Chapter 5

Finite Automata

5.1 Finite State Automata

- Capable of recognizing numerous symbol patterns, the class of regular languages
- Suitable for pattern-recognition type applications, such as the *lexical analyzer* of a compiler
- An abstract (computing) machine *M*, which is *implementation independent*, can be used to determine the acceptability (the outputs) of input strings (which make up the language of *M*)

Lexical Analyzer

- Recognizes occurrences of (valid/acceptable) strings concisely
- Use a (state) transition diagram for producing lexical analysis routines, e.g., Figure 1 (next page) ▶
- Use a transition table whose entries provide a summary of a corresponding transition diagram, which consists of rows (representing *states*), columns (representing symbols) and EOS (End_of_string)
  - Entries of a transition table contain the values "accept", "error", next states. e.g., Figure 3 ▶
- Can be encoded in a program segment, e.g., Figure 2 ▶

Transition Diagram and Table

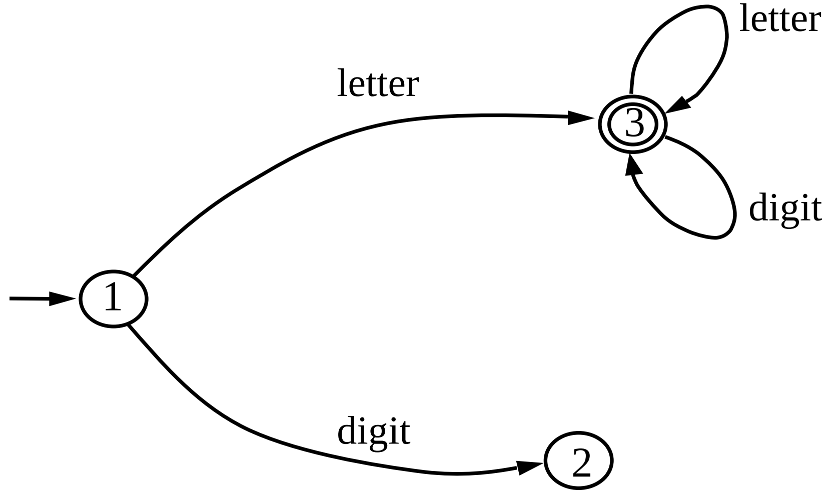


Figure 1. A transition diagram representing the syntax of a *variable name*

	letter	digit	EOS
1	3	2	error
2	error	error	error
3	3	3	accept

Figure 2. A transition table constructed from the transition diagram of Figure 1

Instruction Sequence

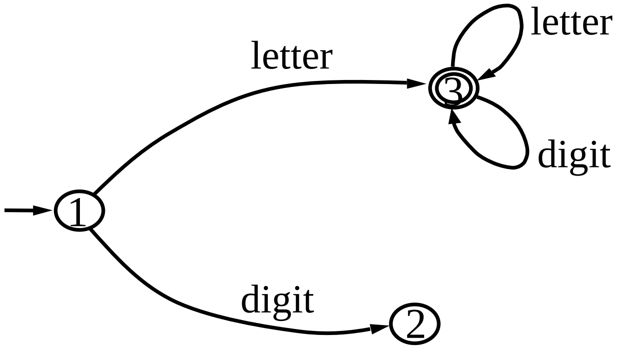
	letter	digit	EOS
1	3	2	error
2	error	error	error
3	3	3	accept

```
State := 1;
Read the next symbol from input;
While not end-of-string do
        Case State of
        1: If the current symbol is a letter then State := 3,
           else if the current symbol is a digit  then State := 2,
           else exit to error routine;
        2: Exit to error routine;
        3: If the current symbol is a letter then State := 3,
           else if the current symbol is a digit then State := 3,
           else exit to error routine;
        Read the next symbol from the input;
End while;
If State not 3 then exit to error routine;
```

Figure 3. An instruction sequence suggested by the transition diagram of Figure 1

5.2 Deterministic Finite Automaton

- DFA (Deterministic Finite Automaton) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where
  1) Q is a finite set of states
  2) Σ is a finite set of (machine) alphabet
  3) δ is a transitive function from $Q \times \Sigma$ to Q, i.e.,
     $\delta: Q \times \Sigma \to Q$
  4) $q_0 \in Q$, is the start state
  5) $F \subseteq Q$, is the set of final (accepting) states



Transition Diagram

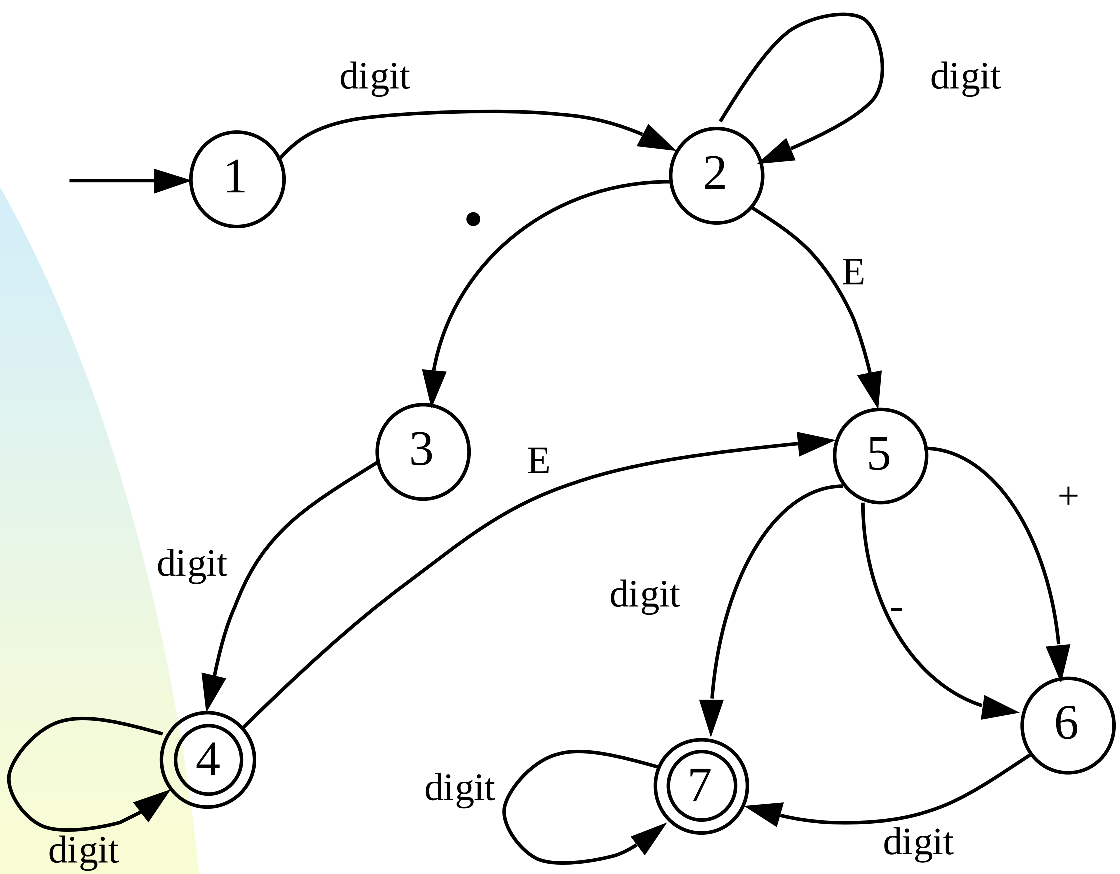


Figure 5. A transition diagram representing the syntax of a *real number*

Transition Table

	digit	•	E	+	-	EOS
1	2	error	error	error	error	error
2	2	3	5	error	error	error
3	4	error	error	error	error	error
4	4	error	5	error	error	accept
5	7	error	error	6	6	error
6	7	error	error	error	error	error
7	7	error	error	error	error	accept

Table 1. A transition table constructed from the transition diagram of the previous figure

Deterministic Finite Automaton

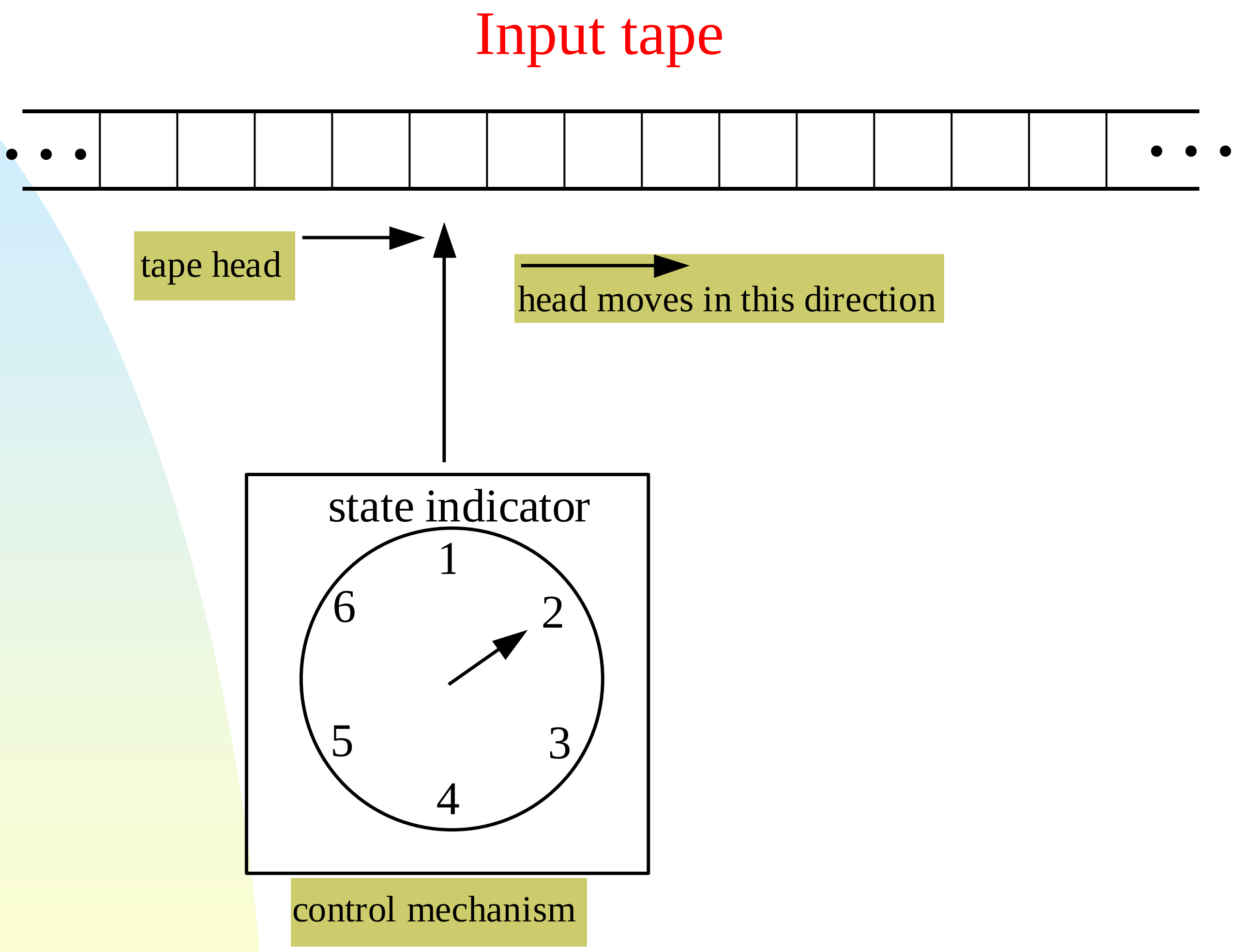


Figure 6. A representation of a deterministic finite automaton

Computation in DFA

M: $Q = \{q_0, q_1\}$ $\delta(q_0, a) = q_1$
$\Sigma = \{a, b\}$ $\delta(q_0, b) = q_0$
$F = \{q_1\}$ $\delta(q_1, a) = q_1$
$\delta(q_1, b) = q_0$

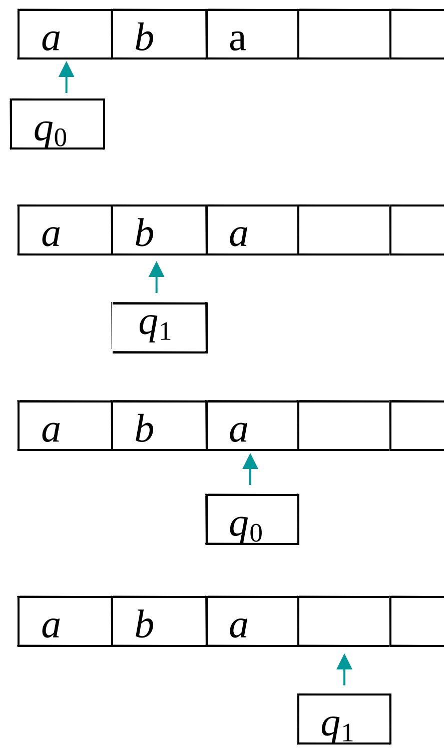


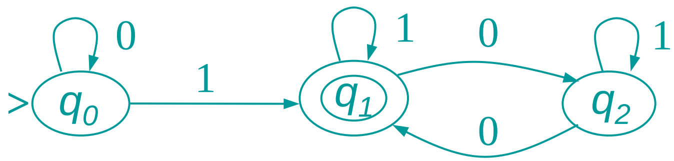
Figure 5.2 Computation in a DFA

State Diagrams

- Defn 5.3.1. The state diagram of a DFA $M = (Q, \Sigma, \delta, q_0, F)$ is a labeled graph G defined by the following:
  i. For each node $N \in G$, $N \in Q$
  ii. For each arc $E \in G$, label(E) $\in \Sigma$
  iii. q_0 is depicted
  iv. For each $f \in F$, f is depicted
  v. For each $\delta(q_i, a) = q_j$, $\exists\ E(q_i, q_j)$ and label(E) = a
     - a transition is represented by an *arc*
  vi. For each $q_i \in Q$ & $a \in \Sigma$, $\exists!\ E(q_i, q_j)$ & label(E) = a, where $q_j \in Q$

- Example: Construct the state diagram of $L(M)$ for DFA M:

  $L(M) = \{w \mid w$ contains at least one 1 and an even number of 0 follow the first 1$\}$



Definitions

- Defn 5.2.2. Let $m = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The language of m, denoted $L(m)$, is the set of strings in Σ^* accepted by m.

- Defn 5.2.3 (Machine configuration). The function $\vdash_M$ ("yields") on $Q \times \Sigma^+$ is defined by

$$[q_i, aw] \vdash_M [\delta(q_i, a), w]$$

where $a \in \Sigma$, $w \in \Sigma^*$, and $\delta \in M$. Also,

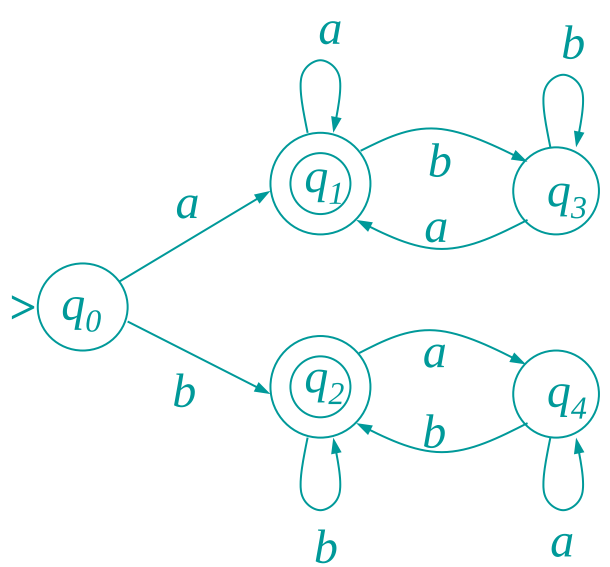
$$[q_i, u] \overset{*}{\vdash}_M [q_j, v]$$

denotes a sequence of 0 or more transitions.

- Defn. 5.2.4. The function $\hat{\delta}$ ($\overset{*}{\vdash}_M$): $Q \times \Sigma^* \rightarrow Q$ of a DFA is called the extended transition function such that
  - $\hat{\delta}(q_i, ua) = \delta(\hat{\delta}(q_i, u), a))$

State Diagrams (Continued)

- <u>Example</u>: Give the state diagram of a DFA *M* such that *M* accepts all strings that <u>start</u> and <u>end</u> with *a*, or that <u>start</u> and <u>end</u> with *b*, i.e., *M* accepts strings that start and end with the same symbol, over the alphabet $\Sigma = \{a, b\}$



- Note: Interchanging the accepting states and non-accepting states of a state diagram for the DFA *M* yields the DFA *M*' that accepts *all* the strings over the same alphabet that are <u>not</u> accepted by *M*.

DFA and State Diagrams

- Construct a DFA that accepts one of the following languages over the alphabet { 0, 1 }

  i. "The set of all strings ending in 00".

  ii. "The set of all strings when interpreted as a binary integer, is a multiple of 5, e.g., strings 101, 1010, and 1111 are in the language, whereas 10, 100, and 111 are not".

State Diagrams

- Theorem 5.3.3. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then $M' = (Q, \Sigma, \delta, q_0, Q - F)$ is a DFA w/ $L(M') = \Sigma^* - L(M)$

  Proof: Let $w \in \Sigma^*$ and $\hat{\delta}$ be the extended transition function constructed form δ. For each $w \in L(M)$, $\hat{\delta}(q_0, w) \in F$. Thus, $w \notin L(M')$. Conversely, if $w \notin L(M)$, then $\hat{\delta}(q_0, w) \in Q - F$ and thus $w \in L(M')$.

  - Examples 5.3.7 and 5.3.8 (page 157)

- An incompletely specified DFA M is a machine defined by a partial function from $Q \times \Sigma$ to Q such that M halts as soon as it is possible to determine that an input string is (not) acceptable.

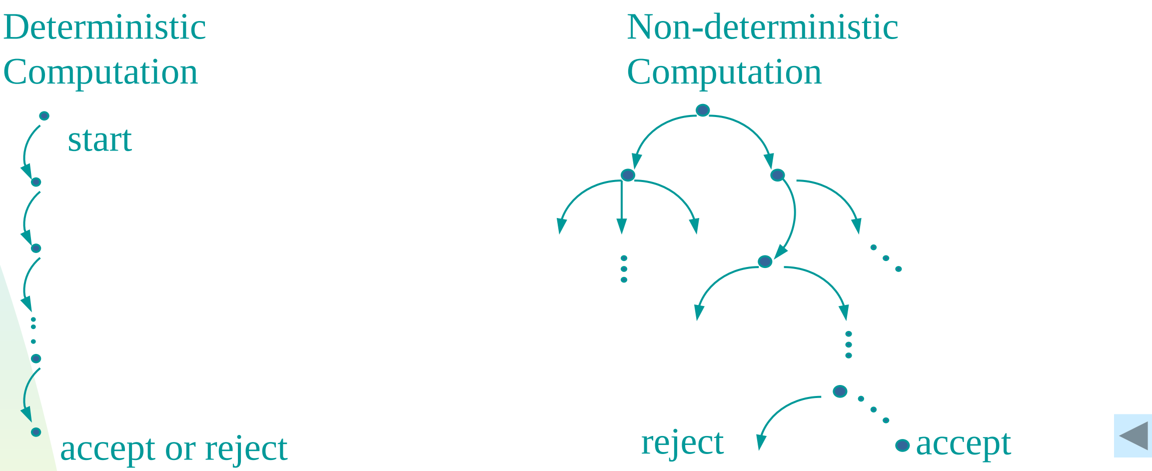
  - M can be transformed into an equivalent DFA by adding a non-accepting "error" state and transitions out of all the states in M with other input symbols to the "error" state.

5.4. Non-deterministic Finite Automata(NFA)

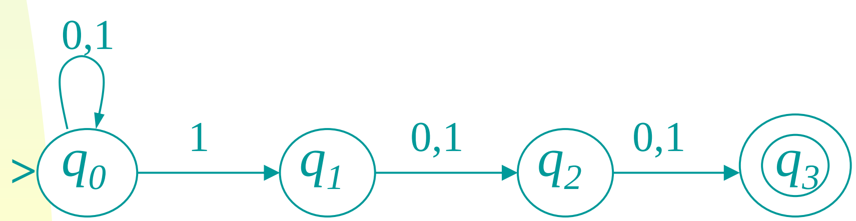
- Relaxes the restriction that all the outgoing arcs of a state must be labeled with *distinct symbols* as in DFAs
- The transition to be executed at a *given state* can be *uncertain*, i.e., > 1 possible transitions, or no applicable transition.
- Applicable for applications that require *backtracking* technique.
- Defn 5.4.1 A non-deterministic finite automaton is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where
  1. Q is a finite set of *states*
  2. Σ is a finite set of symbols, called the *alphabet*
  3. $q_0 \in Q$ the *start state*
  4. $F \subseteq Q$, the set of *final (accepting*) states
  5. δ is a total function from $(Q \times \Sigma)$ to $\wp$**(Q)**, known as the *transition function*

NFA

- Every DFA is an NFA, and vice versa
  - Hence, in an NFA, it is possible to have $(p, a, q_1) \in \delta$ and $(p, a, q_2) \in \delta$, where $q_1 \neq q_2$



- Example. Consider the following state diagram of NFA *M*:

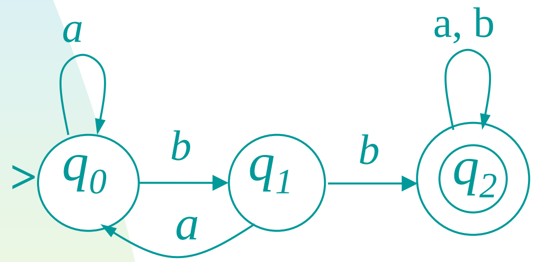


  - *M* stays in the start state until it "guesses" that it is three places from the end of the computation.

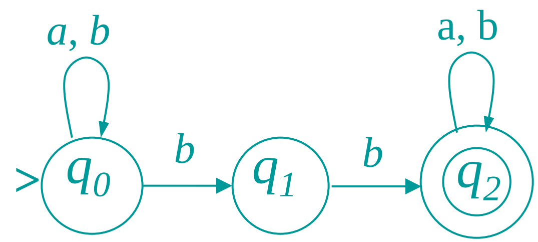
Advantages of NFAs over DFAs

- Sometimes DFAs have many more states, conceptually more complicated
- Understanding the functioning of the NFAs is much easier.
  - Example 5.4.2 M_1(DFA) and M_2(NFA) accept $(a \cup b)^*\ bb\ (a \cup b)^*$

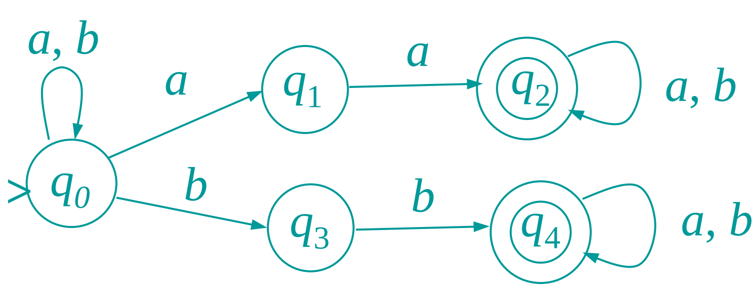
M_1:



M_2:



  - Example 5.4.3 An NFA accepts strings over { *a*, *b* } with substring *aa* or *bb*.



5.5 Lambda Transitions

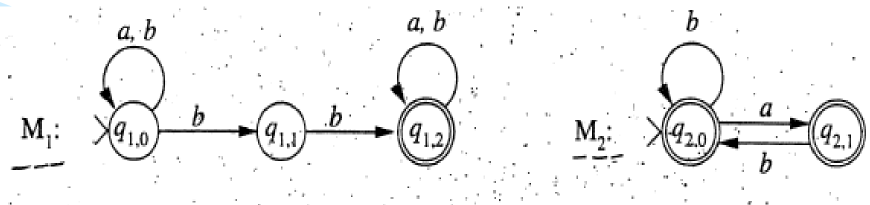
- A transition of any finite automata which shifts from one state to another without reading a symbol from the input tape is known as λ-transition
- λ-transition is labeled by λ on an *arc* in the state transition diagram
- λ-transition represent another form of *non-DFA computations*
- Provide a useful tool for designing finite automata to accept *complex languages*
- Defn. 5.5.1. An NFA with λ-transition, denoted *NFA-λ*, is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

  i) Q, Σ, q_0, and F are the same as in an NFA

  ii) $\delta: Q \times (\Sigma \cup \{ \lambda \}) \rightarrow \wp(Q)$

  - Example 5.5.1 ($\cup$) and compared with the equivalent DFA in Ex. 5.3.3 ▶
  - Example 5.5.2 ($\bullet$) and Example 5.5.3 (*) ▶

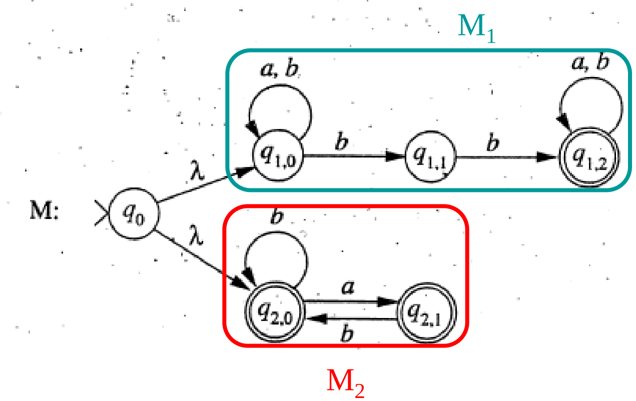
5.5 Lambda Transitions



that accept $(a \cup b)^*bb(a \cup b)^*$ and $(b \cup ab)^*(a \cup \lambda)$, respectively. Composite machines are built by appropriately combining the state diagrams of M_1 and M_2.

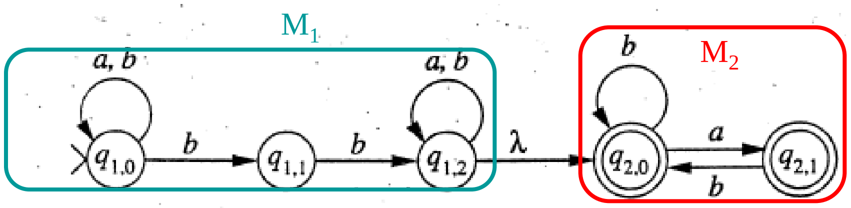
Example 5.5.1

The language of the NFA-λ M is $L(M_1) \cup L(M_2)$.

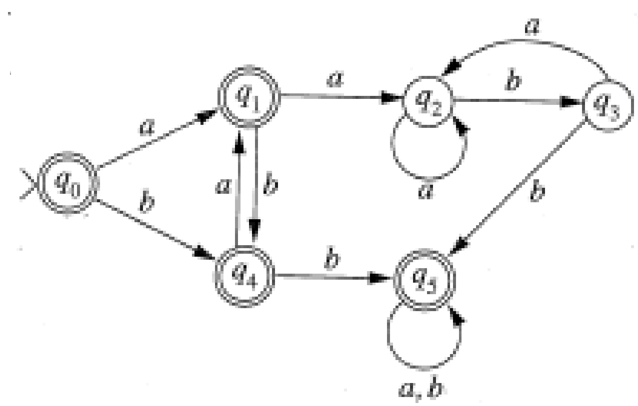


Example 5.5.2

An NFA-λ that accepts $L(M_1)L(M_2)$, the concatenation of the languages of M_1 and M_2, is constructed by joining the two machines with a lambda arc.



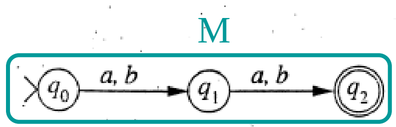
Example 5.3.3

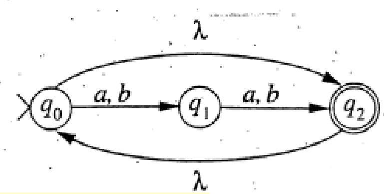


Example 5.5.3

Lambda transitions are used to construct an NFA-λ that accepts all strings of even length over $\{a, b\}$. First we build the state diagram for a machine that accepts strings of length two.

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





5.6. Removing Non-determinism

- Given any NFA(-λ), there is an equivalent DFA.
- Defn 5.6.1. The λ-closure of a state q_i, denoted λ-closure(q_i), is defined recursively by

  (i) Basis: $q_i \in \lambda$-closure(q_i)

  (ii) Recursion: let $q_j \in \lambda$-closure(q_i) and $q_k \in \delta(q_j, \lambda)$
  $\Rightarrow q_k \in \lambda$-closure($q_i$)

  (iii) Closure: each $q_j \in \lambda$-closure(q_i) is obtained by a number of applications of (ii)

- Defn 5.6.2. The input transition function t of an NFA-λ $M = (Q, \Sigma, \delta, q_0, F)$ is a function from $Q \times \Sigma \rightarrow \wp(Q)$ such that

$$t(q_i, a) = \underbrace{\bigcup_{q_j \in \lambda\text{-}closure(q_i)}}_{(1)} \overbrace{\lambda\text{-}closure}^{(3)}(\overbrace{\delta(q_j, a)}^{(2)})$$

  - t is used to construct an equivalent DFA

Removing Non-determinism

- Example: Consider the transition diagram in Fig. 5.3 on p. 171 to compute $t(q_1, a)$:

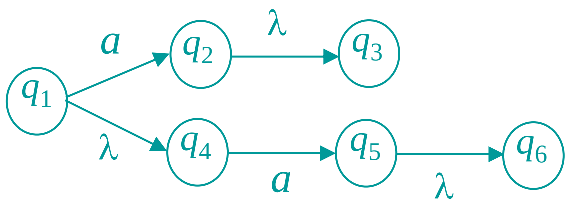
  λ-closure$(q_1) = \{ q_1, q_4 \}$

  $t(q_1, a) = \lambda\text{-closure}(\delta(q_1, a)) \cup \lambda\text{-closure}(\delta(q_4, a))$

  $= \lambda\text{-closure}(\{ q_2 \}) \cup \lambda\text{-closure}(\{ q_5 \})$

  $= \{ q_2, q_3 \} \cup \{ q_5, q_6 \}$

  $= \{ q_2, q_3, q_5, q_6 \}$



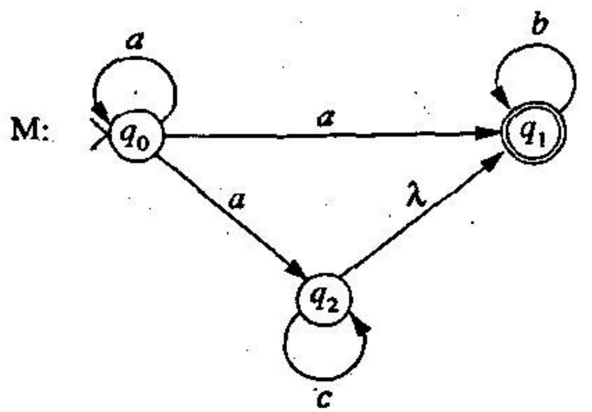
- Given $M = (Q, \Sigma, \delta, q_0, F)$, $t = \delta$ iff there is no λ-transition in δ

- Example 5.6.1.

- To remove the non-determinism in an NFA(-λ), an equivalent DFA simulates the exploration of all possible computations in the NFA (-λ)
  - the nodes of the DFA are sets of nodes from the NFA(-λ)
  - node $Y \subseteq Q$ in NFA(-λ) can be reached from node $X \subseteq Q$ in NFA(-λ) on 'a' if $\exists q \in Y$ and $\exists p \in X$ such that $\delta(p, a) \ni q$ in the DFA

Removing Non-determinism

- <u>Example 5.6.1.</u> Transition tables are given (below) for the *transition function* δ. Compute the *input transition function* *t* of the NFA-λ with state diagram *M*. The language of *M* is $a^+c^*b^*$



δ	a	b	c	λ
q_0	$\{q_0, q_1, q_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1\}$

t	*a*	*b*	*c*
q_0	$\{ q_0,q_1,q_2 \}$	$\{ \}$	$\{ \}$
q_1	$\{ \}$	$\{ q_1 \}$	$\{ \}$
q_2	$\{ \}$	$\{ q_1 \}$	$\{ q_1,q_2 \}$

DFA Equivalent to NFA-λ

- Algorithm 5.6.3. Construction of DM, a DFA Equivalent to NFA-λ

  Input: an NFA-λ $M = (Q, \Sigma, \delta, q_0, F)$, input transition function t of M

  1. Initialize Q' to $\{ \lambda\text{-closure}(q_0) \}$
  2. Repeat

     2.1. IF there is a node $X \in Q'$ and a symbol $a \in \Sigma$ with no arc leaving X labeled a, THEN

     2.1.1. Let $Y = \bigcup_{q_i \in X} t(q_i, a)$

     2.1.2. IF $Y \notin Q'$, THEN set $Q' = Q' \cup \{ Y \}$

     2.1.3. Add an arc from X to Y labeled a

     ELSE *done* := true

     UNTIL *done*
  3. the set of accepting states of DM is

     $F' = \{ X \in Q' \mid X \text{ contains } q_i \in F \}$

Removing Non-determinism

- Example. Consider the *t*-transition table for Example 5.6.1

t	a	b	c
q_0	$\{ q_0, q_1, q_2 \}$	$\{ \}$	$\{ \}$
q_1	$\{ \}$	$\{ q_1 \}$	$\{ \}$
q_2	$\{ \}$	$\{ q_1 \}$	$\{ q_1, q_2 \}$

δ'	a	b	c
$\{q_0\}$	$\{q_0, q_1, q_2\}$	ϕ	ϕ
$\{q_0, q_1, q_2\}^*$	$\{q_0, q_1, q_2\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_1\}^*$	ϕ	$\{q_1\}$	ϕ
$\{q_1, q_2\}^*$	ϕ	$\{q_1\}$	$\{q_1, q_2\}$
ϕ	ϕ	ϕ	ϕ

- Theorem 5.6.4. Let $w \in \Sigma^*$ and $Q_w = \{ q_{w_1}, \ldots, q_{w_j} \}$ be the set of states entered upon the completion of the processing of the string w in M. Processing w in DM terminates in state Q_w. (Prove by induction on $|w|$.)

Determinism and Non-determinism

- Corollary 5.6.5. The finite automata *M* and *DM* (as shown in Algorithm 5.6.3) are ≡.
- Example 5.6.2 and Example 5.6.3 show NFA ⇒ DFA
- (Transformation) Relationships between the classes of finite automata:

$$\begin{array}{ccc} \text{DFA} & \Leftarrow & \text{NFA-}\lambda \\ \subseteq & & \subseteq \\ & \text{NFA} & \end{array}$$