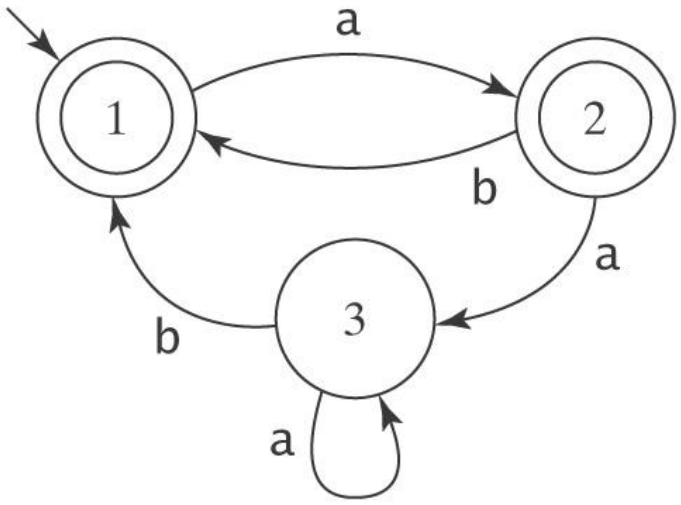
The Algorithm *fsmtoregexheuristic*

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.

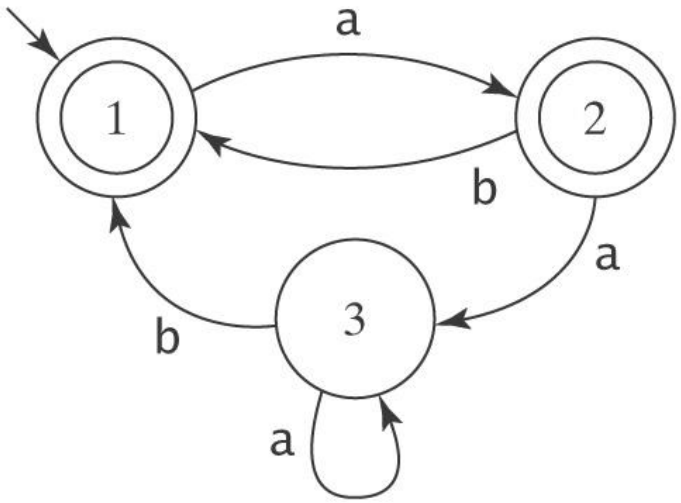
The Algorithm *fsmtoregexheuristic*

Example:



The Algorithm *fsmtoregexheuristic*

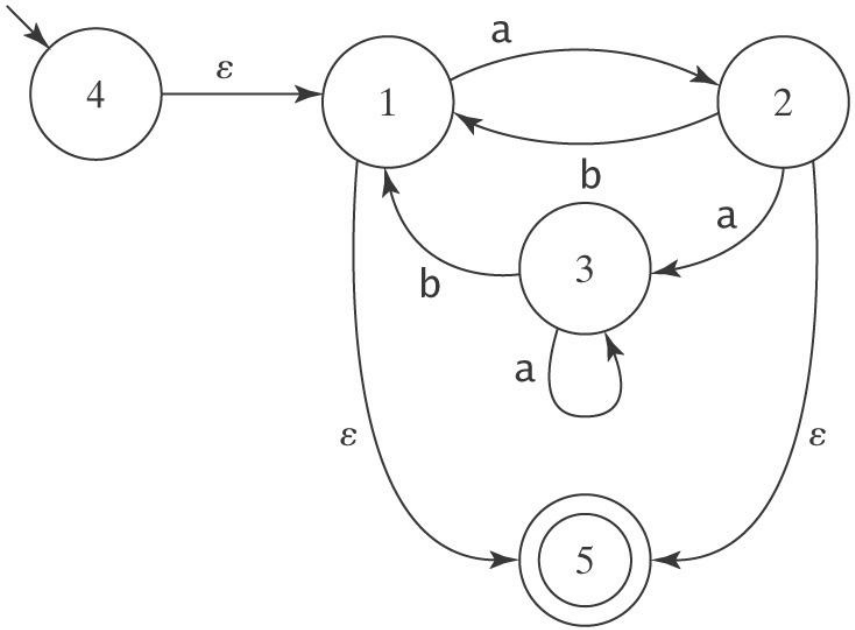
Example:



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

The Algorithm *fsmtoregexheuristic*

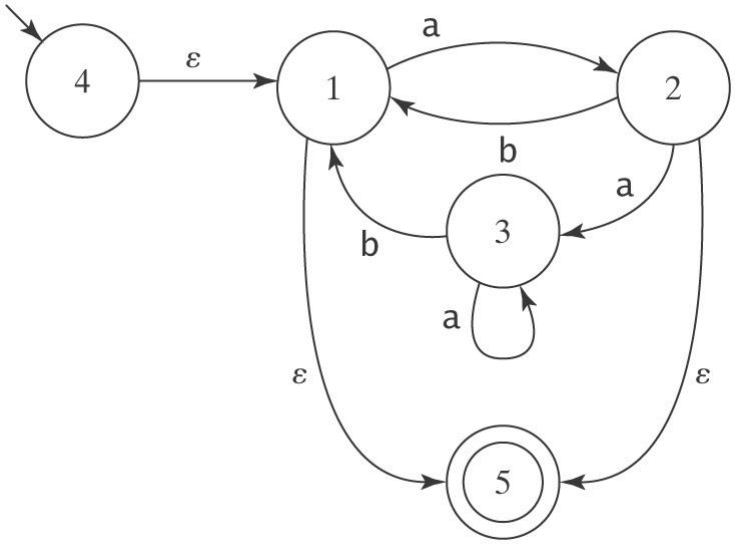
Example continued:



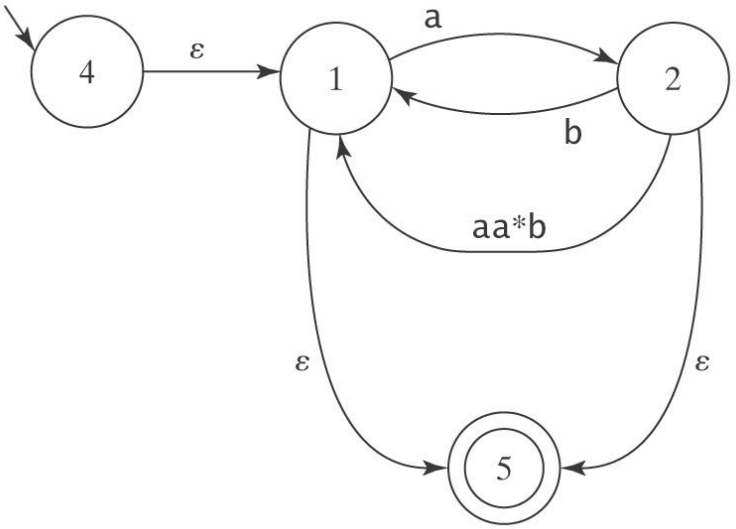
2. Remove states and arcs and replace with arcs labelled with larger and larger regular expressions.

The Algorithm *fsmtoregexheuristic*

Example continued:

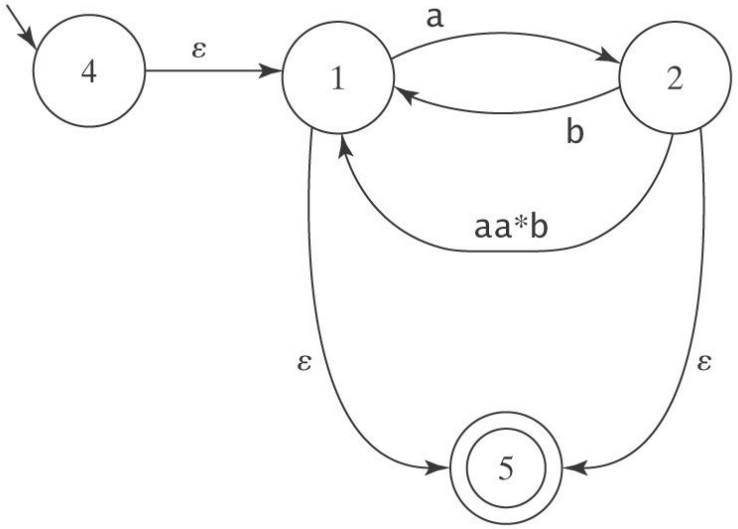


Remove state 3:

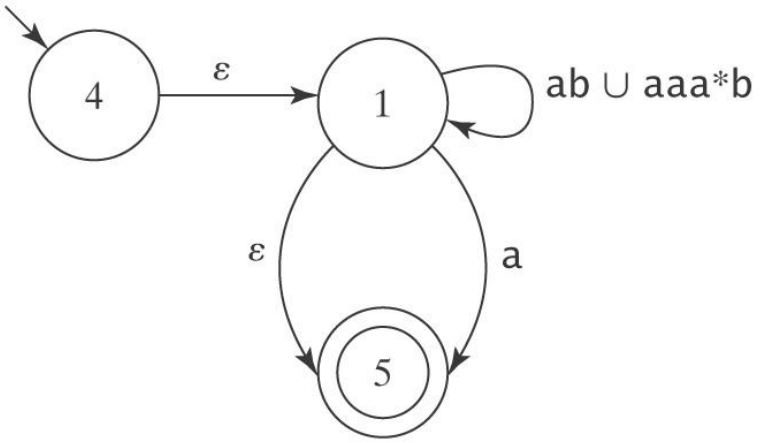


The Algorithm *fsmtoregexheuristic*

Example continued:

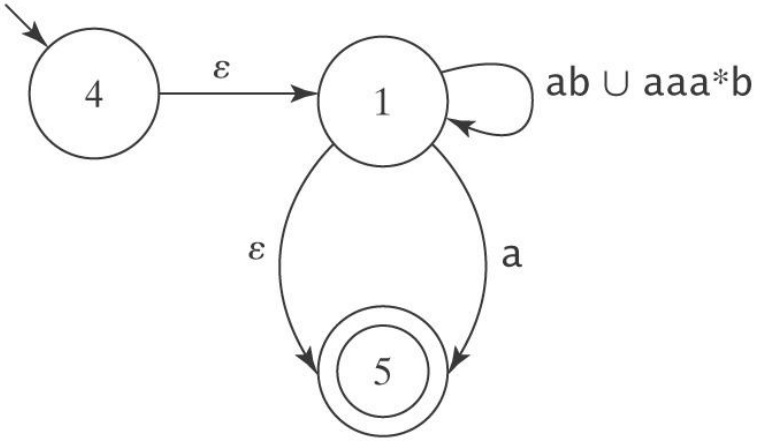


Remove state 2:

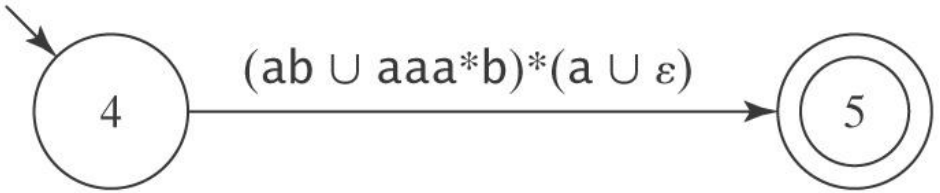


The Algorithm *fsmtoregexheuristic*

Example continued:



Remove state 1:



Further Modifications to *M*

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:
2. If any of the required transitions are missing, add them:
3. Choose a state. Rip it out. Restore functionality

The Algorithm *fsmtoregex*

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*	
	∪	/* 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:

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

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.

Regular Expression Construction

Construction of regular expression from FSM:

$R_{ij}^{(k)}$ – regular expression representing the set of labels of all paths from state i to state j going through intermediate states {1, 2, 3, …, k} only (defined recursively)

$R_{1f}^{(n)}$ – regular expression representing strings accepted by f, f is in F

Equivalent regular expression is the union of all $R_{1f}^{(n)}$

Construction:

BASE:

$R_{ij}^{(0)} = a_1 + a_2 + \ldots + a_k$ where $i \neq j$ and a_m are labels of arcs from state i to state j.

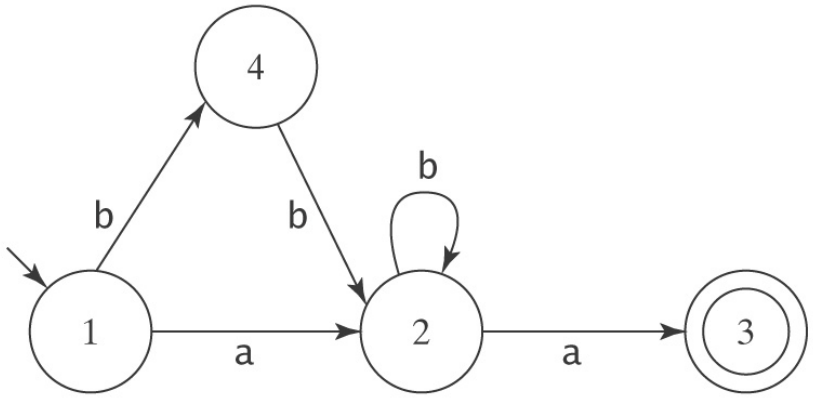
$R_{ij}^{(0)} = \varepsilon + a_1 + a_2 + \ldots + a_k$ where $i = j$ and a_m are labels of arcs from state i to state j.

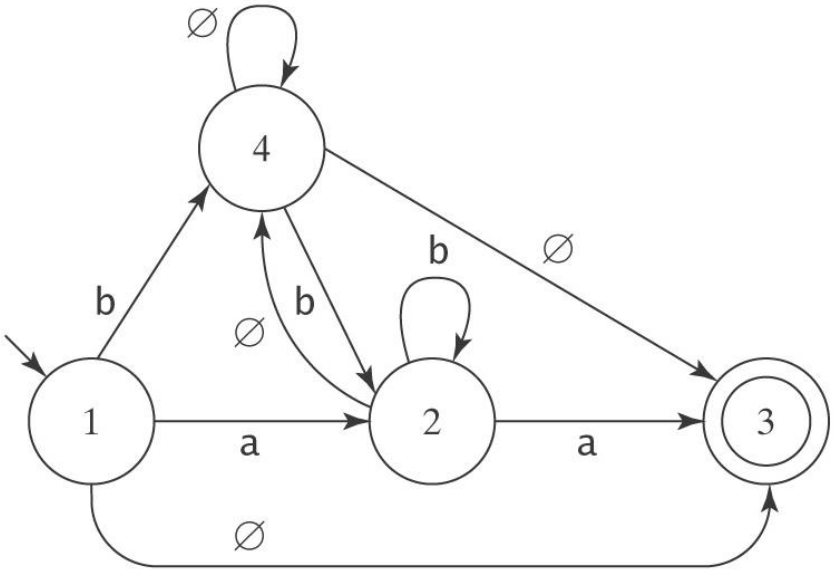
INDUCTON:

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

FSM to Regular Expressions

Example:





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

Let *rip* be state 2. Then:

$$\begin{aligned} R'(1, 3) &= R(1, 3) \cup R(1, rip)R(rip, rip)^*R(rip, 3) \\ &= R(1, 3) \cup R(1, 2)R(2, 2)^*R(2, 3) \\ &= \emptyset \cup \mathtt{a}\ \mathtt{b}^*\ \mathtt{a} \\ &= \mathtt{ab^*a} \end{aligned}$$

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:

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

Applications of regular expressions

- UNIX
- Keyword search
- Given a protein or DNA sequence, find others that are likely to be evolutionarily close to it
- **Using Regular Expressions in the Real World**
  - **Matching numbers**
  - **Matching ip addresses**
  - **Finding doubled words**
  - **Identifying spam**
  - **Trawl for email addresses**
  - **…**