Kleene's theorem

We've seen that every regular expression defines a regular language.

The main goal of today's lecture is to show the converse, that every regular language can be defined by a regular expression. For this purpose, we introduce Kleene algebra: the algebra of regular expressions.

The equivalence between regular languages and expressions is: Kleene's theorem

> *DFAs and regular expressions give rise to exactly the same class of languages (the regular languages).*

As we've already seen, NFAs (with or without ϵ-transitions) also give rise to this class of languages.

So the evidence is mounting that the class of regular languages is mathematically a very 'natural' class to consider.

Kleene algebra

Regular expressions give a textual way of specifying regular languages. This is useful e.g. for communicating regular languages to a computer.

Another benefit: regular expressions can be manipulated using algebraic laws (Kleene algebra). For example:

$$
\begin{array}{rcl}
\alpha + (\beta + \gamma) & \equiv & (\alpha + \beta) + \gamma \\
\alpha + \emptyset & \equiv & \alpha \\
\alpha(\beta\gamma) & \equiv & (\alpha\beta)\gamma \\
\alpha(\beta + \gamma) & \equiv & \alpha\beta + \alpha\gamma \\
\emptyset\alpha & \equiv & \alpha\emptyset \ \equiv \ \emptyset
\end{array}
\qquad
\begin{array}{rcl}
\alpha + \beta & \equiv & \beta + \alpha \\
\alpha + \alpha & \equiv & \alpha \\
\epsilon\alpha & \equiv & \alpha\epsilon \ \equiv \ \alpha \\
(\alpha + \beta)\gamma & \equiv & \alpha\gamma + \beta\gamma \\
\epsilon + \alpha\alpha^* & \equiv & \epsilon + \alpha^*\alpha \equiv \alpha^*
\end{array}
$$

Often these can be used to simplify regular expressions down to more pleasant ones.

Other reasoning principles

Let's write $\alpha \leq \beta$ to mean $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\beta)$ (or equivalently $\alpha + \beta \equiv \beta$). Then

$$\begin{aligned} \alpha\gamma + \beta \leq \gamma \quad &\Rightarrow \quad \alpha^*\beta \leq \gamma \\ \beta + \gamma\alpha \leq \gamma \quad &\Rightarrow \quad \beta\alpha^* \leq \gamma \end{aligned}$$

Arden's rule: Given an equation of the form $X = \alpha X + \beta$, its smallest solution is $X = \alpha^*\beta$.

What's more, if $\epsilon \notin \mathcal{L}(\alpha)$, this is the *only* solution.

Intriguing fact: The rules on this slide and the last form a complete set of reasoning principles, in the sense that if $\mathcal{L}(\alpha) = \mathcal{L}(\beta)$, then '$\alpha \equiv \beta$' is provable using these rules. (Beyond scope of Inf2A.)

Regular Expressions

Every *regular language* can be represented by a *regular expressions*, and every *regular expression* represents a *regular language*.

- $\emptyset$ and ϵ are regular expressions.
- $\forall a \in \Sigma$, a is a regular expression.
- If R_1, R_2 are regular expressions, them the following are regular expressions in order of precedence with the first having the highest precedence.

(R_1)	parentheses
R_1^*	closure
$R_1 R_2$	concatenation
$R_1 + R_2$	alternation

Given regular expression R, $L(R)$ stands for the language represented by R.

Regular Expressions (2)

Some Examples:

Regular Language	Corresponding Regular Expression
$\emptyset$	$\emptyset$
$\{a\}$	a
$\{a, b\}^*$	$(a+b)^*$
$\{aab\}^* a, ab$	$(aab)^*(a+ab)$
$(\{aa, bb\} \cup \{ab, ba\}\{aa, bb\}^*\{ab, ba\})^*$	$(aa+bb+(ab+ba)(aa+bb)^*(ab+ba))^*$

Some properties of regular expressions.

- $R^* = R^* R^* = (R^*)^* = R + R^*$
- $R_1(R_2 R_1)^* = (R_1 R_2)^* R_1$
- $(R_1^* R_2)^* = \epsilon + (R_1 + R_2)^* R_2$
- $(R_1 R_2^*)^* = \epsilon + R_1(R_1 + R_2)^*$

Some More Examples

Let $\Sigma = \{a, b\}$.

Regular Expression	Language	Comments
$(a+b)(a+b)$	$\{aa, ab, ba, bb\}$	$(a+b)(a+b) = aa+ab+ba+bb$
$(a+b)^*$	all strings of a's and b's *including* ϵ	$(a+b)^* = (a^*b^*)^*$
$a + a^*b$	$\{a, b, ab, aab, aaab, \ldots\}$	note order of precedence
$b^*ab^*(ab^*ab^*)^*$	string with an odd number of a's	Other valid expressions are $b^*a(b+ab^*a)^*$ or $(b+ab^*a)^*ab^*$

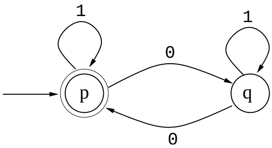
Deterministic Finite Automata

A *deterministic finite automata* (DFA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$ where

- Q is a **finite** set of *states*;
- Σ is a **finite** *input alphabet*;
- $q_0 \in Q$ is the *initial state*;
- $A \subseteq Q$ is the set of *accepting states*;
- $\delta : Q \times \Sigma \to Q$ is the *transition function*.

For any element $q \in Q$ and any symbol $\sigma \in \Sigma$, $\delta(q, \sigma)$ is the state the DFA moves to if it receives input σ while in state q.

DFAs to regular expressions



For each state a, let X_a stand for the set of strings that take us from a to an accepting state. Then we can write some equations:

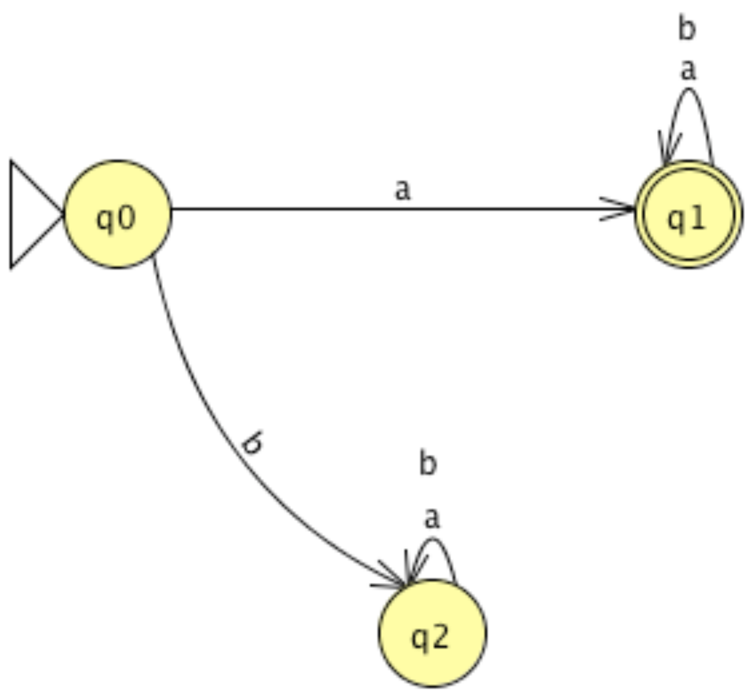
$$\begin{aligned} X_p &= 1.X_p + 0.X_q + \epsilon \\ X_q &= 1.X_q + 0.X_p \end{aligned}$$

Solve by eliminating one variable at a time:

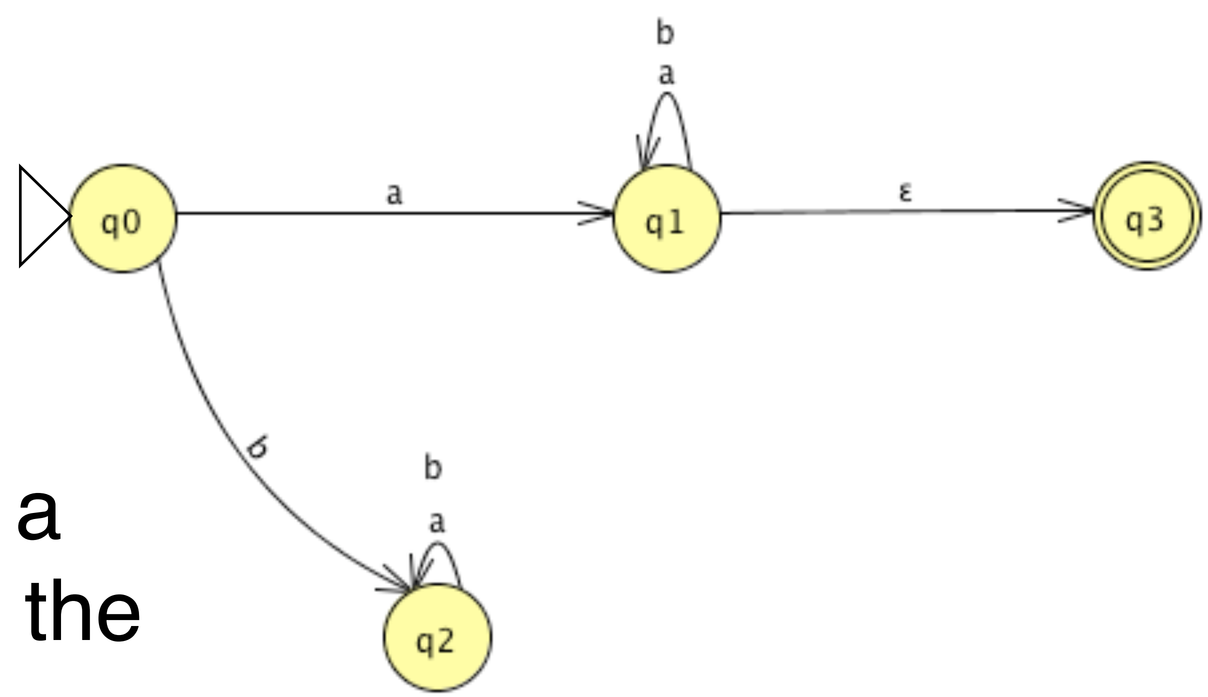
$$\begin{array}{rrcll} & X_q & = & 1^*0.X_p & \text{by Arden's rule} \\ \text{So} & X_p & = & 1.X_p + 01^*0X_p + \epsilon & \\ & & = & (1+01^*0)X_p + \epsilon & \\ \text{So} & X_p & = & (1+01^*0)^* & \text{by Arden's rule} \end{array}$$

Example

0. If there is no arc from state *i* to state *j*, imagine one with label ∅.

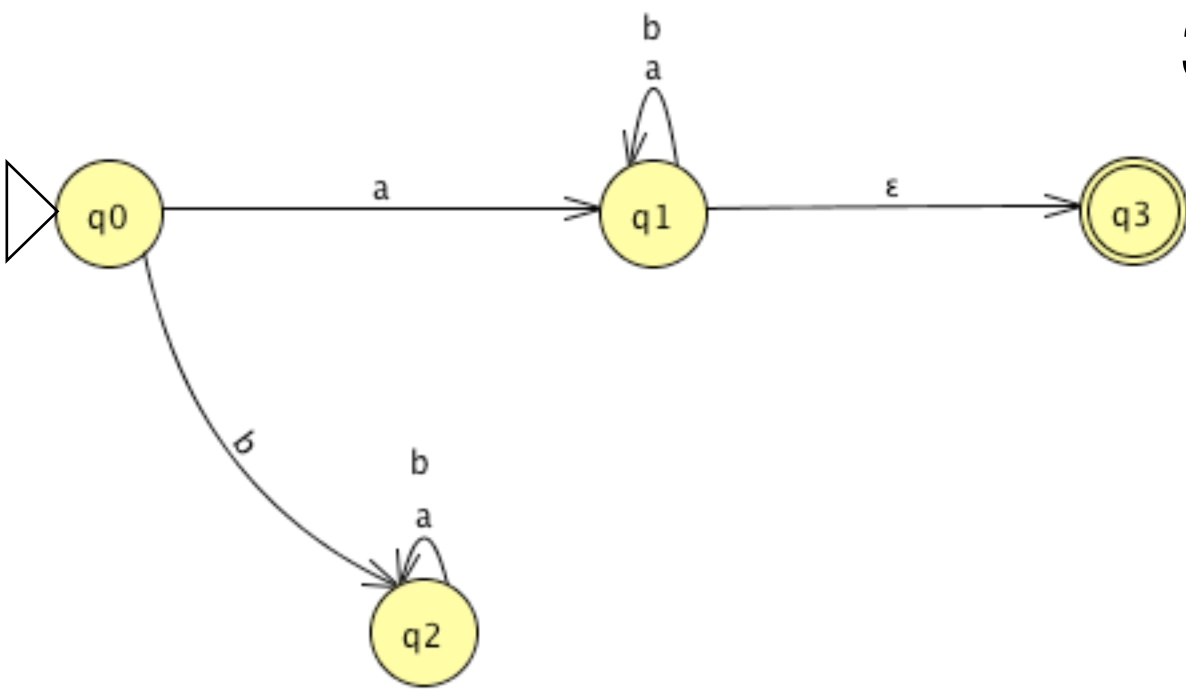


1. If the initial state has a self-transition, create a new initial state with a single ε-transition to the old initial state.

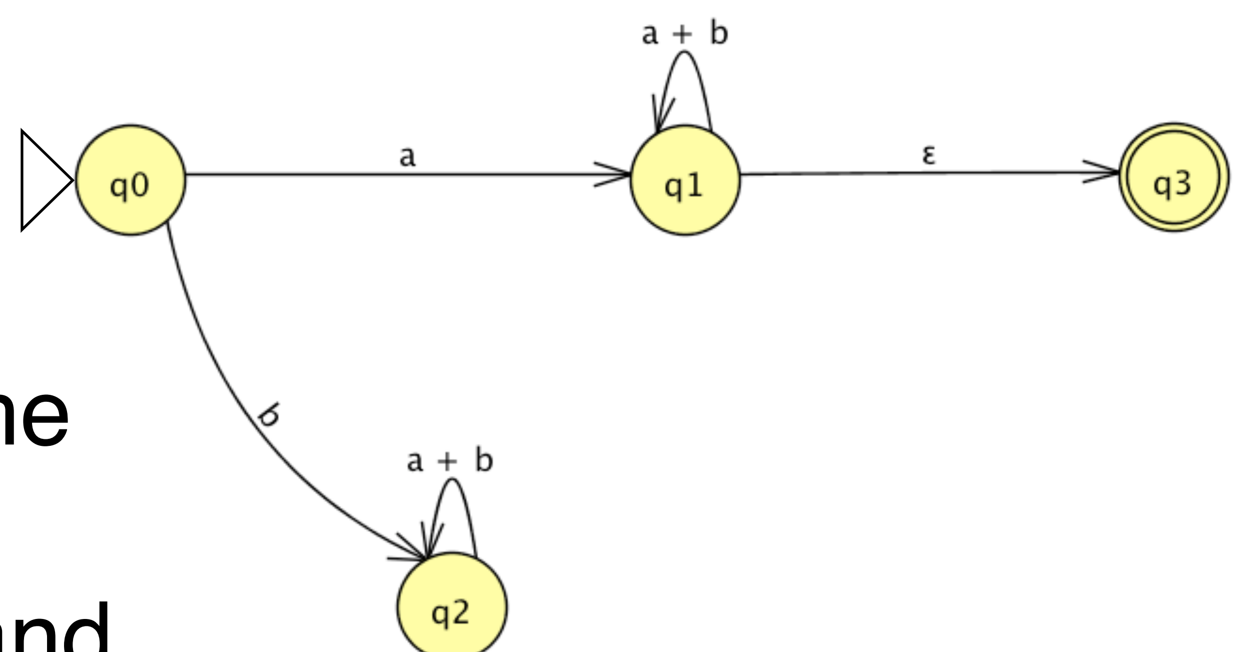


2. Create a new final state with a ε-transition to it from each of the old final states.

Example

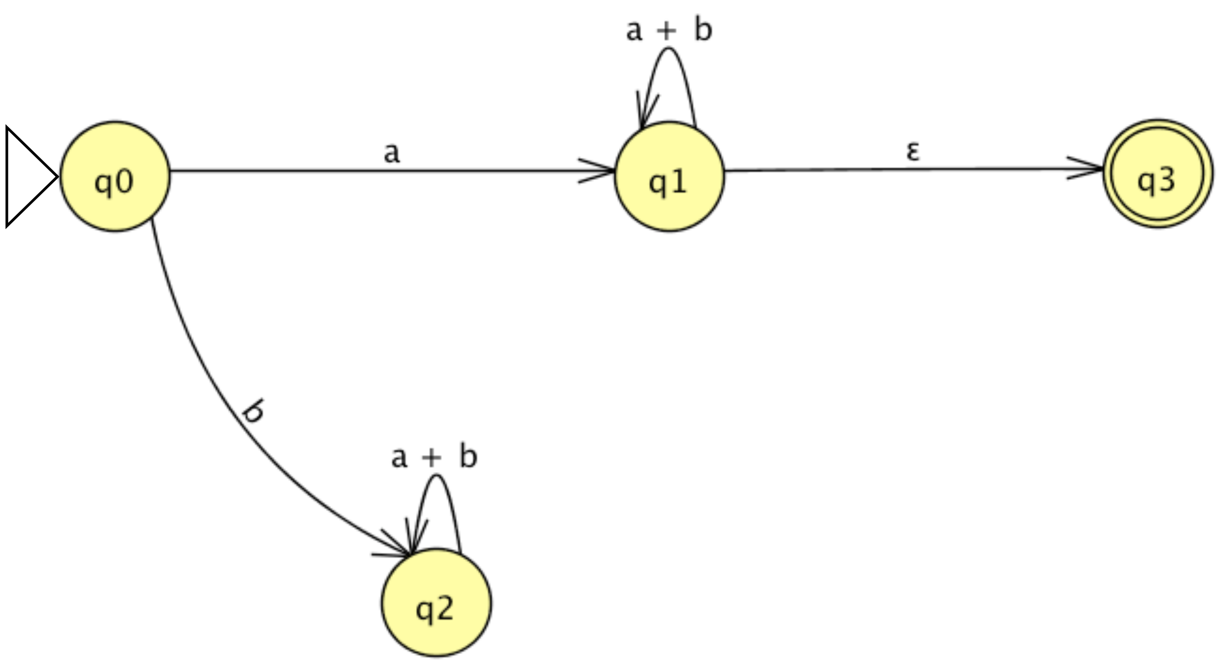


3. For each pair of states *i*, *j* with more than one transition from *i* to *j*, replace them all by a single transition labeled with the r.e. that is the sum of the old labels.



4. Eliminate one state at a time until the only states that remain are the start state and the final state:

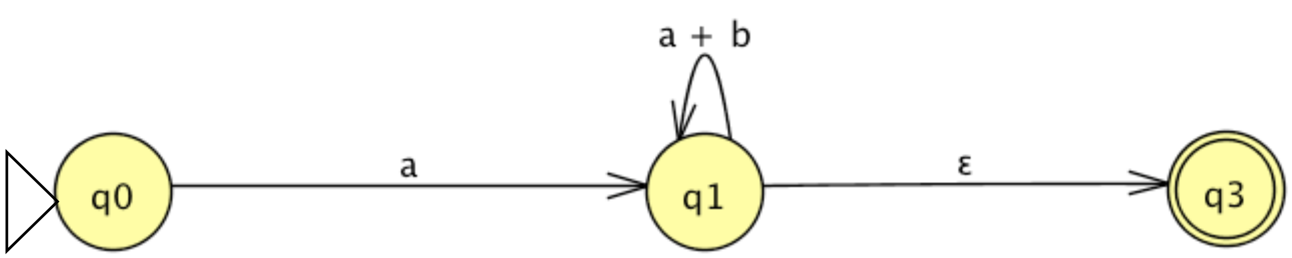
How to Eliminate State *k*



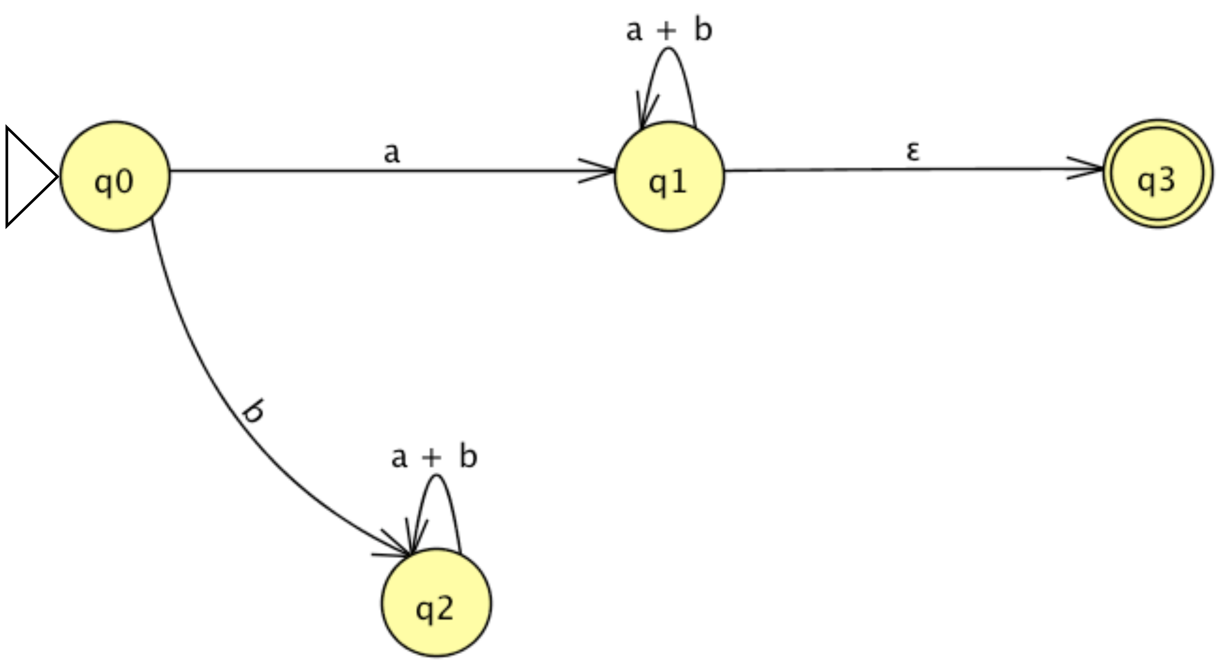
- For each pair of nodes *i, j* (*i* ≠ *k*, *j* ≠ *k*), label the transition from *i* to *j* with:

$$(i, j) + (i, k)(k, k)^*(k, j)$$

- Remove state *k* and all its transitions.



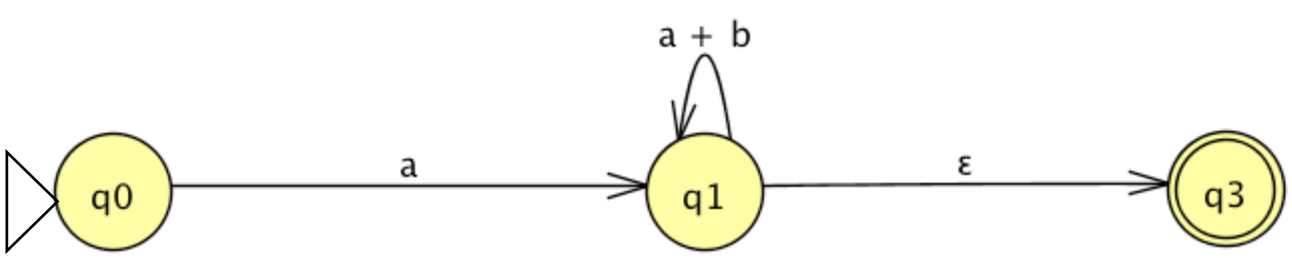
How to Eliminate State *k*

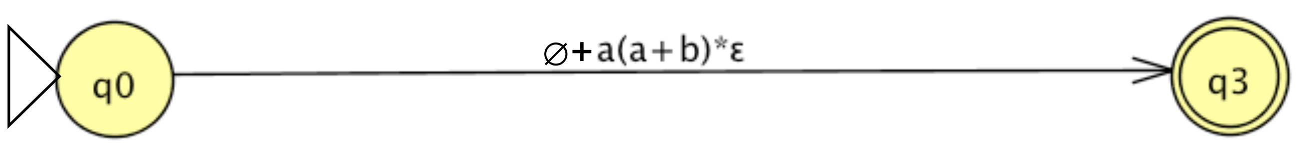


- For each pair of nodes *i, j* (*i* ≠ *k*, *j* ≠ *k*), label the transition from *i* to *j* with:

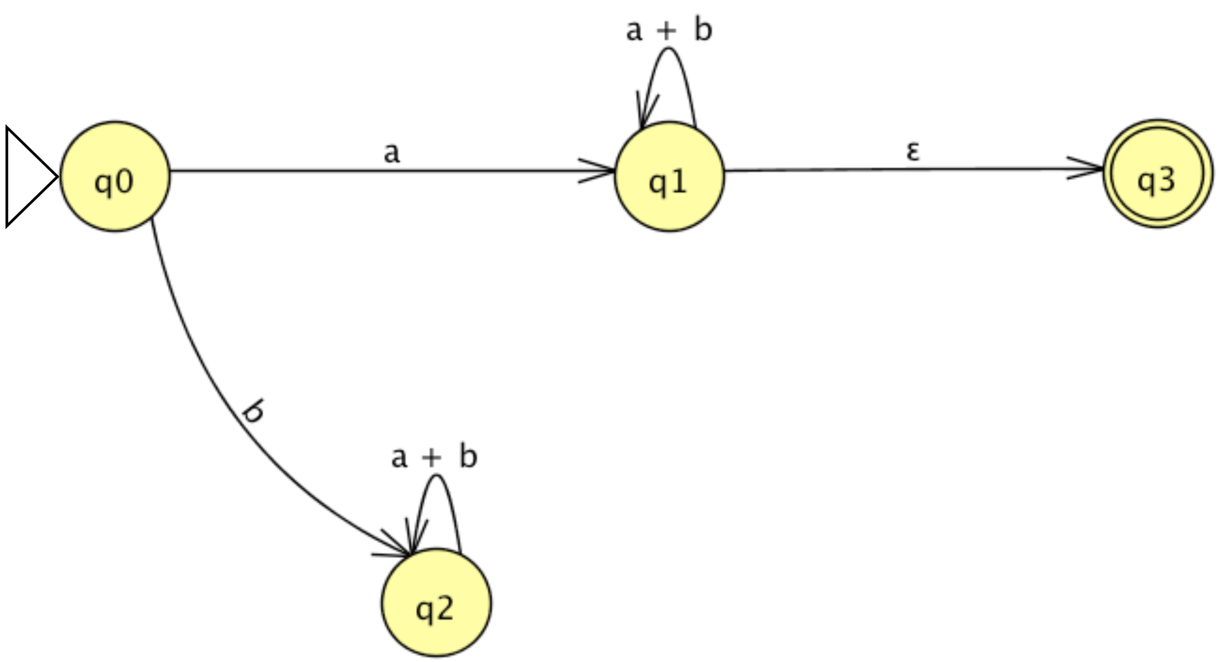
$$(i, j) + (i, k)(k, k)^*(k, j)$$

- Remove state *k* and all its transitions.





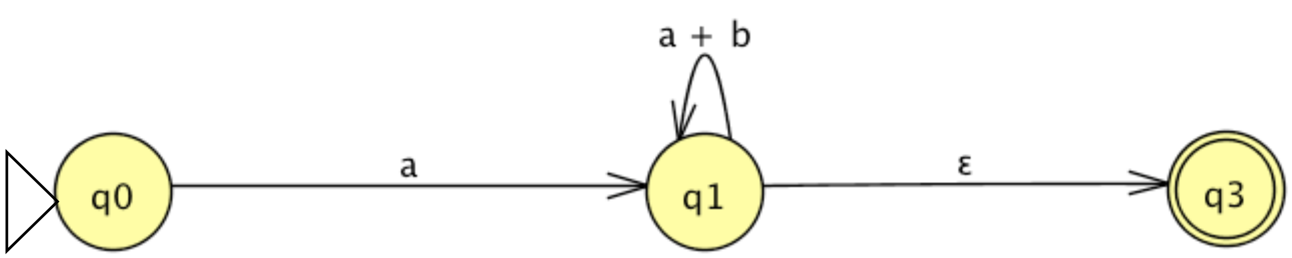
How to Eliminate State *k*

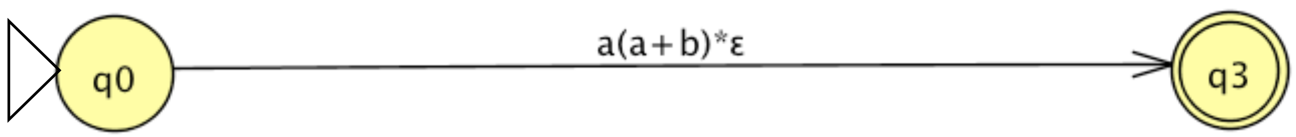


- For each pair of nodes *i, j* (*i* ≠ *k*, *j* ≠ *k*), label the transition from *i* to *j* with:

$$(i, j) + (i, k)(k, k)^{*}(k, j)$$

- Remove state *k* and all its transitions.





The Algorithm from Hein:

Finite Automaton to Regular Expression (11.5)

Assume that we have a DFA or an NFA. Perform the following steps:

1. Create a new start state s, and draw a new edge labeled with Λ from s to the original start state.
2. Create a new final state f, and draw new edges labeled with Λ from all the original final states to f.
3. For each pair of states i and j that have more than one edge from i to j, replace all the edges from i to j by a single edge labeled with the regular expression formed by the sum of the labels on each of the edges from i to j.
4. Construct a sequence of new machines by eliminating one state at a time until the only states remaining are s and f. As each state is eliminated, a new machine is constructed from the previous machine as follows:

Portland State UNIVERSITY

The Algorithm from Hein:

Finite Automaton to Regular Expression (11.5)

Assume that we have a DFA or an NFA. Perform the following steps:

1. Create a new start state s, and draw a new edge labeled with $\boxed{\varepsilon}$ from s to the original start state.
2. Create a new final state f, and draw new edges labeled with $\boxed{\varepsilon}$ from all the original final states to f.
3. For each pair of states i and j that have more than one edge from i to j, replace all the edges from i to j by a single edge labeled with the regular expression formed by the sum of the labels on each of the edges from i to j.
4. Construct a sequence of new machines by eliminating one state at a time until the only states remaining are s and f. As each state is eliminated, a new machine is constructed from the previous machine as follows:

Portland State UNIVERSITY

Eliminate State k

For convenience we'll let old(i, j) denote the label on edge (i, j) of the current machine. If there is no edge (i, j), then set old(i, j) = $\varnothing$. Now for each pair of edges (i, k) and (k, j), where $i \neq k$ and $j \neq k$, calculate a new edge label, new(i, j), as follows:

$$\text{new}(i, j) = \text{old}(i, j) + \text{old}(i, k)\ \text{old}(k, k)^*\ \text{old}(k, j).$$

For all other edges (i, j) where $i \neq k$ and $j \neq k$, set

$$\text{new}(i, j) = \text{old}(i, j).$$

The states of the new machine are those of the current machine with state k eliminated. The edges of the new machine are the edges (i, j) for which label new(i, j) has been calculated.

Now s and f are the two remaining states. If there is an edge (s, f), then the regular expression new(s, f) represents the language of the original automaton. If there is no edge (s, f), then the language of the original automaton is empty, which is signified by the regular expression $\varnothing$.