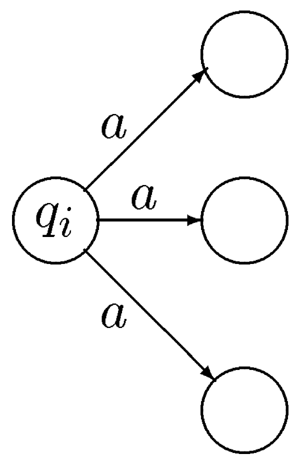
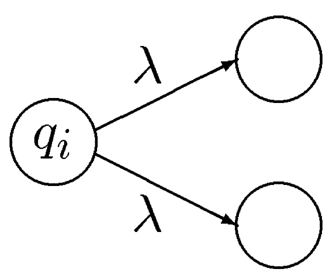
Nondeterministic Finite Automata

Allow transitions from state q_i on symbol a to many states (or none at all):



Also allow transitions on λ:



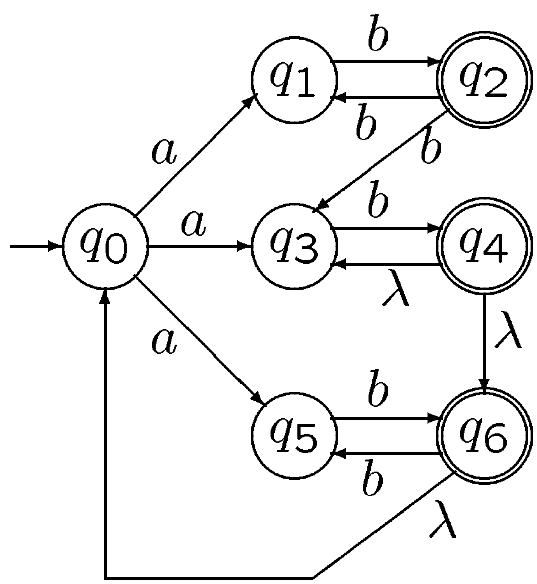
In this case, no input is consumed.

Formal Definition of NFA

A **nondeterministic finite automaton (NFA)** is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of **states;**
- Σ is the **input alphabet;**
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}(Q)$ is the **transition function;**
- $q_0 \in Q$ is the **start state;** and
- $F \subset Q$ is the set of **final** or **accepting states.**

Example NFA



Note that our NFA is an NFA-λ in the text.

Configurations; Yields Relation

A **configuration** of M is an element of $Q \times \Sigma^*$.

For an input $w \in \Sigma^*$, the **initial** or **start configuration** is

$$(q_0, w).$$

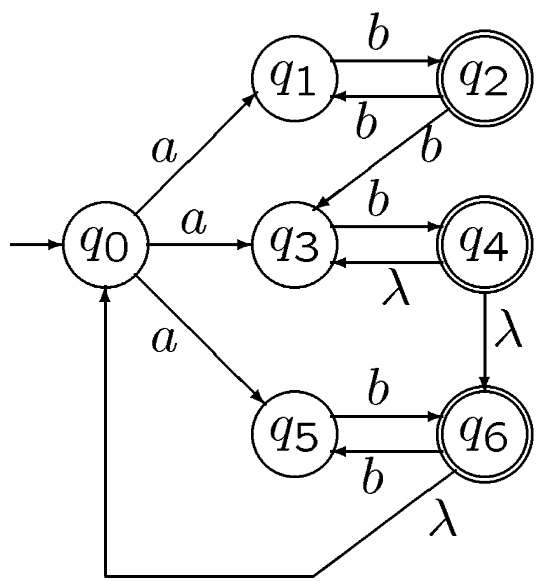
The **yields (in one step) relation** $\underset{M}{\vdash}$ is a binary relation on $Q \times \Sigma^*$. If $q_i, q_j \in Q$, $\sigma \in \Sigma \cup \{\lambda\}$ and $v \in \Sigma^*$, then

$$(q_i, \sigma v) \underset{M}{\vdash} (q_j, v)$$

if and only if

$$q_j \in \delta(q_i, \sigma).$$

Example Computations



$$
\begin{aligned}
(q_0, abbbba) &\vdash (q_1, bbbba) \vdash (q_2, bbba) \\
&\vdash (q_3, bba) \vdash (q_4, ba) \\
&\vdash (q_6, ba) \vdash (q_5, a)
\end{aligned}
$$

$$
\begin{aligned}
(q_0, abbbba) &\vdash (q_3, bbbba) \vdash (q_4, bbba) \\
&\vdash (q_3, bbba) \vdash (q_4, bba) \\
&\vdash (q_6, bba) \vdash (q_5, ba) \\
&\vdash (q_6, a) \vdash (q_0, a) \\
&\vdash (q_1, \lambda)
\end{aligned}
$$

Acceptance

A string $w \in \Sigma^*$ is **accepted** by the NFA M if

$$(q_0, w) \overset{*}{\underset{M}{\vdash}} (q_i, \lambda),$$

for some $q_i \in F$.

The **language** $L(M)$ **accepted** by the NFA M is the set of all strings accepted by M.

Said another way,

$$L(M) = \Big\{ w \in \Sigma^* \mid (q_0, w) \overset{*}{\underset{M}{\vdash}} (q_i, \lambda) \text{ for some } q_i \in F \Big\}.$$

Example

Let

$$M_1 \;=\; (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \delta_1, q_0, \{q_2, q_3\}),$$

where δ_1 is given by this table:

q	$\delta_1(q, \lambda)$	$\delta_1(q, a)$	$\delta_1(q, b)$
q_0	$\emptyset$	$\{q_0, q_1\}$	$\{q_3\}$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_0\}$	$\emptyset$	$\{q_2\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\emptyset$	$\{q_3, q_4\}$

- Draw the state diagram for M_1.

- What is $L(M_1)$?

Exercise

- Give an NFA M_2 to accept the language

$$L_2 = \{w \in \{0,1\}^* \mid w \text{ contains the substring } 100101001\}.$$

- Give an NFA $M_{3,5}$ to accept the language

$$L_{3,5} = \{x \in \{a,b\}^* \mid n_b(x) \equiv 0 \bmod 3 \text{ or } n_b(x) \equiv 0 \bmod 5\}.$$

Lambda Closure

The **lambda closure**

$$\lambda\text{-}\mathit{closure} : Q \to \mathcal{P}(Q)$$

is defined recursively as follows.

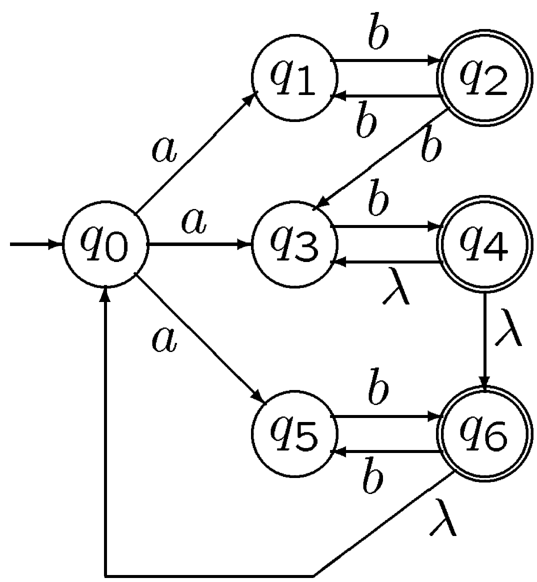
- **Basis:** If $q_i \in Q$, then

$$q_i \in \lambda\text{-}\mathit{closure}(q_i).$$

- **Recursive Step:** If $q_j \in \lambda\text{-}\mathit{closure}(q_i)$ and $q_k \in \delta(q_j, \lambda)$, then

$$q_k \in \lambda\text{-}\mathit{closure}(q_i).$$

Example



What is

$$\lambda\text{-}closure(q_0) = \boxed{\quad ?}$$

What is

$$\lambda\text{-}closure(q_6) = \boxed{\quad ?}$$

What is

$$\lambda\text{-}closure(q_4) = \boxed{\quad ?}$$

Extended Transition Function

The **extended transition function**

$$\widehat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$$

is defined recursively as follows.

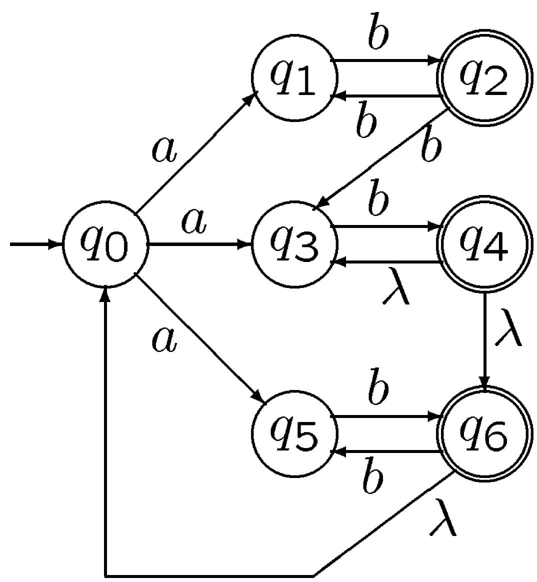
- **Basis:** If $|w| = 0$, then

$$\widehat{\delta}(q_i, w) \;=\; \lambda\text{-}\mathit{closure}(q_i).$$

- **Recursive Step:** If $|w| > 0$, then $w = u\sigma$, where $u \in \Sigma^*$ and $\sigma \in \Sigma$. Define

$$\widehat{\delta}(q_i, w) \;=\; \bigcup_{q_j \in \widehat{\delta}(q_i,u)} \; \bigcup_{q_k \in \delta(q_j,\sigma)} \lambda\text{-}\mathit{closure}(q_k).$$

Example



What is

$$\hat{\delta}(q_0, ba) = \boxed{\quad ?}$$

What is

$$\hat{\delta}(q_0, aba) = \boxed{\quad ?}$$

What is

$$\hat{\delta}(q_0, abbb) = \boxed{\quad ?}$$

Thinking About Nondeterminism

- If there is an accepting path, then M always makes the right "guess."
- Whenever there are multiple choices for the next transition, M splits into parallel machines, each taking one choice.
- At each step, M makes all possible choices. The "state" of M is then a subset of Q.
- Represent the initial configuration and all possible computations in a **computation tree**. When is a computation tree infinite?

On the other hand, **nondeterminism** is not **randomness.**

Normal Form for NFAs

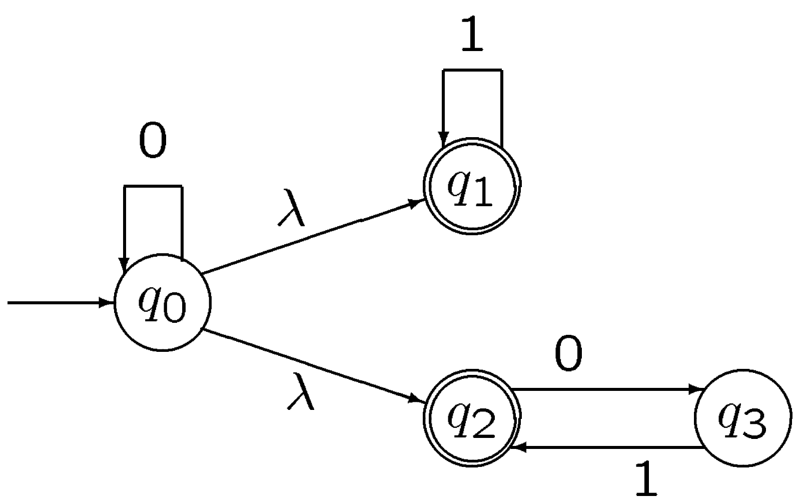
An NFA is in **normal form** if

- The start state has in-degree 0; and
- There is a single final state with out-degree 0.

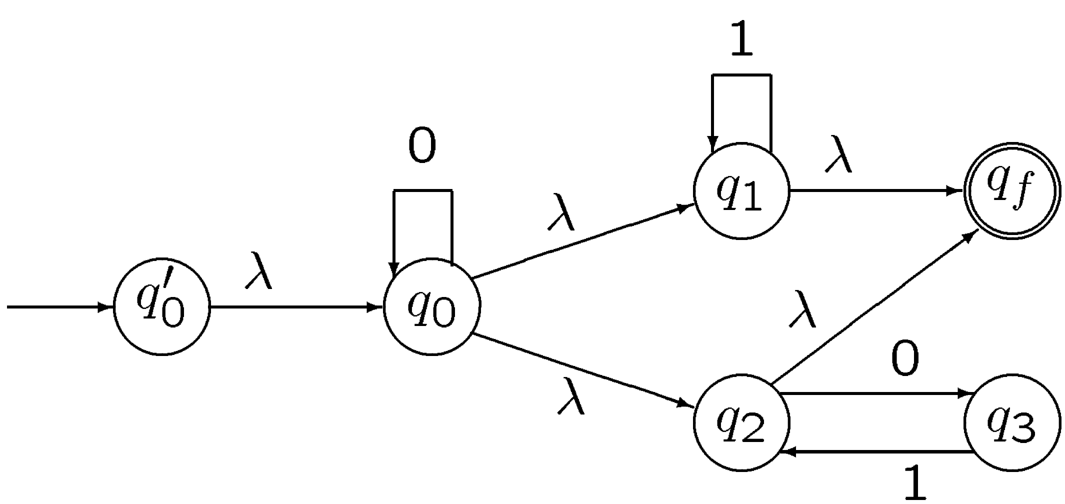
Every NFA has an equivalent NFA (one accepting the same language) that is in normal form.

Normal Form Example

This NFA is not in normal form:



This is an equivalent NFA in normal form:



Normal Form Construction

Start with an NFA $M = (Q, \Sigma, \delta, q_0, F)$.

Extend M to an equivalent NFA
$M' = (Q \cup \{q'_0, q_f\}, \Sigma, \delta', q'_0, \{q_f\})$ where

- q'_0 is the new start state;
- The only nonempty transition from q'_0 is $\delta'(q'_0, \lambda) = \{q_0\}$;
- q_f is a new state that is the only final state;
- There are no nonempty transitions from q_f; and
- The only transitions into q_f are $q_f \in \delta(q_i, \lambda)$ for every $q_i \in F$.

Closure Properties for NFA Languages

Theorem 6.5.3. Let M_1 and M_2 be NFAs. There exist NFAs that accept $L(M_1) \cup L(M_2)$, $L(M_1)L(M_2)$, and $L(M_1)^*$.

Proof: Without loss of generality, we may assume that M_1 and M_2 are in normal form. Now we can construction new NFAs in normal form that accept each of

- $L(M_1) \cup L(M_2)$;
- $L(M_1)L(M_2)$; and
- $L(M_1)^*$.

Regular Languages

Corollary. Any regular set is accepted by some NFA.

Proof:

Let r be a regular expression.

Recall the recursive definition for regular expressions.

If r is gotten from the base case, then we can choose an NFA to accept the language represented by r.

Otherwise, we can construct an NFA to accept the language represented by r by induction. This is because the operations in the recursive step of the definition are exactly the operations in Theorem 6.5.3.

Examples

Construct NFAs to accept the languages represented by these regular expressions:

- λ
- b
- $(\lambda \cup b)$
- abb
- $(abb)^*$

DFAs are NFAs

Any DFA can be viewed as a special NFA with these properties:

- There are no λ transitions; and
- Every transition on a symbol is to a set of cardinality 1.

Removing Nondeterminism

Theorem: Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Then there is a DFA M' that accepts $L(M)$.

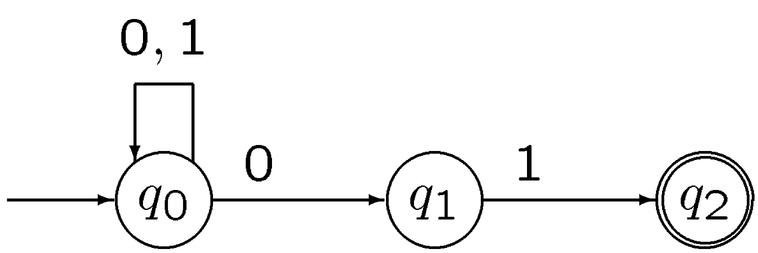
Proof: Define $M' = (\mathcal{P}(Q), \Sigma, \delta', q_0', F')$, where

- $q_0' = \lambda\text{-}\mathit{closure}(q_0)$;
- $F' = \{A \subset Q \mid A \cap F \neq \emptyset\}$; and
- For $A \subset Q$ and $\sigma \in \Sigma$,

$$\delta'(A, \sigma) \;=\; \bigcup_{q_i \in A} \widehat{\delta}(q_i, \sigma).$$

Example

Start with this NFA:



Use the construction in the proof of the theorem to find an equivalent DFA.

The construction will result in some unreachable states, which may be deleted.

Equivalence of Models

One model $\mathcal{M}_1$ of specifying languages is **as powerful as** another model $\mathcal{M}_2$ if every language in the class specified by $\mathcal{M}_2$ can also be specified by $\mathcal{M}_1$.

Two models of specifying languages are **equivalent** if they specify the same class of languages.

Equivalence of Models

Have shown:

- DFAs and NFAs are equivalent.
- NFAs are as powerful as regular expressions.

Want to show equivalent:

- DFAs
- NFAs
- Regular expressions
- Regular grammars

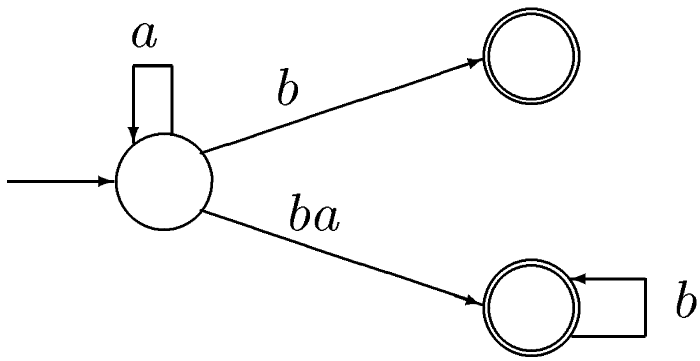
Expression Graphs

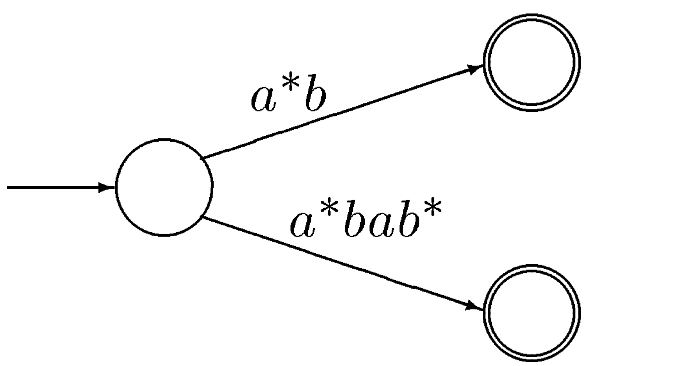
An **expression graph** is a directed graph satisfying these properties:

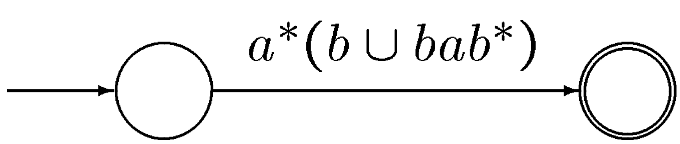
- The nodes are called **states;**
- One node is the **start state;**
- Some of the nodes (possibly none) are **final states;** and
- Every arc is labeled with a **regular expression.**

An expression graph **accepts** a string w if there is a path in the graph labeled by a regular expression that represents w.

Examples







Observations

- We may assume that an expression graph has exactly one arc, labeled $w_{i,j}$, between every pair of nodes (i, j).
- We may assume that an expression graph is in a **normal form** that has exactly one final state, not the start state, and exactly one arc between every pair of nodes.
- Suppose G is an expression graph in normal form having only two states. Then we can construct a regular expression for the language accepted by G.
- The state diagram of an NFA is an expression graph.

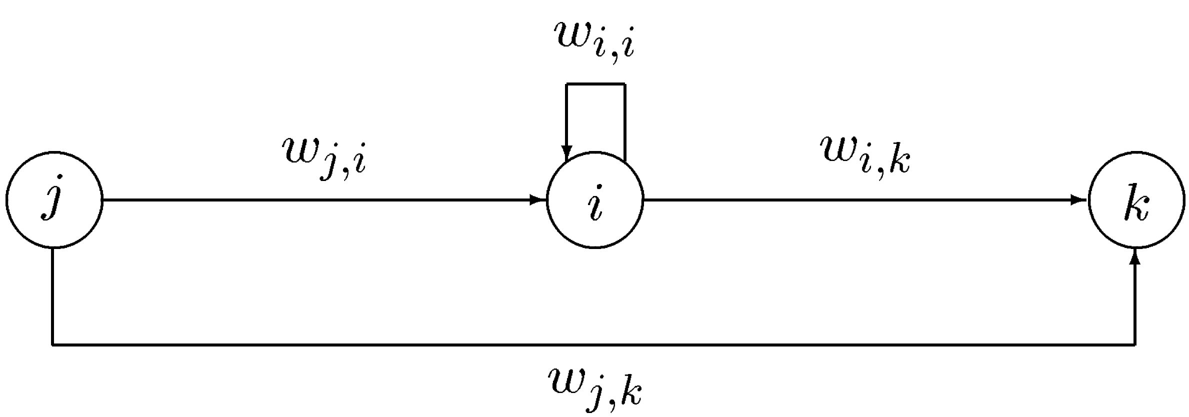
Theorem. The language recognized by an expression graph can be represented by a regular expression.

Proof: By algorithm.

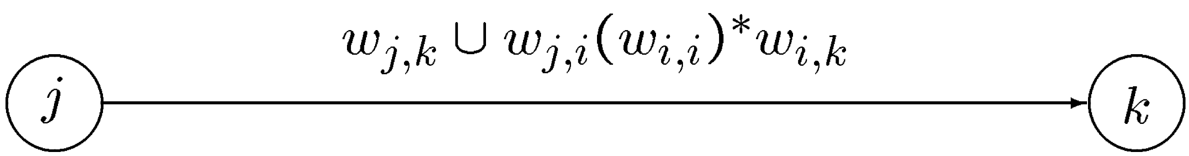
- Start with an expression graph G in normal form.
- Iterate: Eliminate one intermediate (non-start, non-final) state i, resulting in an expression graph in normal form having one fewer state.
- When no intermediate states remain, read the regular expression from the resulting graph.

Eliminating Intermediate State i

Consider every pair of states (j, k) satisfying $i \neq j$ and $i \neq k$. Locally, the picture is this:

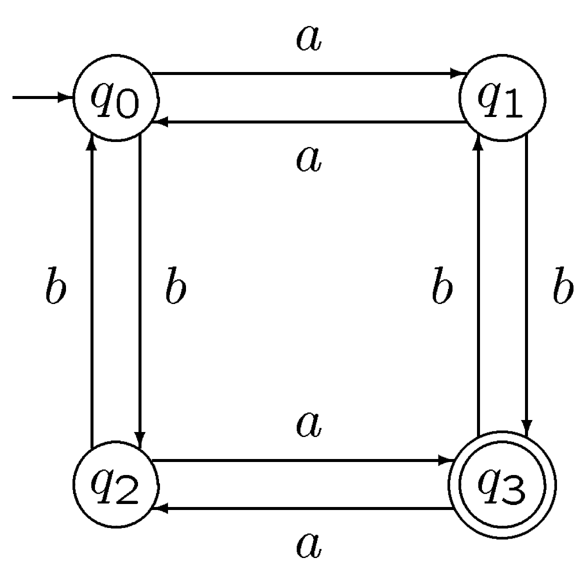


Eliminate state i and replace the label $w_{j,k}$ to obtain this picture:



Example

First convert this NFA (actually DFA) to an expression graph in normal form.



Then apply the algorithm in the proof to the resulting expression graph.

Equivalence to This Point

Corollary. NFAs and regular expressions are equivalent models.

Proof: We already knew that any regular expression has an equivalent NFA. The theorem shows that every NFA has an equivalent regular expression.

Corollary. NFAs, DFAs, and regular expressions are all equivalent.

Regular Grammars

Recall that a **regular grammar** is a CFL where every rule has one of these forms:

1. $A \longrightarrow a$
2. $A \longrightarrow aB$
3. $A \longrightarrow \lambda$

EXAMPLE. Here is an example of a regular grammar:

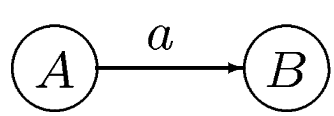
$$\begin{array}{rlcl} G_3: & S & \longrightarrow & aA \mid \lambda \\ & A & \longrightarrow & bB \mid cC \\ & B & \longrightarrow & bB \mid bS \\ & C & \longrightarrow & cC \mid cS \end{array}$$

Theorem: Let $G = (V, \Sigma, P, S)$ be a regular grammar. Then there exists an NFA that accepts $L(G)$.

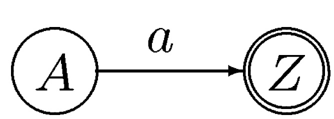
Proof: Construct an NFA M as follows:

- State set is $V \cup \{Z\}$.
- Start state is S.
- Final state is Z.
- Convert rules as given here:

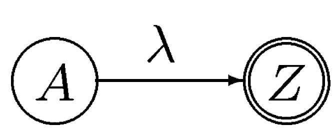
$A \longrightarrow aB$



$A \longrightarrow a$



$A \longrightarrow \lambda$



Argue that $L(M) = L(G)$.

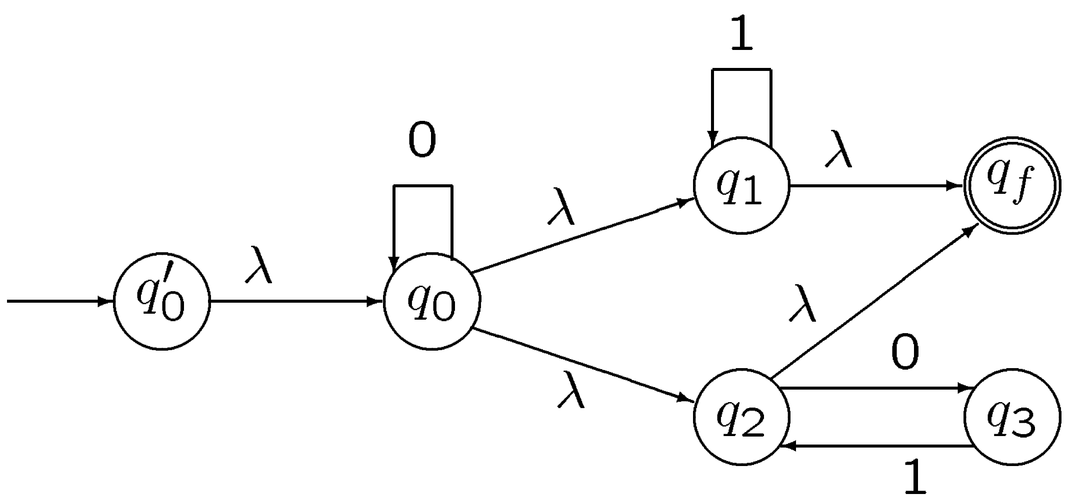
Example

Convert G_3 to an equivalent NFA:

$$
\begin{aligned}
G_3: \quad S &\longrightarrow aA \mid \lambda \\
A &\longrightarrow bB \mid cC \\
B &\longrightarrow bB \mid bS \\
C &\longrightarrow cC \mid cS
\end{aligned}
$$

The Other Direction

EXERCISE. Take this NFA



and construct an equivalent regular grammar.

Theorem: Let M be an NFA. Then there exists a regular grammar that generates $L(M)$.

Equivalence Summary

Corollary. NFAs and regular grammars are equivalent models.

Corollary. NFAs, DFAs, regular expressions, and regular grammars are all equivalent.

All of these equivalent models yield the important class of languages called the REGULAR LANGUAGES.