REGULAR LANGUAGES AND REGULAR GRAMMARS

Chap. 3

Summary

- Two alternative methods for describing regular languages:

  regular expressions and *regular grammars*.
- Regular expressions are convenient in some applications because of their simple string form, but they are restricted and have *no* obvious *extension* to the more complicated languages.
- Regular grammars, on the other hand, are just a *special case* of many different types of grammars.
- Let's explore the essential equivalence of these three modes of describing regular languages:
  - Reg. Expression vs. Reg. Grammar vs. FA
  - the conversion from one form to another.

Learning Objectives

- Identify the language associated with a *regular expression*.
- Find a *regular expression* to describe a given language.
- Construct a NFA to accept the language denoted by a *regular expression*.
- Use generalized transition graphs to construct a *regular expression* that denotes the language accepted by a given finite automaton.
- Identify whether a particular *grammar* is *regular*.
- Construct *regular grammars* for simple languages.
- Construct a NFA that accepts the language generated by a *regular grammar*.
- Construct a *regular grammar* that generates the language accepted by a finite automaton.

Regular Expressions

- A concise way to describe some languages.
- Definition 3.1: *Recursive* Definition of *Regular Expressions.*

  For any alphabet Σ,
  - $\varnothing$ (an empty set), λ (an empty string), and $a \in \Sigma$ are *primitive regular expressions.*
  - If r_1, r_2 are regular expressions,
    - r_1+r_2 (union),
    - $r_1 \cdot r_2$ (concatenation), and
    - $r_1{}^*$ (star closure) and
    - (r_1) of regular expressions are also a regular expression.
  - A string that is derived from primitive regular expressions by *a finite number of these operations* is also a regular expression.
- Example 3.1: $(a+ b \cdot c)^* \cdot (c+\varnothing)$

Languages Associated with Regular Expressions

- A language $L(r)$ denoted by any regular expression r.
- Assume that r_1 and r_2 are regular expressions:
  1. $\varnothing$ is the regular expression denoting the empty set.
  2. λ is the regular expression denoting $\{\lambda\}$.
  3. For any $a \in \Sigma$, a is the regular expression denoting $\{a\}$.
  4. The regular expression r_1+r_2 denotes $L(r_1) \cup L(r_2)$.
  5. The regular expression $r_1 \cdot r_2$ denotes $L(r_1)\, L(r_2)$.
  6. The regular expression (r_1) denotes $L(r_1)$.
  7. The regular expression r_1^* denotes $(L(r_1))^*$.
- Example 3.2: $L(a^* \cdot (a+b)) = L(a^*) \cdot L(a+b) = (L(a))^* \cdot (L(a) \cup L(b))$

  $= \{\lambda, a, aa, aaa, \ldots\} \cdot \{a, b\} = \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}$
- Example 3.3: $\Sigma=\{a,b\}$, a regular expression $r=(a+b)^*(a+bb)$ denotes

  a language $L(r)= \{a, bb, aa, abb, ba, bbb, \ldots\}$.

Determining the Language denoted by a Regular Expression

- By combining regular expressions using the given rules, arbitrarily complex (regular) expressions can be constructed.
- The concatenation symbol $(\cdot)$ is usually omitted.
- Precedence Rules in applying Operations:
  - Star Closure precedes Concatenation.
  - Concatenation precedes Union.
  - Star Closure > Concatenation > Union
- Parentheses are used to override the normal precedence of operators.

Sample Regular Expressions and Associated Languages

Regular Expression	Language
$(ab)^*$	$\{ (ab)^n, n \geq 0 \}$
$a + b$	$\{ a, b \}$
$(a + b)^*$	$\{ a, b \}^*$ (i.e. any string formed with a and b)
$a(bb)^*$	$\{ a, abb, abbbb, abbbbbb, \ldots \}$
$a^*(a + b)$	$\{ a, aa, aaa, \ldots, b, ab, aab, \ldots \}$ (Ex. 3.2)
$(aa)^*(bb)^*b$	$\{ b, aab, aaaab, \ldots, bbb, aabbb, \ldots \}$ (Ex. 3.4)
$(0 + 1)^*00(0 + 1)^*$	$\{w \in \Sigma^* \mid$ w has at least one pair of consecutive zeros$\}$ (Ex. 3.5)

Two regular expressions are *equivalent*

if they denote the same language.

For example, $(a + b)^* \equiv (a^*b^*)^*$.

Regular Expressions and Regular Languages

- Theorem 3.1: For any regular expression r,
  there exists a NFA that accepts the language denoted by r.

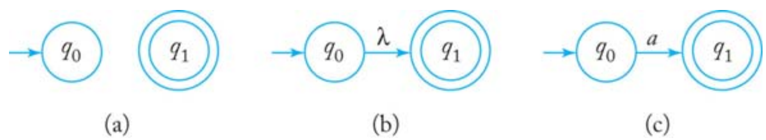
  i.e. $\forall$REX r, $\exists$ NFA N, L(N) = L(r).
- Since Nondeterministic and Deterministic FAs are equivalent, regular expressions are associated precisely with regular languages.
- Proof) A constructive proof for constructing a NFA that accepts the language denoted by any regular expression, by a systematic procedure.

Construction of a NFA to accept a language L(*r*)

Begin with the construction of the simple automata that accept the languages associated with

- the empty set (∅): L(∅) = ∅
- the empty string (λ): L(λ) = {λ}, and
- any individual symbol (*a*): L(*a*) = {*a*}

(i.e. the primitive language).

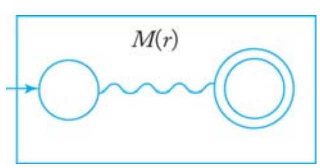


(a) NFA accepts ∅. (b) NFA accepts {λ}. (c) NFA accepts {*a*}.

Construction of a NFA to accept a language *L*(*r*) (cont.)

Given schematic representations for NFA M designed to accept $L(r_1)$ and $L(r_2)$,

a NFA to accept $L(r_1 + r_2)$ (= $L(r_1) \cup L(r_2)$) can be constructed as follows:



Schematic representation of an NFA accepting L(*r*).

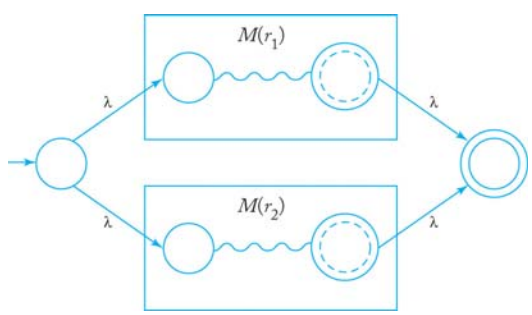


Figure 3.3. an NFA accepting for $L(r_1 + r_2)$

Construction of a NFA to accept a language L(*r*) (cont.)

Given schematic representations for automata designed to accept L(r_1) and (r_2),

a NFA to accept L($r_1 r_2$) (= $L(r_1) \cdot L(r_2)$) can be constructed as follows:

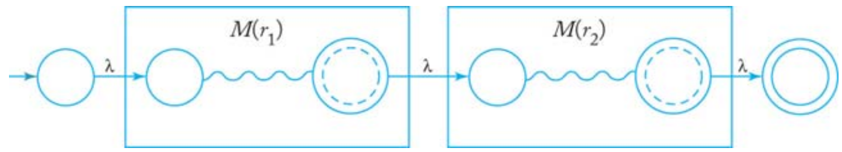


Figure 3.4: an NFA accepting L($r_1 r_2$).

Construction of a NFA to accept a language L(*r*) (cont.)

Given a schematic representation for an automaton designed to accept L(r_1),

a NFA to accept L(r_1*) (= $(L(r_1))^*$) can be constructed as follows:

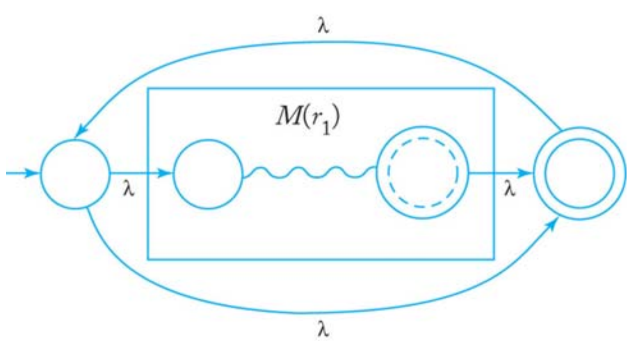
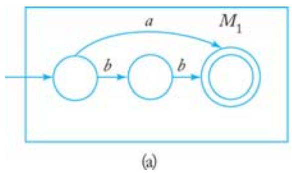


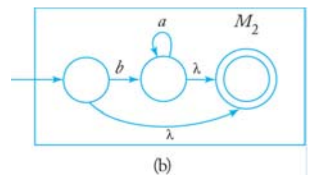
Figure 3.5: an NFA for L(r_1*).

Construction of a NFA to accept L(r) (cont.)

Given the regular expression $r=(a+bb)^*(ba^*+\lambda)$,
an NFA to accept L(r) can be constructed systematically as follows:



(a) M_1 accepts L(a + bb).



(b) M_2 accepts L(ba^* + λ).

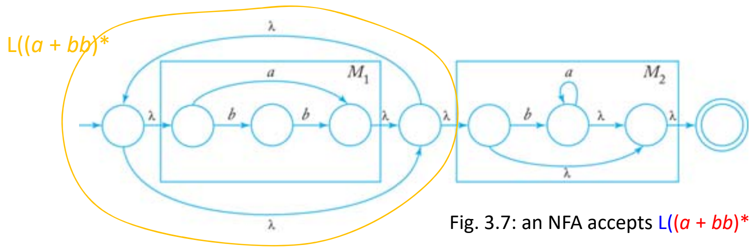
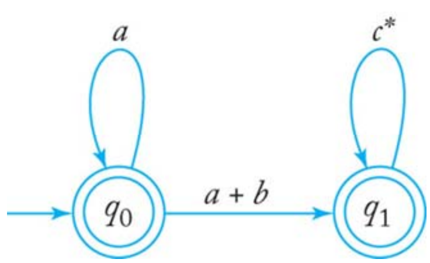


Fig. 3.7: an NFA accepts L((a + bb)* (ba^* + λ)).

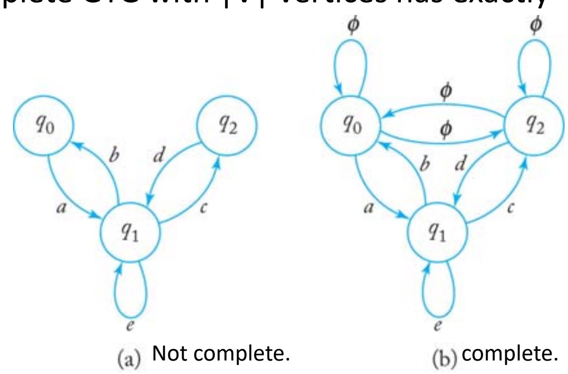
Regular Expressions (REX.) for Regular Languages (REG)

- Theorem 3.2: For every regular language, it is possible to construct a corresponding REX.
- The process can be illustrated with a *generalized transition graph (GTG).*
- A GTG for L(a^* + $a^*(a + b)c^*$) :



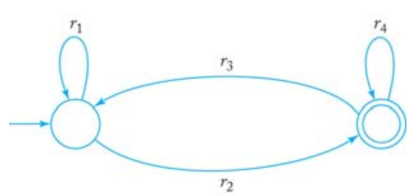
Regular Expressions for Regular Languages

- A *complete GTG* is a graph in which *all edges* are present. If some edges are missing in GTG after conversion from an NFA, we put them in and label them with $\varnothing$.
- Note that a complete GTG with $|V|$ vertices has exactly $|V|^2$ edges.
- Example 3.9:



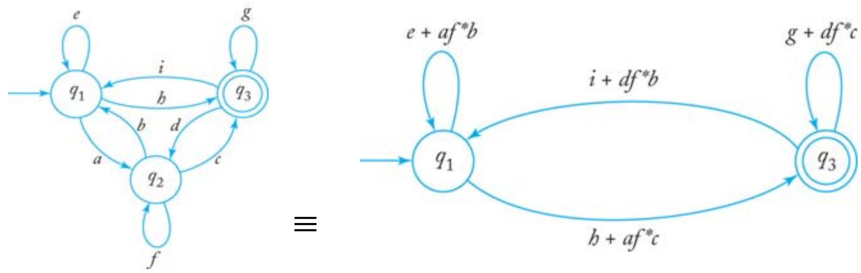
REX for RL (cont.)

- $r = r_1{}^*r_2(r_4 + r_3r_1{}^*r_2)^*$ $\Leftrightarrow$



REX for RL (cont.)

- When a GTG has more than 2 states, we can find an *equivalent graph* by removing one state at a time. How?
- Example 3.10:



- To remove q_2, we introduce some new direct edges.
  - Create an edge $q_1 \to q_1$ and label it $e + af^*b$
  - Create an edge $q_1 \to q_3$ and label it $h + af^*c$
  - Create an edge $q_3 \to q_1$ and label it $i + df^*b$
  - Create an edge $q_3 \to q_3$ and label it $g + df^*c$

REX for RL (cont.)

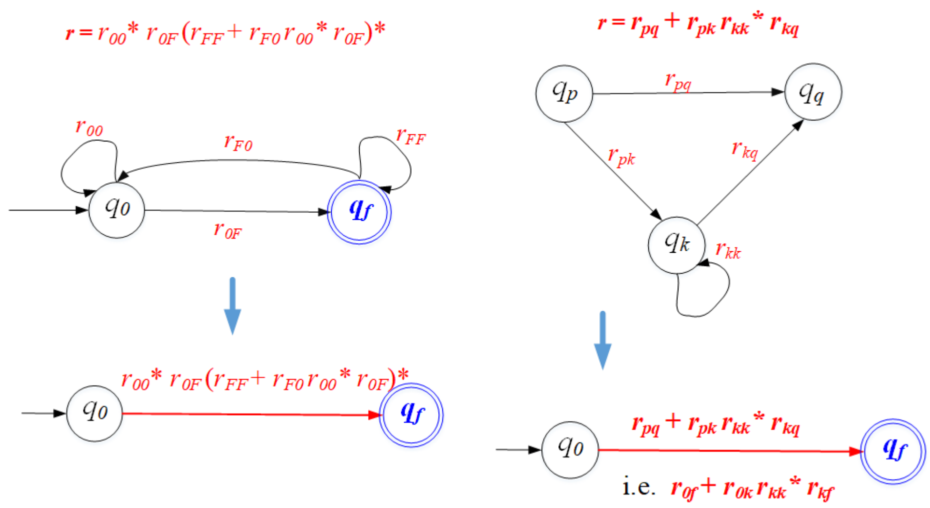
- Procedure: NFA-to-REX:

1. Start with an NFA with states $q_0, q_1, .., q_n$ and a single final state, $q_F (\neq q_0)$.
   Note: if $q_F = q_0$, create a new q_0' and give λ-transition from q_0' to q_0.
2. Convert the NFA into a complete GTG with the labels r_{ij} for $q_i \to q_j$.

Repeat

3. If the GTG has only 2 states q_0 and q_F, , its associated regular expression is $r = r_{00}^* r_{0F} (r_{FF} + r_{F0} r_{00}^* r_{0F})^*$
4. If the GTG has 3 states q_0, q_F, and the 3rd state q_k, introduce new direct edges, labeled $r_{pq} + r_{pk} r_{kk}^* r_{kq}$ for $p,q = 0,F$.
   When this is done, remove q_k and its associated edges.
5. If the GTG has 4 or more states, pick q_k to be removed.
   Apply rule 4 for all pairs of states (q_i, q_j), $i, j \neq k$.
   At each step, apply the simplifying rules.

Until the correct regular expression is obtained between q_0 and q_f.



REX for RL (cont.)

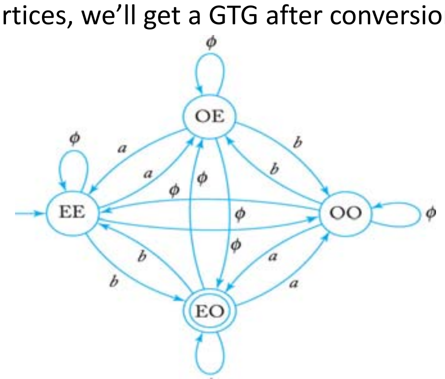
- Example 3.11: Find a regular expression for

$$L = \{w \in \{a,b\}^* \mid n_a(w) \text{ is even and } n_b(w) \text{ is odd}\}.$$

1. Let's label the vertices with

EE: an even # of a's and b's, OE: an odd # of of a's and an even # of b's, etc.

With those vertices, we'll get a GTG after conversion.



REX for RL (cont.)

- <u>Example 3.11</u>: Find a regular expression for
  $L = \{w \in \{a,b\}^* \mid n_a(w)$ is even and $n_b(w)$ is odd$\}$.

2. Convert the NFA in GTG to REX using proc. NFA-to-REX.
   2.1. By rule-5, pick the state OE to remove.
   Then, apply rule-4 for all pairs.

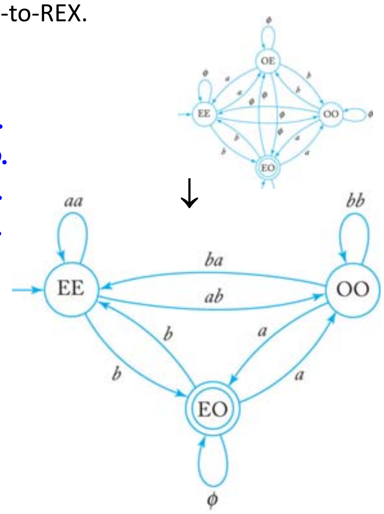
- (EE, EE) : a new edge $r_{EEEE} = \varnothing + a\varnothing^*a = aa.$
- (OO, OO): a new edge $r_{OOOO} = \varnothing + b\varnothing^*b = bb.$
- (EE, OO) : a new edge $r_{EEOO} = \varnothing + a\varnothing^*b = ab.$
- (OO, EE) : a new edge $r_{OOEE} = \varnothing + b\varnothing^*a = ba.$

Note: No changes in

- (EE, EO) : an edge $r_{EEEO} = b + a\varnothing^*\varnothing = b.$
- (EO, EE) : an edge $r_{EOEE} = b + \varnothing\varnothing^*a = b.$
- (OO, EO) : an edge $r_{OOEO} = a + b\varnothing^*\varnothing = a.$
- (EO, OO) : an edge $r_{EOOO} = a + \varnothing\varnothing^*b = a.$

where $\varnothing^* = \lambda$, $a\varnothing = \varnothing a = \varnothing$

$L(\varnothing^*) = \{\lambda\}$, $L(a\varnothing) = L(\varnothing a) = \varnothing$

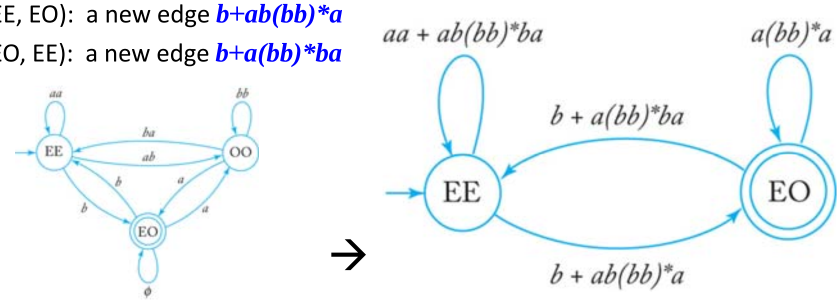


REX for RL (cont.)

- <u>Example 3.11</u>: Find a regular expression for
  $L = \{w \in \{a,b\}^* \mid n_a(w)$ is even and $n_b(w)$ is odd$\}$.

2. Convert the NFA in GTG to REX using proc. NFA-to-REX.
   2.2. By rule-5, pick the state OO to remove.
   Then, apply rule-4 for a pair (EE, EO)

- (EE, EE): a new edge $aa+ab(bb)^*ba$
- (EO, EO): a new edge $\varnothing + a(bb)^*a = a(bb)^*a$
- (EE, EO): a new edge $b+ab(bb)^*a$
- (EO, EE): a new edge $b+a(bb)^*ba$



REX for RL (cont.)

- Example 3.11: Find a regular expression for

  $L = \{w \in \{a,b\}^* \mid n_a(w)$ is even and $n_b(w)$ is odd$\}$

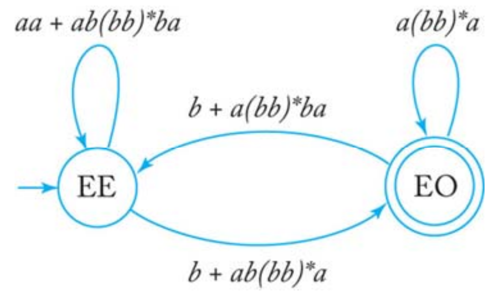
  So, the REX of L $= r_1{}^*r_2(r_4 + r_3r_1{}^*r_2)^*$ where

  $r_1 = aa + ab(bb)^*ba$

  $r_2 = b + ab(bb)^*a$

  $r_3 = b + a(bb)^*ba$

  $r_4 = a(bb)^*a$



Regular Grammars

- Definition 3.3: A grammar G = (V, T, S, P) is said to be *right-linear* if all productions are of the form

  $A \to xB$ or $A \to x$,

  where $A, B \in V$, and $x \in T^*$.

  A grammar is said to be *left-linear* if all productions are of the form

  $A \to Bx$ or $A \to x$.

- In a right-linear grammar, *at most one variable* symbol appears on the right side of any production. If it occurs, it is the *rightmost symbol.*
- In a left-linear grammar, *at most one variable* symbol appears on the right side of any production. If it occurs, it is the *leftmost symbol*.
- A *regular grammar* is one that is either right-linear or left-linear.

Regular Grammars

- A *regular grammar* is one that is *either* right-linear *or* left-linear.
- Example 3.13: A (right-linear) grammar G=(V, T, S, P):
  $V = \{ S \}, T = \{ a, b \}$, *and productions* $S \to abS \mid a$
  is *regular grammar.*
  Derivation of G: $S \Rightarrow abS \Rightarrow ababS \Rightarrow ababa$
- Example 3.14: A grammar G=(V, T, S, P) where
  V = {S, A, B}, T = {*a, b*} with productions
  $S \to A, \quad A \to aB \mid \lambda, \quad B \to Ab$
  is ***not** regular.* *Why?*
  G is an example of a *linear grammar.*
- A *linear grammar* is a grammar in which *at most one* variable can occur on the *right side* of any production, *regardless* the position of this variable.
- A regular grammar is always linear, but not all linear grammars are regular: regular grammar $\subseteq$ linear grammar.

Right-Linear Grammars generate Regular Languages

Theorem 3.3: Let G = (V, T, S, P) be a right-linear grammar.
Then, L(G) is a regular language.

Proof) Assume $V = \{V_0, V_1, ..\}$ where $S=V_0$ and the productions of the form
$V_0 \to v_1V_i$, $V_i \to v_2V_j$, .. *or* $V_n \to v_l$, …. where $T = \{v_1, v_2, \ldots v_l, \ldots v_m\}$

If $w \in$ L(G), $V_0 \Rightarrow v_1V_i \Rightarrow v_1v_2V_j \Rightarrow^* v_1v_2 .. v_kV_n \Rightarrow v_1v_2 .. v_kv_l = w$.

i.e. $L(G) = \{ w \mid S \Rightarrow^* w \}$.

To show that L(G) is a regular language,

we need to construct a FA that accepts the language of L(G),

i.e. $\exists$ FA, M, s.t. L(M) = L(G).

The FA, M, to be constructed will reproduce the derivation by consuming each of these *v*'s in turn.

The initial state of the FA will be labeled V_0, and

for each variable V_i there will be a non-final state labeled V_i.

Right-Linear Grammars generate Regular Languages

Theorem 3.3: Let G = (V, T, S, P) be a right-linear grammar. Then, L(G) is a regular language.

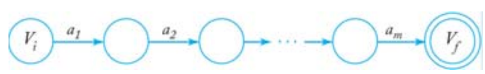
Proof: cont.) (1) Construction of the FA, M, from G s.t. L(M) = L(G).

The initial state of the FA will be labeled V_0, and

for each variable V_i of G, there will be a non-final state labeled V_i.

- For each production $V_i \rightarrow a_1a_2...a_mV_j$, a FA, M, will have transition***s*** from V_i to V_j, *i.e.* $\delta^*(V_i, a_1a_2...a_m) = V_j$.

- For each production $V_i \rightarrow a_1a_2...a_m$, a FA M's transition will be $\delta^*(V_i, a_1a_2...a_m) = V_F$ where V_F is a final state.



Right-Linear Grammars generate Regular Languages (cont.)

Proof (cont.) : Suppose $w \in$ L(G), so $V_0 \Rightarrow v_1V_i \Rightarrow v_1v_2V_j \Rightarrow^* v_1v_2 .. v_kV_n \Rightarrow v_1v_2 .. v_kv_l = w$.

(2) Prove $\forall w \in$ L(G) iff $\forall w \in$ L(M) $\Leftrightarrow$ L(G) = L(M)

Claim: *w* is generated by *G* if and only if *w* is accepted by *M* :

$\rightarrow$) Show that w is accepted by M that is constructed in accordance with G.

If $w \in$ L(G), $V_0 \Rightarrow v_1V_i \Rightarrow v_1v_2V_j \Rightarrow^* v_1v_2 .. v_kV_n \Rightarrow v_1v_2 .. v_kv_l = w$.

In M, there is a path from V_0 to V_i labeled v_1, a path from V_i to V_j labeled v_2, etc.

i.e. $\delta^*(V_0, v_1) = V_i$, $\delta^*(V_i, v_2) = V_j$, $\delta^*(V_j, v_3...v_k) = V_n$, $\delta^*(V_n, v_l) = V_F$

So, $V_F \in \delta^*(V_0, w)$ and *w* is accepted by M, i.e. $w \in$ L(*M*).

$\leftarrow$) Show that *w* is generated by G if *w* is accepted by M.

By the way M was constructed in accordance with G, M has to pass through a sequence of states V_0, V_i, ... to V_F, using paths labeled v_1, v_2, ... v_k, v_l.

Therefore, *w* must have the form $w = v_1v_2 \cdots v_kv_l$ and

the derivation $V_0 \Rightarrow v_1V_i \Rightarrow v_1v_2V_j \Rightarrow^* v_1v_2 .. v_kV_n \Rightarrow v_1v_2 .. v_kv_l$ (= *w*) is possible.

Thus, $w \in$ L(G). Q.E.D.

Right-Linear Grammars generate Regular Languages

i.e.

Theorem 3.3: it is always possible to construct a FA to accept the language generated by a regular grammar G:

- Label the FA start state with S and a final state V_F
- For every variable symbol $V_i \in$ G, create a FA state and label it V_i
- For each production of the form $A \to aB$, label a transition from state A to B with symbol a, $\delta^*(A, a) = B$.
- For each production of the form $A \to a$, label a transition from state A to V_F with symbol a (may have to add intermediate states for productions with more than one terminal on RHS), $\delta^*(A, a) = V_F$.

Example 3.15:

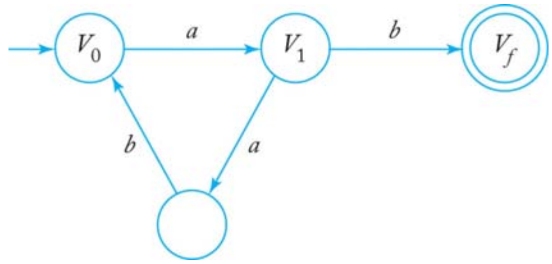
Construction of a FA to accept a language L(G)

Given the regular grammar G with productions

$V_0 \to aV_1$ and $V_1 \to abV_0 \mid b$,

where V_0 is the start variable,

Construct a FA that accepts L(G).



Right-Linear Grammars for Regular Languages

Theorem 3.4: A language, L, is regular iff there exists a (right-linear) regular grammar G s.t. $L = L(G)$.

(1) It is always possible to construct a (right-linear) regular grammar G that generates the language accepted by a DFA M:

- Each *state* in the DFA corresponds to a *variable symbol* in G.
- For each DFA *transition* from state A to state B labeled with symbol a, $A \xrightarrow{a} B$, there is a production of the form $A \rightarrow aB$ in G.
- For each final state F_i in the DFA, there is a corresponding production $F_i \rightarrow \lambda$ in G.

(2) Further show that L(M) = L(G).

Thm. 3.5. A language is regular iff there exists a (left-linear) regular grammar G s.t. $L = L(G)$.

Left-Linear Grammars for Regular Languages

Theorem. 3.5. A language is regular iff there exists a (left-linear) regular grammar G s.t. $L = L(G)$.

Proof) Given any left-linear grammar with productions of the form

$$A \rightarrow Bv \quad \text{or} \quad A \rightarrow v,$$

We construct from it a right-linear grammar G' by replacing every such production of G with

$$A \rightarrow v^R B \quad \text{or} \quad A \rightarrow v^R, \text{ respectively.}$$

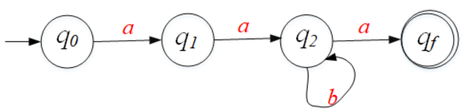
Then, $L(G) = (L(G'))^R$.

Since the reverse of any regular language is also regular, L(G') is regular which is a reverse of the regular language, L(G').

Thus, so are $L(G')^R$ and $L(G)$. Q.E.D.

Example 3.16: Construction of a regular grammar G to generate a language L(M)

Given the language $L(aab^*a)$, the table shows the transition function for a DFA that accepts the language and the productions for the corresponding regular grammar.



$\delta(q_0, a) = \{q_1\}$	$q_0 \rightarrow aq_1$
$\delta(q_1, a) = \{q_2\}$	$q_1 \rightarrow aq_2$
$\delta(q_2, b) = \{q_2\}$	$q_2 \rightarrow bq_2$
$\delta(q_2, a) = \{q_f\}$	$q_2 \rightarrow aq_f$
$q_f \in F$	$q_f \rightarrow \lambda$

Equivalence of Regular Languages and Regular Grammars

Theorem 3.6: A language *L* is regular iff there exists a regular grammar G s.t. $L = L(G)$.