1 Abstract machines

- Basic components of a computer: (i) central processing unit; (ii) memory; (iii) input-output devices.
- We are concerned with abstract machines, i.e. mathematical descriptions of computing machinery.

2 Deterministic finite automata

- A **dfa** consists of a **reading head**, a **tape** and a **finite control**.
- The tape consists of squares which can hold a symbol.
- The finite control consists of a set of finite states and an indicator which shows the current state of the automaton.
- The reading head starts at the leftmost position on the tape. The automaton operates through discrete steps. At each step the reading head advances one square right reading the symbol in its starting position, and updating the state of the machine according to a **transition function** which maps pairs of symbols and states to states.

  Definition 2.1 (Deterministic finite automaton)
  A dfa is a quintuple $\langle Q, \Sigma, \delta, q_0, F\rangle$ where

  Q is a set of **states**,

  Σ is an **alphabet**,

  $q_0 \in Q$ is the **initial state**,

  $F \subseteq Q$ is the set of **final states**.

  and $\delta : Q \times \Sigma \mapsto Q$ is the **transition function**.

Exercise 2.2
Take the fa $M = \langle \{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_0\}\rangle$ where δ is,

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

Trace the processing of the string *aabba* by M.

- Here are some formal definitions concerning the computations of fa:

  As made clear by exercise 2.2, given any point during the computation of an fa, the state the fa will end up when it runs out of input is determined by the state at that point and the string yet to be read.

  Call (q, w) a **configuration** of an fa M, where q is the current state and w is the string on the tape including the current symbol and what lies to the right of it.

  For an fa $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ the relation **yields in one step**, defined over set of configurations and designated with $\models_M$, is such that,

  $$(q, w) \models_M (q', w') \text{ iff } w = \sigma w' \text{ for some } \sigma \in \Sigma \text{ and } \delta(q, \sigma) = q'$$

  The relation **yields**, designated as $\models_M^*$, is the reflexive transitive closure of $\models_M$,

  An fa $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ **accepts** a string $w \in \Sigma^*$ iff $(q_0, w) \models_M^* (q_n, \varepsilon)$ for some $q_n \in F$.

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

3 State diagrams for fa

- Finite automata can be conveniently represented by directed graphs called **state diagrams**. Let's draw a state diagram for the language accepted by the fa in Exercise 2.2.

Exercise 3.1
Draw the state diagrams of an accepting dfa for each language below:

1. $\{w \in \{a,b\}^* \mid \text{each } a \text{ is immediately followed by } b\}$.
2. $\{w \in \{a,b\}^* \mid w \text{ has } abab \text{ as a substring}\}$.
3. $\{w \in \{a,b\}^* \mid w \text{ does not contain three consecutive } b\text{'s}\}$.
4. $\{w \in \{a,b\}^* \mid w \text{ has both } ab \text{ and } ba \text{ as substrings}\}$.
5. $\{w \in \{a,b\}^* \mid w \text{ has odd number of } a\text{'s and an even number of } b\text{'s}\}$.
6. $\{w \in \{a,b\} \mid w \text{ has an even length and odd number of } b\text{'s}\}$.
7. $\{w \in \{a,b\} \mid w \text{ has at least three } a\text{'s and at least two } b\text{'s}\}$.
8. $\{w \in \{0,1\} \mid w \text{ starts with 0 and has odd length, or starts with 1 and has even length }\}$.
9. $\{w \in \{0,1\} \mid w \text{ starts and ends with the same symbol}\}$.
10. $\{w \in \{a,b\} \mid w \text{ does not begin with } aaa\}$.
11. $\{w \in \{a,b,c\} \mid w \text{ has an odd number of occurrences of } ab\}$.

□

Exercise 3.2
Write equivalent dfa's for the following regular expressions (reminder: $w^+ = ww^*$):

1. $(ab \cup aba)^*$
2. $a(abb) \cup b$
3. $a^+ \cup (ab)^+$
4. $(a \cup b^+)a^+b^+$
5. $(ab)^*ba$
6. $(ab)^*(ba)^*$
7. $aa(a \cup b)^+bb$
8. $((aa)^+bb)^*$
9. $(ab^*a)^*$

□

Exercise 3.3
Construct a deterministic finite automaton and write a regular expression for each of the following languages ($\Sigma = \{a,b\}$).

1. The set of strings that either begin or end (or both) with ab.
2. The set of strings where the number of a's ≥ 1 (modulo 3).[1]
3. The set of strings where every odd position is an a.
4. The set of strings with at least two a's and at most one b.
5. The set of strings not in a^*b^*.

□

---

4 Non-deterministic finite automata

- A **non-deterministic finite automaton** extends the notion of dfa in the following respects:

  i. δ is no longer required to be a function;

  ii. **empty transitions** are allowed, i.e. the finite control is allowed to advance the state without reading any symbol from the input tape;

  ii. transitions are not limited to symbols, an nfa can read $w \in \Sigma^*$.

Exercise 4.1
Provide an nfa for the languages 1, 2 and 6 of exercise 3.1.

[1] See the entry "modulo operation" in Wikipedia, in case you are not familiar with modular arithmetic.

- Finite automata M_1 and M_2 are **equivalent** if and only if $L(M_1) = L(M_2)$.

  Theorem 4.2
  For each non-deterministic finite automaton there exists an equivalent deterministic finite automaton.

- There exists an algorithm to construct a dfa from a given nfa. The basic logic of the algorithm is to treat *sets* of states of the nfa as states of the dfa. We will not study the algorithm in detail.

  Exercise 4.3
  Provide nfa's that accept the following languages:

  (a) $(ab)^*(ba)^* \cup aa^*$

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

  (c) $((a^* b^* a^*)^* b)^*$

  (d) $(ba \cup b^*) \cup (bb \cup a)^*$

5 Equivalence of FALs and regular languages

- We call a language $\mathscr{L}$ a **finite automaton language** (or fal), if and only if there exists an fa M such that $\mathscr{L} = L(M)$.

- Finite automata and regular expressions are two alternative ways of characterizing the same class of languages.

  Theorem 5.1 (Kleene)
  A set of strings is a finite automaton language if and only if it is a regular language.

- We will prove by induction only the *regular expression-to-finite automaton* side of the theorem:

  - We start by showing that the three basic regular expressions, $\emptyset$, ε and the unit language **a** for an arbitrary symbol a, have equivalent NFAs.
  - Then we show that finite automata are closed under union, concatenation and closure.

6 Properties of regular languages

- We have seen above that regular languages and FALs are closed under the operations of union, concatenation and closure. Now we look at intersection and complementation.

- If a language $L \subseteq \Sigma^*$ is regular,[2] then its complement, namely $\Sigma^* - L$ is also regular. How can we prove this? (Hint: make use of finite automata.)

- From the above result, it follows that regular languages are closed under intersection as well. Why? (Hint: DeMorgan!)

- These properties can be used to decide whether a given language is regular or not.

  Exercise 6.1
  Is the language L over $\{a,b\}$ whose strings has an even number of a's and odd length regular?

7 A Way to determine that a language is not regular

- Given a language L, we can sometimes tell that it is not a FAL (or a regular language) by using a simple theorem.

  Theorem 7.1 (Pumping Theorem for FALs)
  If L is an infinite FAL over an alphabet Σ, then there are strings $x,y,z \in \Sigma$, such that $y \neq \varepsilon$ and $xy^n z \in L$ for all $n \geq 0$.

  Example 7.2
  Let us take a language that we know to be a FAL, namely
  $\{w \in \{a,b\}^* \mid w \text{ has an odd number of } b\text{s}\}$ and observe how the Pumping Theorem applies.

- Pumping Theorem can *sometimes* be used to prove that a language is not a FAL.

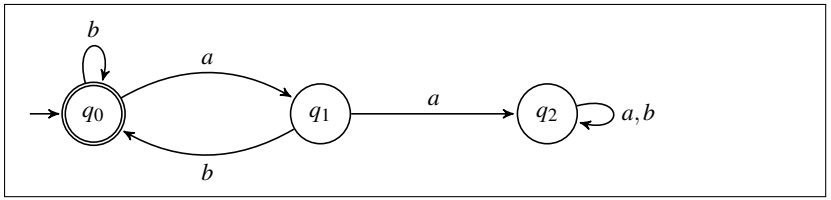
  Exercise 7.3
  Prove that $\{w \in \{a,b\}^* \mid w = a^n b^n \text{ for } n \geq 0\}$ is not a FAL.

[2] Note the typo in Partee et al. p. 477, which says $L \in \Sigma^*$ instead of $L \subseteq \Sigma^*$.

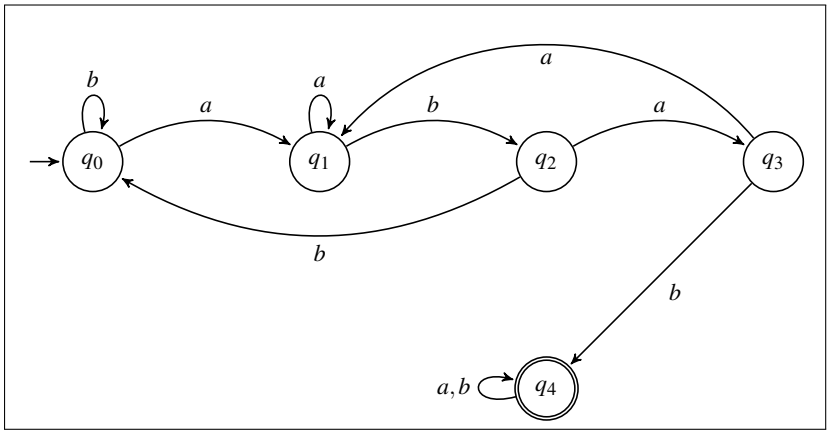
A Answers for selected exercises

3.1

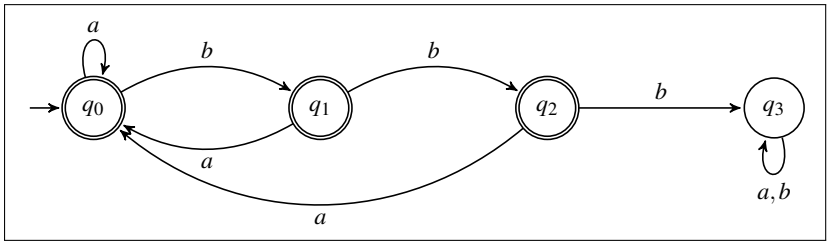
1.



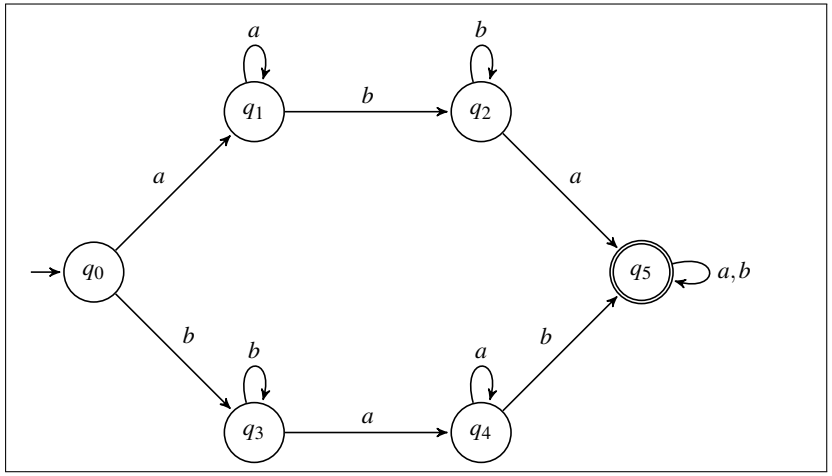
2.



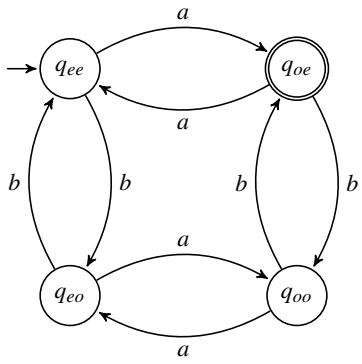
3.



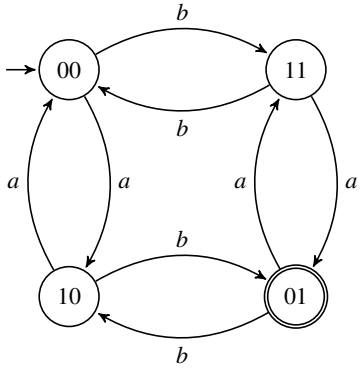
4.



5. Let's agree on a simple convention: 'ee' means the number of a's and b's are both even; 'oe' means odd number of a's and even number of b's; 'eo' means even a's and odd b's; 'oo' means both are odd. These four situations exhaust the logically possible states that a dfa can be in while processing a string of a's and b's. We will construct our automaton on the basis of this observation. We name our states according to our convention. Therefore the state named 'oe' – odd a even b – will be an accepting (=final) state. **Please observe that the names we give to our states is just to make the machine more understandable for humans. The names do not mean anything to the machine. Any other naming would be equally good, provided that the state transition function stays the same.**



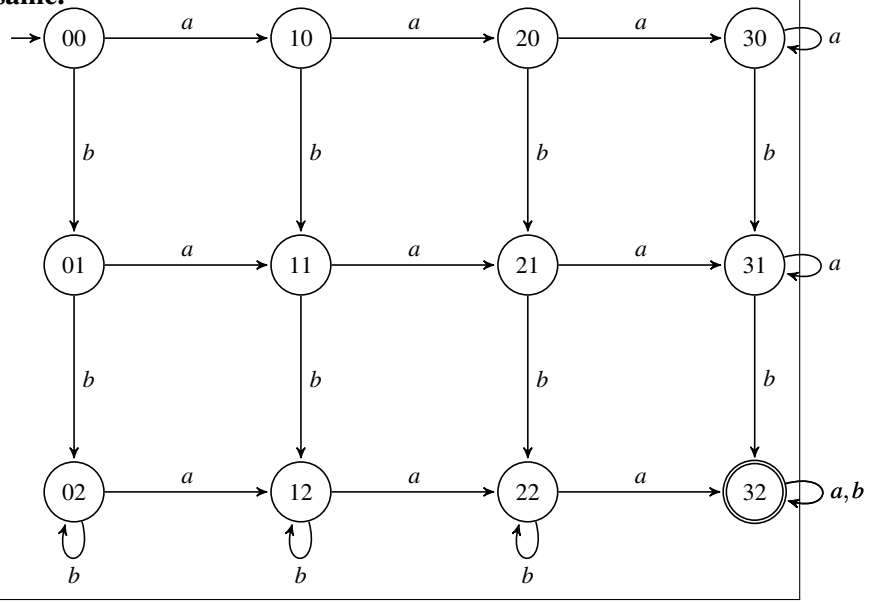
6. Let us designate state names according to the following convention: The set of names is the language over the alphabet $\{0,1\}$ defined by the regular expression $(0 \cup 1)(0 \cup 1)$. In other words, our state names are picked from the set $\{00, 10, 01, 11\}$. 0 means even and 1 means odd, where the first digit applies to the length of the string to be read by the dfa, the second digit applies to the number of b's. With this convention, 01 is the state for an even length input with an odd number of b's; 00 is for even length with even number of b's, and so on. Here is the machine operating over these four states, recognizing the strings of a's and b's with an even length and odd number of b's. **Please observe that the names we give to our states is just to make the machine more understandable for humans. The names do not mean anything to the machine. Any other naming would be equally good, provided that the state transition function stays the same.**



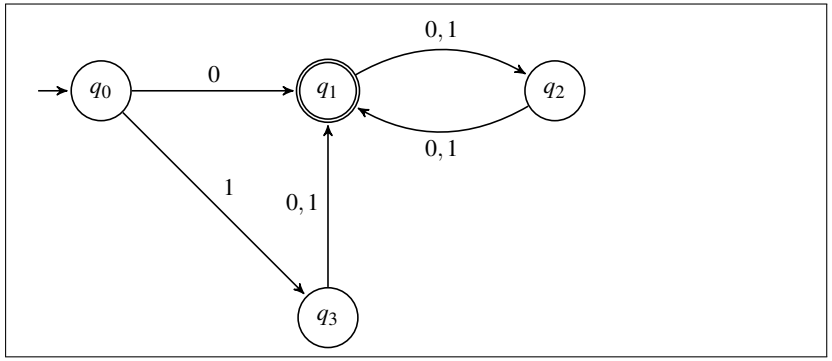
7.

Let us designate our states with two numbers xy. We will use x to designate the number of a's encountered. For $0 < x < 3$, it will stand directly for the number of a's, for $x = 3$ it will mean "we encountered at least 3 a's." A similar convention will apply to y and the number of b's encountered. The only difference is that $y = 2$ will mean "we encountered at least 2 b's", and the values 0 and 1 for y will directly represent the number of b's encountered so far. With this naming convention we have 0, 1, 2, and 3 as possible values for x; 0, 1, and 2 for y. This means we can have 12 states to designate all the possibilities. We will construct our machine on these possibilities.

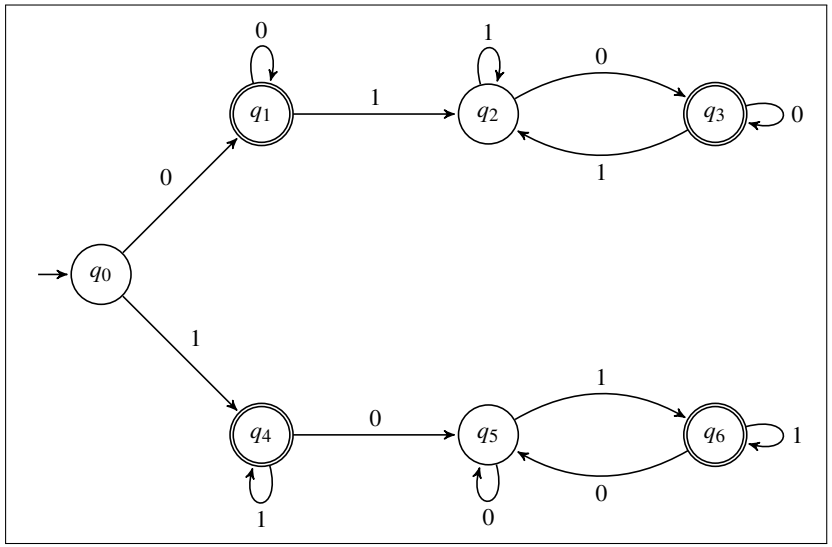
Please observe that the names we give to our states is just to make the machine more understandable for humans. The names do not mean anything to the machine. Any other naming would be equally good, provided that the state transition function stays the same.



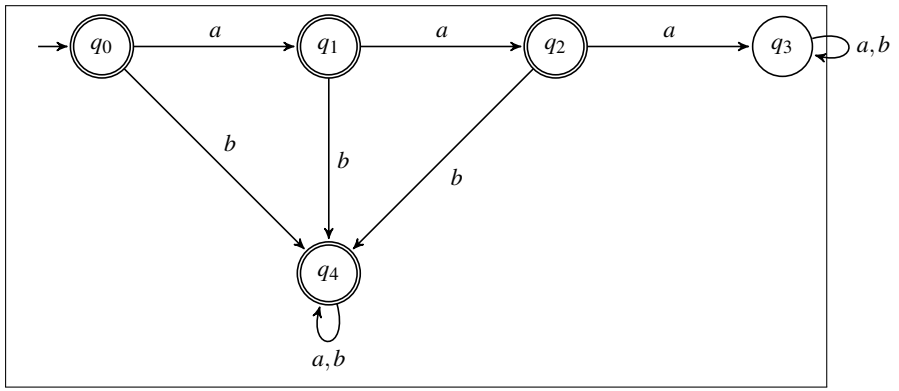
8.



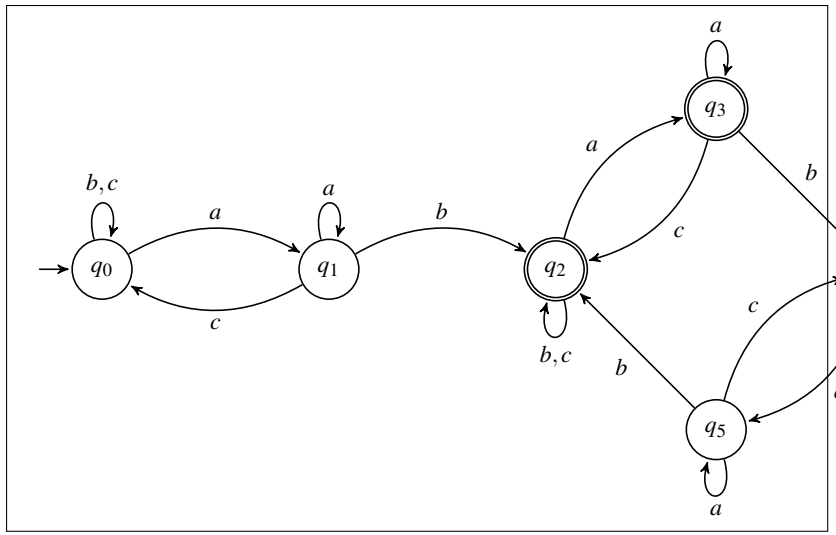
9.



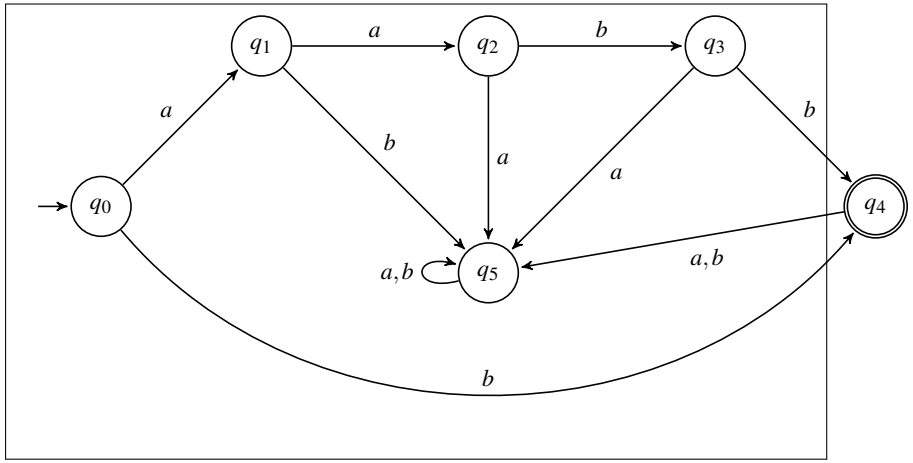
10.



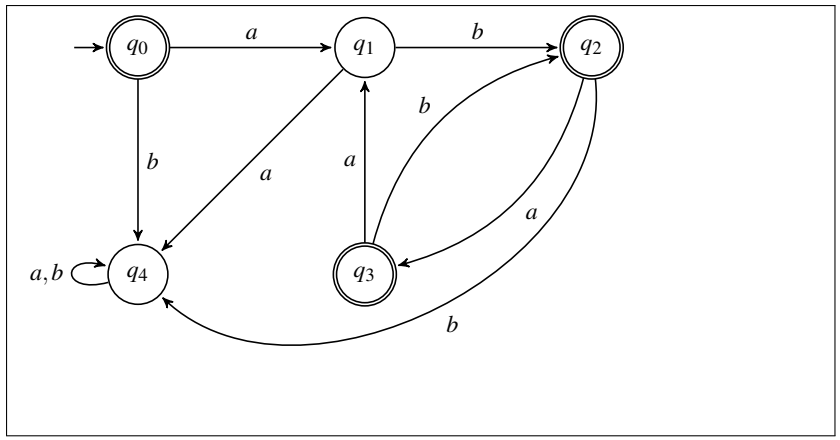
11.


q_0
q_1
q_2
q_3
q_4
q_5
b,c
a
b
c

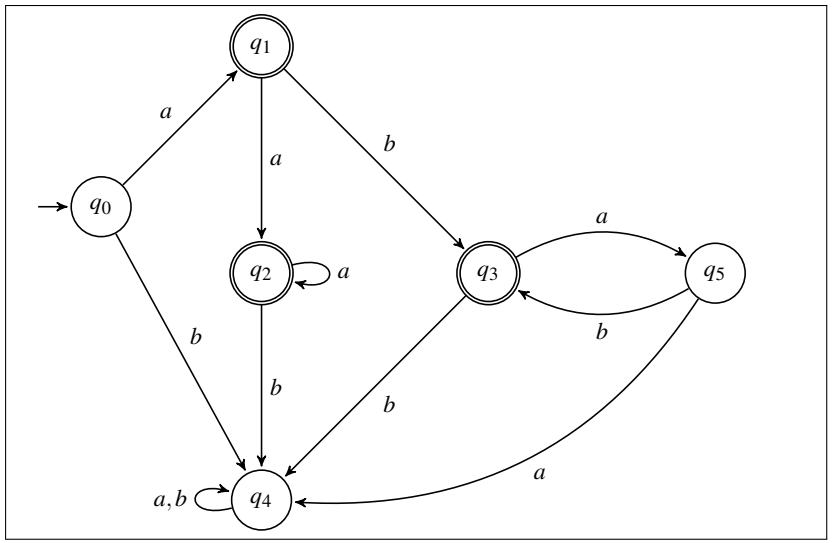
2.


q_0
q_1
q_2
q_3
q_4
q_5
a
b
a,b

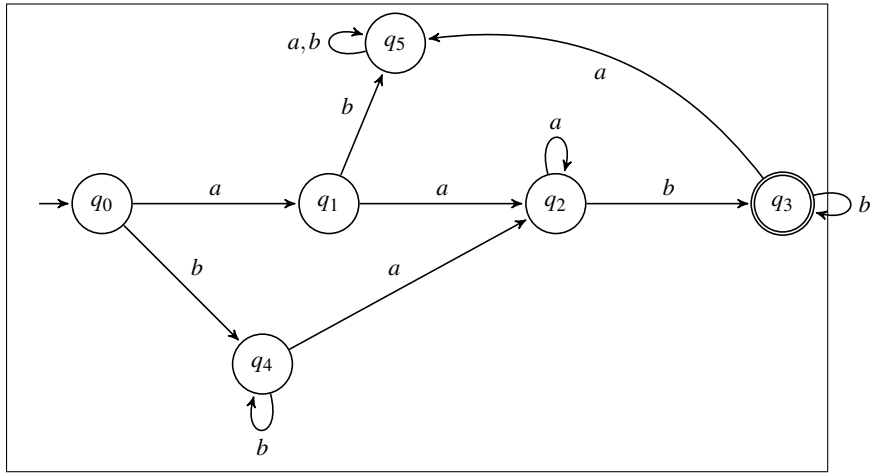
3.2 1.


q_0
q_1
q_2
q_3
q_4
a
b
a,b

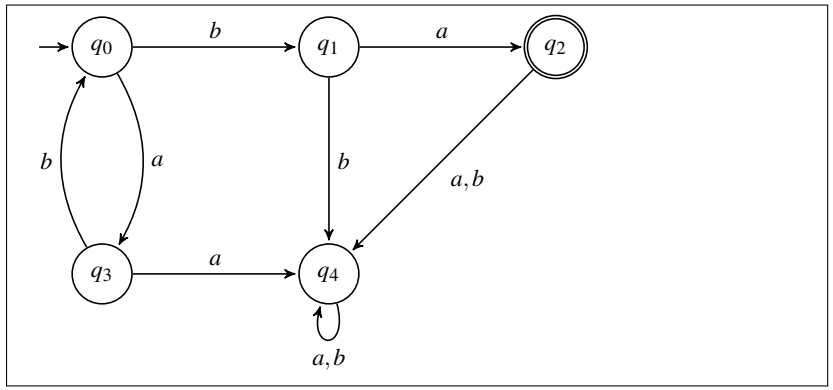
3.


q_0
q_1
q_2
q_3
q_4
q_5
a
b
a,b

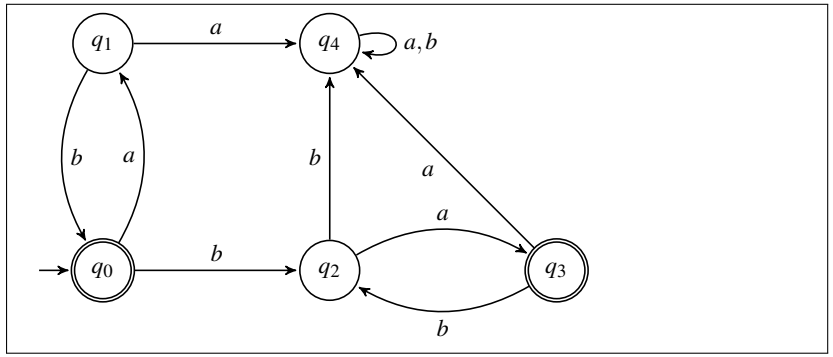
4.


q_0
q_1
q_2
q_3
q_4
q_5
a
b
a,b

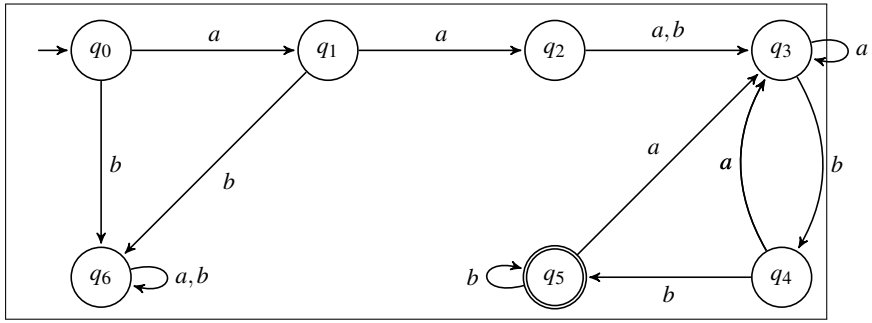
5.


q_0
q_1
q_2
q_3
q_4
a
b
a,b

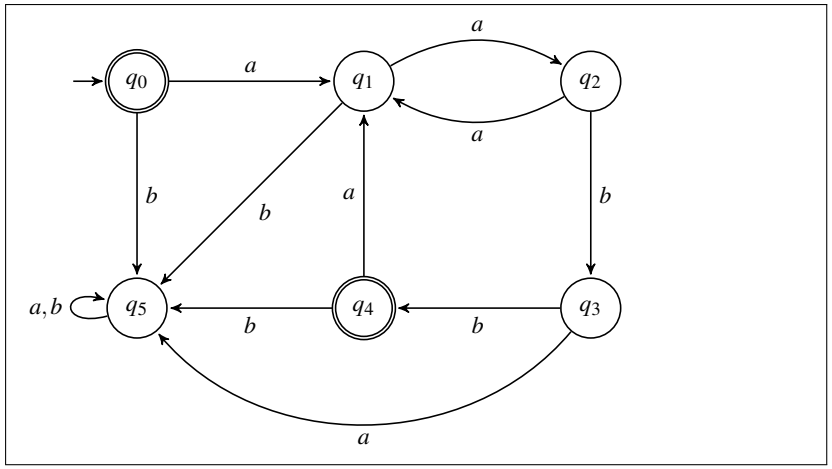
6.


q_0
q_1
q_2
q_3
q_4
a
b
a,b

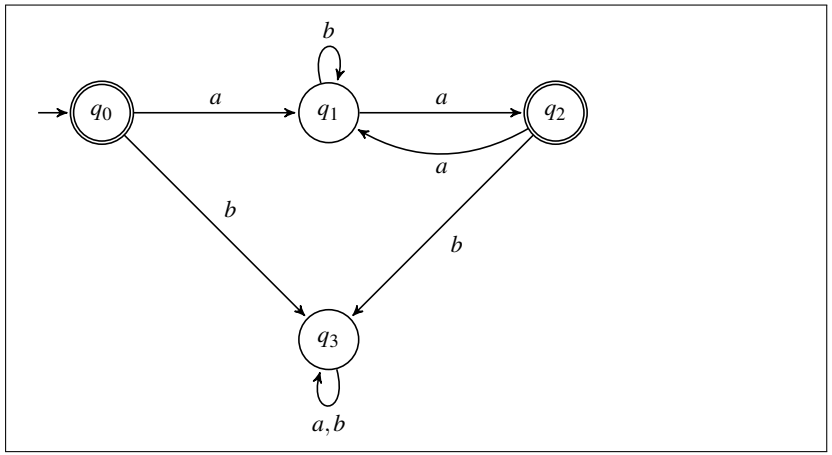
7.


q_0
q_1
q_2
q_3
q_4
q_5
q_6
a
b
a,b

8.

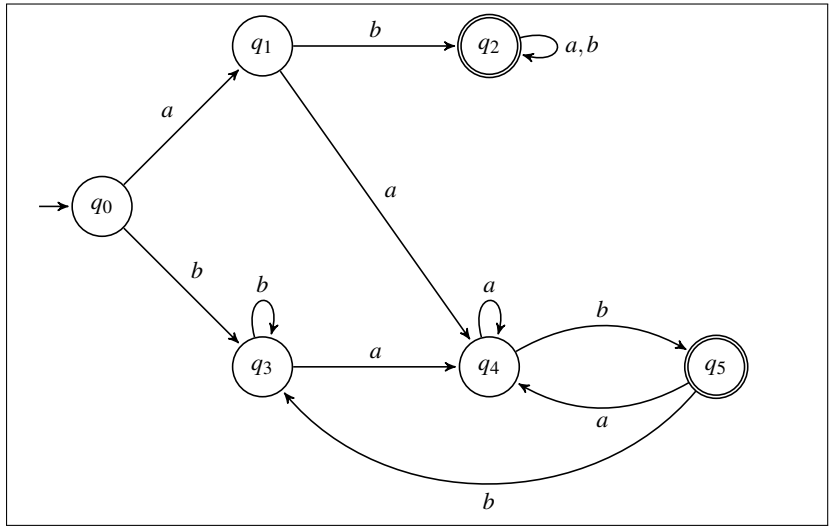

q_0
q_1
q_2
q_3
q_4
q_5
a
b
a,b

9.

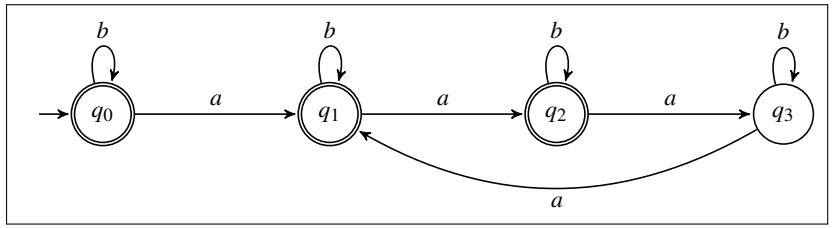

q_0
q_1
q_2
q_3
a
b
a,b

3.3

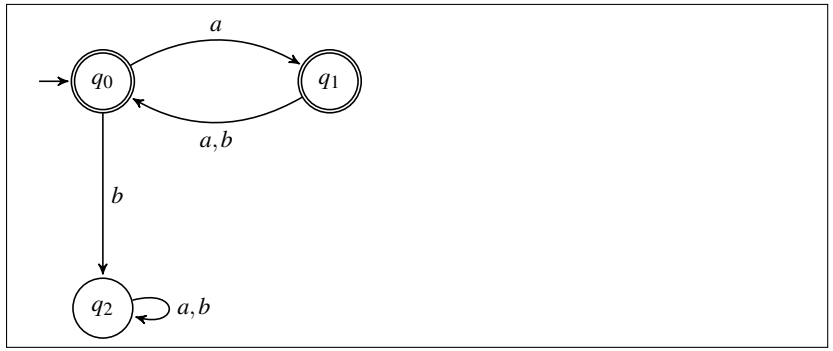
1.


q_0
q_1
q_2
q_3
q_4
q_5
a
b
a,b

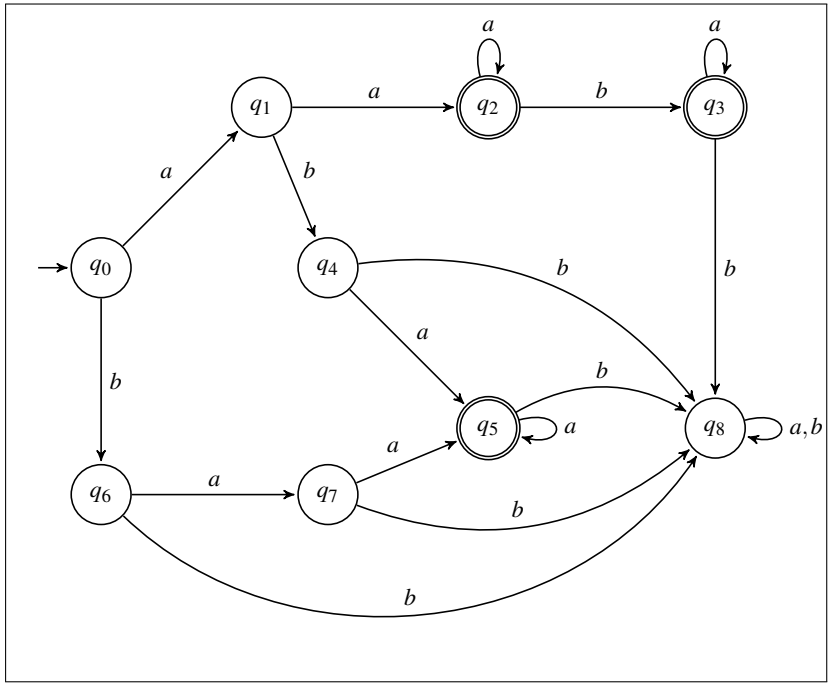
2.


q_0
q_1
q_2
q_3
a
b

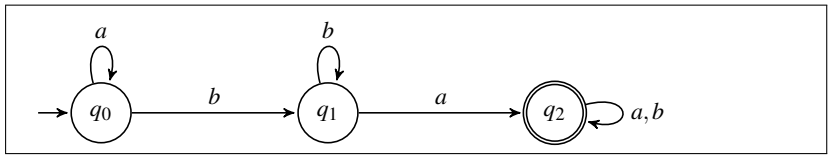
3.


q_0
q_1
q_2
a
b
a,b

4.


q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
a
b
a,b

5.


q_0
q_1
q_2
a
b
a,b