Chapter 1, Part 3

Regular Expression

The regular expressions are equivalent to the finite automata.

Regular Expression

An expression R is a **regular expression** if R is

1. a for some a in some alphabet Σ,
2. ϵ,
3. $\emptyset$,
4. $(R_1 \cup R_2)$ for some regular expressions R_1 and R_2,
5. $(R_1 \circ R_2)$ for some regular expressions R_1 and R_2, or
6. $(R_1)^*$ for some regular expression R_1.

When the meaning is clear from the context, $()$ and $\circ$ can be removed from the expression.

The Language Represented by a Regular Expression

For a regular expression R, $L(R)$ denotes the language R expresses.

1. For each $a \in \Sigma$, $L(a) = \{a\}$.
2. $L(\epsilon) = \{\epsilon\}$
3. $L(\emptyset) = \emptyset$.
4. $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$.
5. $L(R_1 \circ R_2) = L(R_1) \circ L(R_2) = \{uv \mid u \in R_1 \text{ and } v \in R_2\}$.
6. $L(R_1^*) = \{\epsilon\} \cup \{u_1 \cdots u_k \mid u_1, \ldots, u_k \in R_1\}$.

Regular Expression Examples

- $L(a) = \{a\}$ (for a single-element regular expression, you may simply write the element)

- $L(abab \cup bc) = \{abab, bc\}$

- $L((abab \cup abc)^*) = \{\epsilon\} \cup \{w_1 \cdots w_k \mid k \geq 1$ and $w_1, \cdots, w_k \in \{abab, abc\}\}$.

- $L((abab \cup abc)^* \cup c^* \cup abc(abca)^*)$
  $= \{\epsilon\} \cup \{w_1 \cdots w_k \mid k \geq 1$ and $w_1, \cdots, w_k \in \{abab, abc\}\}$
  $\cup \{w \mid w$ is a repetition of c's $\}$
  $\cup \{abc, abcabca, abcabcaabca, \cdots\}$.

Finite Automata $\equiv$ Regular Expression

Theorem. The regular languages are precisely those that are expressed by regular expressions.

We show:

- **Regular expressions can be recognized by finite automata.**
- **Regular languages can be expressed as regular expressions.**

Regular expressions can be recognized by finite automata

By induction

Base Cases

1. a for some a in some alphabet Σ,
2. ϵ,
3. $\emptyset$,

Induction

4. $(R_1 \cup R_2)$ for some regular expressions R_1 and R_2,
5. $(R_1 \circ R_2)$ for some regular expressions R_1 and R_2, or
6. $(R_1)^*$ for some regular expression R_1.

Proof

The Base Case

The following are all regular:

- any set consisting only of a single letter,
- the set consisting only of the empty string,
- the empty set.

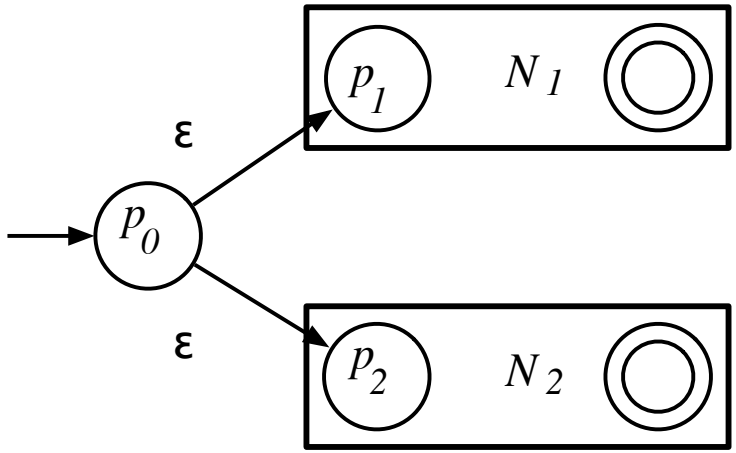
The Induction Step

We have only to show the following:

If L_1 is accepted by an FA $N_1 = (Q_1, \Sigma, \delta_1, p_1, F_1)$ and L_2 is accepted by an FA $N_2 = (Q_2, \Sigma, \delta_2, p_2, F_2)$ (here F_1 and F_2 are disjoint and Q_1 and Q_2 are disjoint), then there are NFA for $L_1 \cup L_2$, $L_1 \circ L_2$, and L_1^*.

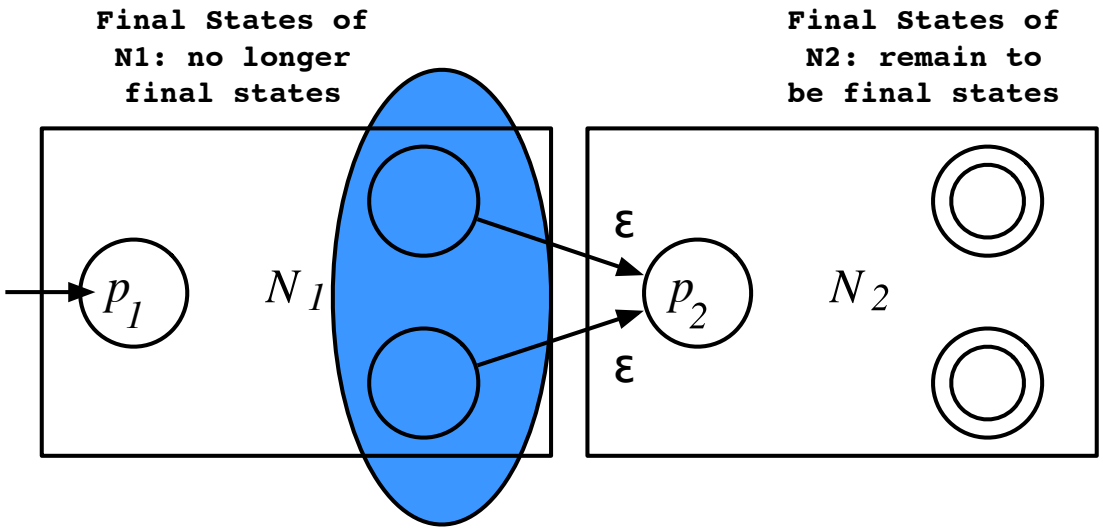
Union

The new NFA has an ϵ-arrow from its initial state p_0 to p_1 and to p_2.
Its final state set is $F_1 \cup F_2$.



Concatenation

The state set of the new NFA is the union of the state sets of N_1 and N_2.
The initial state is p_1.
There is an ϵ-arrow from each $s \in F_1$ to p_2.
The final state set F_0 is F_2.



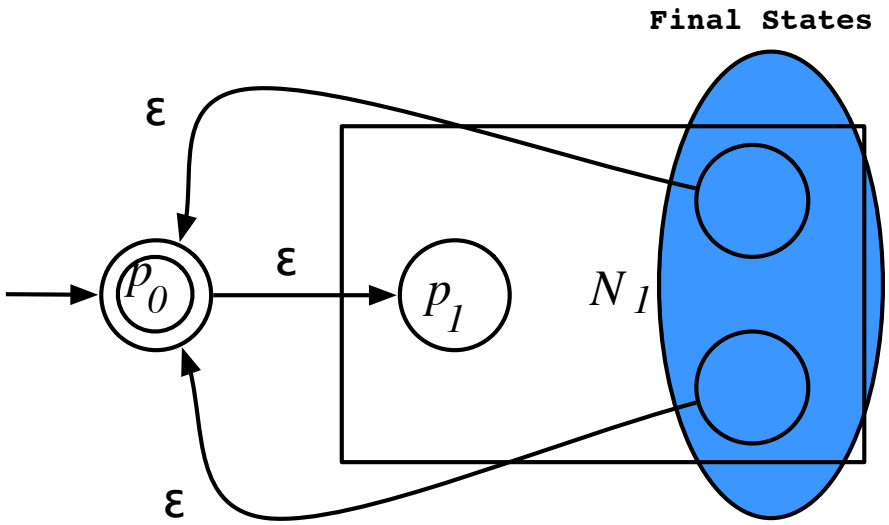
L_1^*

Introduce a new initial state p_0.

Add an ϵ-arrow from p_0 to p_1.

Add an ϵ-arrow from each $s \in F_1$ to p_0.

The final state set is $\{p_0\}$.



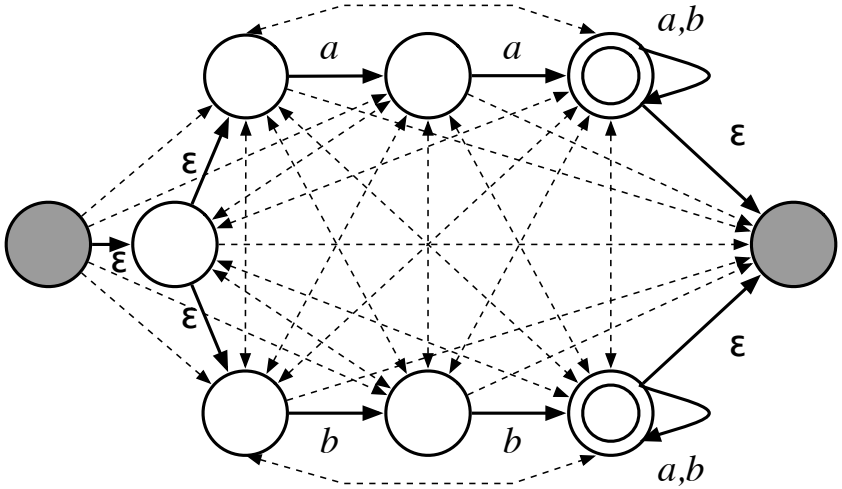
Regular languages can be expressed as regular expressions.

A **general nondeterministic finite automaton (GNFA)** is a kind of NFA such that:

- There is a unique start state and is a unique accept state.
- Every pair of nodes are connected by an arrow in each direction, each labeled with a regular expression. Exceptions are:
  - The start state has no incoming edges.
  - The accept state has no outgoing edges.

The language accepted by the GNFA is the union of all $L(R)$ such that R is the regular expression constructed by concatenating all the regular expressions appearing on the path from the start state to the accept state in the order they appear.

An Example of GNFA



Here the dashed arrows represent arrows with $\emptyset$ as the label.

Proof Plan

1. Show that any NFA can be converted to an equivalent GNFA.
2. Show that any GNFA can be converted to a regular expression.

From an NFA to a GNFA

Given an NFA N, construct a GNFA G as follows:

1. Add a special start state q_{start} and connect it to the initial state of N with ϵ as the label. Connect q_{start} to each remaining state with $\emptyset$ as the label.

From an NFA to a GNFA

Given an NFA N, construct a GNFA G as follows:

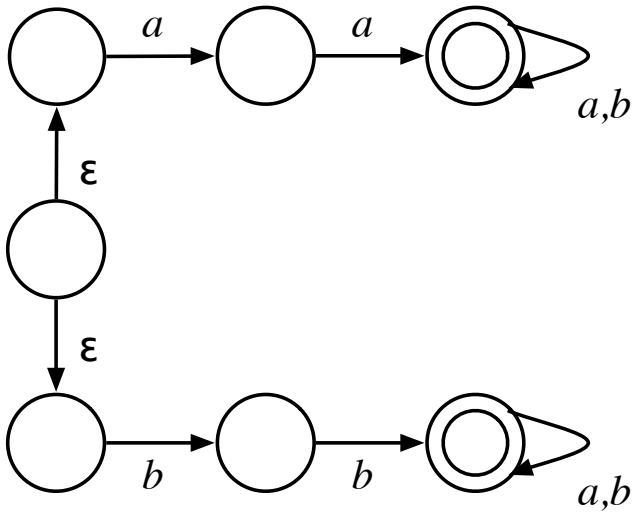
1. Add a special start state q_{start} and connect it to the initial state of N with ϵ as the label. Connect q_{start} to each remaining state with $\emptyset$ as the label.
2. Add a special accept state q_{accept} and connect to it from each final state of N with ϵ as the label. Connect from each of the remaining state of N to q_{accept} with $\emptyset$ as the label.

From an NFA to a GNFA

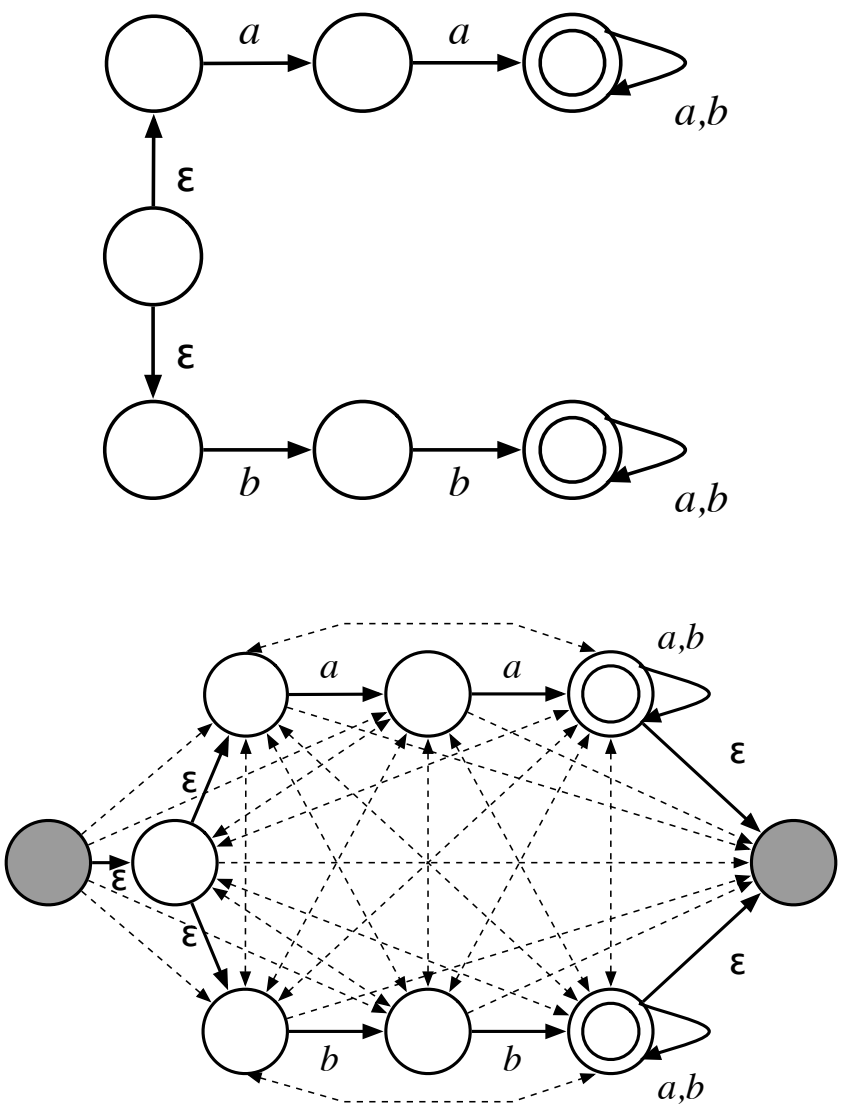
Given an NFA N, construct a GNFA G as follows:

1. Add a special start state q_{start} and connect it to the initial state of N with ϵ as the label. Connect q_{start} to each remaining state with $\emptyset$ as the label.
2. Add a special accept state q_{accept} and connect to it from each final state of N with ϵ as the label. Connect from each of the remaining state of N to q_{accept} with $\emptyset$ as the label.
3. Add an arrow with $\emptyset$ as the label wherever necessary.

Before and After



Before and After



What Have We Done?

We have constructed our initial GNFA G so that:

(I) For each word w in $L(N)$, there is a path $[u_0, \ldots, u_m]$ in G from q_{start} to q_{accept} such that
   (a) w is decomposed as $w_1 w_2 \cdots w_m$;
   (b) for all i, $1 \leq i \leq m$, w_i is a member of the language represented by the regular expression of the arrow (u_{i-1}, u_i).

What Have We Done?

We have constructed our initial GNA G so that:

(I) For each word w in $L(N)$, there is a path $[u_0, \ldots, u_m]$ in G from q_{start} to q_{accept} such that
   (a) w is decomposed as $w_1 w_2 \cdots w_m$;
   (b) for all i, $1 \leq i \leq m$, w_i is a member of the language represented by the regular expression of the arrow (u_{i-1}, u_i).

(II) For each word w that can be decomposed as in the above, w is a member of $L(N)$.

Convert a GNFA to a Regular Expression

We will remove intermediate nodes one after the other while preserving the two properties.

Repeat the following until there is no state left other than the start state and the accept state.

- Select an arbitrary state s.
- Produce from the current GNFA an equivalent GNFA by:
  - For each arrow (p, q) such that $p, q \neq s$, replace its label with the label corresponding to the paths $[p, q], [p, s, q], [p, s, s, q], [p, s, s, s, q], \ldots$.
  - Remove s and all edges attached to it.

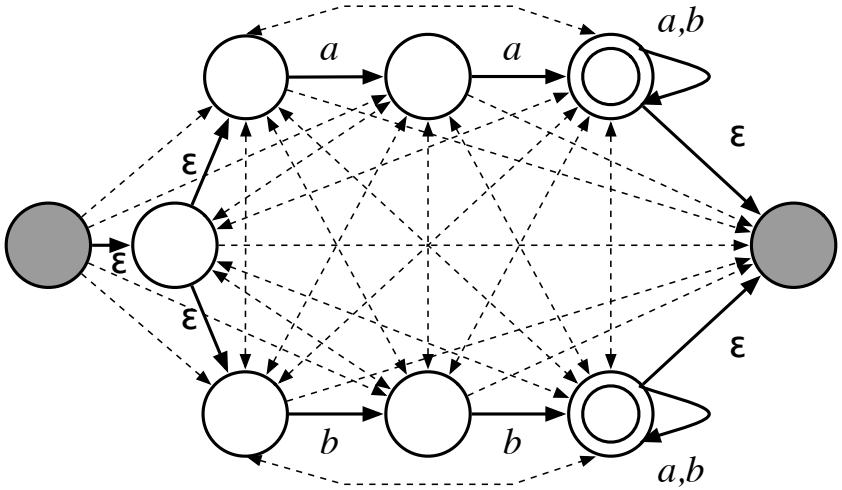
The Procedure

Let G be the current GNFA. Replace G with a new GNFA G' constructed from G as follows:

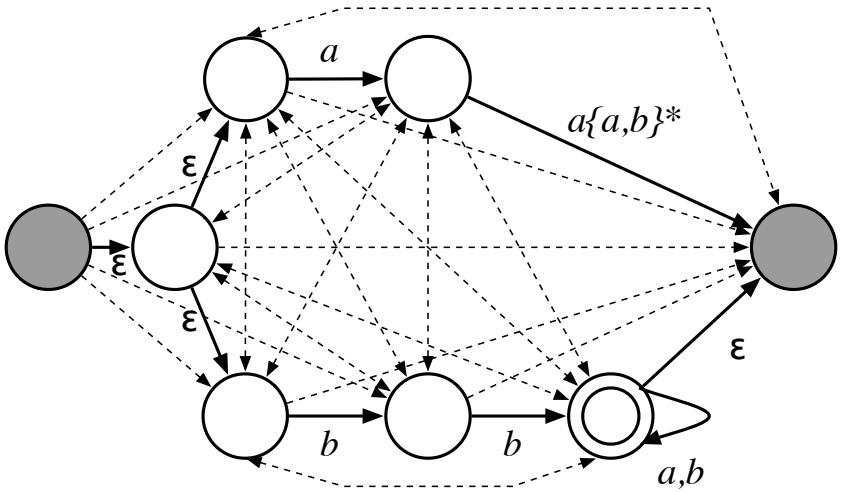
- G' has all states of G except for s.
- For each pair (p, q) of states, $p, q, \neq s$, label the arrow (p, q) of G' by $u \cup xy^*z$, where
  - **u is the label of (p, q) in G,**
  - **x is the label of (p, s) in G,**
  - **y is the label of (s, s) in G,**
  - **z is the label of (s, q) in G,**
  - **if at least one of x, y and z is $\emptyset$, the label is u.**

The two properties (I) and (II) are preserved.

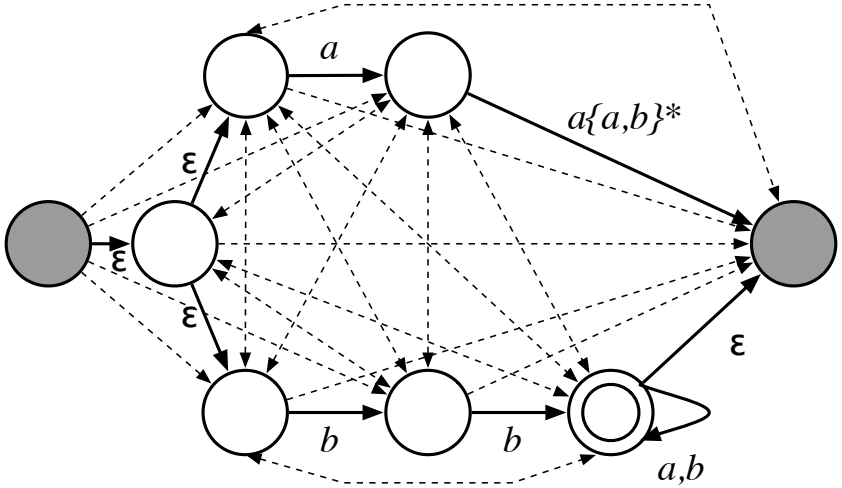
State Elimination



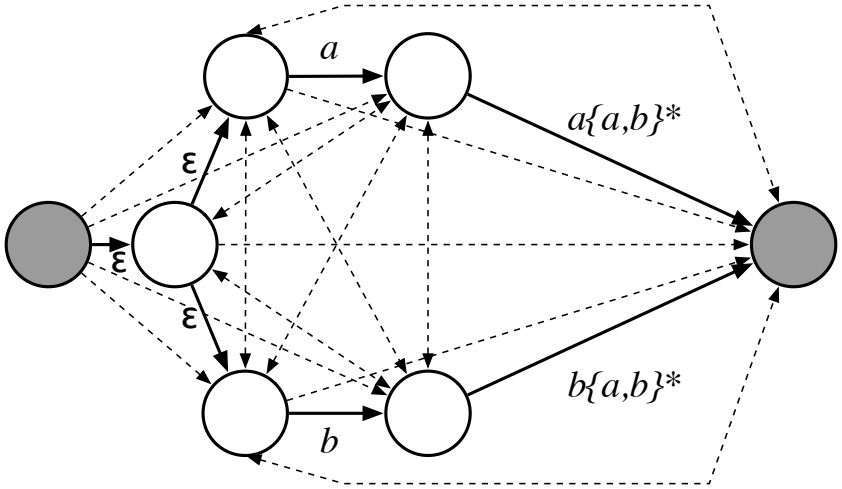
becomes



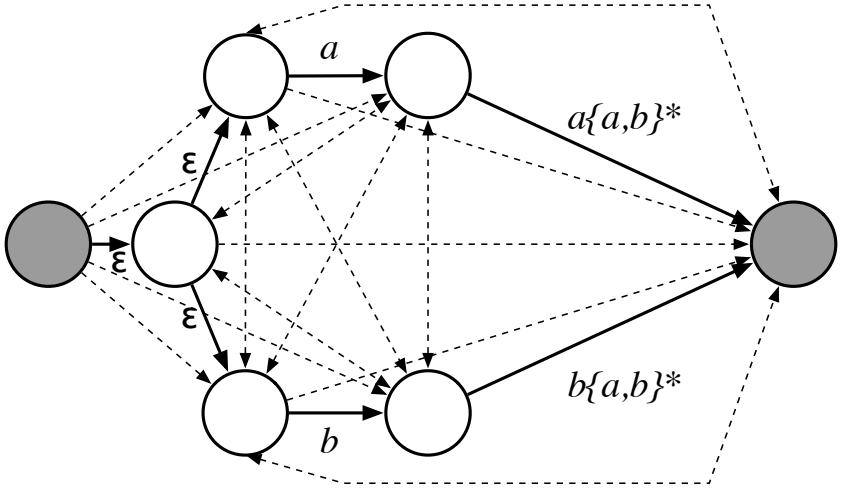
State Elimination



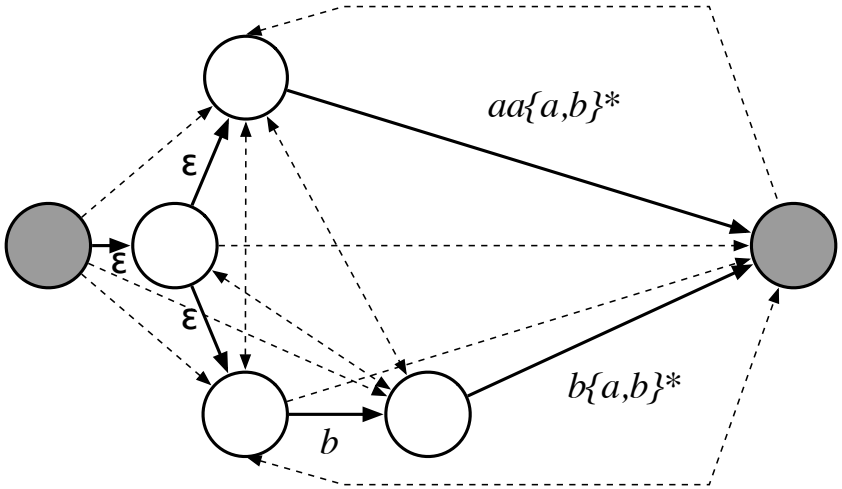
becomes



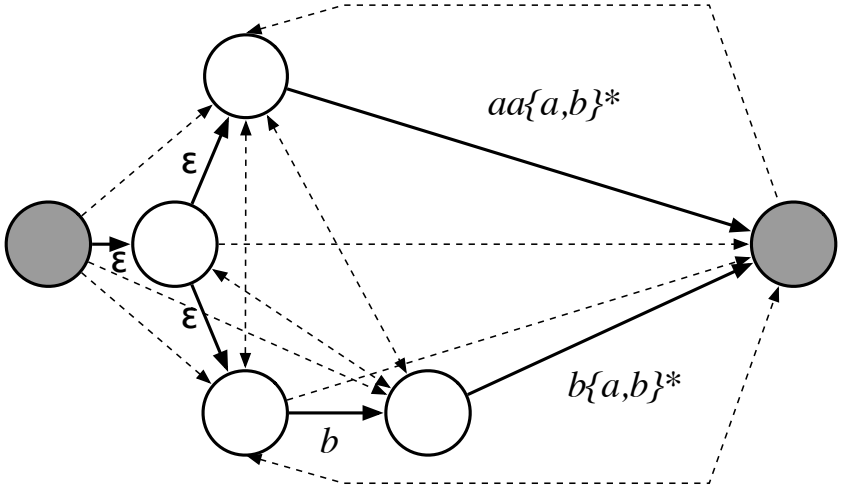
State Elimination



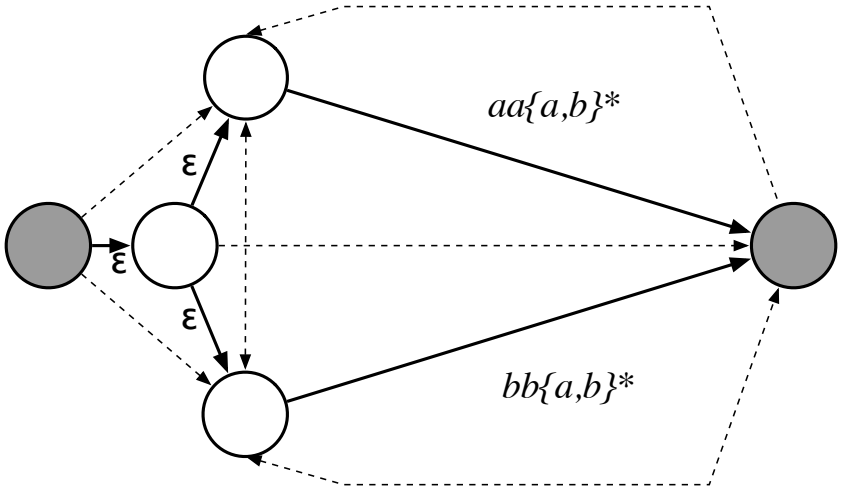
becomes



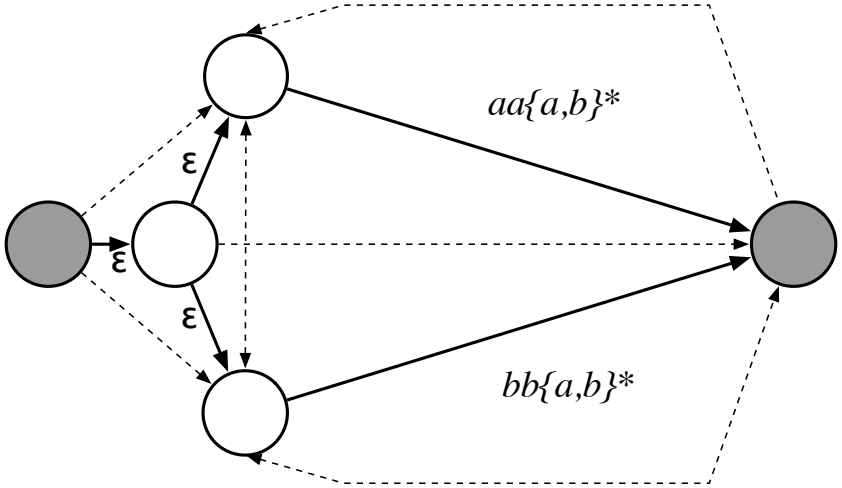
State Elimination



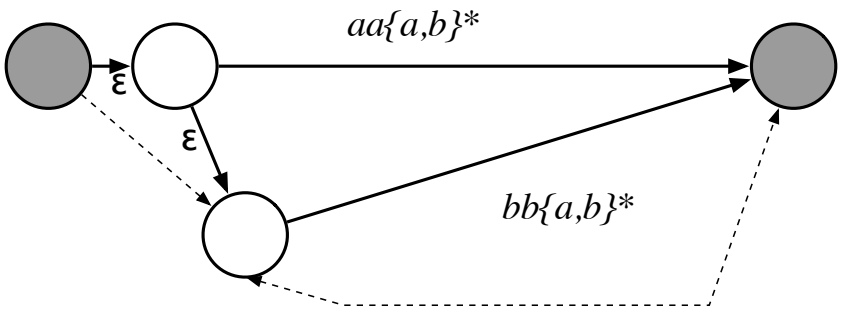
becomes



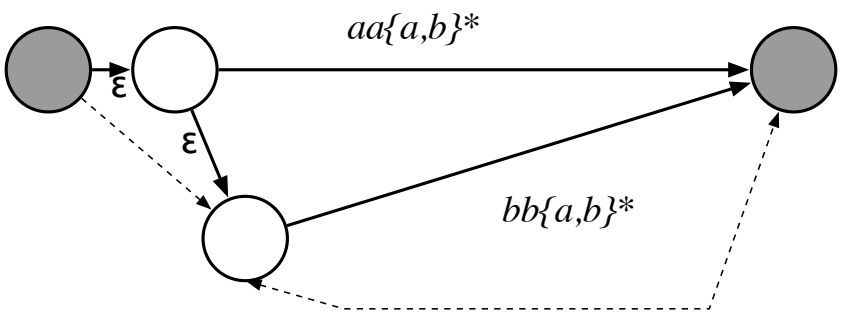
State Elimination



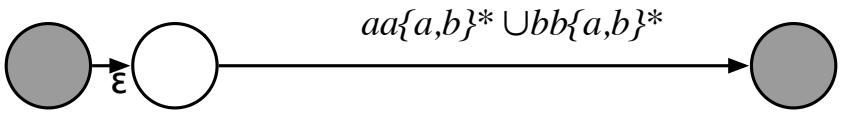
becomes



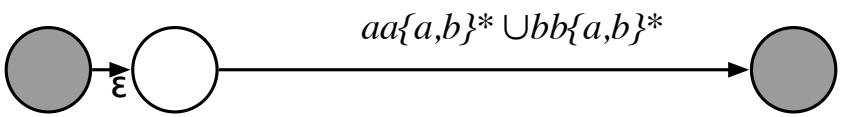
State Elimination



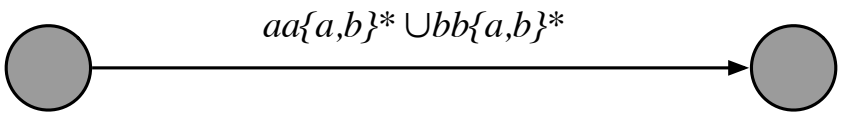
becomes



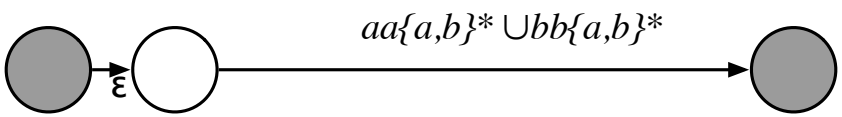
State Elimination



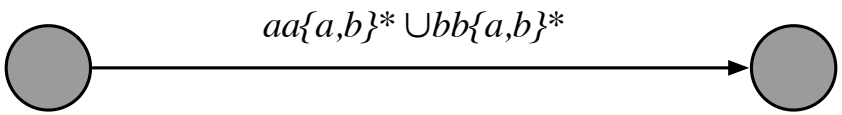
becomes



State Elimination



becomes



The conversion has been completed.

The Regular Expression May Not Be Unique

Note that the regular expression is dependent on the order in which the states are eliminated and thus may not be unique.

Closure Properties of Regular Languages

Theorem. **The regular languages are closed under complement, union, intersection, concatenation, and star.**

Proof The closure properties under union, concatenation, and star follow from the fact that **the regular languages are those that are expressible with regular expressions**.

For complement, note that the complement of a language accepted by a DFA $(Q, \Sigma, \delta, p_0, F)$ is accepted by a DFA $(Q, \Sigma, \delta, p_0, Q \setminus F)$.

For intersection, use **DeMorgan's Law.** ∎