BLG311E Formal Languages and Automata

Automata

A.Emre Harmancı   Tolga Ovatman   Ö.Sinan Saraç

2012

Outline

Definitions

Automaton

An automaton is an abstract model of a machine that perform computations on an input by moving through a series of states or configurations. At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration. As a result, once the computation reaches an accepting configuration, it accepts that input. The most general and powerful automata is the Turing machine.

Definitions

Deterministic Finite Automata

A DFA accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string. *Deterministic* refers to the uniqueness of the computation.

The major objective of automata theory is to develop methods by which computer scientists can describe and analyze the dynamic behavior of discrete systems, in which signals are sampled periodically.

Formal Definition of a DFA

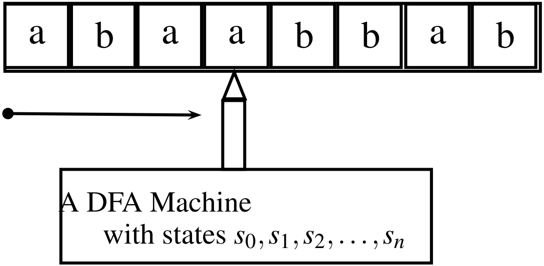
A deterministic finite state machine is a quintuple $M = (\Sigma, S, s_0, \delta, F)$, where:

- S: A finite, non-empty set of states where $s \in S$.
- Σ: Input alphabet (a finite, non-empty set of symbols)
- s_0: An initial state, an element of S.
- δ: The state-transition function $\delta : S \times \Sigma \rightarrow S$
- F: The set of final states where $F \subseteq S$.

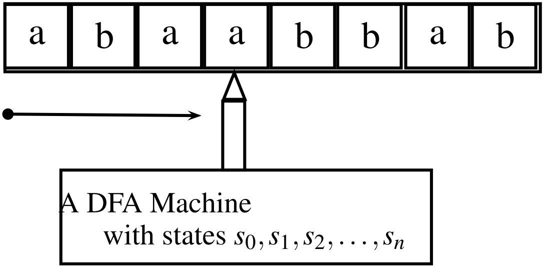
This machine is a Moore machine where each state produces the output in set $Z = \{0, 1\}$ corresponding to the machine's accepting/rejecting conditions.

DFA as a machine

Consider the physical machine below with an *input tape*. The tape is divided into cells, one next to the other. Each cell contains a symbol of a word from some finite alphabet. A machine that is modeled by a DFA, reads the contents of the cell successively and when the last symbol is read, the word is said to be accepted if the DFA is in an accepted state.



DFA as a machine



A run can be seen as a sequence of compositions of transition function with itself. Given an input symbol $\sigma \in \Sigma$ when this machine reads σ from the strip it can be written as $\delta(s,\sigma) = s' \in S$ or alternatively $\delta_\sigma(s) = s' \in S$.

$\forall \sigma \in \Sigma \;\; \exists \delta_\sigma : S \to S \land \delta = \{\delta_\sigma | \sigma \in \Sigma\}$

Configuration

A computation history is a (normally finite) sequence of configurations of a formal automaton. Each configuration fully describes the status of the machine at a particular point.
$s, \omega \in S \times \Sigma^*$

Configuration derivation is performed by a relation $\vdash_M$. If we denote the tuples in $\vdash_M$ as (s, ω) and (s', ω'), the relation can be defined as:

a $\omega = \sigma\omega' \wedge \sigma \in \Sigma$

b $\delta(s, \sigma) = s'$

A transition defined by this relation is called *derivation in one step* and denoted as $(s, \omega) \vdash_M (s', \omega')$. Following definitions can be defined based on this:

- Derivable configuration: $(s, \omega) \vdash_M^* (s', \omega')$ where $\vdash_M^*$ is the reflexive transitive closure of $\vdash_M$
- Recognized word: $(s_0, \omega) \vdash_M^* (s_i, \Lambda)$ where $s_i \in F$. Therefore we can deduce that $\vdash_M$ is a function from $S \times \Sigma^+$ to $S \times \Sigma^*$
- Execution: $(s_0, \omega_0) \vdash (s_1, \omega_1) \vdash (s_2, \omega) \vdash \ldots \vdash (s_n, \Lambda)$ where Λ is the empty string.
- Recognized Language: $L(M) = \{\omega \in \Sigma^* | (s_0, \omega) \vdash_M^* (s_i, \Lambda) \wedge s_i \in F\}$

Language Recognizer

The reflexive transitive closure of $\vdash_M$ is denoted as ${\vdash_M}^*$.
$(q,\omega){\vdash_M}^*(q',\omega')$ denotes that (q,ω) yields (q',ω') after some number of steps.
$(s,\omega){\vdash_M}^*(q,\Lambda)$ denotes that $\omega \in \Sigma^*$ is recognized by an automaton if $q \in F$. In other words $L(M) = \{\omega \in \Sigma^* | (s,\omega){\vdash_M}^*(q_i,\Lambda) \wedge q_i \in F\}$

Example 1

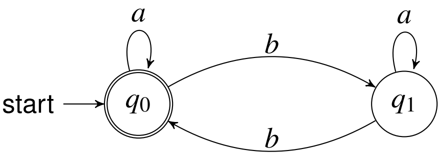
$S = \{q_0, q_1\}$

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

$s_0 = q_0$

$F = \{q_0\}$

q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0



$$(q_0, aabba) \vdash_M (q_0, abba)$$
$$(q_0, abba) \vdash_M (q_0, bba)$$
$$(q_0, bba) \vdash_M (q_1, ba)$$
$$(q_1, ba) \vdash_M (q_0, a)$$
$$(q_0, a) \vdash_M (q_0, \Lambda)$$

Example 1

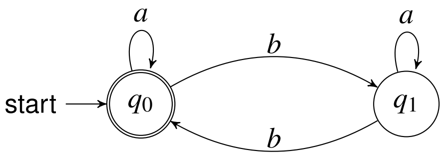
$S = \{q_0, q_1\}$

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

$s_0 = q_0$

$F = \{q_0\}$

q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0



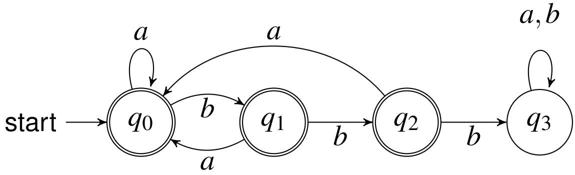
$L(M) = (a \vee ba^*b)^*$. We can write the grammar as:

$$V = S \cup \Sigma$$
$$I = \Sigma = \{a, b\}$$
$$s_0 = q_0 = n_0$$
$$< q_0 >::= \Lambda | a < q_0 > | b < q_1 > | a$$
$$< q_1 >::= b < q_0 > | a < q_1 > | b$$

Example 2

$L(M) = \{\omega | \omega \in \{a,b\}^* \wedge \omega$ should not include three successive b's$\}$
$S = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a,b\}, s_0 = q_0, F = \{q_0, q_1, q_2\}$

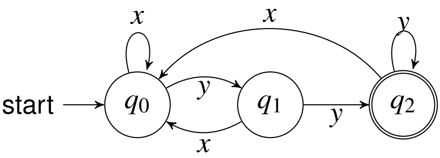
q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_2
q_2	a	q_0
q_2	b	q_3
q_3	a	q_3
q_3	b	q_3



We have a dead state q_3 where the automaton is not able to change state once it visits the dead state. $L(M) = [(\Lambda \vee b \vee bb)a]^*(\Lambda \vee b \vee bb)$

Example 3

$$S = \{q_0, q_1, q_2\}, \Sigma = \{x, y\}, s_0 = q_0 = n_0, F = \{q_2\}$$



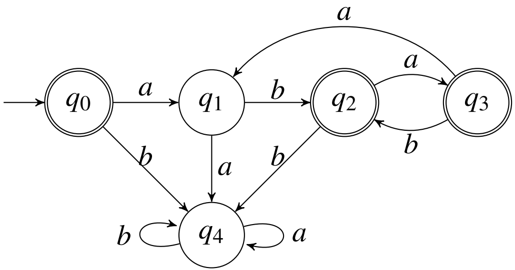
$$< q_0 > ::= x < q_0 > \mid y < q_1 >$$
$$< q_1 > ::= y < q_2 > \mid \; y \; \mid x < q_0 >$$
$$< q_2 > ::= y \; \mid y < q_2 > \mid x < q_0 >$$

$$L(M) = ((x \vee yx)^* yy^+ x)^* (x \vee yx)^* yy^+$$
$$L(M) = ((\Lambda \vee y \vee yy^+)x)^* yy^+ = (y^* x)^* yy^+$$
$$L(M) = (x \vee yx \vee yy^+ x)^* yyy^*$$

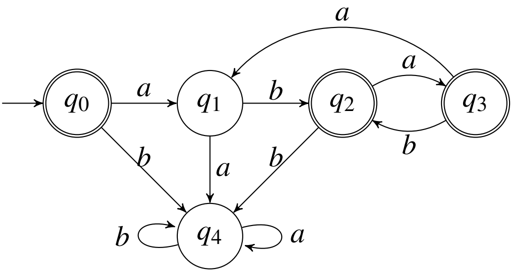
Non-deterministic Finite Automata(NFA)

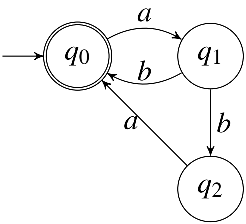
In an NFA, given an input symbol it is possible to jump into several possible next states from each state.

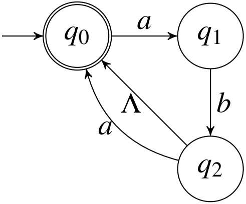
A DFA recognizing $L = (ab \vee aba)^*$ can be diagramatically shown as:

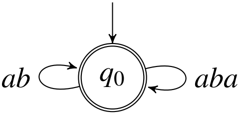


We may construct three different NFAs recognizing the same language.









Formal Definition of an NFA

A non-deterministic finite state automata is a quintuple $M = (\Sigma, S, s_0, \Delta, F)$, where:

- S: A finite, non-empty set of states where $s \in S$.
- Σ: Input alphabet (a finite, non-empty set of symbols)
- s_0: An initial state, an element of S.
- Δ: The state-transition relation $\Delta \subseteq S \times \Sigma^* \times S((q,u,b) \in \Delta \wedge u \in \Sigma^*)$
- F: The set of final states where $F \subseteq S$.

A configuration is defined as a tuple in set $S \times \Sigma^*$. Considering the definition of derivation in one step:

$(q,\omega) \vdash_M (q',\omega') \Rightarrow \exists u \in \Sigma^* (\omega = u\omega' \wedge (q,u,q') \in \Delta)$

For deterministic automata $\Delta \subseteq S \times \Sigma^* \times S$ relation becomes a function $S \times \Sigma \to S$. For (q,u,q') triples $|u| = 1 \wedge (\forall q \in S \wedge \forall u \in \Sigma) \exists! q' \in S$

The language that an NFA recognizes is

$L(M) = \{\omega | (s,\omega) \vdash_m{}^* (q,\Lambda) \wedge q \in F\}$

An example NFA

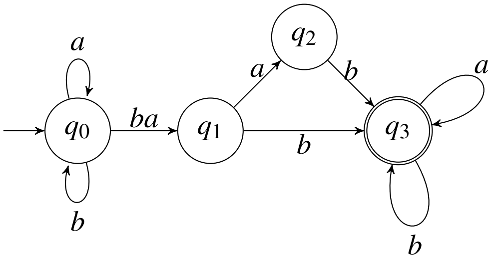
Build an NFA that recognizes languages including bab or baab as substrings.

$S = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$s_0 = q_0$
$F = \{q_3\}$
$\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, ba, q_1), (q_1, b, q_3), (q_1, a, q_2),$
$(q_2, b, q_3), (q_3, a, q_3), (q_3, b, q_3)\}$

$M = (S, \Sigma, \Delta, s_0, F)$
$< q_0 >::= a < q_0 > |b < q_0 > |ba < q_1 >$
$< q_1 >::= b < q_3 > | \ b \ |a < q_2 >$
$< q_2 >::= b < q_3 > |b$
$< q_3 >::= a \ | \ b \ | \ a < q_3 > \ | \ b < q_3 >$

An example NFA

$$< q_0 > ::= a < q_0 > | b < q_0 > | ba < q_1 >$$
$$< q_1 > ::= b < q_3 > | \ b \ | a < q_2 >$$
$$< q_2 > ::= b < q_3 > | b$$
$$< q_3 > ::= a \ | b \ | a < q_3 > | b < q_3 >$$



A possible derivation may follow the path:

$(q_0, aaabbbaabab) \mapsto$
$(q_0, aabbbaabab) \mapsto$
$(q_0, abbbaabab) \mapsto$
$(q_0, bbbaabab) \mapsto$
$(q_0, bbaabab) \mapsto$
$(q_0, baabab) \mapsto$
$(q_1, abab) \mapsto (q_2, bab) \mapsto$
$(q_3, ab) \mapsto (q_3, b) \mapsto$
(q_3, Λ)

Lemma

$M = (S, \Sigma, \Delta, s_0, F) \wedge q, r \in S \wedge x, y \in \Sigma^*$
$\exists p \in S \wedge (q, x) \vdash_M{}^* (p, \Lambda) \wedge (p, y) \vdash_M{}^* (r, \Lambda) \Rightarrow (q, xy) \vdash_M{}^* (r, \Lambda)$

Definition

Regular Grammar: All the production rules are of type-3.
Regular Language: Languages that can be recognized by regular grammars.
Regular Expression: $\varnothing, \{\Lambda\}, \{a|a \in \Sigma\}, A \vee B, A.B, A^*$
Regular set : The sets which can be represented by regular expressions are called regular sets.

Regular grammars can be represented by NFAs.

Definition

Regular Grammar: All the production rules are of type-3.
Regular Language: Languages that can be recognized by regular grammars.
Regular Expression: $\varnothing, \{\Lambda\}, \{a|a \in \Sigma\}, A \vee B, A.B, A^*$
Regular set : The sets which can be represented by regular expressions are called regular sets.

Regular grammars can be represented by NFAs.

a) Non-terminal symbols are assigned to states
b) Inititial state corresponds to initial symbol
c) Accepting states sorresponds to the rules that end with terminal symbols
d) If Λ should be recognized, initial state is an accepting state.

Languages recognized by finite automata(Regular Languages) are closed under union, concatanation and Kleene star operations.

Kleene Theorem

Every regular language can be recognized by a finite automaton and every finite automaton defines a regular language.

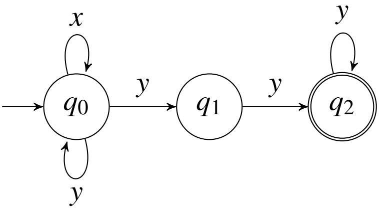
$M = (S, \Sigma, \Delta, s_0, F) \Leftrightarrow G = (N, \Sigma, n_0, \mapsto), L = L(G)$ a grammar of type-3.

$S = N \wedge F \subseteq N$

$s_0 = n_0$

$\Delta = \{(A, \omega, B) : (A \mapsto \omega B) \in \mapsto \wedge (A, B \in N) \wedge \omega \in \Sigma^*\} \cup \{(A, \omega, f_i) : (A \mapsto \omega) \in \mapsto \wedge A \in N \wedge f_i \in F \wedge \omega \in \Sigma^*\}$

Example



$$< q_0 > ::= x < q_0 > \mid y < q_0 > \mid y < q_1 >$$
$$< q_1 > ::= y \mid y < q_2 >$$
$$< q_2 > ::= y < q_2 > \mid y$$

Kleene Theorem

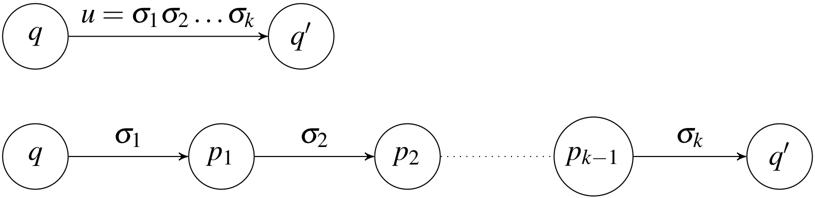
For every NFA an equivalent DFA can be constructed.

For the NFA $M = (S, \Sigma, \Delta, s_0, F)$ our aim is to...

(a) In $(q, u, q') \in \Delta$ there shouldn't be any $u = \Lambda$ and $|u| > 1$
(b) An input should be present for all symbols in all states
(c) There shouldn't be more than one transitions for each configuration.

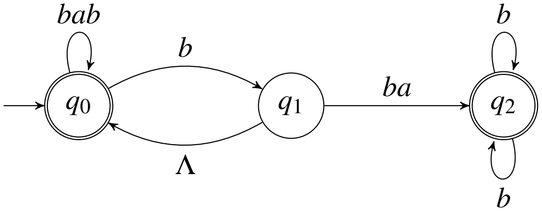
NFA/DFA equivalency-Phase 1

Interim steps are populated to eliminate the $|u| > 1$ in (q, u, q') of Δ.

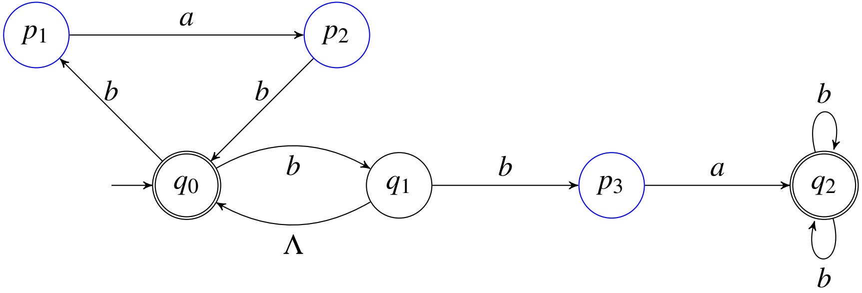


This expansion transforms Δ into Δ' by replacing triples of (q, u, q') with triples like $(q, \sigma_1, p_1), (p_1, \sigma_2, p_2), \ldots, (p_{k-1}, \sigma_k, q')$. A new machine is formed $M' = (S', \Sigma, \Delta', s_0', F')$ where $F' \equiv F$ and $s_0' \equiv s_0$

NFA/DFA equivalency-Phase 1



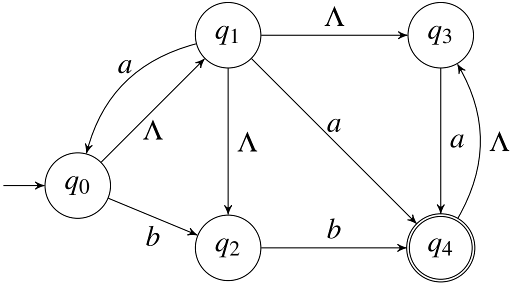
is equivalent to



NFA/DFA equivalency-Phase 2

Reachability set of a state

$R(q) = \{p \in S' | (q, \Lambda) \vdash_{M'}{}^{*} (p, \Lambda)\}$ or
$R(q) = \{p \in S' | (q, \omega) \vdash_{M'}{}^{*} (p, \omega)\}$

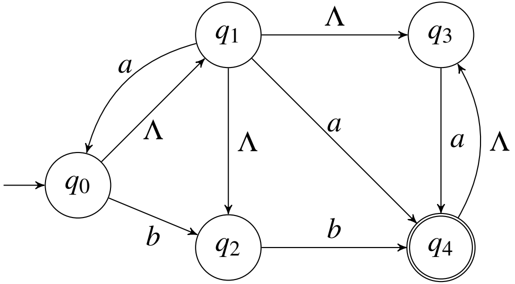


$R(q_0) = \{q_0, q_1, q_2, q_3\}$
$R(q_1) = \{q_1, q_2, q_3\}$
$R(q_2) = \{q_2\}$
$R(q_3) = \{q_3\}$
$R(q_4) = \{q_3, q_4\}$

NFA/DFA equivalency-Phase 2

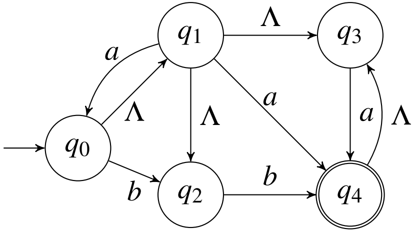
Constructing an equivalent deterministic machine:
$M'' = (S'', \Sigma, \delta'', F'')$
$S'' = \mathscr{P}(S') = 2^{S'}$
$s_0'' = R(s_0')$ The states that can be reached from the initial state by Λ transitions
$F'' = \{Q \subseteq S' | Q \cap F' \neq \varnothing\}$

NFA/DFA equivalency-Phase 2



Constructing an equivalent deterministic machine, definition of δ'':
$\forall Q \subseteq S' \wedge \forall \sigma \in \Sigma$
$\delta''(Q,\sigma) = \bigcup_P \{R(p) | \forall q \in Q \wedge \forall p \in S' \wedge \forall (q,\sigma,p) \in \Delta'\}$
Let's write all the possible triplets except empty string:
Transitions with a : $(q_1,a,q_0),(q_1,a,q_4),(q_3,a,q_4),$
Transitions with b : $(q_0,b,q_2),(q_2,b,q_4)$

NFA/DFA equivalency-Phase 2



Let's write all the possible triplets except empty string:

Transitions with a : $(q_1, a, q_0), (q_1, a, q_4), (q_3, a, q_4),$

Transitions with b : $(q_0, b, q_2), (q_2, b, q_4)$

Let's build δ'' using those transitions:

$s_0'' = R(s_0) = \{q_0, q_1, q_2, q_3\}(d_0)$

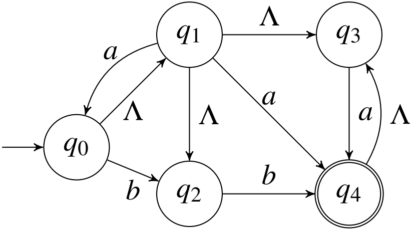
$\delta''(d_0, a) = R(q_0) \cup R(q_4) = \{q_0, q_1, q_2, q_3, q_4\}(d_1)$

$\delta''(d_0, b) = R(q_2) \cup R(q_4) = \{q_2, q_3, q_4\}(d_2)$

$\delta''(d_1, a) = \{q_0, q_1, q_2, q_3, q_4\}(d_1)$

$\delta''(d_1, b) = \{q_2, q_3, q_4\}(d_2)$

NFA/DFA equivalency-Phase 2



Let's write all the possible triplets except empty string:

Transitions with a : $(q_1, a, q_0), (q_1, a, q_4), (q_3, a, q_4),$

Transitions with b : $(q_0, b, q_2), (q_2, b, q_4)$

Let's build δ'' using those transitions:

$\delta''(d_2, a) = R(q_4) = \{q_3, q_4\}(d_3)$

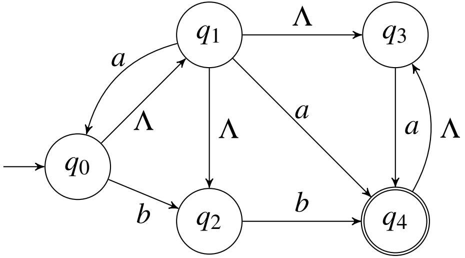
$\delta''(d_2, b) = R(q_4) = \{q_3, q_4\}(d_3)$

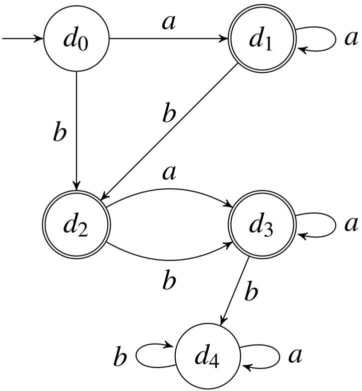
$\delta''(d_3, a) = R(q_4) = \{q_3, q_4\}(d_3)$

$\delta''(d_3, b) = \varnothing(d_4)$

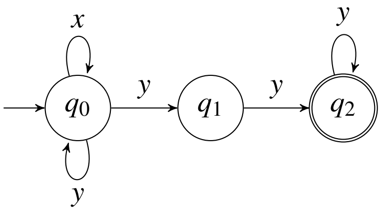
$\delta''(d_4, a) = \delta''(d_4, b) = \varnothing(d_4)$

NFA/DFA equivalency-Phase 2





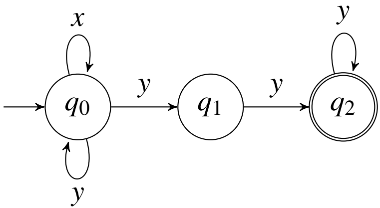
Example for NFA/DFA equivalency



An NFA recognizing the language $L(M) = (x \vee y)^* yy^+$. Let's build an equivalent DFA.

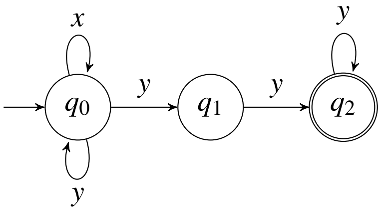
$R(q_0) = \{q_0\}$
$R(q_1) = \{q_1\}$
$R(q_2) = \{q_2\}$

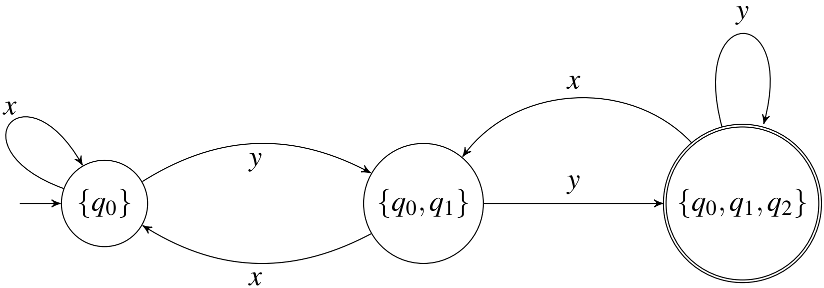
Example for NFA/DFA equivalency



$$\Delta' = \{(q_0,x,q_0),(q_0,y,q_0),(q_0,y,q_1),(q_1,y,q_2),(q_2,y,q_2)\}$$
$$s_0'' = R(q_0) = \{q_0\}$$
$$\delta(s_0'',x) = R(q_0) = \{q_0\}$$
$$\delta(s_0'',y) = R(q_0) \cup R(q_1) = \{q_0,q_1\}$$
$$\delta(\{q_0,q_1\},x) = R(q_0) = \{q_0\}$$
$$\delta(\{q_0,q_1\},y) = R(q_0) \cup R(q_1) \cup R(q_2) = \{q_0,q_1,q_2\}$$
$$\delta(\{q_0,q_1,q_2\},x) = R(q_0) = \{q_0\}$$
$$\delta(\{q_0,q_1,q_2\},y) = R(q_0) \cup R(q_1) \cup R(q_2) = \{q_0,q_1,q_2\}$$

Example for NFA/DFA equivalency





$$L(M) = ((x \vee yx)^* yy^+ x)^* (x \vee yx)^* yy^+$$

Kleene Theorem (cont.)

Any language is regular which is constructed by applying closed language operations on regular languages. e.g. if we know L_1 and L_2 are regular languages, any language built by applying a closed operation on them produces a new regular language.

Kleene Theorem (cont.)

Regular languages recognized by a finite automaton is closed under the following operations

(a) Union
(b) Concatenation
(c) Kleene star
(d) Complement
(e) Intersection

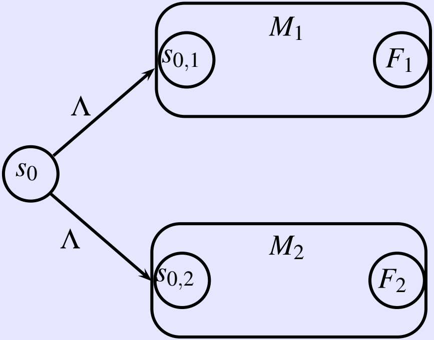
Union

$M_1 = (S_1, \Sigma, \Delta_1, s_{0,1}, F_1) \leftarrow L(M_1)$ Non-deterministic
$M_2 = (S_2, \Sigma, \Delta_2, s_{0,2}, F_2) \leftarrow L(M_2)$ Non-deterministic

$M_= (S, \Sigma, \Delta, s_0, F) \leftarrow L(M_1) \cup L(M_2)$ Non-deterministic

$S = S_1 \cup S_2 \cup \{s_0\}$ $F = F_1 \cup F_2$
$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s_0, \Lambda, s_{0,1}), (s_0, \Lambda, s_{0,2})\}$

Union

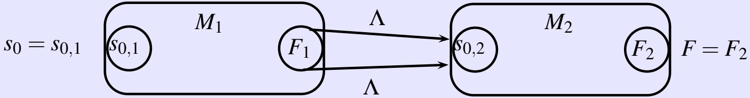


Concatanation (Non-deterministic)

$L(M_1).L(M_2) = L(M)$

$S = S_1 \cup S_2$
$s_0 = s_{0,1}$
$F = F_2$
$\Delta = \Delta_1 \cup \Delta_2 \cup (F_1 \times \{\Lambda\} \times \{s_{0,2}\})$

Concatanation

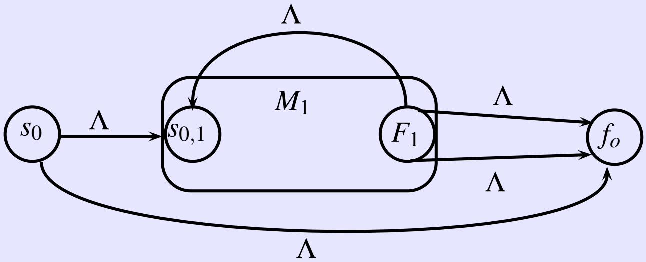


Kleene Star

$L(M_1)^* = L(M)$

$S = S_1 \cup \{s_0\}$
$F = \{f_o\}$
$\Delta = \Delta_1 \cup (F_1 \times \{\Lambda\} \times \{s_{0,1}\}) \cup (s_0, \Lambda, s_{0,1}) \cup (F_1 \times \{\Lambda\} \times F) \cup (s_0, \Lambda, f_o)$

Kleene Star



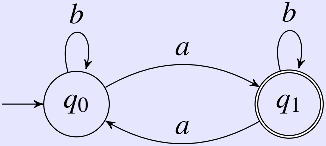
Complement (deterministic)

$\overline{L(M_1)} = L(M)$

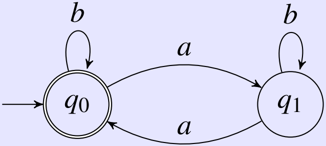
$F = \{S_1 - F_1\}$

Languages that contain odd number of a's in $\Sigma = \{a, b\}$

Complement



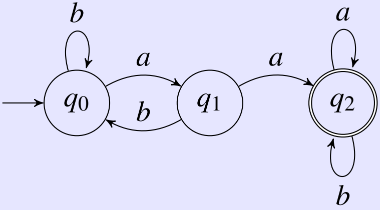
$q_1 = ba^*(b \vee ab^*a)^*$

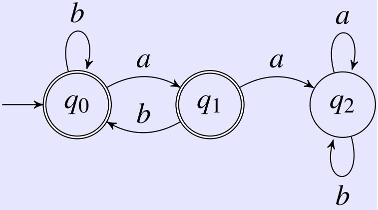


$q_0 = (b \vee ab^*a)^*$

Languages that contain at least one aa couple in $\Sigma = \{a, b\}$

Complement





$q_2 = (b \vee ab)^* aa(a \vee b)^*$

$q_0 \vee q_1 = (b \vee ab)^* (a \vee \Lambda)$

Intersection

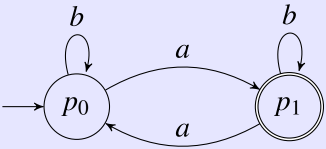
$L(M_1) \wedge L(M_2) = L(M)$

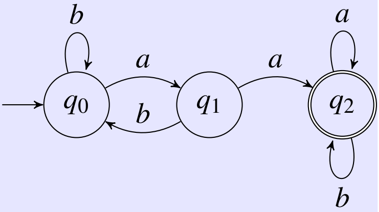
$S \subseteq S_1 \times S_2$
$s_0 = (s_{0,1}, s_{0,2})$

$F \subseteq F_1 \times F_2$ where
$[(p_1, p_2) \in F] \iff [[p_1 \in F_1] \wedge [p_2 \in F_2]]$

$\Delta \subseteq \Delta_1 \cup \Delta_2$ where
$[((p_1, p_2), \sigma, (q_1, q_2)) \in \Delta] \iff [[(p_1, \sigma, q_1) \in \Delta_1] \wedge [(p_2, \sigma, q_2) \in \Delta_2]]$

Intersection





$$S = \{p_0, p_1\}$$
$$s_0 = \{p_0\}$$
$$F = \{p_1\}$$
$$\Delta = \{(p_0, b, p_0), (p_0, a, p_1), (p_1, b, p_1), (p_1, a, p_0)\}$$

$$S = \{q_0, q_1, q_2\}$$
$$s_0 = \{q_0\}$$
$$F = \{q_2\}$$
$$\Delta = \{(q_0, b, q_0), (q_0, a, q_1), (q_1, b, q_0), (q_1, a, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$$

Intersection

$M_1:$

$S = \{p_0, p_1\}$

$s_0 = \{p_0\}$

$F = \{p_1\}$

$\Delta = \{(p_0, b, p_0), (p_0, a, p_1), (p_1, b, p_1), (p_1, a, p_0)\}$

$M_2:$

$S = \{q_0, q_1, q_2\}$

$s_0 = \{q_0\}$

$F = \{q_2\}$

$\Delta = \{(q_0, b, q_0), (q_0, a, q_1), (q_1, b, q_0), (q_1, a, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$

$M_1 \cap M_2:$

$S = \{(p_0, q_0), (p_0, q_1), (p_0, q_2), (p_1, q_0), (p_1, q_1), (p_1, q_2)\}$

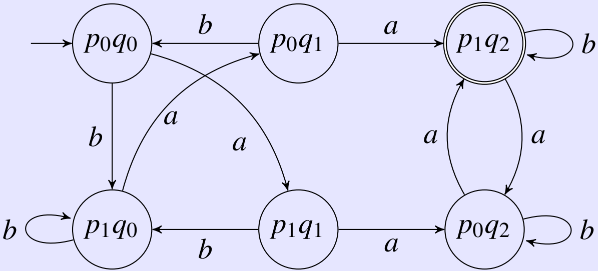
$s_0 = \{(p_0, q_0)\}$

$F = \{(p_1, q_2)\}$

$$\begin{aligned}\Delta = \{&[(p_0, q_0), b, (p_0, q_0)], [(p_0, q_1), b, (p_0, q_0)], [(p_0, q_2), b, (p_0, q_2)]\\ &[(p_1, q_0), b, (p_1, q_0)], [(p_1, q_1), b, (p_1, q_0)], [(p_1, q_2), b, (p_1, q_2)]\\ &[(p_0, q_0), a, (p_1, q_1)], [(p_0, q_1), a, (p_1, q_2)], [(p_0, q_2), a, (p_1, q_2)]\\ &[(p_1, q_0), a, (p_0, q_1)], [(p_1, q_1), a, (p_0, q_2)], [(p_1, q_2), a, (p_0, q_2)]\}\end{aligned}$$

Intersection

$M_1 \cap M_2$:



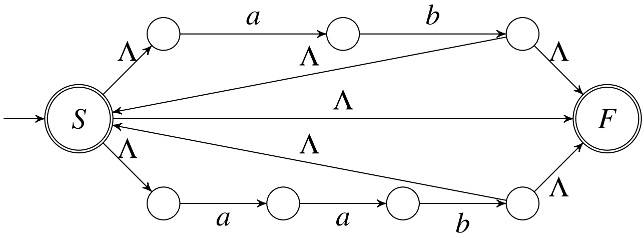
$$\Delta = \{[(p_0,q_0),b,(p_0,q_0)],[(p_0,q_1),b,(p_0,q_0)],[(p_0,q_2),b,(p_0,q_2)]$$
$$[(p_1,q_0),b,(p_1,q_0)],[(p_1,q_1),b,(p_1,q_0)],[(p_1,q_2),b,(p_1,q_2)]$$
$$[(p_0,q_0),a,(p_1,q_1)],[(p_0,q_1),a,(p_1,q_2)],[(p_0,q_2),a,(p_1,q_2)]$$
$$[(p_1,q_0),a,(p_0,q_1)],[(p_1,q_1),a,(p_0,q_2)],[(p_1,q_2),a,(p_0,q_2)]\}$$

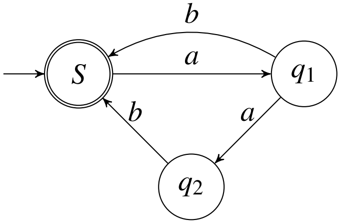
Kleene Theorem (cont.)

Regular languages are closed under complement and intersection operations.

$$
\begin{aligned}
L_1 \cap L_2 &= \overline{\overline{L_1} \cup \overline{L_2}} \\
&= \Sigma^* - \left((\Sigma^* - L_1) \cup (\Sigma^* - L_2) \right)
\end{aligned}
$$

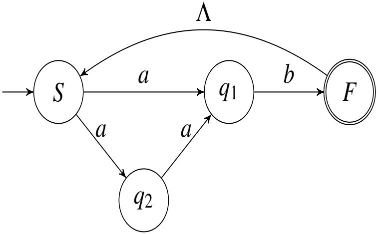
$(ab \vee aab)^*$

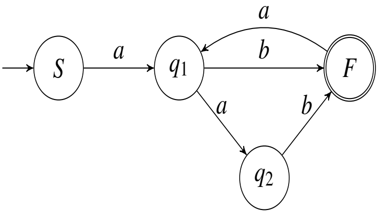




$$< S >::= \Lambda \mid a < q_1 >$$
$$< q_1 >::= b < S > \mid a < q_2 >$$
$$< q_2 >::= b < S >$$

$$(ab \vee aab)^{+} = (ab \vee aab)(ab \vee aab)^{*}$$



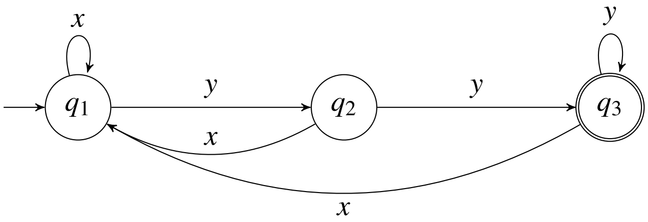


$$< S > ::= a < q_1 >$$
$$< q_1 > ::= b | b < F > | a < q_2 >$$
$$< q_2 > ::= b | b < F >$$
$$< F > ::= a < q_1 >$$

Systematic way to find the regular language recognized by a DFA

Remember the theorem that states the one and only solution to the equation $X = XA \cup B \;\; \wedge \;\; \Lambda \notin A$ is $X = BA^*$.
Let's rewrite the statement using regular expressions:
$x = xa \vee b \;\; \wedge \;\; \Lambda \notin A \Rightarrow x = ba^*$
We shall use this theorem in finding the regular language recognized by a DFA

Example



$q_1 = q_1 x \vee q_2 x \vee q_3 x \vee \Lambda$
$q_2 = q_1 y$
$q_3 = q_2 y \vee q_3 y$
We can use the theorem for q_3
$(q_3) = (q_3) y \vee q_2 y$ than we have $q_3 = q_2 y y^* \Rightarrow q_3 = q_1 y y^+$

Example

$(q_3) = (q_3)y \vee q_2y$ than we have $q_3 = q_2yy^* \Rightarrow q_3 = q_1yy^+$
Using this equality:
$q_1 = q_1x \vee q_1yx \vee q_1yy^+x \vee \Lambda$
$q_1 = q_1(x \vee yx \vee yy^+x) \vee \Lambda$
$q_1 = (x \vee yx \vee yy^+x)^* = (y^*x)^*$
$q_3 = (y^*x)^*yy^+$
The example automaton is actually the DFA equivalent of the NFA given in the previous examples. Heuristically we have found $(x \vee y)^*yy^+$ as the language of the NFA. Let's show that these two are equivalent.

Proof of Example

We are going to prove $(y^*x)^*yy^+ = (x \vee y)^*yy^+$

$(x \vee y)^*yy^+ = (x \vee y)^*y^*yy$

$(y^*x)^*yy^+ = (y^*x)^*y^*yy$

$(x \vee y)^*y^* \stackrel{?}{=} (y^*x)^*y^*$

We need to prove

a) $(y^*x)^*y^* \subseteq (x \vee y)^*y^*$ and

b) $(x \vee y)^*y^* \subseteq (y^*x)^*y^*$

Proof of Example

The proof of (a):

$(y^*x)^* \subseteq (y^*x^*)^*$

$(y^*x)y^* \subseteq (y^*x^*)^*y^* = (x^*y^*)^*y^*$

$(x^*y^*)^* = \Lambda \vee x^*y^* \vee (x^*y^*)^2 \vee \ldots \vee (x^*y^*)^n \vee \ldots$

$(x^*y^*)^*y^* = y^* \vee x^*y^*y^* \vee \ldots \vee (x^*y^*)^{n-1}x^*y^*y^* \vee \ldots$

$(x^*y^*)^*y^* = \Lambda \vee y^+ \vee x^*y^*y^* \vee \ldots \vee (x^*y^*)^{n-1}x^*y^*y^* \vee \ldots$

$(x^*y^*)^*y^* = \Lambda \vee y^+ \vee x^*y^* \vee \ldots \vee (x^*y^*)^n \vee \ldots$

$(x^*y^*)^*y^* = \Lambda \vee x^*y^* \vee \ldots \vee (x^*y^*)^n \vee \ldots = (x^*y^*)^*$

Proof of Example

The proof of (b):

$(x \vee y)^* y^* \subseteq (y^* x)^* y^*$

$(x \vee y)^* = \Lambda \vee (x \vee y) \vee (x \vee y)^2 \vee (x \vee y)^3 \vee \ldots$

Let's use induction

$(x \vee y)^0 = \Lambda \subseteq (y^* x)^* y^*$

$(x \vee y)^1 \subseteq (y^* x)^* y^*$

$\ldots$

Inductive step: $(x \vee y)^n \subseteq (y^* x)^* y^*$

$(x \vee y)^n (x \vee y) = (x \vee y)^n x \vee (x \vee y)^n y \overset{?}{\subseteq} (y^* x)^* y^*$

i) $(x \vee y)^n x \subseteq (y^* x)^* y^* x = (y^* x)^+ \subseteq (y^* x)^* \subseteq (y^* x)^* y^*$

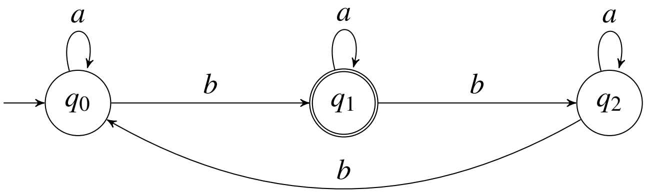
ii) $(x \vee y)^n y \subseteq (y^* x)^* y^* y = (y^* x)^* y^+ \subseteq (y^* x)^* y^*$

P.S.:

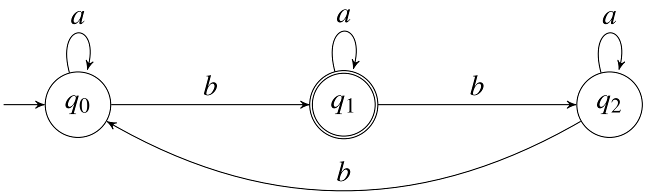
- $X = XA \cup \{\Lambda\}$'s solution is $X = A^*$
- $X = XA$ has no solution since $B = \varnothing$

Example

Construct an automaton recognizing strings containing $3k+1$ b symbols and discover the corresponding regular expression.
$\Sigma = \{a, b\}$



Example



One possible solution is $a^*ba^*[(ba^*)^3]^*$

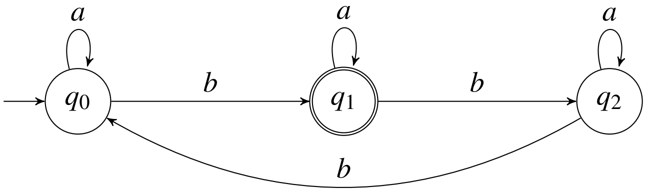
$q_0 = q_0a \vee q_2b \vee \Lambda$

$q_1 = q_0b \vee q_1a$

$q_2 = q_1b \vee q_2a$

$$\begin{array}{l} q_2 = q_2a \vee q_1b \Rightarrow q_2 = q_1ba^* \\ q_1 = q_1a \vee q_0b \Rightarrow q_1 = q_0ba^* \\ \hline q_2 = q_0(ba^*)(ba^*) \end{array}$$

Example



One possible solution is $a^*ba^*[(ba^*)^3]^*$

$q_0 = q_0a \vee q_2b \vee \Lambda$

$q_1 = q_0b \vee q_1a$

$q_2 = q_1b \vee q_2a$

$q_0 = q_0a \vee q_0(ba^*)^2b \vee \Lambda$

$q_0 = q_0(a \vee (ba^*)^2b) \vee \Lambda$

$q_0 = (a \vee (ba^*)^2b)^*$

$q_1 = (a \vee (ba^*)^2b)^*b \vee q_1a$

$q_1 = (a \vee (ba^*)^2b)^*ba^*$

$q_1 = (a \vee (ba^*)^2b)^*(ba^*)$

Pumping Lemma

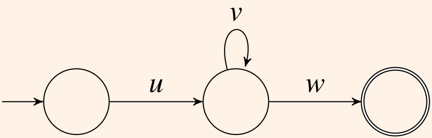
Theorem

If L is a regular language with unrestricted word length than in this language we can build any word longer than n using substrings u, v, w in following form: uv^iw. The following conditions should apply

i v should not be empty string

ii $|uv| \leq n$

iii u and w can be empty strings.



Proof

Each regular language L, can be recognized by a deterministic finite automaton.

- $M = \{S, \Sigma, \delta, s_0, F\}$
- $|s| = n$

$X = \sigma_1\sigma_2\ldots\sigma_\ell$, is a word of L and $|X| = \ell > n$

$(s_0, \sigma_1\sigma_2\ldots\sigma_\ell) \vdash_M (s_1, \sigma_2\ldots\sigma_\ell) \vdash_M \ldots \vdash_M (s_{\ell-1}, \sigma_\ell) \vdash_M (s_\ell, \Lambda)$ ve $s_\ell \in F$

If $\ell > n \wedge |S| = n$ than according to the pigeonhole principle $\exists(i,j) : s_i = s_j \wedge 0 \leq i < j \leq n^* \wedge i \neq j$

*If there is more than one couple, the couple with minimum $|i - j|$ should be selected.

Proof

In this case for our couple of states s_i, s_j
$\sigma_1\sigma_2\ldots\sigma_i = u$ $\sigma_i\ldots\sigma_j = v$ ve $\sigma_j\ldots\sigma_\ell = w$

X can be written as $X = uvw$; this makes $X_0 = uw$ and
$X_m = uv^m w \ \ (m \in \mathbb{N}^+)$ strings to be recognized by M

If $|X_0| = |uw| \geq n$ than proof can be repeated by using uw instead of X. The diagrammatic representation of M is a connected graph which makes the longest length of a path at most n.

$$(s_0, uv^m w) \vdash_M^* (s_i, v^m w) \vdash_M^* (s_i, v^{m-1} w) \vdash_M^* (s_i, w) \vdash_M^* (s_\ell, \Lambda)$$

Proving Using Pumping Lemma

A proof using the pumping lemma that L cannot be accepted by a finite automaton is a proof by contradiction.

1. We assume, for the sake of contradiction, that L can be accepted by M, an FA with n states.
2. We try to select a string in L with length at least n so that statements 1-3 in Theorem lead to a contradiction.

If we don't get a contradiction, we haven't proved anything, and so we look for a string x that will produce one.

There doesn't exist any language which doesn't satisfy "Pumping lemma".[†]

[†] One way implication

Example 1

$$L(M) = a^n b^n | n \in \mathbb{N}^+$$

To simplify, let's assume M automaton has three states. Let's try to split $aaabbb$ word into u, v, w substrings[‡].

1. If $u = a$ $v = aa$ $w = bbb$ then $uv^2w = a\ aaaa\ bbb$ is not in L
2. If $u = aaa$ $v = bb$ $w = b$ then $uv^2w = aaa\ bbbb\ b$ is not in L
3. If $u = aa$ $v = ab$ $w = bb$ then $uv^2w = aa\ abab\ bb$ is not in L

$L(M)$ is not a regular language.

[‡] $v \leq 3$

Example 2

Let L be the language

$$L = a^{i^2} | i \geq 0$$

Assumptions:

- Suppose that there is an FA M having n states and accepting L
- Choose $x = a^{n^2}$

By pumping lemma

- $x = uvw$ and $0 < |v| \leq n$
- $n^2 = |uvw| < |uv^2w| = n^2 + |v| \leq n^2 + n < n^2 + 2n + 1 = (n+1)^2$
- Condition 3 says that $|uv^2w|$ must be i^2 for some integer i, but there is no integer i whose square is strictly between n^2 and $(n+1)^2$

Example 3

Let L be the set of legal C programs, the string

`main() ...`

with m occurrences of "{" and n occurrences of "}", is a legal C program precisely if $m = n$. Assumptions:

- Suppose that there is an FA M having n states and accepting L
- Let x be the string `main()` $\{^n\}^n$

By pumping lemma

- $x = uvw$
- if $i = 0$ in condition 3 string v cannot contain any right brackets because of condition 1
- if the shorter string uw is missing any of the symbols in "main()", then it is not a legal C program
- if it is missing any of the left brackets, then the two numbers don't match.