THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

Theory of Computation
Week 3

Much of the material on this slides comes from the recommended textbook by Elaine Rich

Announcement

Release of Assignment 1

- Release 13/03/2020
- Due: 29/03/2020

Detailed content

Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques
- ✓ Week 2 – Languages and strings. Hierarchies. Computation. Closure properties

Week 3 – Finite State Machines: non-determinism vs. determinism

- ❑ Week 4 – Regular languages: expressions and grammars
- ❑ Week 5 – Non regular languages: pumping lemma. Closure
- ❑ Week 6 – Context-free languages: grammars and parse trees
- ❑ Week 7 – Pushdown automata
- ❑ Week 8 – Non context-free languages: pumping lemma and decidability. Closure
- ❑ Week 9 – Decidable languages: Turing Machines
- ❑ Week 10 – Church-Turing thesis and the unsolvability of the Halting Problem
- ❑ Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)
- ❑ Week 12 – Revision of the hierarchy. Safety-critical systems
- ❑ Week 13 – Extra revision (if needed)

Week 03 Videos

You already know

- ❑ Deterministic Finite State Machine (DFSM)
  - ❑ Informal Definition
  - ❑ Accept / Reject
  - ❑ Formal Definition
  - ❑ Dead state
- ❑ Non Deterministic Finite State Machine (NDFSM)
  - ❑ Difference between DFSM and NDFSM

 Videos to watch before lecture

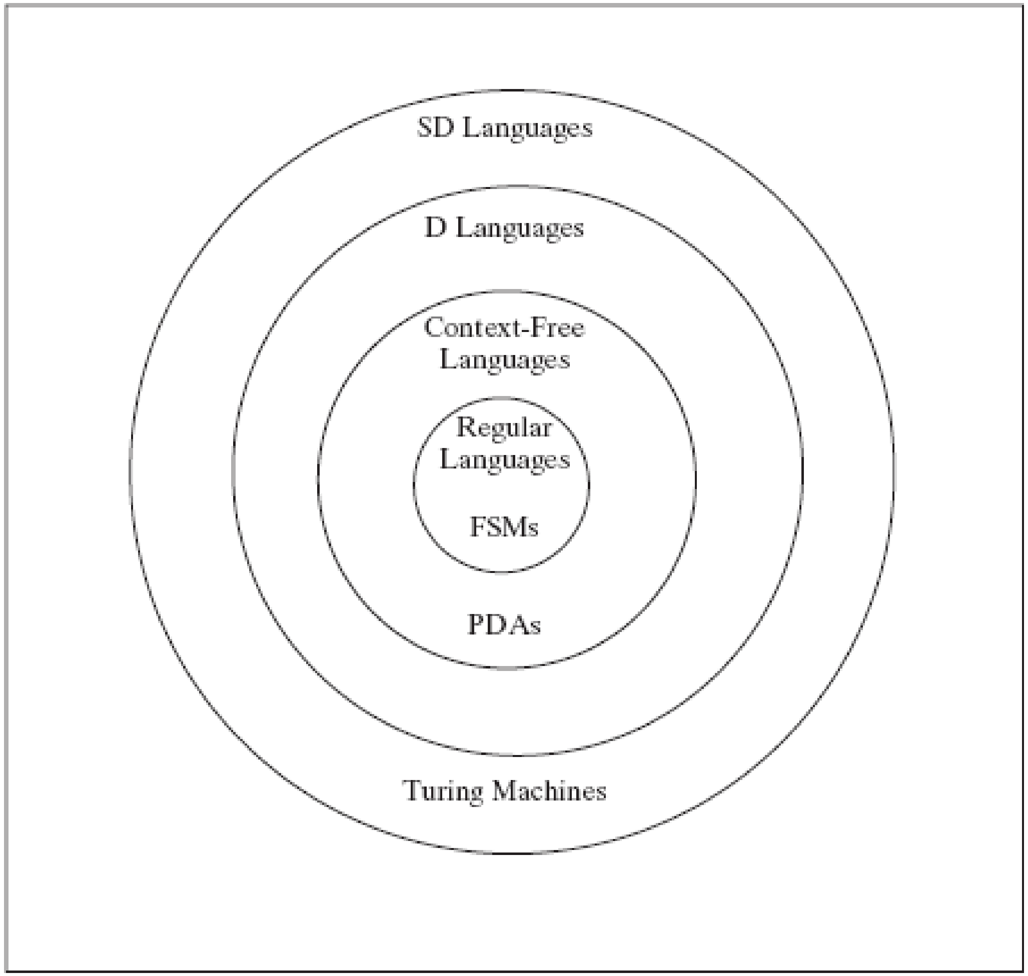
 Additional videos to watch for this week

Week 03 Lecture Outline

Finite State Machines: non-determinism vs. determinism

- ❑ Finite State Machine (Deterministic) - DFSM
- ❑ How to construct FSM for a language?
- ❑ Some Theorems on FSM
- ❑ Minimizing FSM
- ❑ Non-deterministic Finite State Machine (NDFSM)
- ❑ Difference between DFSM and NDFSM
- ❑ How to construct NDFSM for a language?
- ❑ Examples showing advantage of NDFSM
- ❑ Equivalence of DFSM and NDFSM

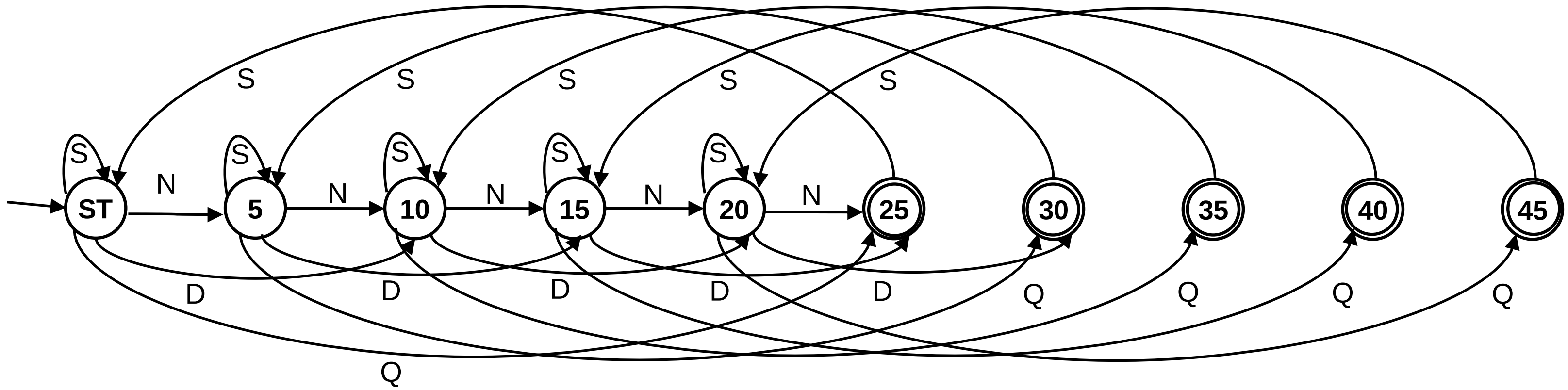
THE HIERARCHY



ON OFF Switch

Vending Machine as a FINITE STATE MACHINE

- A vending machine controller to accept $.25 for a drink:



S:soda; **D**:dime; **N**:nickel; **Q**:quarter;

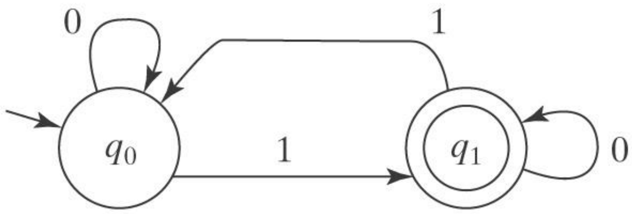
FINITE STATE MACHINES
Definition

- A Finite State Machine M is a quintuple

$\boldsymbol{M = (K, \Sigma, \delta, s, A)}$, where:

- – $\boldsymbol{K}$ is a finite set of states
- – Σ is an alphabet
- – δ is the transition function from $(K \times \Sigma)$ to K
- – $\boldsymbol{s} \in K$ is the initial state, and
- – $\boldsymbol{A} \subseteq K$ is the set of accepting states

K=$\{q_0,q_1\}$
$\Sigma=\{0,1\}$
$\delta=\{(q_0,0)=q_0,\ (q_0,1)=q_1,$
$(q_1,0)=q_1,\ (q_1,1)=q_0\}$
s=q_0
A=$\{q_1\}$



FINITE STATE MACHINES

- Informally, ***M*** accepts a string *w* iff *M* winds up in some element of ***A*** when it has finished reading *w*.

- The language accepted by ***M***, denoted ***L*(*M*)**, is the set of all strings accepted by *M*.

FINITE STATE MACHINES

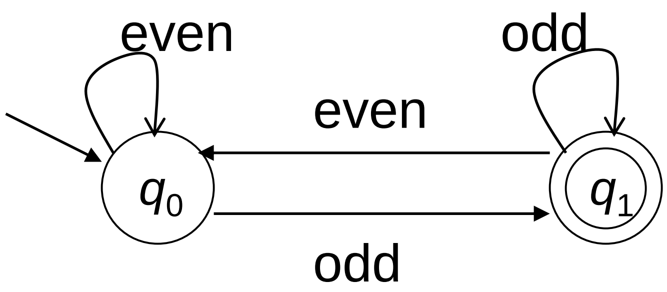
- Informally, *a* ***configuration*** of a DFSM M is an element of:

$$K \times \Sigma^*$$

- It captures the two things that can make a difference to M's future behavior:
  - its current state
  - the input that is still left to read.
- The ***initial configuration*** of a DFSM M, on input w, is:

$$(s_M, w)$$

FINITE STATE MACHINES
Example: Yields relation



On input 235, the configurations are:

$(q_0, 235) \ \vdash_M \ (q_0, 35) \ \vdash_M \ (q_1, 5) \ \vdash_M \ (q_1, \varepsilon)$

$C_0 \ \vdash_M \ C_1 \ \vdash_M \ C_2 \ \vdash_M \ C_3$

Thus $(q_0, 235) \vdash_M^* (q_1, \varepsilon)$

FINITE STATE MACHINES

A ***computation*** by M is a finite sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that:

- C_0 is an initial configuration,
- C_n is of the form (q, ε), for some state $q \in K_M$,
- $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$.

FINITE STATE MACHINES

- A DFSM M ***accepts*** a string w iff:
  $(s, w) \vdash_M{}^* (q, \varepsilon)$, for some $q \in A_M$.

- A DFSM M ***rejects*** a string w iff:
  $(s, w) \vdash_M{}^* (q, \varepsilon)$, for some $q \notin A_M$.

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

REGULAR LANGUAGES

A language is ***regular*** iff it is accepted by some FSM.

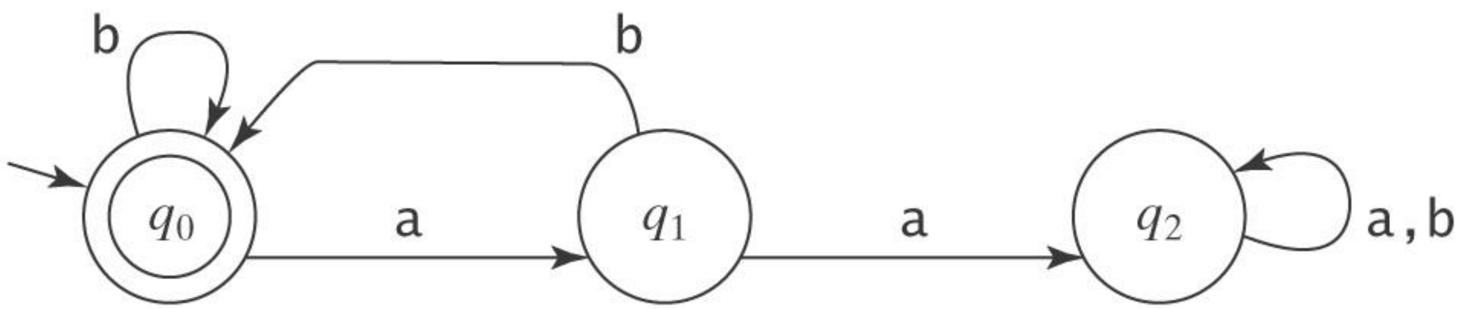
- Example: $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* :$

  every $\mathtt{a}$ is immediately followed by a $\mathtt{b}\}$.

FINITE STATE MACHINES

- Example: $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$:

  every $\mathtt{a}$ is immediately followed by a $\mathtt{b}\}$.



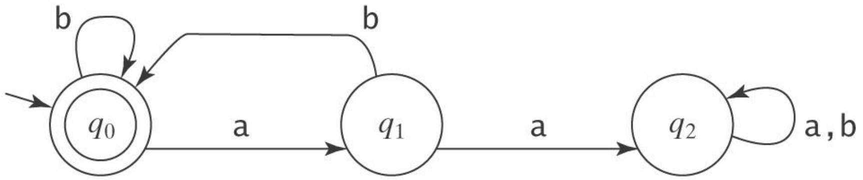
FINITE STATE MACHINES

- Example: $L = \{w \in \{\texttt{a}, \texttt{b}\}^*$:

  every $\texttt{a}$ is immediately followed by a $\texttt{b}\}$.

DFSM $M = (\{q_0, q_1, q_2\}, \{a,b\}, \delta, q_0, \{q_0\})$ where:

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



FINITE STATE MACHINES

- ***Theorem:*** Every DFSM M, on input s, halts in $|s|$ steps.

- **Proof:** On input string s, M executes a computation $C_0 \mid\!-_M C_1 \mid\!-_M C_2 \mid\!-_M \dots \mid\!-_M C_n$, where C_0 is the initial configuration and C_n is either an accepting of rejecting configuration. So M will halt when it reaches C_n. Each step consumes a character of s, so $n=|s|$. Thus M will halt after $|s|$ steps.

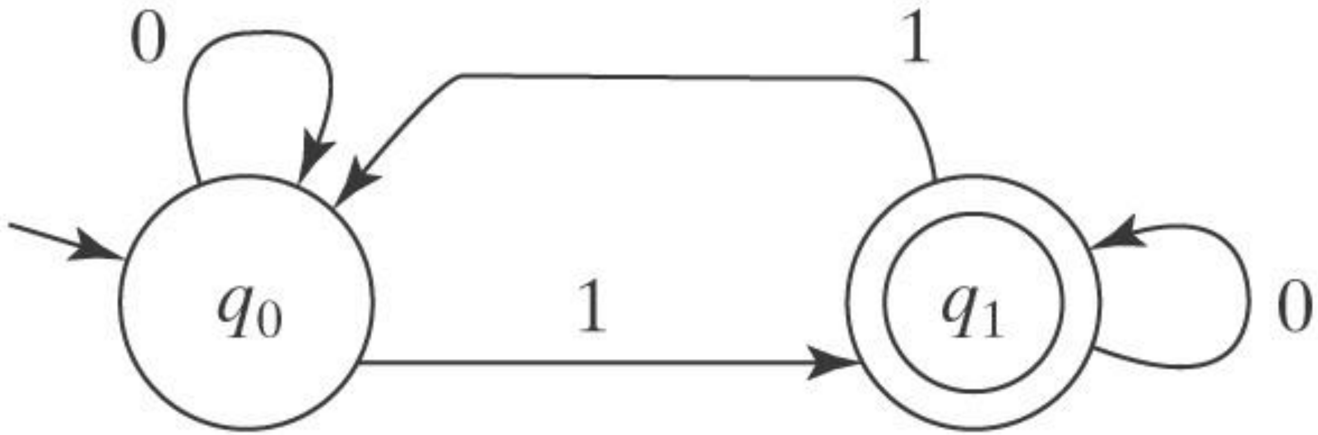
FINITE STATE MACHINES
Example: Parity Checking

- $L = \{w \in \{\mathtt{0}, \mathtt{1}\}^* : w \text{ has odd parity}\}$

- A binary string has odd parity iff the number of 1’s in it is odd

FINITE STATE MACHINES
Example: Parity Checking

- $L = \{w \in \{\texttt{0}, \texttt{1}\}^* : w \text{ has odd parity}\}$.

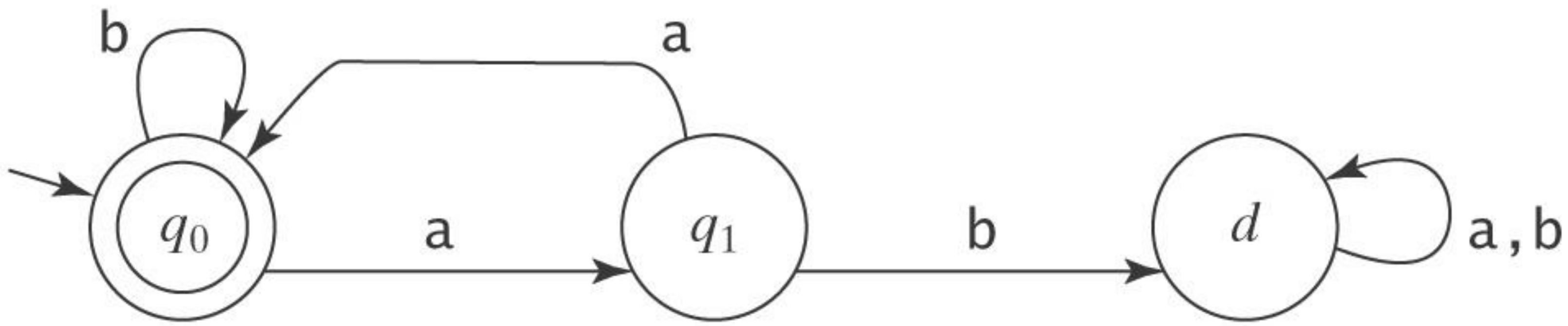


FINITE STATE MACHINES
Dead states

- If we look at the language on slide 16, we can see that once a string enters state q_2, it can never exit it to become an accepted string.
- We call these type of states “dead states”
- To describe DFSMs sometimes we omit the dead states, as this makes them look ‘neater’
- By convention if there is no transition specified for a (state, input-char) we assume there is a dead state

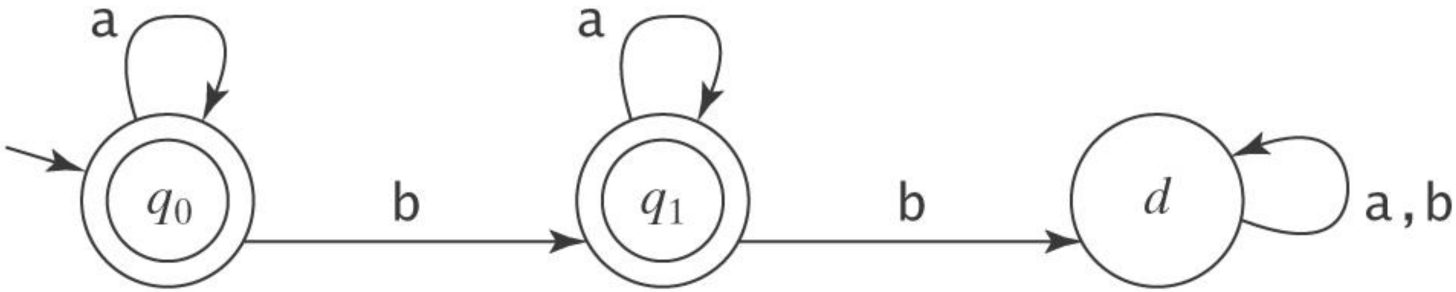
FINITE STATE MACHINES
Dead states

- $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* :$ every $\mathtt{a}$ region in w is of even length$\}$



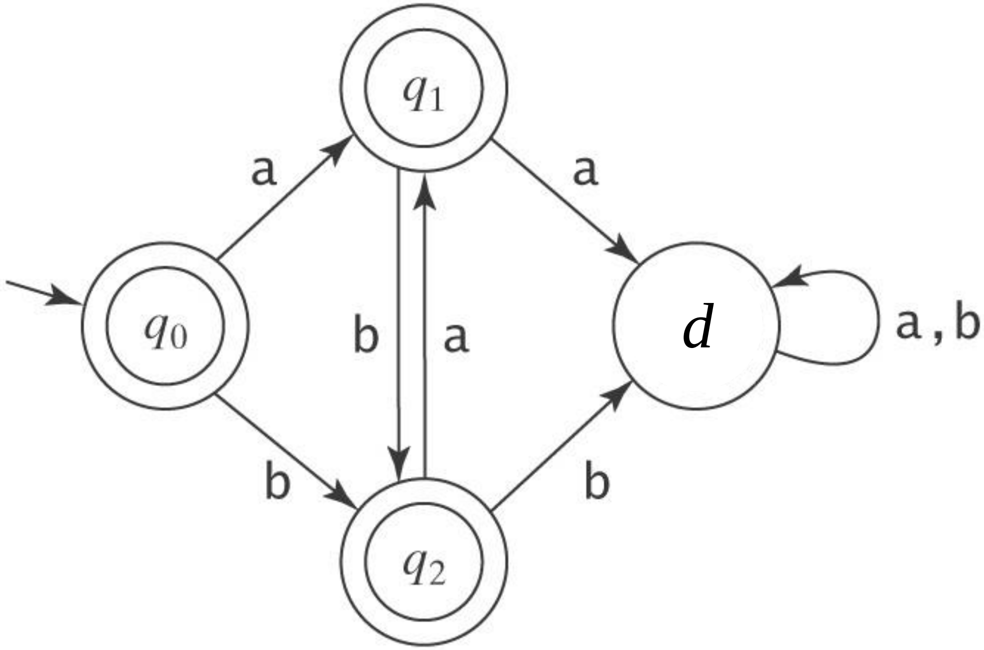
FINITE STATE MACHINES
Example: No more than one b

- $L = \{w \in \{\texttt{a}, \texttt{b}\}^* : w$ contains no more than one $\texttt{b}\}$.



FINITE STATE MACHINES
Example: Consecutive characters

- $L = \{w \in \{\texttt{a}, \texttt{b}\}^* :$ no two consecutive characters are the same$\}$.

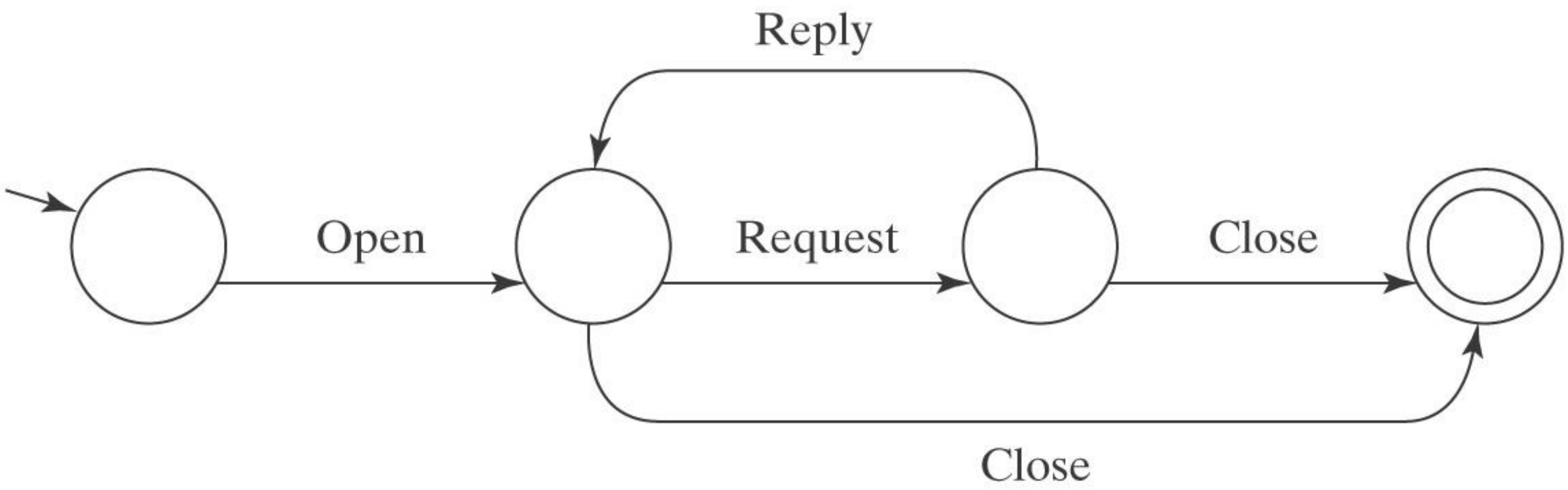


FINITE STATE MACHINES
Dead states

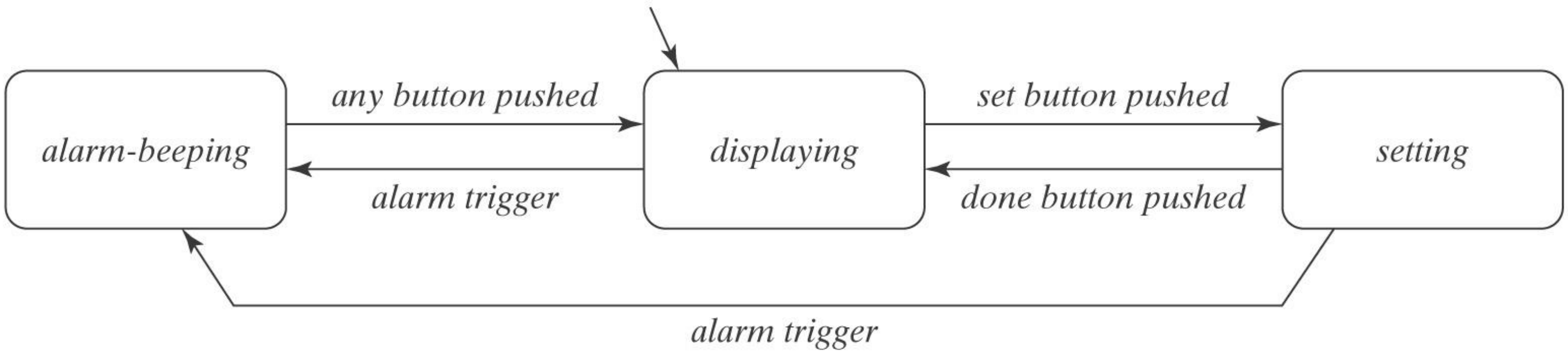
- $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* :$ every $\mathtt{b}$ in w is surrounded by $\mathtt{a}$'s$\}$

FINITE STATE MACHINES
Another example

- A simple communication protocol
  - Σ={Open, Request, Reply, Close}

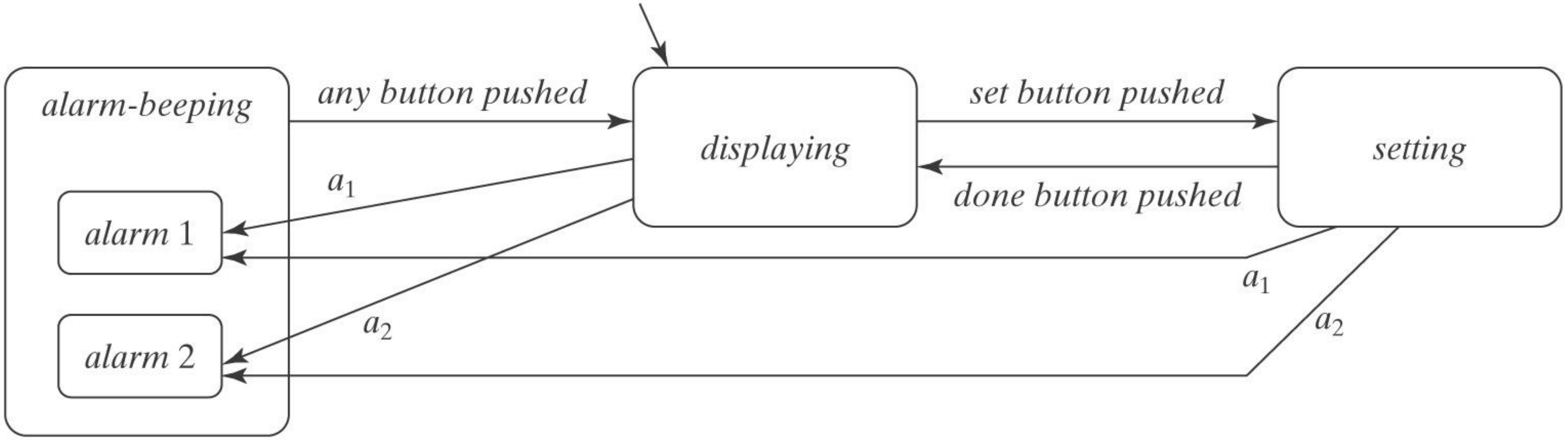


FSM REPRESENTATIONS SOFTWARE ENGINEERING



A high-level state chart model of a digital watch.

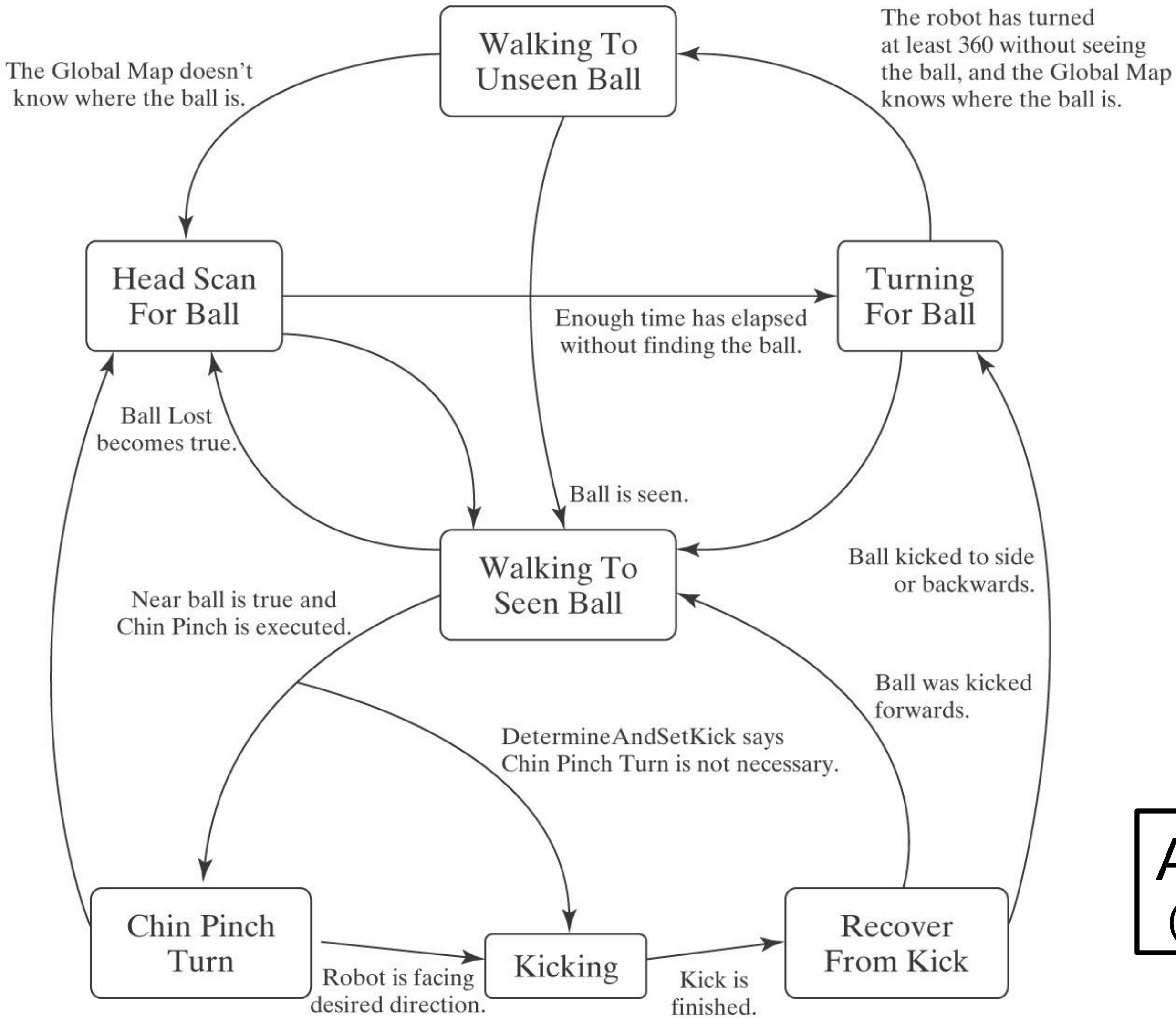
FSM REPRESENTATIONS SOFTWARE ENGINEERING



Hierarchical model

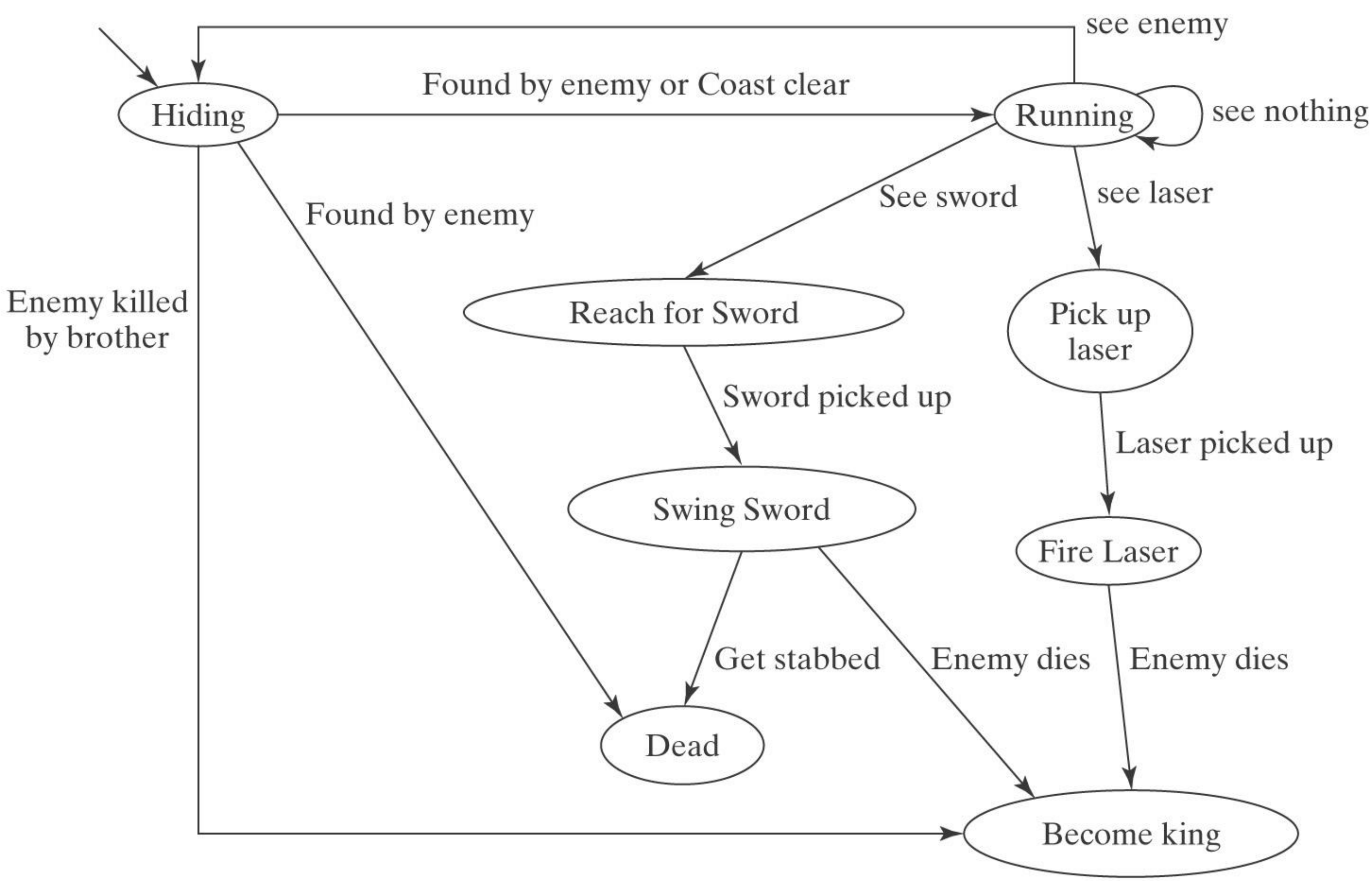
Controlling a Soccer-Playing Robot

Controlling a Soccer-Playing Robot



Appendix P
(page:1062-63)

A Finite State Model of a Game Character

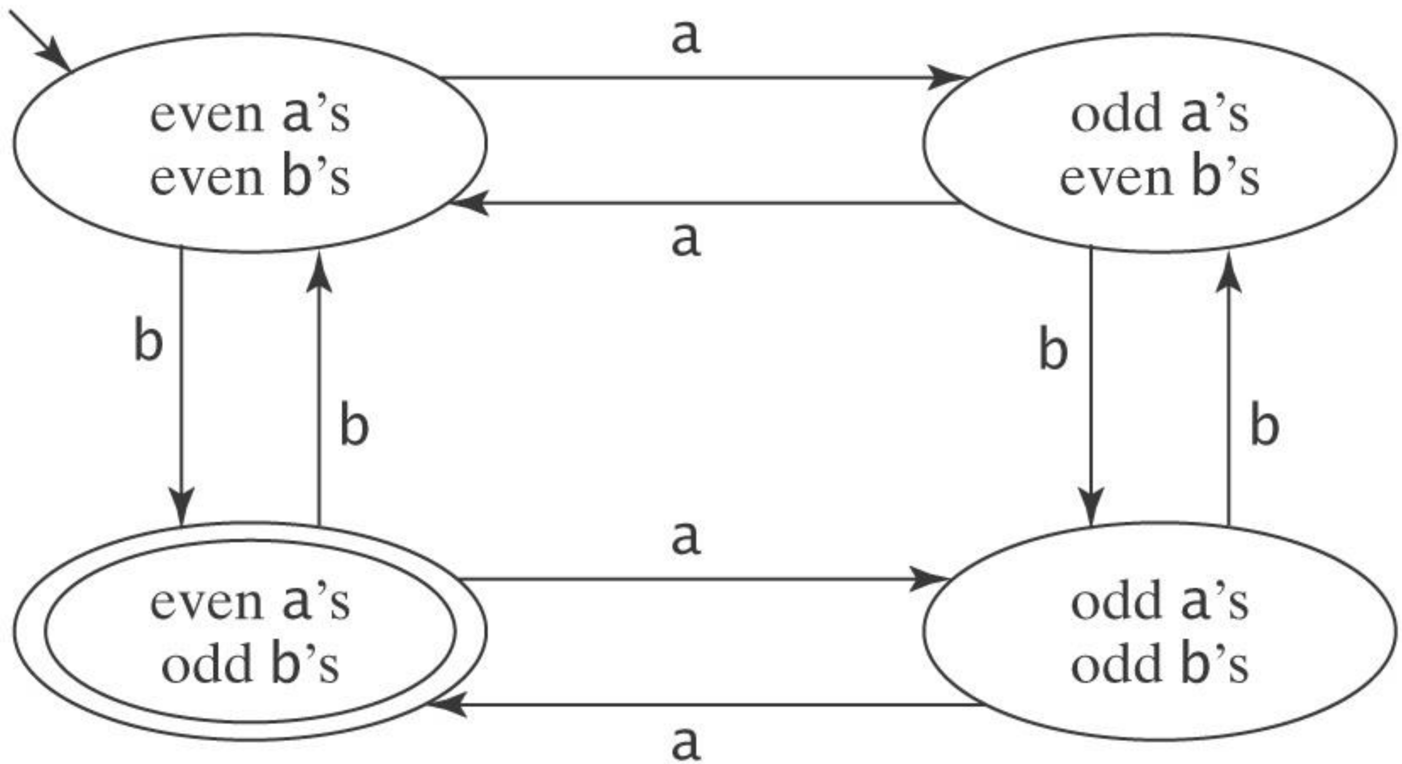


PROGRAMMING FSMs
Tricks

- Cluster strings that share a "future".

- Let $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : w$ contains an even number of $\mathtt{a}$'s and an odd number of $\mathtt{b}$'s$\}$

PROGRAMMING FSMs
Tricks



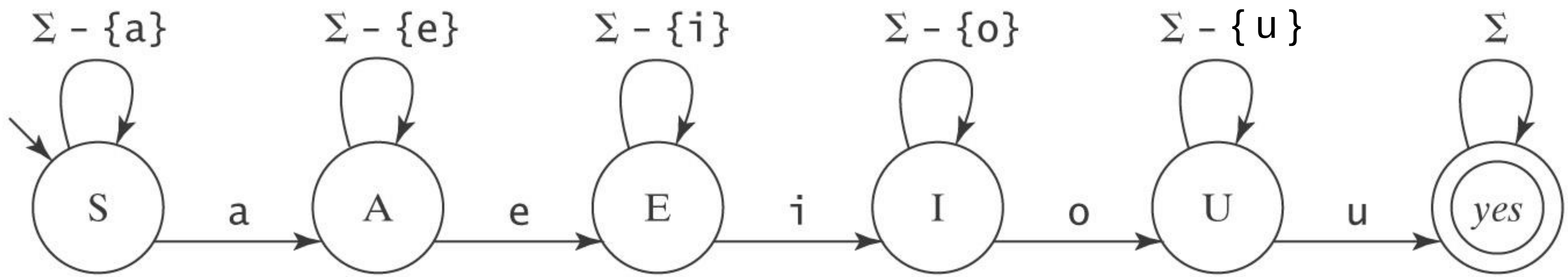
PROGRAMMING FSMs
Tricks

- $L = \{w \in \{\texttt{a} - \texttt{z}\}^*$: all five vowels, $\texttt{a}$, $\texttt{e}$, $\texttt{i}$, $\texttt{o}$, and $\texttt{u}$, occur in w in alphabetical order$\}$.

- abstemious, facetious, sacrilegious $\in$ L
- tenacious $\notin$ L

PROGRAMMING FSMs
Tricks

- $L = \{w \in \{$a - z$\}^*$: all five vowels, a, e, i, o, and u, occur in w in alphabetical order}.



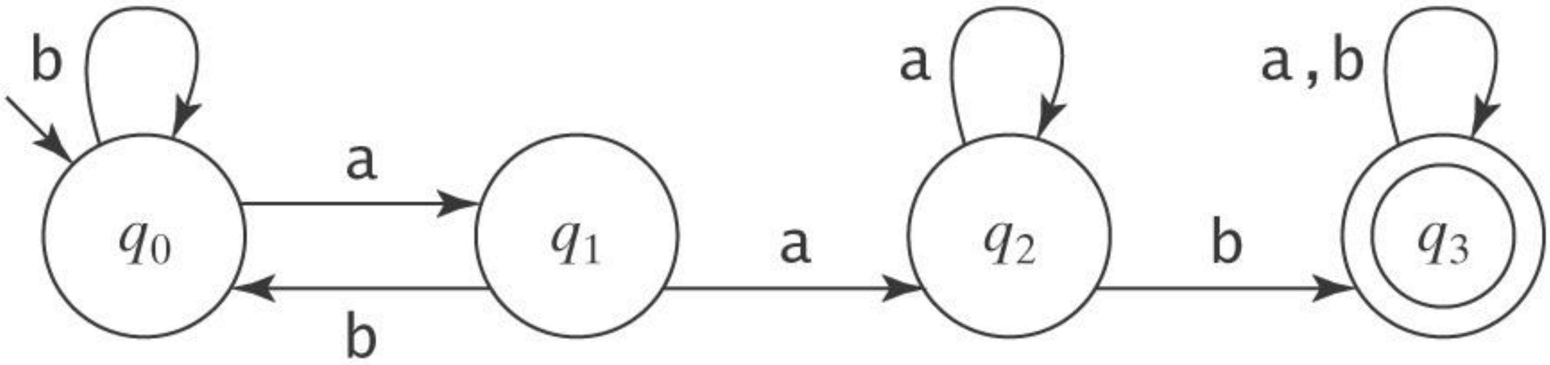
PROGRAMMING FSMs
Tricks

- Sometimes to design a FSM is better to begin designing its complement
- $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : w$ does not contain the substring $\mathtt{aab}\}$.

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

PROGRAMMING FSMs
Tricks

- $L = \{w \in \{\texttt{a}, \texttt{b}\}^* : w$ does not contain the substring `aab`$\}$.



How much has to be changed?

PROGRAMMING FSMs
Tricks

Let Σ = {`a`, `b`, `c`, `d`}.

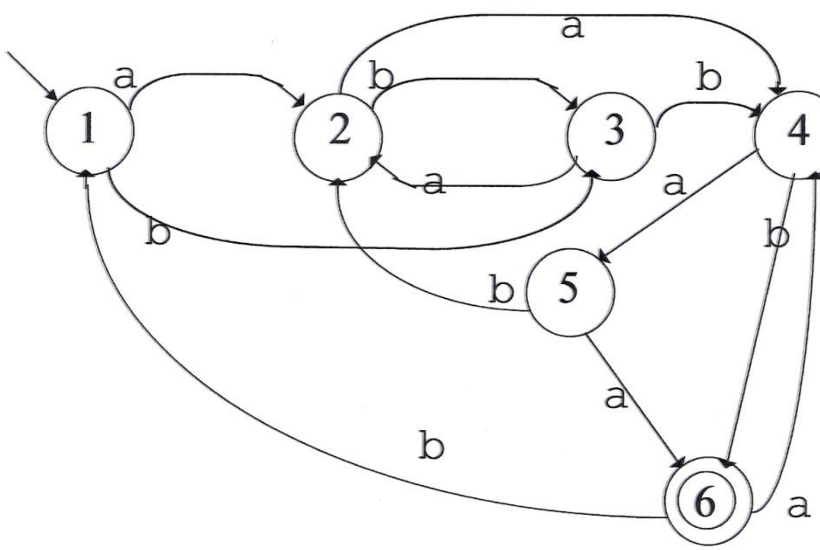
Let $L_{Missing}$ =
{w : there is a symbol $a_i \in \Sigma$ not appearing in w}.

Try to make a DFSM for $L_{Missing}$

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

PROGRAMMING FSMs
State Minimisation

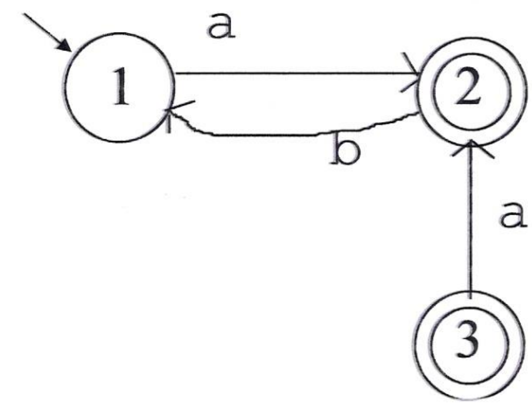
Consider:



Is it a minimal machine?

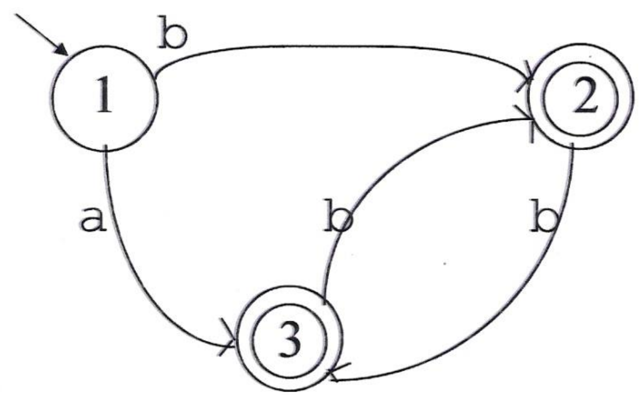
PROGRAMMING FSMs
State Minimisation

Step (1): Get rid of unreachable states.



State 3 is unreachable.

Step (2): Get rid of redundant states.

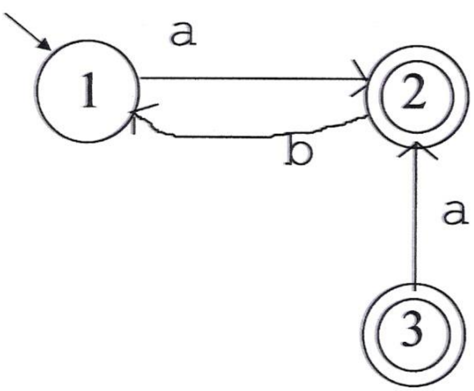


States 2 and 3 are redundant.

PROGRAMMING FSMs
State Minimisation

We can't easily find the **unreachable states** directly. But we can find the reachable ones and determine the unreachable ones from there.

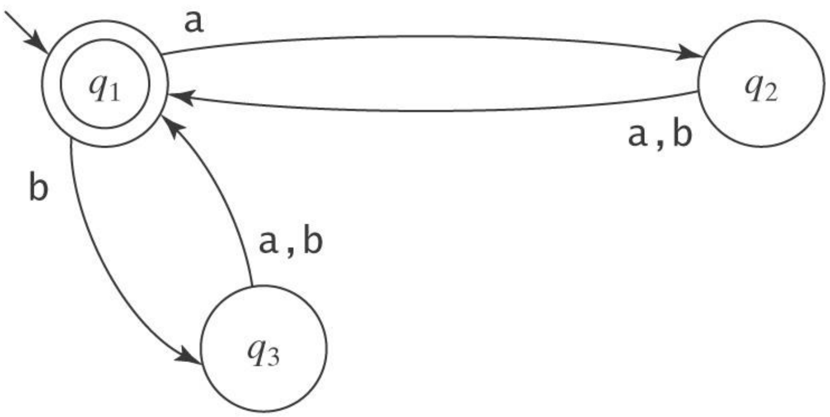
An algorithm for finding the reachable states:



PROGRAMMING FSMs
State Minimisation

Intuitively, two states are **equivalent** to each other (and thus one is redundant) if all strings in Σ^* have the same fate, regardless of which of the two states the machine is in. But how can we tell this?

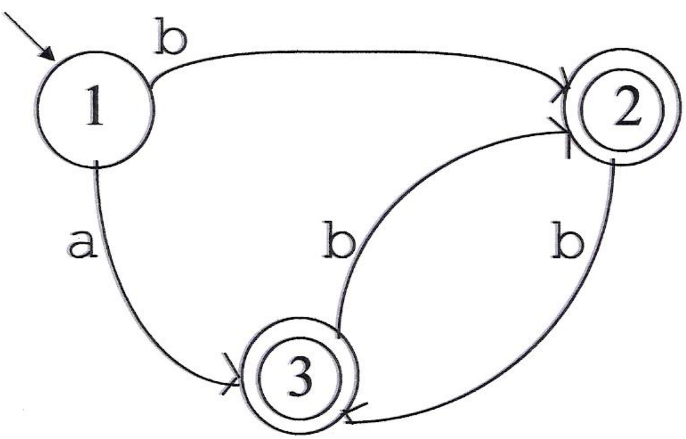
The simple case:



Two states have identical sets of transitions out.

PROGRAMMING FSMs
State Minimisation

The harder case:



The outcomes in states 2 and 3 are the same, even though the states aren't.

MinDFSM

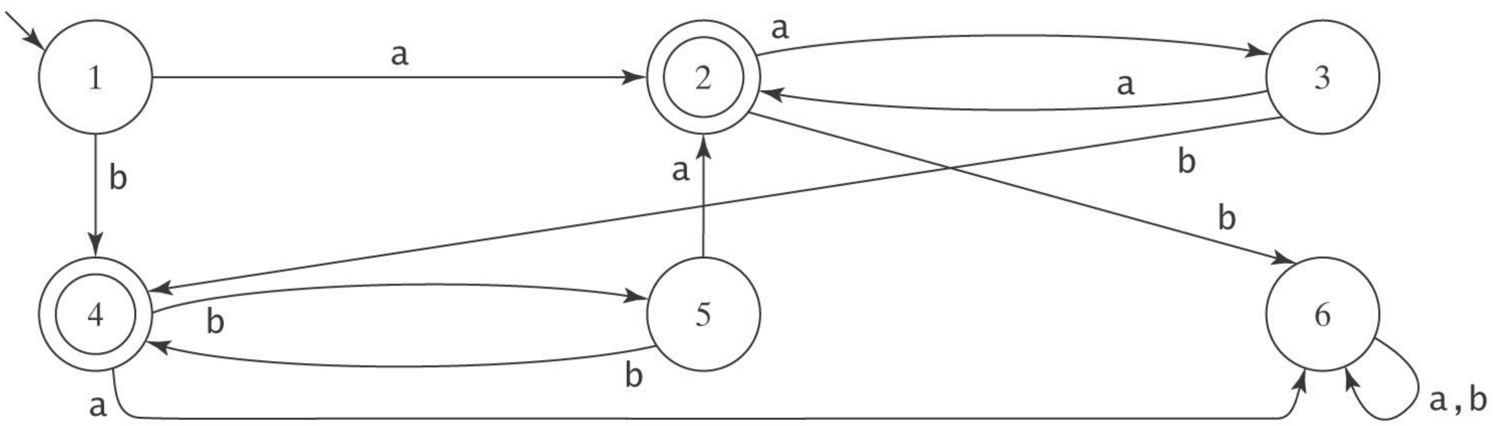
MinDFSM(*M*: DFSM) =

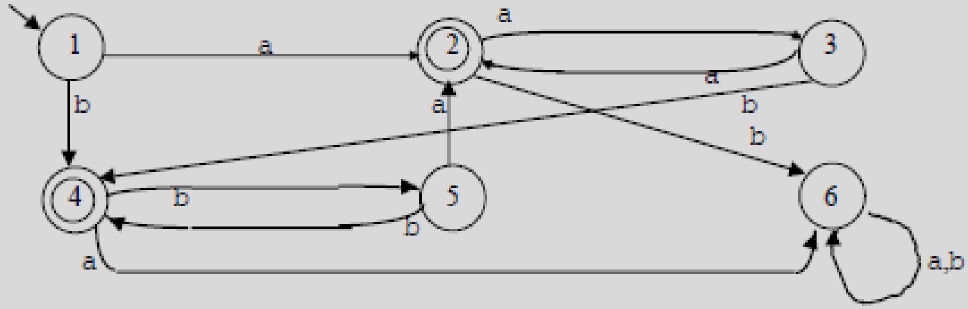
1. *classes* := {*A*, *K*-*A*};
2. Repeat until no changes are made
   - 2.1. *newclasses* := $\varnothing$;
   - 2.2. For each equivalence class *e* in *classes*, if *e* contains more than one state do
     - For each state *q* in *e* do
       - For each character *c* in Σ do
         - Determine which element of *classes* *q* goes to if *c* is read
     - If there are any two states *p* and *q* that need to be split, split them. Create as many new equivalence classes as are necessary. Insert those classes into *newclasses*.
     - If there are no states whose behavior differs, no splitting is necessary. Insert *e* into *newclasses*.
   - 2.3. *classes* := *newclasses*;
3. Return $M^* = (classes, \Sigma, \delta, [s_M], \{[q]\text{: the elements of } q \text{ are in } A_M]\})$, where δ_{M^*} is constructed as follows:

   if $\delta_M(q, c) = p$, then $\delta_{M^*}([q], c) = [p]$

MinDFSM: Example

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





We will show the operation of *minDFSM* at each step:

Initially, *classes* = {[2, 4], [1, 3, 5, 6]}.

At step 1:

((2, a), [1, 3, 5, 6])	((4, a), [1, 3, 5, 6])		No splitting required here.
((2, b), [1, 3, 5, 6])	((4, b), [1, 3, 5, 6])		

((1, a), [2, 4])	((3, a), [2, 4])	((5, a), [2, 4])	((6, a), [1, 3, 5, 6])
((1, b), [2, 4])	((3, b), [2, 4])	((5, b), [2, 4])	((6, b), [1, 3, 5, 6])

There are two different patterns, so we must split into two classes, [1, 3, 5] and [6]. Note that, although [6] has the same behavior as [2, 4] after reading a single character, it cannot be combined with [2, 4] because they do not share behavior after reading no characters.

Classes = {[2, 4], [1, 3, 5], [6]}.

At step 2:

((2, a), [1, 3, 5])	((4, a), [6])		These two must be split.
((2, b), [6])	((4, b), [1, 3, 5])		

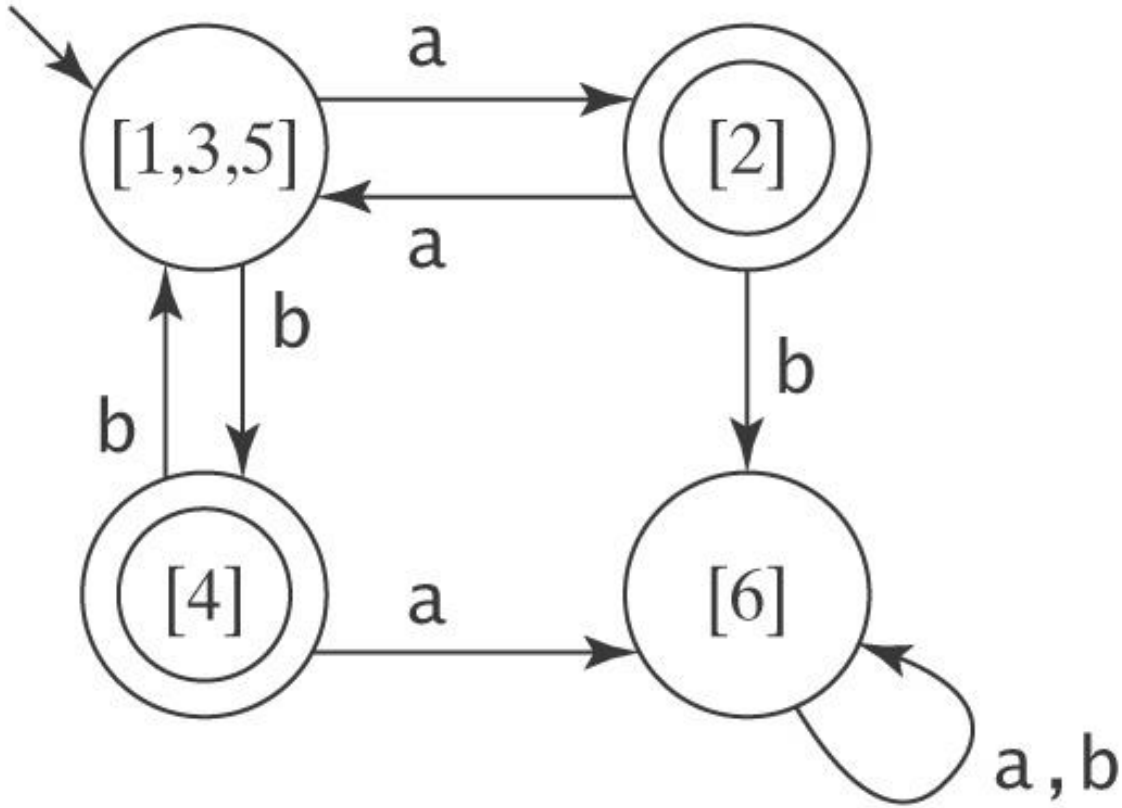
((1, a), [2, 4])	((3, a), [2, 4])	((5, a), [2, 4])	No splitting required here.
((1, b), [2, 4])	((3, b), [2, 4])	((5, b), [2, 4])	

Classes = {[2], [4], [1, 3, 5], [6]}.

At step 3:

((1, a), [2])	((3, a), [2])	((5, a), [2])	No splitting required here.
((1, b), [4])	((3, b), [4])	((5, b), [4])	

MinDFSM: Example



PROGRAMMING FSMs
State Minimisation

Theorem: Let L be a regular language and let M be a DFSM that accepts L. The number of states in M is greater than or equal to the number of equivalence classes of $\approx_L$.

Proof: Suppose that the number of states in M were less than the number of equivalence classes of $\approx_L$.

Then, by the pigeonhole principle, there must be at least one state q that contains strings from at least two equivalence classes of $\approx_L$.

But then M's future behavior on those strings will be identical, which is not consistent with the fact that they are in different equivalence classes of $\approx_L$.

PROGRAMMING FSMs
State Minimisation

***Theorem*:** Let L be a regular language over some alphabet Σ. Then there is a DFSM M that accepts L and that has precisely n states where n is the number of equivalence classes of $\approx_L$. Any other FSM that accepts L must either have more states than M or it must be equivalent to M except for state names.

Proof: in your textbook, but please, look at it!

FINITE STATE MACHINES
Definition

- A Finite State Machine M is a quintuple

$M = (K, \Sigma, \delta, s, A)$, where:

  - K is a finite set of states
  - Σ is an alphabet
  - δ is the transition function from $(K \times \Sigma)$ to K
  - $s \in K$ is the initial state, and
  - $A \subseteq K$ is the set of accepting states

NONDETERMINISTIC FSMs
Definition

- A Nondeterministic Finite State Machine M is a quintuple

  $M = (K, \Sigma, \Delta, s, A)$, where:

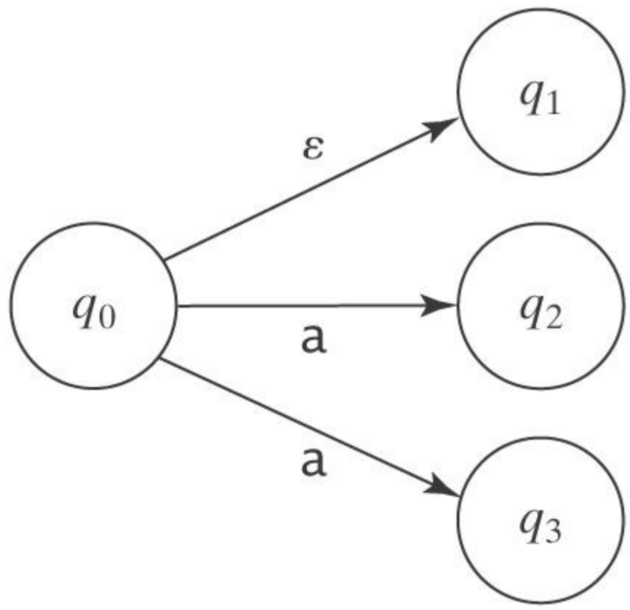
  - K is a finite set of states
  - Σ is an alphabet
  - Δ is the transition relation. It is a finite subset of $(K \times (\Sigma \cup \{\varepsilon\}) \times K$
  - $s \in K$ is the initial state, and
  - $A \subseteq K$ is the set of accepting states

NONDETERMINISTIC FSMs

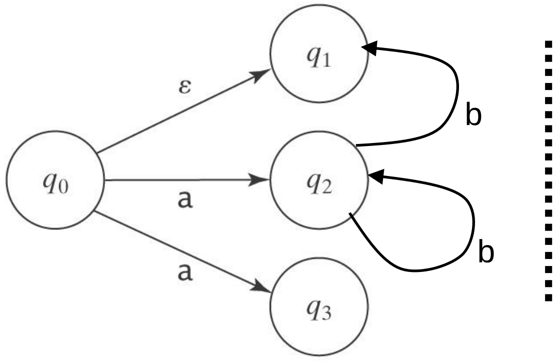
- A Finite State Machine *M* accepts a string *w* iff there exists <u>*some path*</u> along which *w* drives *M* to some element of *A*.

- The language accepted by *M*, denoted *L*(*M*), is the set of all strings accepted by *M*.

NONDETERMINISTIC FSMs
Sources of Nondeterminism

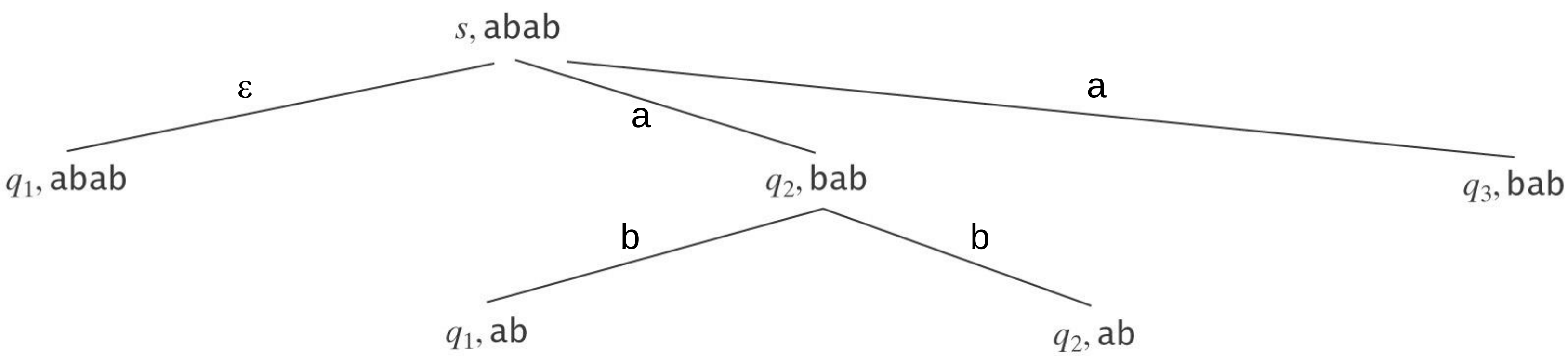


NONDETERMINISTIC FSMs
Analysing Nondeterminism



Two approaches:

- Explore a search tree:



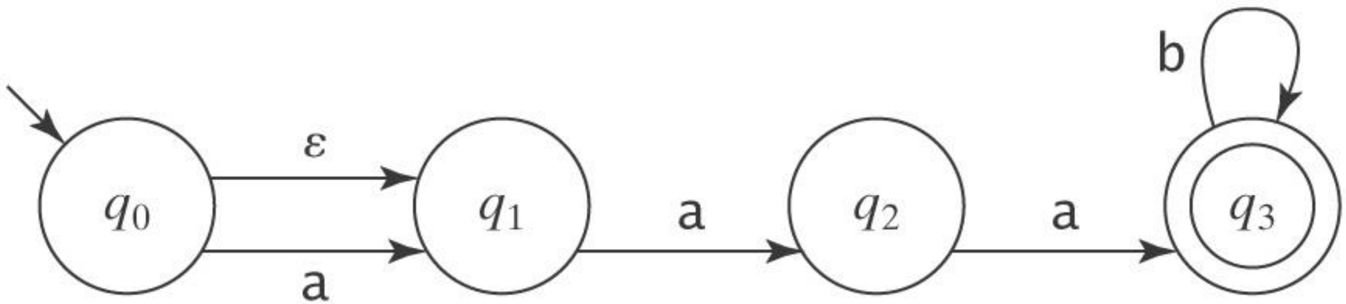
- Follow all paths in parallel

DFSM VS NDFSM
Key Differences

- NDFSM may enter a configuration in which input symbols left but no move available
  - NDFSM will simply halt without accepting
- NDFSM may enter a configuration from which two or more competing transitions are available
  - ε-transition
  - More than one transition for a single input character

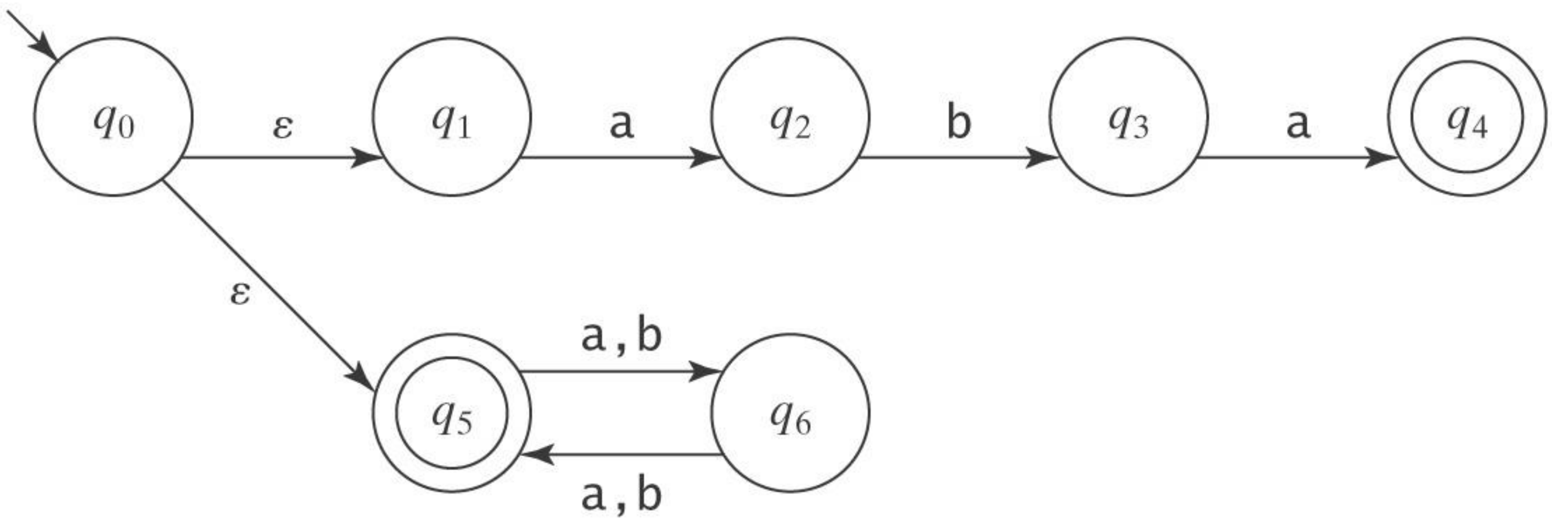
NONDETERMINISTIC FSMs
Example

- $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : w$ is made up of an optional a followed by aa followed by zero or more b's$\}$.



NONDETERMINISTIC FSMs
Another example

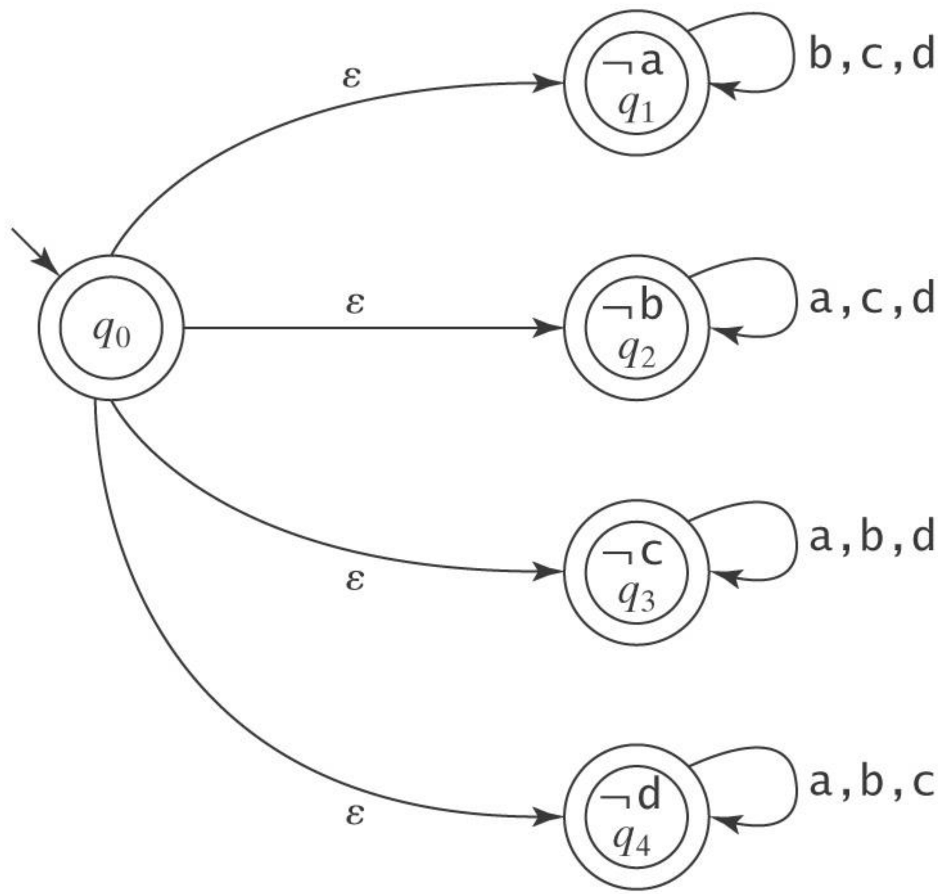
- $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : w = \mathtt{aba} \text{ or } |w| \text{ is even}\}$.



NONDETERMINISTIC FSMs
Back to $L_{Missing}$

- Let Σ = {`a`, `b`, `c`, `d`}. Let $L_{Missing}$ = {*w* : there is a symbol $a_i \in \Sigma$ not appearing in *w*}

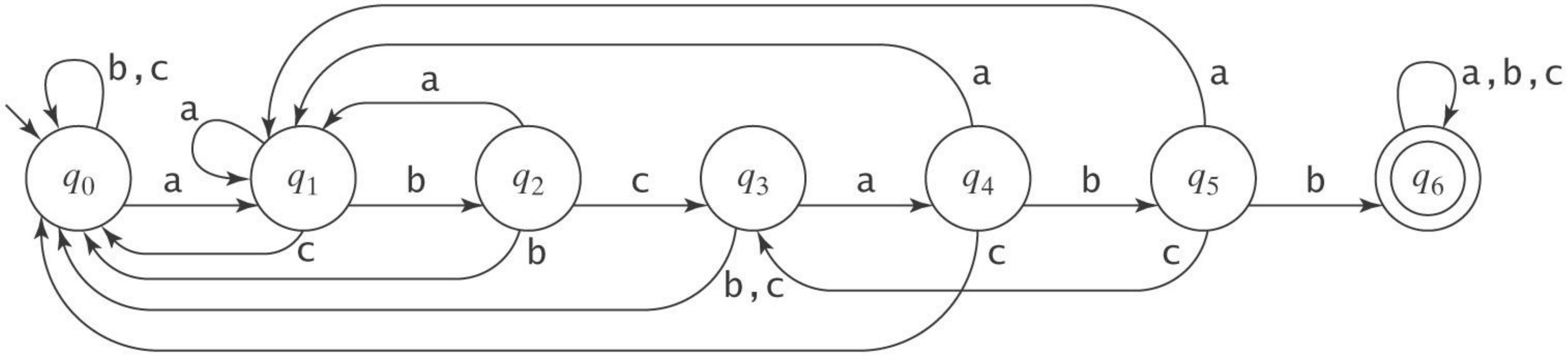
NONDETERMINISTIC FSMs
Back to $L_{Missing}$



NONDETERMINISTIC FSMs
Pattern Matching

$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* : \exists x, y \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \ (w = x \ \mathtt{abcabb} \ y)\}.$

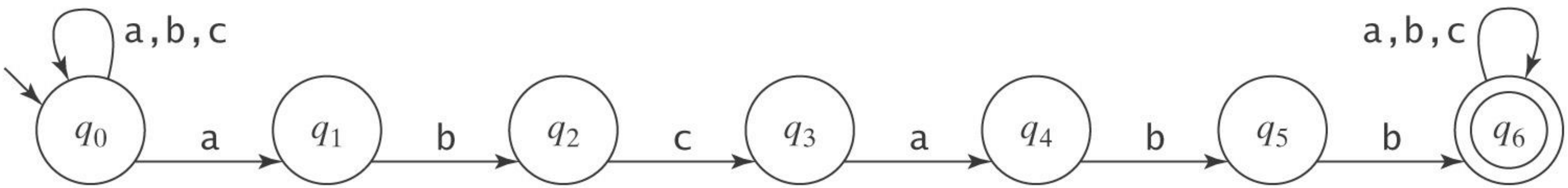
A DFSM:



NONDETERMINISTIC FSMs
Pattern Matching

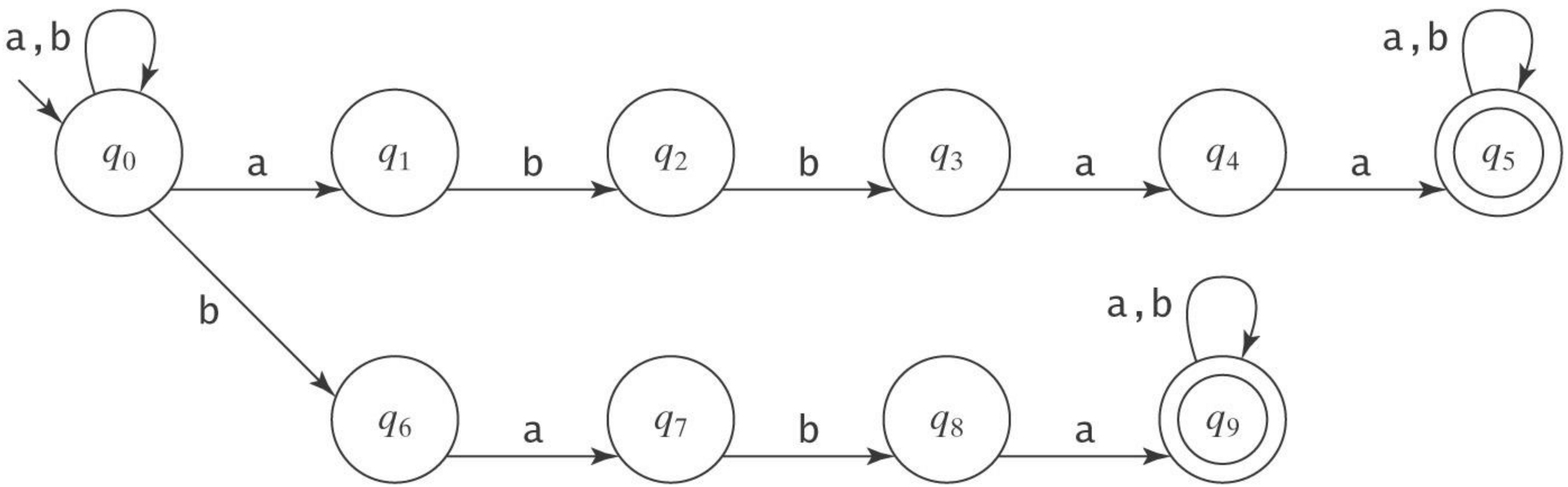
$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* : \exists x, y \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \ (w = x\ \mathtt{abcabb}\ y)\}.$

An NDFSM:



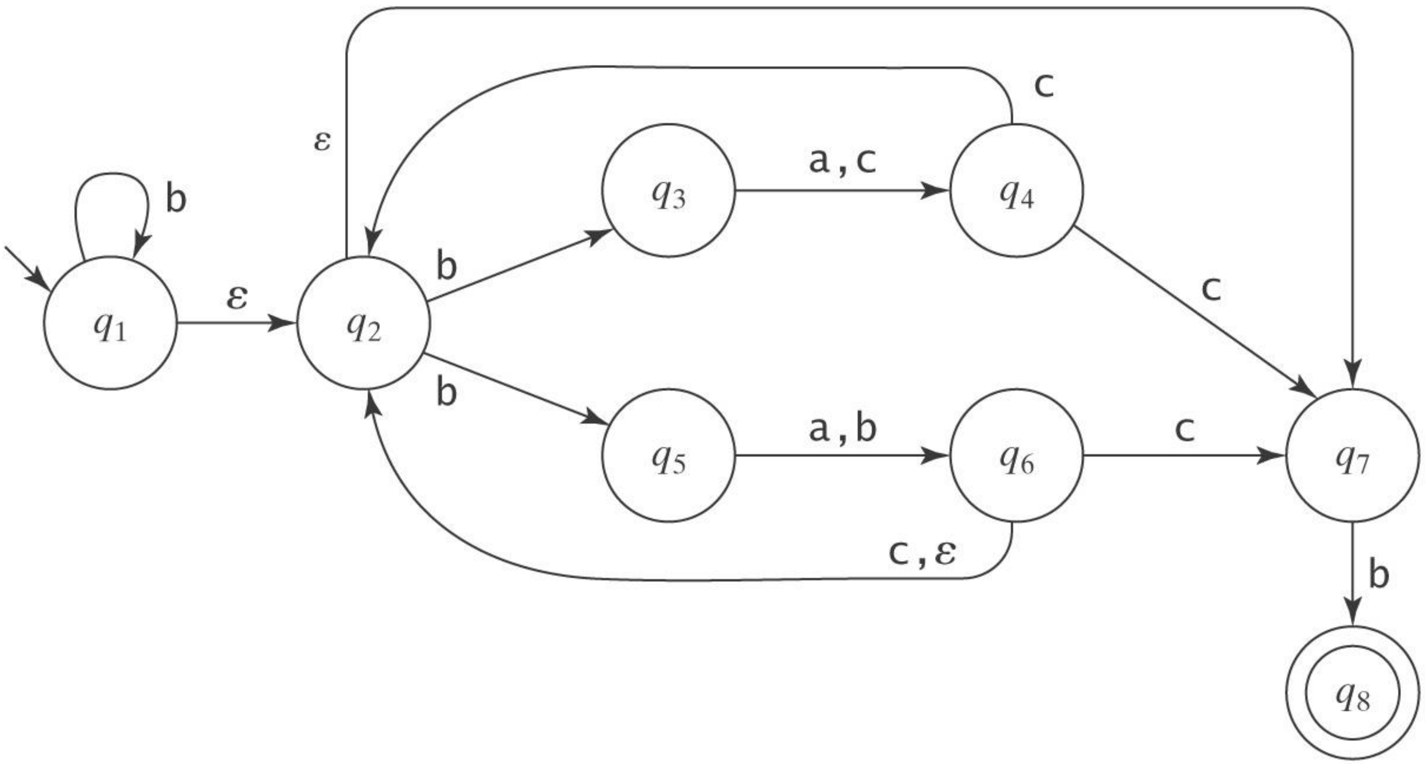
NONDETERMINISTIC FSMs
Pattern Matching

$$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* : \exists x, y \in \{\mathtt{a}, \mathtt{b}\}^* ((w = x\,\mathtt{abbaa}\,y) \lor (w = x\,\mathtt{baba}\,y))\}$$



NONDETERMINISTIC FSMs

$$b^* (b(a \cup c)c \cup b(a \cup b)\ (c \cup \varepsilon))^* b$$



NONDETERMINISTIC FSMs
Dealing with ε transitions

$eps(q) = \{p \in K : (q, w) \mid\!-^*_M (p, w)\}$.

$eps(q)$ is the closure of $\{q\}$ under the relation
$\{(p, r) :$ there is a transition $(p, \varepsilon, r) \in \Delta\}$.

How shall we compute $eps(q)$?

NONDETERMINISTIC FSMs
Dealing with ε transitions

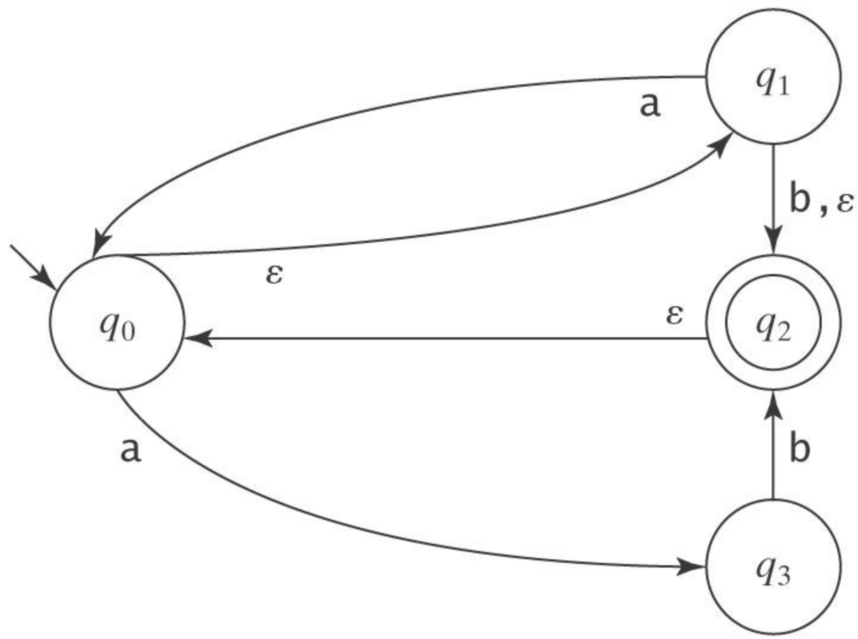
eps(*q*: state) =

result = {*q*}.

While there exists some $p \in$ *result* and
some $r \notin$ *result* and
some transition $(p, \varepsilon, r) \in \Delta$ do:
Insert *r* into *result.*

Return *result.*

NONDETERMINISTIC FSMs
Example of *eps*



$eps(q_0) =$
$eps(q_1) =$
$eps(q_2) =$
$eps(q_3) =$

NONDETERMINISTISM vs. DETERMINISM

Clearly: {Languages accepted by a DFSM} $\subseteq$
{Languages accepted by a NDFSM}

More interestingly:

***Theorem**:*

For each NDFSM, there is an equivalent DFSM.

NONDETERMINISTISM vs. DETERMINISM

***Proof*:** By construction:

Given a NDFSM $M = (K, \Sigma, \Delta, s, A)$,
we construct $M' = (K', \Sigma, \delta', s', A')$, where

$$K' = \mathcal{P}(K)$$
$$s' = eps(s)$$
$$A' = \{Q \subseteq K : Q \cap A \neq \varnothing\}$$

$$\delta'(Q, a) = \bigcup\{eps(p) : p \in K \text{ and } (q, a, p) \in \Delta \text{ for some } q \in Q\}$$

NONDETERMINISTISM vs. DETERMINISM
Algorithm

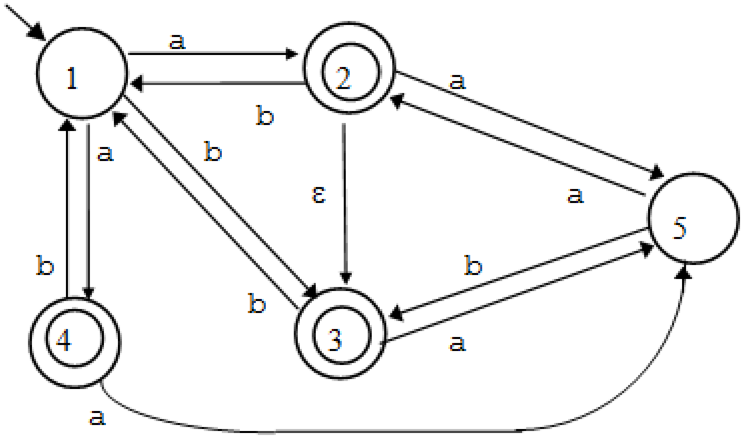
1. Compute the $eps(q)$'s.
2. Set $s' = eps(s)$.
3. Compute δ'.
4. Compute K' = a subset of $\mathcal{P}(K)$.
5. Compute $A' = \{Q \in K' : Q \cap A \neq \emptyset\}$.

NONDETERMINISTISM vs. DETERMINISM
Algorithm

ndfsmtodfsm(M: NDFSM) =

1. For each state q in K_M do:
    - 1.1 Compute $eps(q)$.
2. $s' = eps(s)$
3. Compute δ':
    - 3.1 *active-states* = $\{s'\}$.
    - 3.2 $\delta' = \varnothing$.
    - 3.3 While there exists some element Q of *active-states* for which δ' has not yet been computed do:
        - For each character c in Σ_M do:
            - *new-state* = $\varnothing$.
            - For each state q in Q do:
                - For each state p such that $(q, c, p) \in \Delta$ do:
            - *new-state* = *new-state* $\cup$ $eps(p)$.
            - Add the transition (Q, c, *new-state*) to δ'.
            - If *new-state* $\notin$ *active-states* then insert it.
4. K' = *active-states.*
5. $A' = \{Q \in K' : Q \cap A \neq \varnothing\}$.

NONDETERMINISTISM vs. DETERMINISM
Algorithm: example



1. Compute $eps(q)$ for each state q in K_M:

$eps(1) = \{1\}$
$eps(2) = \{2, 3\}$
$eps(3) = \{3\}$
$eps(4) = \{4\}$
$eps(5) = \{5\}$
$eps(6) = \{6\}$

2. Start state

$s' = eps(s) = eps(1) = \{1\}$.

NONDETERMINISTISM vs. DETERMINISM
Algorithm: example

3. Compute δ'

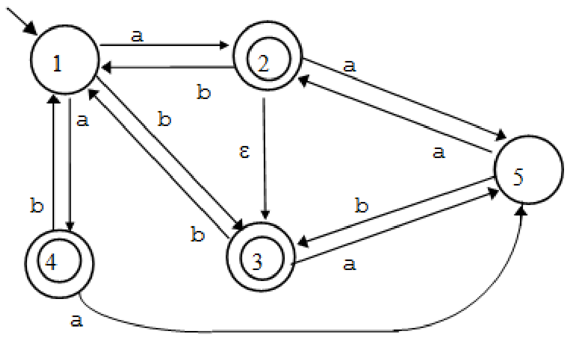
Active states: {{1}}. Consider {1}
eps ({1}, a) = {2, 3, 4}
eps ({1}, b) = {3}

Active states: {{1},{2,3,4},{3}}. Consider {2,3,4}
eps ({2, 3, 4}, a) = {5}
eps ({2, 3, 4}, b) = {1}

Active states: {{1},{2,3,4},{3},{5}}. Consider {3}
eps ({3}, a) = {5}
eps ({3}, b) = {1}

Active states: {{1},{2,3,4},{3},{5}}. Consider {5}
eps ({5}, a) = {2, 3}
eps ({5}, b) = {3}

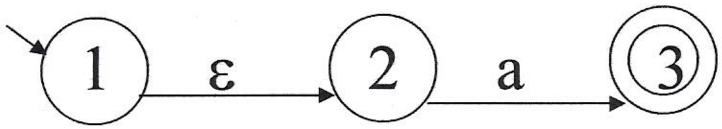
Active states: {{1},{2,3,4},{3},{5},{2,3}}. Consider {2,3}
eps ({2, 3}, a) = {5}
eps ({2, 3}, b) = {1}



4. K′ = **{{1},{2,3,4},{3},{5},{2,3}}**

5. **A**′ = {{2,3,4},{2,3},{3}}

NONDETERMINISTISM vs. DETERMINISM
The ‘real’ meaning

Let M = 

Is M deterministic?

An FSM is ***deterministic***, in the most general definition of determinism, if, for each input and state, there is at most one possible transition.

- DFSMs are always deterministic. Why?
- NDFSMs can be deterministic (even with ε-transitions and implicit dead states), but the formalism allows nondeterminism, in general.
- Determinism implies uniquely defined machine behavior.

SUMMARY

- A FSM M is a quintuple $\boldsymbol{M = (K, \Sigma, \delta, s, A)}$
- Every DFSM *M*, on input *s*, halts in |*s*| steps.
- A language accepted by some DFSM is regular.
- Given any regular language *L*, there exists a minimal DFSM *M* that accepts *L*.
  - *M* is unique up to the naming of its states.
- Given any DFSM *M*, there exists an algorithm *minDFSM* that constructs a minimal DFSM that also accepts *L*(*M*).
- For each NDFSM, there is an equivalent DFSM.

References

- **Automata, Computability and Complexity. Theory and Applications**

  - By Elaine Rich

- Chapter 5:
  - Page : 54~79, 82~94.