Compiler Construction: Principles and Practice
by Kenneth C. Louden

Chapter 2 Exercise Answers

Exercise 2.1

(a) **a (a | b | c | ... | z)* a | a**

(c) **(1 | 2 | 3 | ... | 9) (0 | 1 | 2 | ... | 9)* | 0 | ε**

(e) **(0 | 1 | 2 | ... | 8)* (0 | 1 | 3 | 4 | ... | 9)***

(g) **b*ab*(ab*ab*)* | a*ba*(ba*ba*)***

(i) Impossible (see Example 2.3, page 39).

Exercise 2.2

(a) All strings of *a*'s and *b*'s whose last or next-to-last character is an *a*.

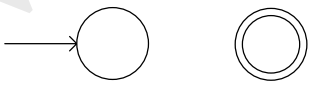
(c) All strings of *a*'s and *b*'s in which no single *a*'s are followed by single *b*'s.

Exercise 2.4

The reason is that set union, which describes the result of applying the | operator, is associative. Similarly, set product, which defines the result of concatenation, is also an associative operation.

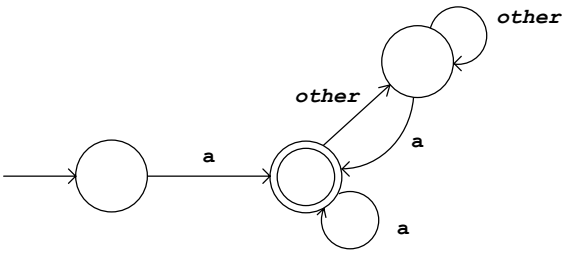
Exercise 2.7

The following DFA has no transitions to the accepting state, and so is equivalent to ϕ:



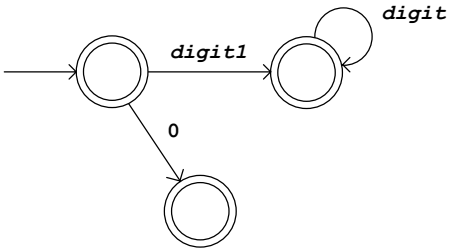
Exercise 2.8

(a)



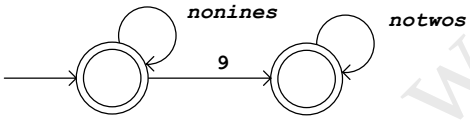
(*other* = b|c|...|z)

(c)



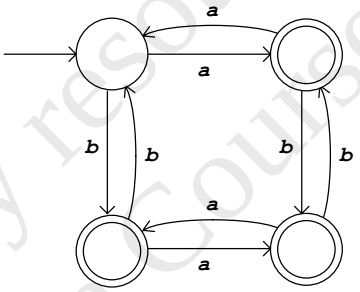
(*digit1* = 1|2|3|...|9)
(*digit* = 0|1|...|9)

(e)



(*nonines* = 1|2|3|...|8)
(*notwos* = 0|1|3|4|...|9)

(g)



(i) Since DFAs and regular expressions are equivalent concepts, a DFA is impossible, since a regular expression is impossible (see the answer to 2.1(i) above).

Exercise 2.10

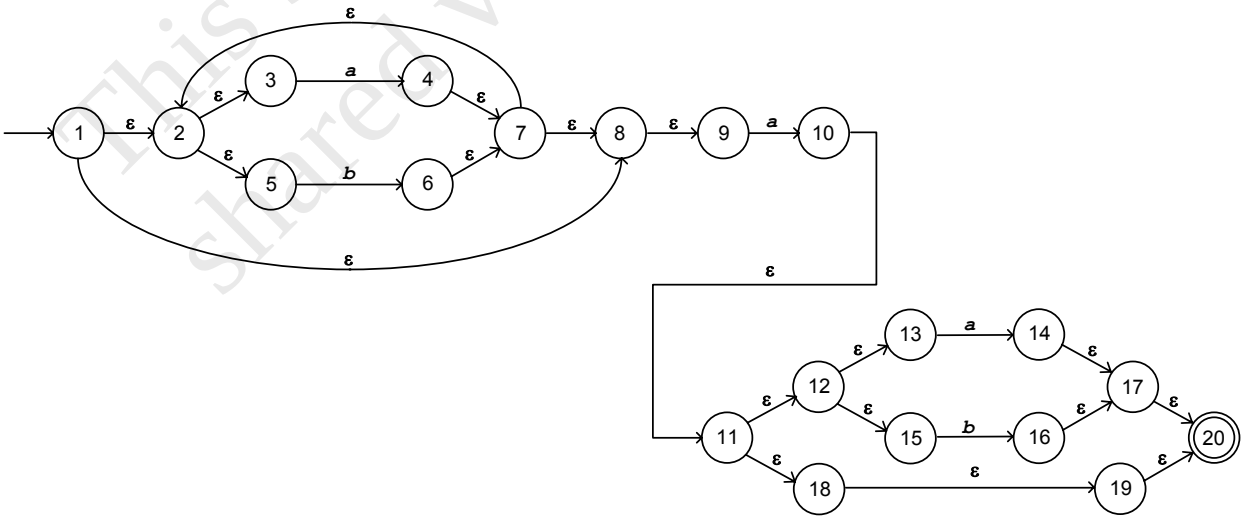
state := 1; { start }
while *state* = 1, 2, 3 *or* 4 **do**
  case *input character* **of**
  "/": **case** *state* **of**
        1: *advance the input* ;
          state := 2 ;
        2: *state* := ... { error or other } ;
        3: *advance the input* { and stay in state 3 } ;
        4: *advance the input* ;

```
            state := 5 ;
        end case;
    "*": case state of
            1: state := ... { error or other } ;
            2: advance the input ;
               state := 3 ;
            3: advance the input ;
               state := 4 ;
            4: advance the input { and stay in state 4 };
        end case ;
    else case state of
            1, 2: state := ... { error or other } ;
            3, 4: advance the input ;
                  state := 3 ;
        end case;
     end case
end while;
if state = 5 then accept else declare error ;
```

If each input character must be tested in each state, then the number of tests remains the same at *n***m* (*n* = the number of states, *m* = the number of input chars), regardless of whether we write the state tests or the character tests on the outside. However, if in each state only a few characters need testing (and all the others can be handled by defaults), while for each character all states still need to be distinguished, then writing the state tests on the outside will be better. This is usually the case for language scanners. However, if the total number of transitions on each character is small compared to the number of states, then writing the character tests on the outside will be better.

Exercise 2.12

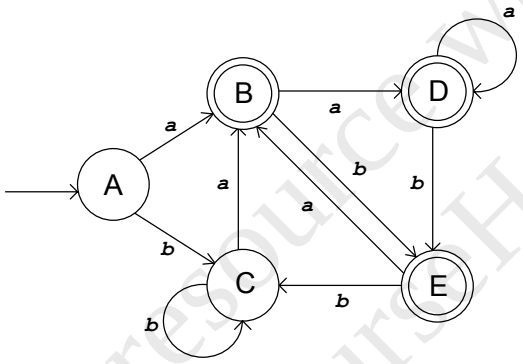
(a)



(b) The subset construction leads to the following sets of states of the machine of (a), to which we assign new names:

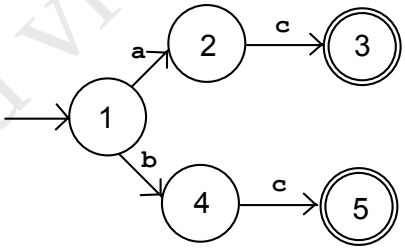
Subset	New Name
{1, 2, 3, 5, 8, 9}	A
{2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 18, 19, 20}	B
{2, 3, 5, 6, 7, 8, 9}	C
{2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20}	D
{2, 3, 5, 6, 7, 8, 9, 16, 17, 20}	E

The DFA computed by the subset construction is then as follows:

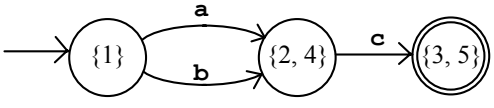


Exercise 2.16

(a) Given the following numbering of the states



the minimized DFA is



Exercise 2.18

(a)

```
"/*""/"*("*"*[^*/]"/"*)*"*"*"*/"
```