CHAPTER 5 FINITE STATE MACHINES

SECTION 5.1 FSMs WITH OUTPUTS

a finite state machine (FSM) with outputs

A FSM (also called a finite automaton) with outputs is an abstract device consisting of a finite number of states (one of which is called the starting state), a finite input alphabet and a finite output alphabet. Initially the FSM is in its starting state. It receives a symbol from its input alphabet, in response emits a symbol from its input alphabet and moves to a next state.

A FSM is considered to have memory (as you'll see in examples) because the current output depends not only on the current input but on the present state of the machine which itself is a summary of the past history of the machine.

The state diagram in Fig 1 shows a FSM with input alphabet 0,1 and output alphabet p,q,r. The states are A (the starting state), B and C. To see how the FSM works let's find the output corresponding to the input string

0 1 1 0 1 0 1.

The leftmost symbol in the string is fed in first. The FSM is initially in its starting state A and when input 0 is received, it produces output p and moves to next state A (i.e., stays in A). After the next input, 1, the FSM produces output q and moves to next state B. The table in Fig 2 lists the successive outputs and next states. All in all, the output string is

p q p r p p q

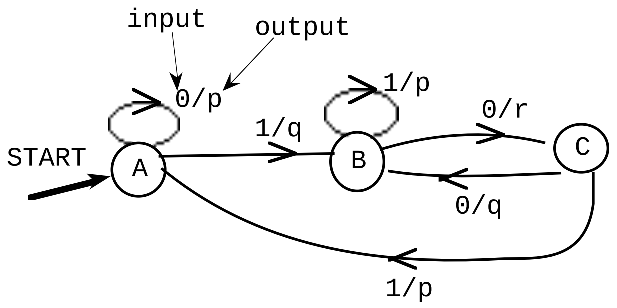


FIG 1

input	0	1	1	0	1	0	1
output	p	q	p	r	p	p	q
next state	A	B	B	C	A	A	B

FIG 2

The state table in Fig 3 contains the same information as Fig 1, but in different form. For example the second line of the table in Fig 3 says that if the FSM is in state B and the input is 0 then the output is r and the machinemoves to next state C; if the input is 1 the output is p and the next state is B.

state	output		next state	
	input 0	input1	input 0	input 1
A	p	q	A	B
B	r	p	C	B
C	q	p	B	A

FIG 3

finite state machines as recognizers

Look at an input string of symbols, called a *word.* The FSM *recognizes* the word if the last output is 1.

It's often necessary to design a FSM to recognize a class of words (e.g., the compiler in a computer must to be able to recognize inputs of a certain form).

example 1

Suppose you want a FSM with input and output symbols 0,1 which recognizes words ending in 101, i.e., which outputs a final 1 if a word ends in 101 and outputs a final 0 otherwise.

Note that if a FSM is to recognize words ending in 101 then it must output a 1 *after every occurrence* of 101 in an input string in case the word suddenly ends there.

Fig 4 shows a sample input string with the corresponding desired outputs.

```
input   0  0  1  1  1  0  1  0  1  1  1  1  0  1  0  0  0
output  0  0  0  0  0  0  1  0  1  0  0  0  0  1  0  0  0
```

FIG 4

Fig 5 shows the FSM recognizer.

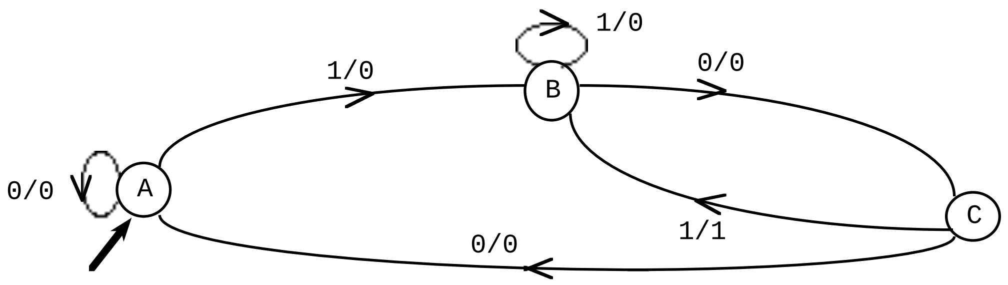


FIG 5 Recognizes words ending in 101

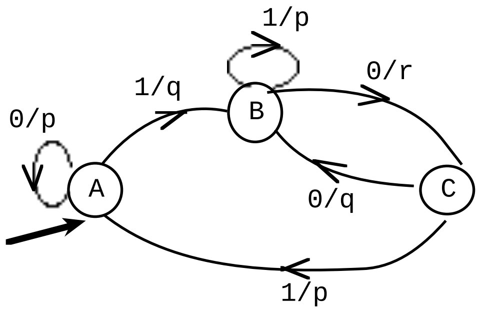
State A "remembers" that (i.e., is reached when) neither of the last two inputs were 1's so that a 101 block isn't in the making yet. The machine won't switch to another state until you do get an input 1, and a 101 block starts to grow.
State B remembers that the last input was 1. The FSM stays in state B until it receives an input 0 and a 101 starts to grow further.
State C remembers that the last two inputs were 10.

If the next input is 1, the output is a 1 to signal that a 101 was completed, and the next state is B since the last input 1 may be the beginning of a new 101.
If the next input is 0 then the output is 0 indicating that a 101 was not achieved, and the FSM sends you back to A to try again.

PROBLEMS FOR SECTION 5.1

1. Fill in the following table for the given FSM

input	0	0	1	1	0	0	0	1	0
output									
next state									



2. Let the input and output symbols be 0,1. Design a FSM to recognize the set of words which
 (a) contain an even number of 1's
(remember that a word with no 1's, i.e., with zero 1's, should be recognized since 0 is an even number)
 (b) contain an odd number of 1's
 (c) contain no 1's
 (d) contain exactly two 1's
 (e) contain at least two 1's
 (f) end in 000

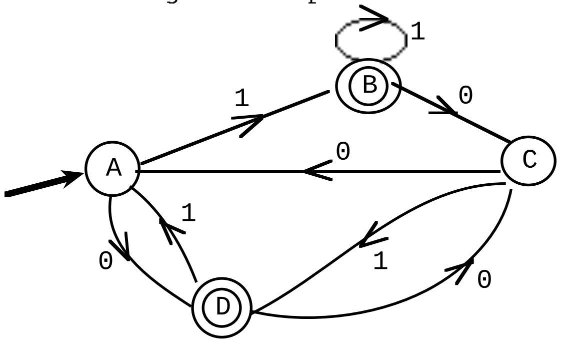
SECTION 5.2 FINITE STATE MACHINES WITH ACCEPTING STATES INSTEAD OF OUTPUTS

definition of a FSM with accepting states

Suppose a FSM is designed with no outputs. Instead, some of the states are called *accepting states*, denoted by a double circle. In Fig 1, states B and D are accepting, A and C are non-accepting. The numbers on the edges are the inputs.

A FSM with outputs (preceding section) recognizes (accepts) an input word if the last output is 1. A FSM with accepting states (this section) recognizes (accepts) a word if the machine is left in an accepting state after the word has been processed.

Fig 1 shows a FSM with accepting states instead of outputs and a table for the input string 1 0 1 1 1.
The string is accepted because the machine ends up in B, an accepting state.



input	1	0	1	1	1
next state	B	C	D	A	$\boxed{B}$

FIG 1

example 1

Fig 2 shows a FSM with accepting states which accepts the set of strings ending in 101

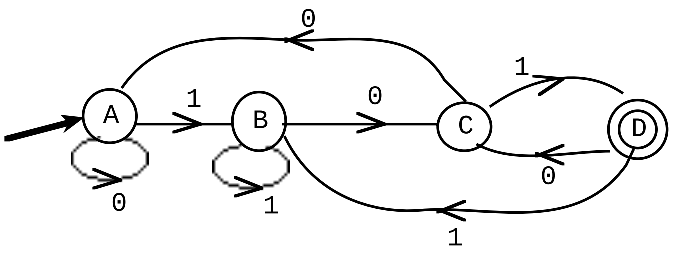
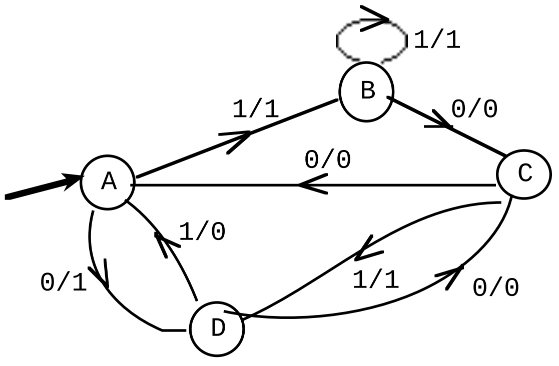


FIG 2

converting a FSM with accepting states to a FSM with outputs

Make the output 1 for each transition into an accepting state and 0 for each transition in a non-accepting state. Fig 3 shows the FSM with accepting states from Fig 1 converted to a FSM with outputs.
The table shows that the string 1 0 1 1 1 is accepted because the last output is 1.



input	1	0	1	1	1
output	1	0	1	0	$\boxed{1}$
next state	B	C	D	A	B

FIG 3 The output version of Fig 1

and vice versa --- converting a FSM with outputs to a FSM with accepting states

I'll illustrate how to do it with the FSM in Fig 4.
When the FSM in Fig 4 enters state A it emits output 0 or 1, depending on the state you're coming from. To make an equivalent FSM with accepting states, replace state A by non-accepting A_0 and accepting state A_1, the former to be entered only as the machine emits 0 and the latter only as the machine emits 1. Similarly B is replaced by non-accepting state B_0 and accepting state B_1 so that the no-output version will have four states, not two. The table in Fig 5 shows how pieces of the original FSM are replaced by new pieces. Fig 6 shows the new FSM with accepting states.

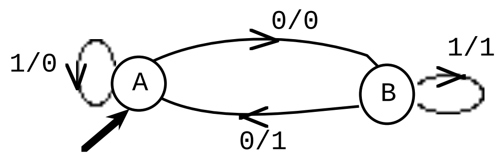


FIG 4 FSM with outputs

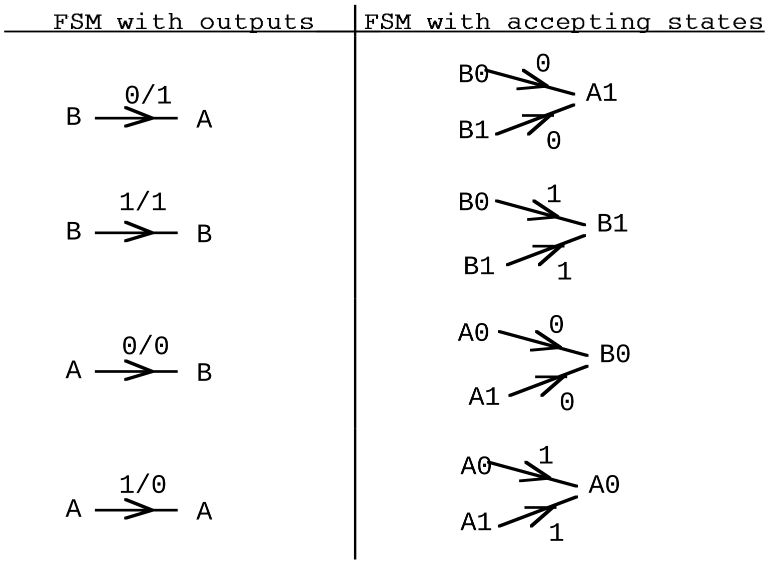
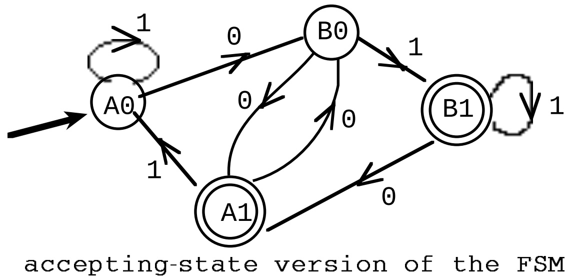


FIG 5



Note that in the new FSM the 0-exits from B0 and B1 must go to the *same* state, (happens to be A1 here).
And the 1-exits from B0 and B1 must go to the same state (happens to be B1 here)

accepting-state version of the FSM in Fig 4
FIG 6

In general, if there are transitions into a state Z with outputs of 0 *and* 1 then Z gets split into non-accepting state Z_0 to receive the transitions with output 0 and accepting state Z_1 to receive the transitions that had output 1.
And Z_0 and Z_1 each have the same transitions *out* as the original Z.

recognition capabilities of FSMs with outputs vs. FSMs with accepting states

For every FSM with accepting states there is a FSM with outputs that recognizes the same words. And vice versa.

So *you do not lose or gain recognition capability by switching from FSMs with outputs to FSMs with accepting states.*

footnote Well, actually you do gain recognition capability, in particular the ability to recognize the empty string, denoted λ. But don't worry about it.

The advantage of FSMs with accepting states is that they are easier to prove theorems about and since the two types have the same recognition capabilities, theorems coming later about FSMs with accepting states will also hold for FSMs with outputs.

the empty string

The empty string, i.e., the string with no symbols in it, is denoted λ.

If the starting state of a FSM is an accepting state then the FSM is said to recognize λ.

choosing the starting state in the equivalent no-output FSM

The starting state of the FSM with outputs in Fig 4 is A, so either A_0 or A_1 should be the starting state of the FSM with accepting states in Fig 6

If you use A_1 as the starting state, the FSM with accepting state accepts the empty string λ, since it's in an accepting state before any symbols comes in. But the original FSM in Fig 5 doesn't recognize λ (no FSM with outputs can do that).

So choose A_0 as the starting state rather than A_1.

inventing a new starting state if necessary

I'll find a FSM with accepting states that is equivalent to (recognizes the same strings as) the FSM in Fig 7.

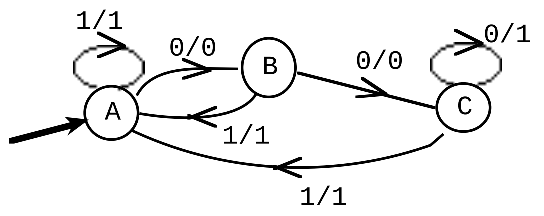


FIG 7

All transitions into A emit output 1 so don't split A into two states; just make A an accepting state in the new version. Similarly all transitions into B emit output 0 so B becomes a non-accepting state (no splitting).

But C is split into a non-accepting state C_0 and an accepting state C_1.

So far you have states A, B, C_0 and C_1 in Fig 8. But you don't want to make A the start since it's an accepting state and you don't want to recognize λ (because the original FSM in Fig 7 does not recognize λ). So add a new state NewStart to serve as the start. *Make NewStart non-accepting, with no transitions in and with the same exits as A.*

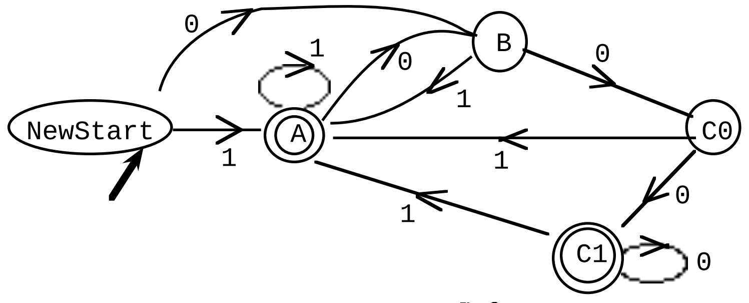


FIG 8

mathematical catechism (you should know the answers to these questions)

question 1

Why bother converting from a FSM with outputs to a FSM with accepting states and vice versa.

answer

The fact that we *can* convert proves that the two collections of FSMs have the same recognition capabilities (give or take recognizing λ).

question 2

Why do we care that the FSMs with outputs and the FSMs with accepting states have the same recognition capabilities.

answer

When we prove a theorem about FSMs with accepting states (what they can and cannot recognize) the same theorem will hold for FSMs with outputs.

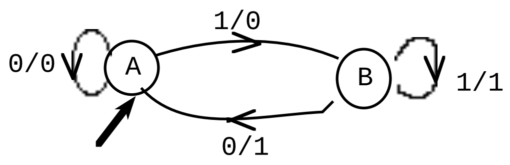
PROBLEMS FOR SECTION 5.2

1. Let the input alphabet be 0,1.
Find a FSM with accepting states to recognize the set of strings
(a) containing at least one occurrence of 101
(b) ending in 000
(c) where the number of 1's is a multiple of 3

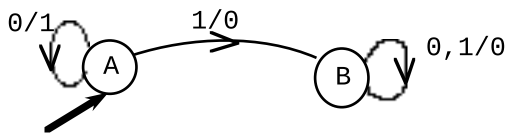
2. Let the input alphabet be a, b, c.
Find a FSM with accepting states to recognize
(a) words ending in abc
(b) words containing at least one occurrence of abc
(c) just the word abc

3. For each of the following FSMs use the official method in this section to find an equivalent FSM (i.e., one which recognizes the same strings) with accepting states instead of outputs.

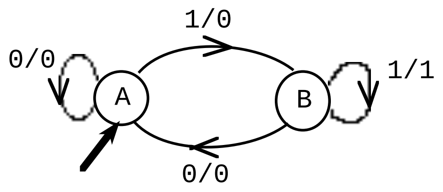
(a)



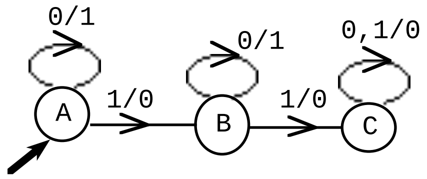
(b)



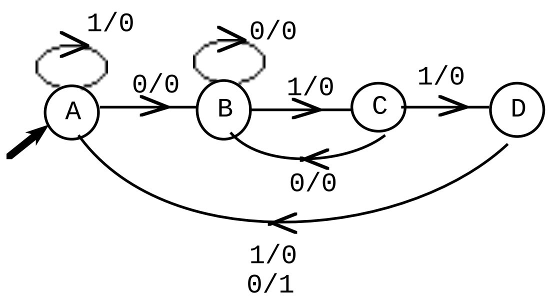
(c)



(d)



(e)



SECTION 5.3 NONDETERMINISTIC FINITE STATE MACHINES

definition of a nondeterministic FSM (assuming the alphabet is 0, 1)

The FSM with accepting states from the last section is a *deterministic finite state machine* (DFSM) meaning that the next state is uniquely determined by the present state and input. From each state there is *one* 0-exit and *one* 1-exit.

In a *nondeterministic finite state machine* (NFSM) there may be more than one 0-exit or 1-exit from a state, i.e., there may be several next states. There may be *no* 0-exit or 1-exit from a state, i.e., the machine may shut down and be in no state at all. For the NFSM in Fig 1, from state A the input 1 allows any of the three next states A, B, C. From state C, any input causes a shutdown.

An input string is recognized (accepted) by a NFSM if the NFSM *may* be left in an accepting state after the word has been processed. You have to look at all the possible tracks to see if any of them reach an accepting state.

For example, try the string 100 in the NFSM in Fig 1. The several possible paths are shown in Fig 2. Since the second path leads to an accepting state, the string is accepted.

warning

Don't say that the NFSM in Fig 1 *might* recognize 100 depending on which track in Fig 2 is followed. The definition says that the NFSM *does* recognize 100 because at least one of the tracks leads to an accepting state.

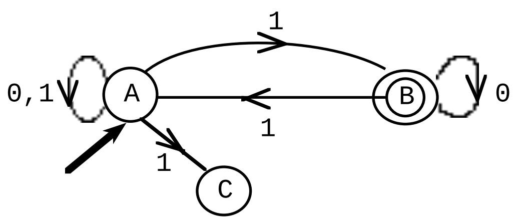


FIG 1 Example of a NFSM

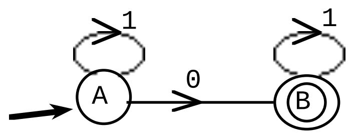
$$\begin{tikzcd}
A \arrow[r, "1"] & A \arrow[r, "0"] & A \arrow[r, "0"] & A \\
A \arrow[r, "1"] & B \arrow[r, "0"] & B \arrow[r, "0"] & B \\
A \arrow[r, "1"] & C \arrow[r, "0"] & \text{none}
\end{tikzcd}$$

FIG 2 Multiple tracks for input 100

shutdowns

Look at the NFSM in Fig 3 and try input 001. The string is not accepted since the only track leaves the machine in *no* state. (In fact once a string begins with 00 it's not accepted since the machine shuts down after the initial 00.)

Note that from state B with input 0 the next state is "none", *not* B; the machine shuts down and is *not* left in state B.



in		0	0	1
next state	A	B	shutdown	still shutdown

FIG 3

example 1

Fig 4 is a NFSM which recognizes words containing at least one occurrence of cat.

For example, if the input is xcaz*cat*d then there is a track ending in an accepting state (Fig 5) so the string is accepted.

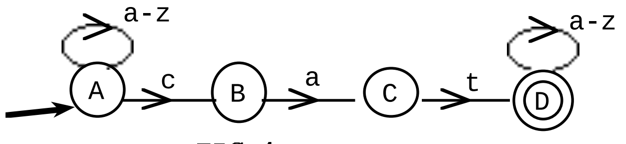


FIG 4

$$\begin{tikzcd}
A \arrow[r, "x"] & A \arrow[r, "c"] & A \arrow[r, "a"] & A \arrow[r, "z"] & A \arrow[r, "c"] & B \arrow[r, "a"] & C \arrow[r, "t"] & D \arrow[r, "d"] & D
\end{tikzcd}$$

FIG 5

the empty string λ

The empty string or null string, i.e., the string containing no symbols, is denoted by λ. Any FSM whose starting state is an accepting state recognizes λ since in that case the FSM is in an accepting state before there is any input at all.

Note that since λ is an empty string, the strings 0λλ1, λ0λ1, 01λλλ are all the same as 01.

NFSMs with λ-moves (λNFSMs)

The NFSM in Fig 6 allows transitions from A to B, B to E, D to B with "input" λ, i.e., with no input. These transitions are called λ-moves.

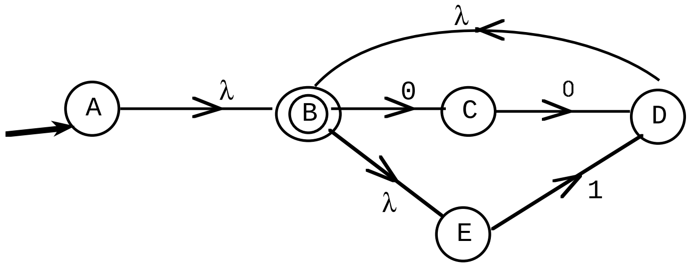


FIG 6

In a NFSM with λ-moves the state Q is said to be *λ-reachable* from state P if there is a sequence of λ-moves leading from P to Q. In Fig 5, E is λ-reachable from A (see the path ABE), B is λ-reachable from D (see the path DB). Furthermore *each state is considered to be λ-reachable from itself*.

In a small FSM like Fig 6 you can tell by inspection which states are λ-reachable from which other states. Here's how to do it in a large FSM. First consider the FSM as a digraph with the states as vertices and the λ-moves as directed edges (ignore other moves). Find the *λ-adjacency matrix* M in which an entry of 1 in row P, col Q indicates a λ-move from P to Q; by convention 1's are inserted along the diagonal to indicate λ-moves from each state to itself. Then use Warshall's algorithm to find the *λ-reachability matrix* M_∞ in which an entry of 1 in row P, col Q indicates a path of λ-moves from P to Q, i.e., indicates that Q is λ-reachable from P.

Here are the λ-adjacency matrix and the λ-reachability matrix for the NFSM in Fig 6.

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 1 & 1 & 0 & 0 & 0 \\ B & 0 & 1 & .0 & 0 & 1 \\ C & 0 & 0 & 1 & 0 & 0 \\ D & 0 & 1 & 0 & 1 & 0 \\ E & 0 & 0 & 0 & 0 & 1 \end{array}$$

λ-adjacency matrix M for Fig 6

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 1 & 1 & 0 & 0 & 1 \\ B & 0 & 1 & 0 & 0 & 1 \\ C & 0 & 0 & 1 & 0 & 0 \\ D & 0 & 1 & 0 & 1 & 1 \\ E & 0 & 0 & 0 & 0 & 1 \end{array}$$

λ-reachability matrix M_∞ for Fig 6

how to tell whether a NFSM accepts or rejects a string

In a small NFSM you can tell by looking at all the parallel tracks whether or not a string is accepted. Here's an algorithm for deciding automatically.

For a NFSM without λ-moves, keep a record of the *set* of next states (as opposed to the *unique* next state in a DFSM); a string is recognized by the NFSM if you end up with a set of next states which includes an accepting state.

I'll test the string 110 in the NFSM in Fig 7. From the starting state A, the input 1 leads to possible next states A,C (Fig 8). With another input of 1, from A the possible next states are A,C and from C the only possible next state is B; so the set of next states is A,B,C. With the input 0, from A the possible next states are A,B, from B the only possible next state is B and from C there is no next state; so the set of next states is A,B. Since one of them, namely B, is accepting, the string is accepted.

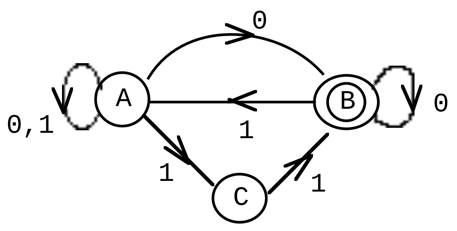


FIG 7

input	1	1	0	
next states	A,C	A,B,C	A, $\boxed{B}$	accept

FIG 8

For a NFSM with λ-moves there are extra steps because if the NFSM is in say state P then even before the next input appears the NFSM may move to any state λ-reachable from P. So before the first input you should move from the starting state to the set of states that are λ-reachable from the start. And after an input, if the current state is say any of B,C,D then before the next input you should move to the set of states that are λ-reachable from B,C D. This amounts to testing a string as if there were λ's before and after each character in the string .

I'll test 100 in the NFSM in Fig 9.
The starting state is A and states A,B,E are λ-reachable from A so the FSM is considered to be in any of states A,B,E as the first input, 1, is processed (Fig 10). Of these three states only E has a 1-transition, namely to D. Fig 10 continues to show inputs and next states. The string 100 leads to the possible states B,D,E. Since one of these, namely B, is accepting, the string is recognized (in particular see path A B E D B C D B).

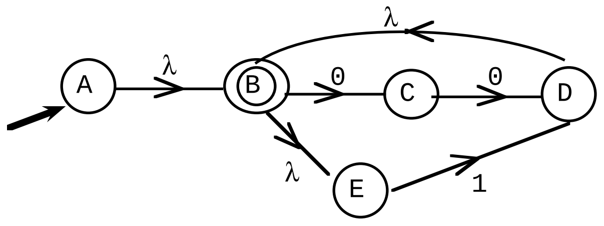


FIG 9

input		λ's	1	λ's	0	λ's	0	λ's	
next states	A	A,B,E	D	B,D,E	C	C	D	$\boxed{B}$,D,E	accept

FIG 10 The FSM in Fig 9 accepts 100

Let's try the string 110 in the FSM in Fig 9. Fig 11 shows that all tracks lead to the non-accepting state C. So the string is not accepted

input		λ's	1	λ's	1	λ's	0	λ's	
next states	A	A,B,E	D	B,D,E	D	B,D,E	C	C	reject

FIG 11 The FSM in Fig 9 rejects 110

NFSMs versus DFSMs

A DFSM can be physically constructed by engineers so that passage from state to state actually occurs inside the device. A NFSM doesn't exist as a physical device but it is realistic in that its ability to recognize strings can be simulated by a computer program, in particular by the preceding algorithm.
A DFSM can be thought of as a special case of a NFSM, the case where the options of having no next state, more than one next state and λ moves are not exercised. From that point of view it would seem that the class of NFSMs is "larger" than the class of DFSMs. But I'll show:

(1) For every NFSM there is a DFSM which recognizes the same words.

So we *gain no extra recognition capability (and lose none) by expanding from DFSMs to NFSMs*.

The advantage of the NFSMs is that it's easier to prove theorems about NFSMs than about DFSMs and since the two types have the same recognition capabilities, certain theorems coming later about NFSMs also hold for DFSMs.

why (1) holds --- how to convert a NFSM to a DFSM

I'll illustrate the idea by finding a DFSM that recognizes the same strings as the NFSM in Fig 12 (where the starting state is C).

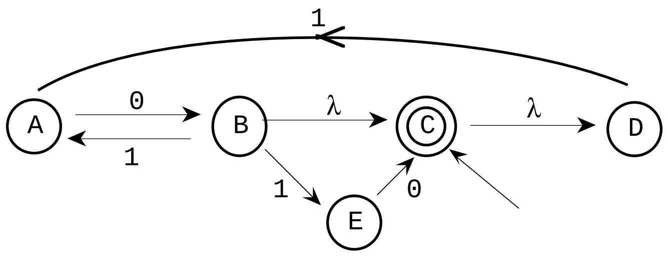


FIG 12

The DFSM will have as its states some or all of the states named

ϕ, A, B, C, D, E, AB, AC, AD, AE, BC, BD, BE, CD, CE, DE, ABC, ABD, ..., ABCD, ..., ABCDE

For convenience you can think that the names correspond to subsets of {A,B,C,D,E}. Here's how to choose the particular states and transitions.

In the NFSM the states that are λ-reachable from the starting state C are C (every state is λ-reachable from itself) and D. So for the new DFSM the starting state will be named CD.

To get the 0-exit from state CD in the new FSM look at 0-exits from C and D in the old NFSM.

C -> none
D -> none

Choose to have a state named ϕ in the DFSM. And use it as the 0-exit from CD.

To find the 1-exit from state CD in the new DFSM look at 1-exits from C and D in the old NFSM.

C -> none
D -> A -> (lambda moves) A

Choose to have a state named A as the next state
(I'm starting to get the DFSM in Fig 13.)

To get the 0-exit from state A in the new DFSM look at 0-exits from A in the old NFSM.

A -> B -> (λ moves) B, C, D

Next state is named BCD.

warning Don't forget to include these λ moves when they exist.
In other words, After A $\xrightarrow{0}$ B, continue from B to all states, including B itself, that are λ-reachable from B, namely B, C, D.

To get the 1-exit from state A in the new DFSM look at 1-exits from A in the old NFSM.

A -> none

Next state is ϕ.

General Rule

(1) (how to get started)
If the starting state in the NFSM is X and the states λ-reachable from X are E, Z then the starting state in the new DFSM is named EXZ.

(2) (how to keep going)
To find the next state from say the state APQ in the new DFSM when the input is 0, look at next states from A, P, Q in the old NFSM when the input is 0.

If say

```
        A -> C -> (λ moves) C, X, Y
        P -> none
        Q -> A, Z -> (λ moves) A, Z, D
```

then the next state in the new DFSM would be named ACDXYZ

If

```
         A -> none
         P -> none
         Q -> none
```

then the next state in the new DFSM would be ϕ.

(3) (transitions from ϕ (if ϕ is a state in the DFSM).
All exits go back to ϕ.

(4) (accepting states)
A state in the new DFSM is called accepting if at least one of the letters in its name was an accepting state in the old NFSM.

Here's the rest of the construction.

To find the 0-exit from BCD, look at 0-exits in the NFSM:

```
     B -> none
     C -> none
     D -> none
```

Next state is ϕ.

To get the 1-exit from BCD, look at 1-exits in the NFSM:

```
      B -> E -> (lambda moves) E
      C -> none
      D-> A -> (lambda moves) A
```

Next state is AE

To get the 0-exit from AE:

```
     A -> B -> (lambda moves) B, C, D
     E -> C -> (λ moves) C, D
```

Next state is BCD

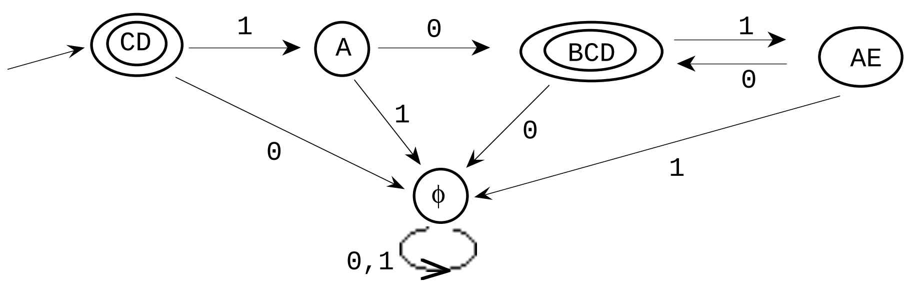
warning Don't forget the λ moves.
If you leave out the λ moves you will mistakenly think that next state is BC.
Messes everything up.

To get the 1-exit from AE:

```
      A -> none
      E -> none
```

Next state is ϕ.

And finally, C was the only accepting state in the old NFSM so the accepting states in the new DFSM state are the ones whose name contains the letter C (Fig 13).



WARNING

Don't forget the exits from ϕ

And don't forget to say which are the accepting states

FIG 13 DFSM equivalent to Fig 12

To illustrate that the DFSM in Fig 13 recognizes the same strings as the NFSM in Fig 12, I'll test the string 101.

Fig 14A shows the progress of the string through the NFSM in Fig 12. It's rejected because the final set of states, A,E, does not include an accepting state.

Fig 14B shows the progress of the same string through the DFSM in Fig 13. It's rejected because the unique final state, AE, is not an accepting state.

Note that the name of each *unique* next state in the DFSM is an amalgam of the names in the *set* of next states in the NFSM which is why the conversion works.

input		λ's	1	λ's	0	λ's	1	λ's	
next states	C	C, D	A	A	B	B,C,D	A,E,	A,E	reject

FIG 14A

input		1	0	1	
next state	CD	A	BCD	AE	reject

FIG 14B

example 2

Find a DFSM equivalent to (i.e., which recognizes the same strings as) the NFSM in Fig 15.

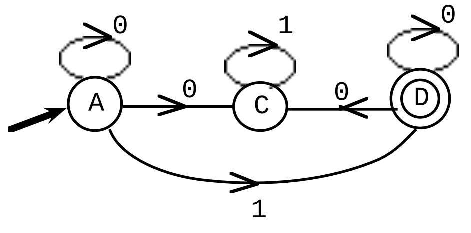


FIG 15

The starting state in the NFSM is A and there are no λ-moves from A so the starting state in the new DFSM will be named A also.

To get the 0-exit from A:
A –> A, C.
Next state is AC

Note: There are no lambda moves in the whole NFSM so I can leave them out.

To get the 1-exit from A:
A –> D.
Next state is D

```
To get the 0-exit from AC:
         A -> A, C
         C -> C
       Next state is AC

To get the 0-exit from D:
          D -> C, D
       Next state is CD

To get the 1-exit from D:
        D -> none
      Next state is ϕ

To get the 0-exit from CD:
          C -> none
          D -> C, D
       Next state is CD

To get the 1-exit from CD:
           C -> C
           D -> none
        Next state is C

To get the 0-exit from C:
            C -> none
           Next state is ϕ

To get the 1-exit from C:
            C -> C
           Next state is C
```

The DFSM is in Fig 16. The accepting states are D and CD.

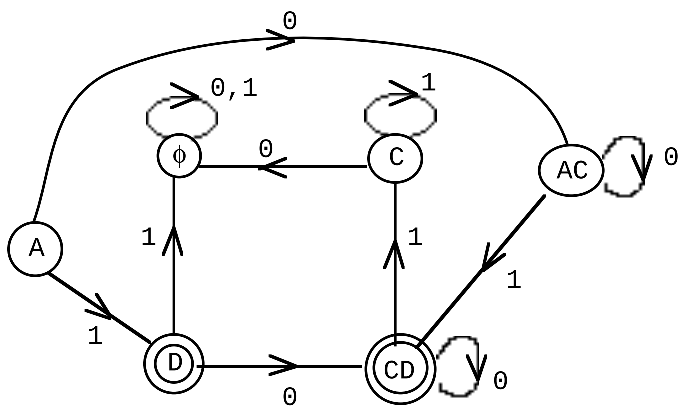


FIG 16

mathematical catechism (you should know the answers to these questions)

question 1

Why bother getting the DFSM recognizer in Fig 16 when it is messier than the NFSM recognizer in Fig 15.

answer The fact that we can always get a DFSM recognizer proves that we don't gain any recognition capability when we allow NFSMs.

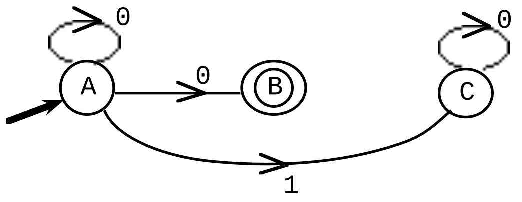
question 2

Why do we care about not gaining capability.

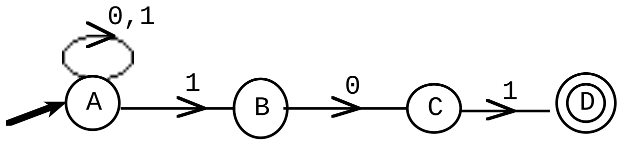
answer When we prove a theorem about NFSMs (what they can and cannot recognize) the same theorem will hold for DFSMs.

PROBLEMS FOR SECTION 5.3

1. Look at the NFSM in the diagram below and decide by inspection whether or not the NFSM recognizes (a) 000 (b) 01



2. What set of strings is accepted by the FSM in the diagram below.

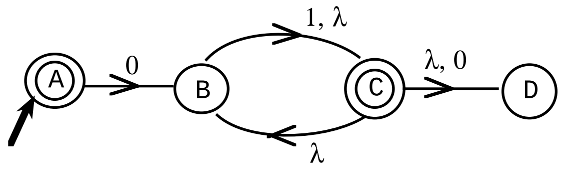


3. Let the alphabet be 0,1. Find FSMs (with accepting states) to recognize the following sets of strings. Try to make them as simple as possible by allowing them to be non-deterministic but do it without using λ moves.

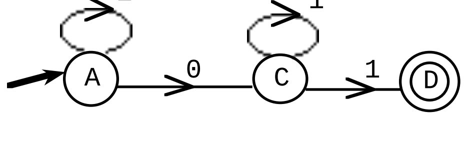
(a) strings ending in 000
(b) strings beginning with 000
(c) just the string 000
(d) strings containing at least one occurrence of 000
(e) strings containing no occurrences of 000
(f) strings containing an even number of 1's
(g) strings with no 1's
(h) strings containing exactly two 1's
(i) strings containing at least two 1's
(j) strings where the number of 1's is a multiple of 3

4. Let the alphabet be a, b, c.
Find FSM recognizers for
(a) the set of strings containing no occurrences of abc
(b) (harder) the set of strings containing exactly one occurrence of abc
Start with a FSM that ends up in an accepting state after the *first* occurrence of abc. And then tack on the FSM from (b) to prevent any more occurrences of abc.
(c) the set of strings containing at most one occurrence of abc.
Suggestion Adjust the FSM from part (b)

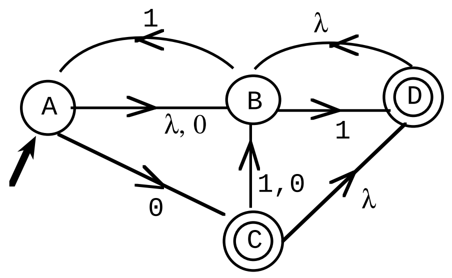
5.(a) Decide by inspection if 011, 0, 00, λ are accepted by the FSM in the diagram.
(b) Find the λ-adjacency matrix and the λ-reachability matrix.



6. Keep track of all next states as in Figs 8, 10, 11 to see if 01110 is accepted by the FSM in the diagram.



7. (a) Keep track of all next states to see if 11 and 00 are accepted by the FSM in the diagram.
   (b) Find the λ-adjacency matrix
   (c) Find the λ-reachability matrix.



8. A NFSM has starting state A, accepting state D, input alphabet a,b and the following a-adjacency matrix (an entry of 1 in row P, col Q indicates an a-transition from P to Q), b-adjacency matrix and λ-adjacency matrix.

a-adjacency

	A	B	C	D
A				
B			1	
C				
D				

b-adjacency

	A	B	C	D
A				
B				
C				1
D		1		

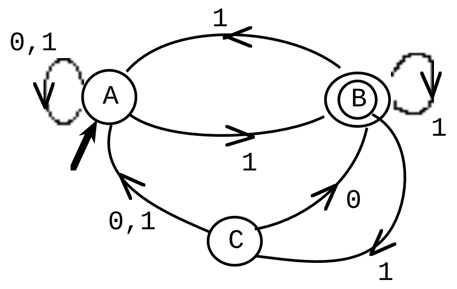
λ-adjacency

	A	B	C	D
A	1	1		1
B		1		
C		1	1	1
D				1

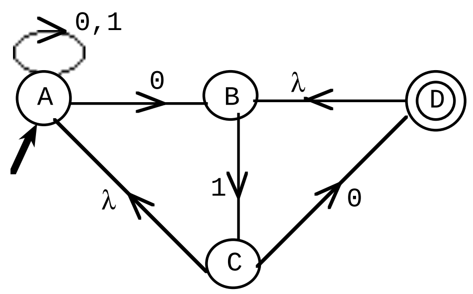
(a) Find the λ-reachability matrix.
(b) Test to see if the string aa is accepted.

9. Find an equivalent DFSM (using the converting algorithm from this section).

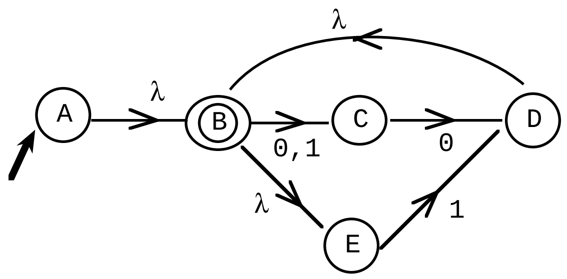
(a)



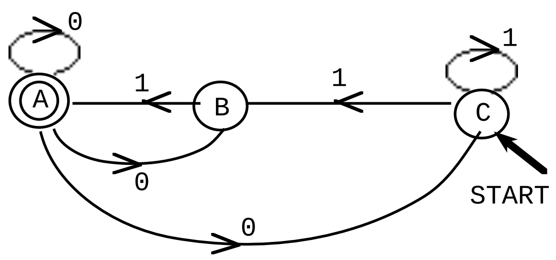
(b)



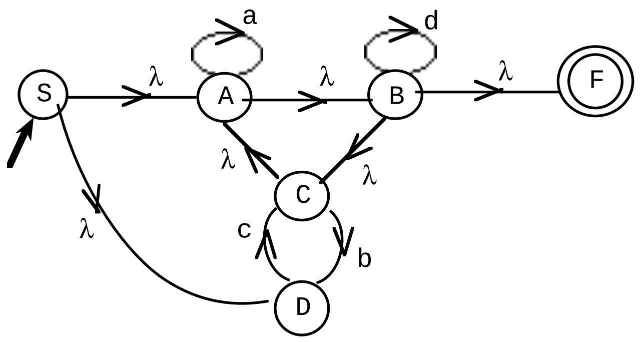
(c)



(d)

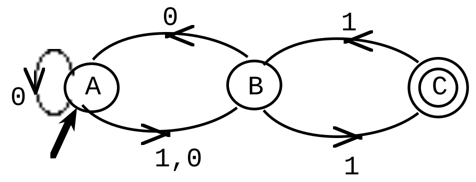


10. If a NFSM contains a λ-cycle then it can be simplified by inspection. See if you can do it for the NFSM in the diagram which has λ-cycle ABCA.



11. Suppose a FSM recognizes a bunch of strings.
The *reverse* of the FSM is a new FSM which recognizes precisely the reverse strings. For example if the first FSM recognizes cat then the reverse FSM recognizes tac.
Turns out that for every FSM there always is a reverse FSM.

Find the reverse of the FSM in the diagram which conveniently has only one accepting state.



12. Suppose you start with a NFSM and are going to find an equivalent DFSM using the method of this section.
(a) If the NFSM has 10 states what is the maximum number of states in the DFSM.
(b) If the NFSM has 10 states, no parallel tracks but does have shutdowns (i.e., there are never two possible next states but sometimes there is no next state) how many states will the DFSM have.

13. It can be shown that for every FSM with *more than one* accepting state there is an equivalent FSM with just *one* accepting state.
To illustrate the idea let's try to find the one-accepting-state version of the FSM with two accepting states in Fig A below.
(a) I tried to do by merging the two original accepting states (Fig B).
Explain why it doesn't work.
(b) Do it right (pretty easily) (think λ).

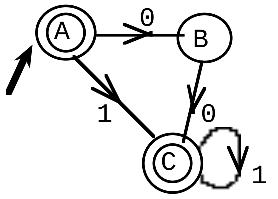


FIG A

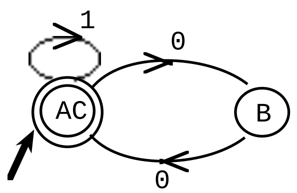


FIG B

SECTION 5.4 REGULAR SETS

concatenation of strings and sets

The concatenation of two *strings* is their juxtaposition; e.g., the concatenation of pq and abc (in that order) is pqabc. The concatenation of the empty string λ with abc (in either order) is just abc, since λ contains no symbols.

If A and B are *sets* of strings then the concatenation AB is the set obtained by concatenating all the strings in A with all the strings in B (in that order).

For example, let A = {a,b,cb} and B = {pq,r}. Then

AB = {apq, ar, bpq, br, cbpq, cbr}

BA = {pqa, ra, pqb, rb, pqcb, rcb}

BB = B^2 = {pqpq, pqr, rr, rpq}

the union of two sets of strings

The union of two sets of strings A and B is denoted A + B.
If A = {a, 00} and B = {pq,c} then A + B = {a, 00, pq, c}

notation I'll write aaaa as a^4 and 01 01 01 as $(01)^3$.

the closure of a set of strings

If A is a set of strings then the (Kleene) closure, denoted A^*, is the set you get by concatenating the strings in A with each other as often as you like in any order you like.

A^* always contains the empty string λ because the definition includes the possibility of concatenating no times

And A^* always contains everything that was in A because the definition allows concatenating just once..

For example, if A = {x} then $A^* = \{\lambda, x, x^2, x^3, \dots\}$

If A = {a, bc} then words in A^* are formed by stringing together a's and bc's in any order, as many of each as you like. So some strings in A^* are

λ, a, bc, abc, bca, $a^4(bc)^7a^5(bc)^8$, $bca^7(bc)^8$ etc.

notation (ambiguous but convenient)

The set {0} is denoted by 0.

So 0 stands both for the single string 0 and also for the set containing the string 0.

Similarly the set {1} is denoted by 1, the set {b} is denoted by b, etc.

01 can mean just the string 01. Or it can mean the set {01}, the set containing the string 01. It can also be thought of as the concatenation of the string 0 and the string 1.

example 1

0 + 1 is the union of the sets {0} and {1} so it's the set {0, 1}

01 + 0 is the union of {01} and {0} so it's the set {01, 0}

$0^* = \{\lambda, 0, 0^2, 0^3, \dots\}$.

And especially

$$\boxed{(0+1)^* = \{0,1\}^* = \text{set of } \textit{all} \text{ strings of 0's and 1's including the string } \lambda}$$

example 2

To get the members of $(a + b^*)^*$, first get

$$a + b^* = \{a, \lambda, b, b^2, b^3, \dots\}$$

and then do a lot of concatenating. The result is the set of *all* strings of a's and b's including λ; i.e., $(a + b^*)^*$ is the same as $(a + b)^*$

example 3

$(ab)^* c^* d$ is the set of strings of the form $(ab)^n c^m d$ where $n \ge 0$, $m \ge 0$.
Some of the members of the set are d, abd, $(ab)^5 d$, cd, $c^7 d$, abcd, $(ab)^4 c^{70} d$.

λ versus ϕ

λ stands for the empty string.
By abuse of notation it also stands for the set containing λ.
ϕ stands for the empty set.
The sets ϕ and λ are not the same. One way to see the difference is to look at the different recognizers for them. The FSM in Fig 1 recognizes the string λ, since it's in an accepting state with no input, and recognizes nothing else,. So Fig 1 recognizes the set λ. On the other hand, the FSM in Fig 2 is never in an accepting state (there are no entrances to B), it doesn't accept any strings so the set of strings accepted is ϕ.

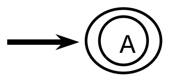


FIG 1 Recognizes the set λ

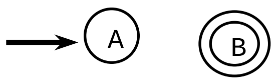


FIG 2 Recognizes the set ϕ

Furthermore λ and ϕ act differently when you concatenate or add with them:
$\lambda A = A\lambda = A$
$\phi A = A\phi = \phi$
$A + \phi = A$
$A + \lambda$ might or might not be A (it is A if and only if λ is in A to begin with)
$A - \phi = A$
$A - \lambda$ might or might not be A (it is A iff λ is not in A to begin with)

example 4

To get the members of $(ab^*)^*$, start with $ab^* = \{a, ab, ab^2, ab^3, \ldots\}$ and then concatenate as many times as you like in any order you like. The result is the string λ plus all strings beginning with a. For example, the string $a^5 b^7 a^4 ba$ is in the set since it's of the form $a^4 (ab^7)\ a^3 (ab) a$

The set of strings beginning with a, together with the string λ can also be written as $a(a + b)^* + \lambda$, i.e., $(ab^*)^* = a(a + b)^* + \lambda$

warning

Remember that the closure of any set contains the empty string λ.

regular sets

Let the input alphabet be 0,1 (similarly for any other alphabet).

> If a set of strings can be expressed in terms of the sets 0, 1, λ and a finite number of the operations +, * and concatenation then the set is called regular and the expression itself is a regular expression.
> Furthermore, the set ϕ is defined as regular.

Equivalently, here's a recursive definition of a regular set.

> (I) (the basic regular sets) The sets 0, 1, λ, ϕ are regular.
> (II) (building new regular sets from old)
>
> If r_1 and r_2 are regular then so are $r_1 + r_2$, $r_1 r_2$ and r_1^* .

All the sets in examples 1-4 are regular sets.

example 5

The set of strings ending in 10 is regular since it can be written as $(0 + 1)^*10$

The set of strings containing exactly one 1 is regular since it has the regular expression $0^*1\,0^*$

The set of strings containing at least one 1 is regular because it can be written as $(0 + 1)^*\ 1\ (0 + 1)^*$ [also as $0^*\ 1(0 + 1)^*$]

PROBLEMS FOR SECTION 5.4

1. Let A = {ab,c,d) and B = {0,11}. Find A + B, AB, B^2, and find some members of B^*.

2. Let the alphabet be 0,1. Describe in ordinary English the members of the following sets.

(a) 0^* (b) 00^* (c) 0^*0 (d) $(00)^*$ (e) $(0 + 1)^*\ 111$

(f) $0^*\ 111$ (g) $0^*10^*10^*$ (h) $(0^*10^*10^*)^*$ (i) 01 + 111

3. Two sets P and Q can overlap in any of the five following ways.

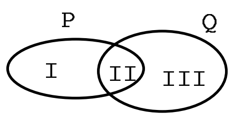


FIG A

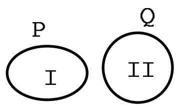


FIG B

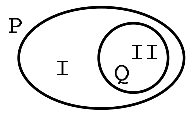


FIG C

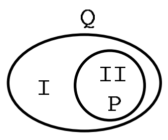


FIG D

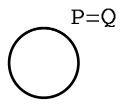


FIG E

Suppose the alphabet is a,b,c.
Which picture holds for each of the following pairs of P,Q.
And find some strings in subregions I, II, III in each case.

(a) $P = abc(abc)^*,\ \ Q = a^*(bc)^*$

(b) $P = a^* + (bc)^*,\ \ Q = (a + bc)^*$

(c) $P = (abc)^*,\ \ Q = a^* + (bc)^*$

4.(a) Is λ in $a^*b^*c^*$.

(b) Is $01^3\,01^3$ in $(1^*01)^*(11 + 0^*)$.

5. The expression $a^*(a + bb)^*$ can be simplified to $(a + bb)^*$ since the factor a^* in front doesn't contribute anything extra. Simplify the following if possible (by erasing part of the expression).

(a) $(a + bb)^* + bb + b$

(b) $(a + bb)^*bb$

(c) $\big((01)^* + 1\big)^*$

6. Let A and B be arbitrary sets of strings. Simplify if possible,

(a) $(A^*)^*$ (b) A^*A^* (c) $(A^* + B^*)^*$ (d) $AA^* + \lambda$

7. Let A, B, C be arbitrary sets of strings. True or False?

(a) $AA^* = A^*A$

(b) $AB = BA$

(c) $(A^*B^*)^* = (A + B)^*$

(d) $AA^* = A^*$

(e) $A(B + C) = AB + AC$

(f) $1A = A$

(g) $(A + B)^* = A^* + B^*$

8. Let the alphabet be 0,1. Find a regular expression for the set of strings with
(a) no 1's
(b) exactly one 1
(c) least one 1
(d) at most one 1
(e) exactly two 1's
(f) at least two 1's
(g) at most two 1's
(h) an even number of 1's (note that 0 is an even number)
(i) length at least 2 and alternating 0's and 1's such as 10101, 010, 1010, 01
(j) at least one occurrence of 1001

9. Let the alphabet be a,b,c.
Show that the following are regular by finding regular expressions for them.
(a) $\{ab^n a : n \ge 1\}$
(b) $\{a,b,c\}$
(c) set of strings with at least one occurrence of cba
(d) $\{\lambda, a, (bc)^n : n \ge 1\}$
(e) set of strings beginning with a and ending with b
(f) set of strings with second symbol b such as abcca, bb, ab, cbba.

10. (a) The set $(a + bb)^*$ contains λ (every closure contains λ). Find a regular expression for the new set obtained by removing the λ.
You would like to write $(a + bb)^* - \lambda$ but the definition of a regular expression does not allow minus signs. So think of some other way to do it.
(b) The set $a + bc^*$ does not contain λ. Find a regular expression for the new set obtained by including λ.

11. Why is $0^*(0 + 1)\,0^*(0 + 1)\,0^*$ not the set of strings containing at most two 1's and how close does it come.

12. I once asked on an exam for a regular expression for the set of strings in which each 0 is immediately followed by at least one 1, e.g., λ, 1, 11, 011, 1010111 but not 0, 10, 1001.
I got a lot of answers including the following. Which are correct? If one is wrong, explain (to the dope) why it's wrong.

(a) $(1 + 01)^*$

(b) $\left[1^*(01)^*\right]^*$

(c) $\left[1^*01\,1^*\right]^*$

(d) $1^* + (011^*)^*$

(e) $1^*(011^*)^*$

(f) $1^*(01)^*1^*$

SECTION 5.5 KLEENE'S THEOREM PART I

Kleene's theorem

Regular sets are the language of FSMs in the following sense. (By FSM I mean DFSMs, in particular, deterministic FSM with accepting states.)

(I) For every regular set of strings there is a FSM recognizer.
(II) Conversely, for every FSM, the set of strings it recognizes is a regular set.

In other words, a set of strings is regular if and only if it is the set of strings recognized by some FSM.

footnote
Kleene's theorem almost holds for the FSMs with outputs from Section 5.1. The quibble is that a FSM with outputs cannot recognize the empty string λ, so for example, a FSM with outputs can recognize $(01)(01)^*$ but not $(01)^*$

Kleene's construction method to prove (I)

Although the theorem is about DFSMs, for the proof it's easier to find NFSM recognizers with λ-moves allowed. This is sufficient since I showed in Section 5.3 that for every NFSM with λ-moves there is an equivalent DFSM.

Furthermore, for convenience the proof will use NFSMs which have just one accepting state (which won't be the starting state). This is sufficient since for every FSM with more than one accepting state there is an equivalent NFSM with just one accepting state (Problem 14 in Section 5.3).

In other words I'll show that for every regular set of strings there is a λNFSM with just one accepting state (that isn't the start) that recognizes that set.

I'll show how to construct such a FSM for any regular set by showing two things:

(1) Each of the basic regular sets 0, 1, λ, ϕ has such a FSM recognizer.
(2) Once you have such FSM recognizers for the regular sets r_1 and r_2 you can construct such FSM recognizers for the new regular sets $r_1 + r_2$, $r_1 r_2$ and r_1^* .

The proof of (1) is in Figs 1–4 which show FSMs (with just one accepting state that isn't the starting state) that recognize the sets 0, 1, λ, ϕ.

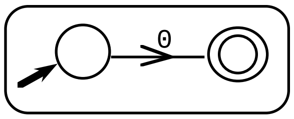


FIG 1
Recognizes 0

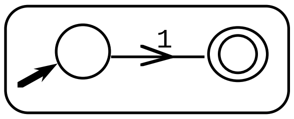


FIG 2
Recognizes 1

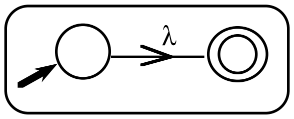


FIG 3
Recognizes λ

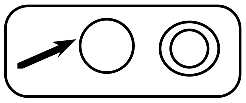


FIG 4
Recognizes ϕ

To prove (2), assume you already have FSMs which recognize r_1 and r_2 (Fig 5).

Recognizes r_1

Recognizes r_2

FIG 5

To make a FSM which recognizes $r_1 + r_2$, change the accepting states in the r_1 and r_2 recognizers to non-accepting and then hook them up in parallel with a new starting state and a new accepting state (Fig 6).

footnote The thing in Fig 6 labeled "r1-recognizer" really is an "r1-recognizer but with its accepting state changed to a non-acc state". But that label was too long to write.

The procedure is illustrated in Fig 7 which shows the recognizers for 0 and 1 from Figs 1 and 2 (with their accepting states made non-accepting) linked in parallel to get a FSM recognizing 0 + 1 (not the most efficient FSM recognizing 0+1 but who cares).

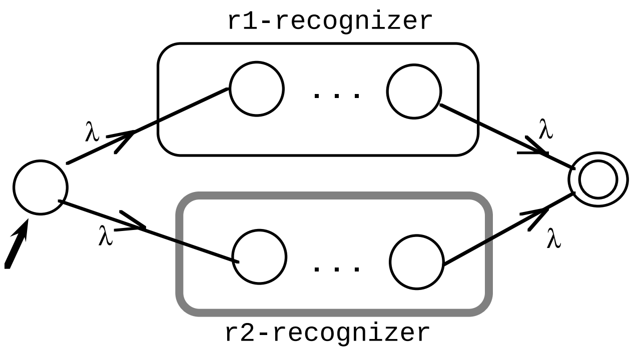


FIG 6 Recognizes $r_1 + r_2$

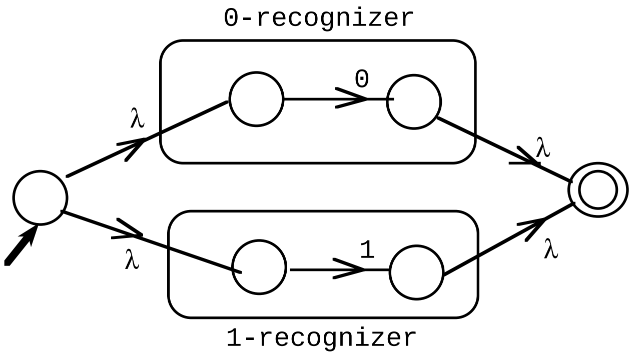


FIG 7 Recognizes 0+1

To find a FSM which recognizes $r_1 r_2$, change the accepting state in the r_1 recognizer to non-accepting and hook up the two recognizers in series (Fig 8).

The procedure is illustrated in Fig 9 which shows the recognizers for 0+1 and 0 from Fig 7 and Fig 1 hooked in series to get a recognizer for (0+1)0.

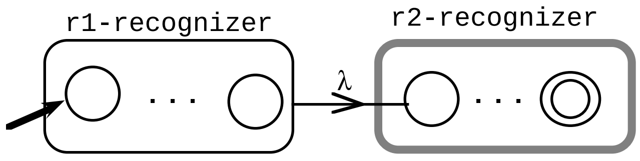


FIG 8 Recognizes $r_1 r_2$

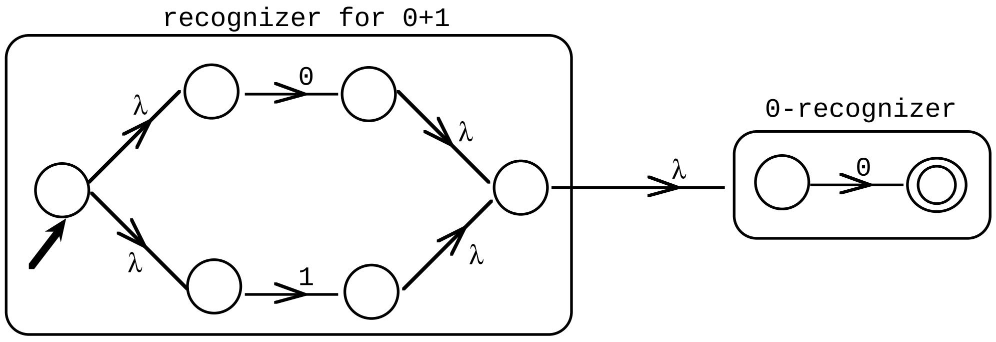


FIG 9 Recognizes (0+1)0

warning

Don't merge the "last" state of the 0+1 recognizer with the "first" state of the 0-recognizer, like Fig 9A.

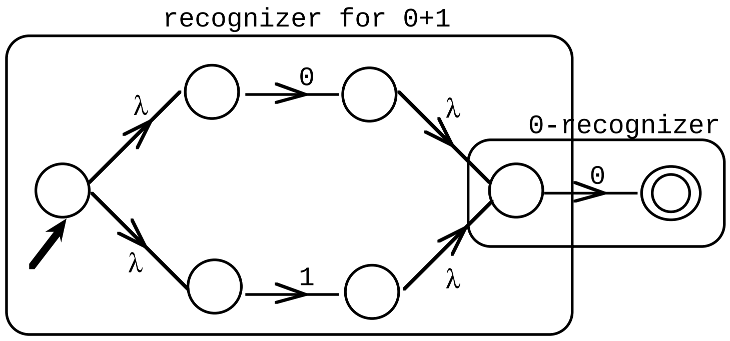


FIG 9A

In this instance, it actually does make a correct recognizer for (0+1)0 *but* i doesn't work in general (see problem #4) so it isn't the way Kleene did it; i.e., it isn't like Fig 8.

To convert the r_1 recognizer to a recognizer for r_1^* (Fig 10) add a new start and a new accepting state connected by a λ-move (so that λ is recognized) and hook up the original machine "circularly" by adding a λ-move from the old accepting state to the old start.

The procedure is illustrated in Fig 11 which shows the recognizer for 0 + 1 adjusted to get a recognizer for $(0+1)^*$.

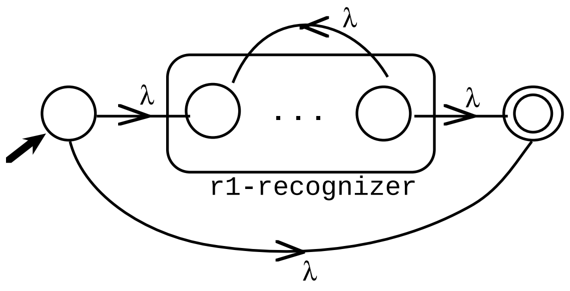


FIG 10 Recognizes r_1^*

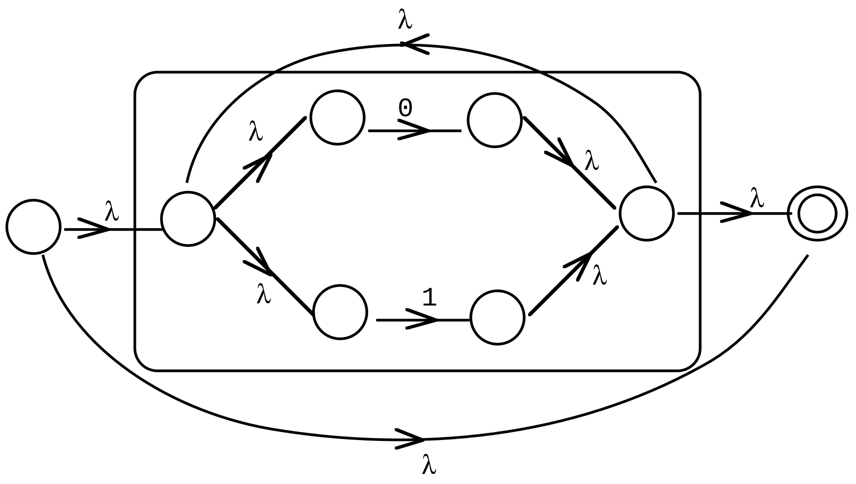


FIG 11 Recognizes (0+1)*

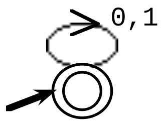


FIG 12

These construction methods are foolproof and are the guarantee that every regular set has a FSM recognizer even though they do not necessarily produce the *simplest* recognizer for a regular set. Fig 12 shows a recognizer for (0+1)* (the set of all binary strings) that is much simpler than the one in Fig 11.

example 1

Figs 13—19 show the Kleene construction of a FSM to recognize (a + b*c)* starting with the FSMs for a,b,c.

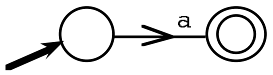


FIG 13 Recognizes a

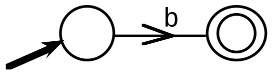


FIG 14 Recognizes b

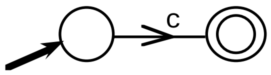


FIG 15 Recognizes c

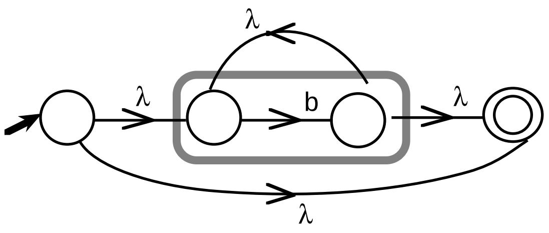


FIG 16 Recognizes b*

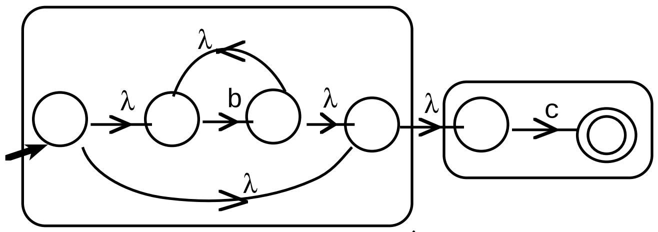


FIG 17 Recognizes b*c

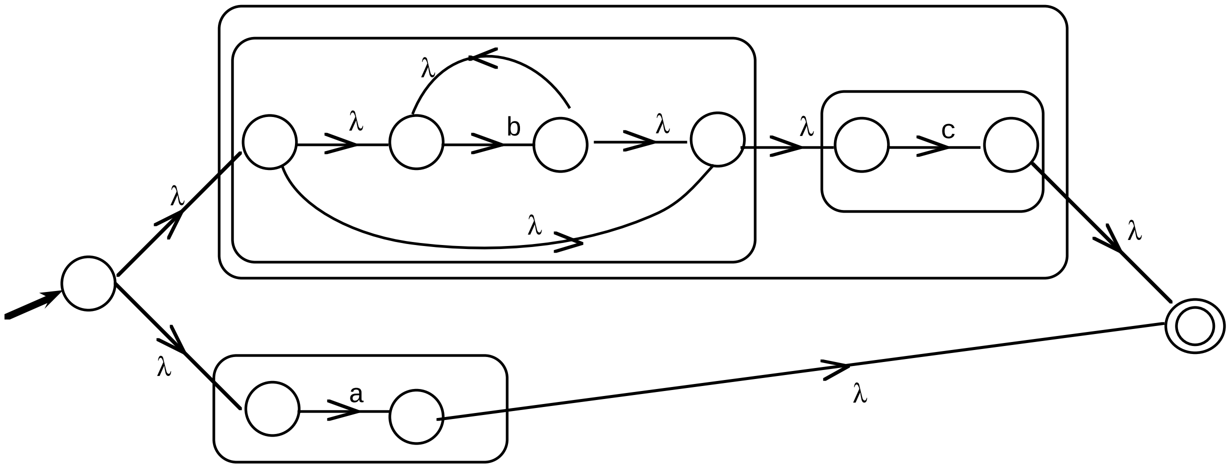


FIG 18 Recognizes a + b*c

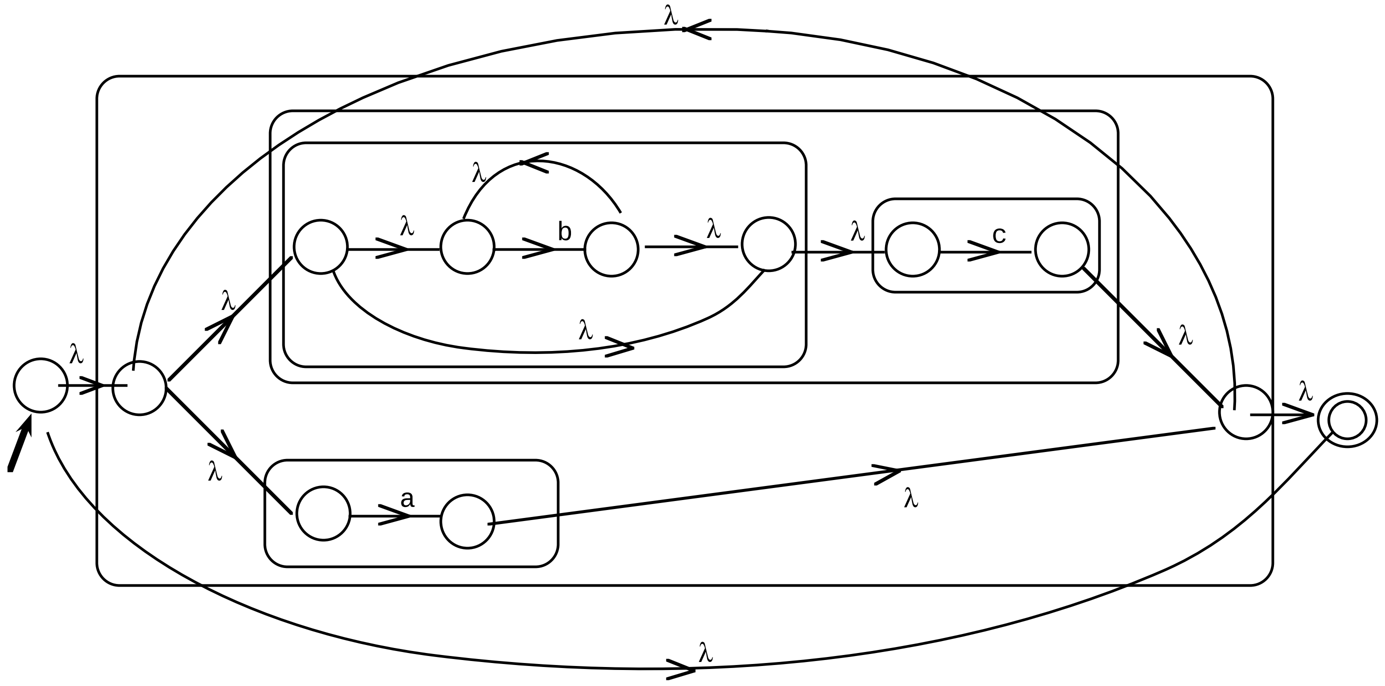


FIG 19 Recognizes (a + b*c)*

mathematical catechism (you should know the answers to these questions)

question

What good is the Kleene construction method when you end up with a monster FSM.

answer

The Kleene construction is part of the proof that every regular set has a FSM recognizer. Kleene is for showing the existance of a recognizer, not for getting the best recognizer.

PROBLEMS FOR SECTION 5.5

1. Use the Kleene construction method to find a FSM recognizer for each set.
   (a) $(01)^*$ (b) $(01)^*1$ (c) $(01)^* + 1$

2. (a) Find recognizers for the following sets using the Kleene construction method and then find simpler ones by inspection.
   (i) $1^* + 1^*0$ (ii) $01^* + \lambda$

(b) What's the point of the Kleene construction when it produces such an unsimple FSM?

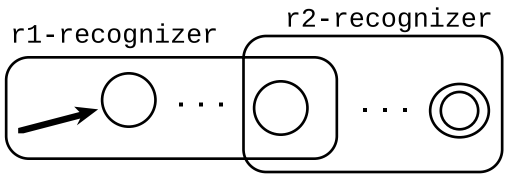
3. Write a regular expression for each set and, by inspection, find a FSM recognizer without λ-moves.

   (a) $\{(01)^n 1^{2m} : n \geq 0, m \geq 0\}$

   (b) $\{(01)^n 1^{2m} : n \geq 1, m \geq 1\}$

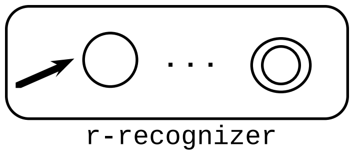
   (c) $\{0^n 1 0^m : n \geq 0, m \geq 0\} \cup \{0^k : k \geq 3\}$

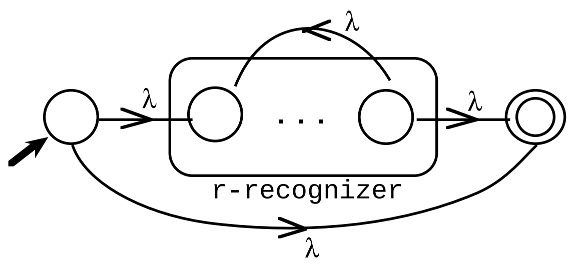
4. Fig 8 showed Kleene's recognizer for $r_1 r_2$. Here's a proposed streamlined version which merges the accepting state of r_1 with the starting state of r_2 instead of putting in a λ-move from one to the other. Does it work?



Proposed recognizer for $r_1 r_2$

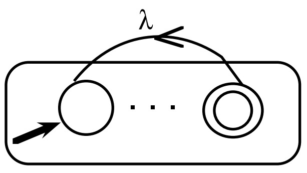
5. On the left below is a hypothetical recognizer for r and on the right is Kleene's recognizer for r* (this is a repeat of Fig 10).



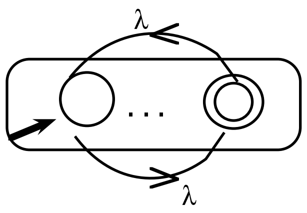


Here are some *other*, sometimes simpler, proposed recognizers for r*.
Do they work? If not, why not?

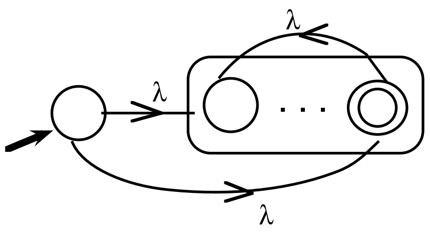
(a)



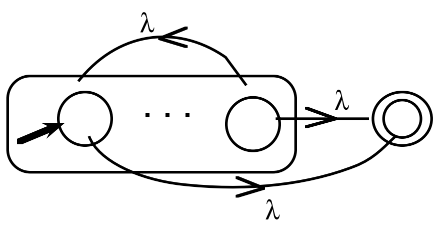
(b)



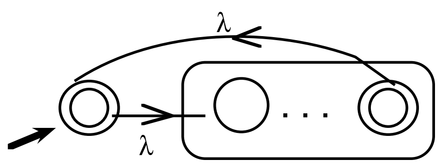
(c)



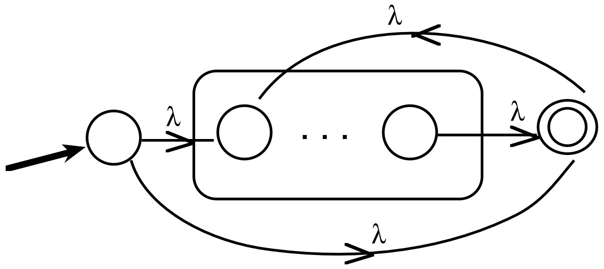
(d)



(e)

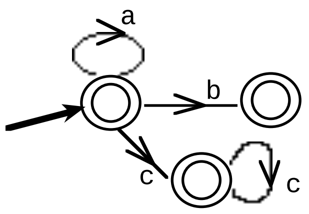


(f)



6. The set $a^*b + c^*$ is a regular set so it must have a FSM recognizer.

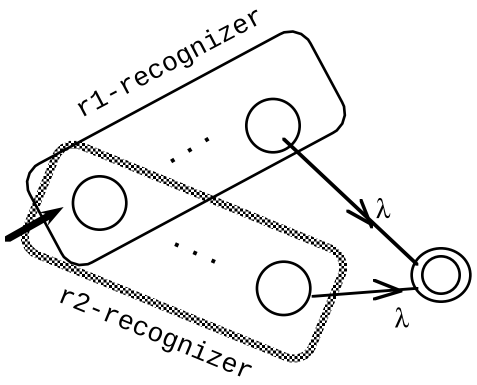
(a) What's wrong with the proposed recognizer below



(b) Find a correct FSM recognizer without λ moves and with as few states as possible (by inspection).

7. By inspection, find a FSM recognizer without λ moves and with as few states as possible for
(a) a^*bc^* (b) $(ab + c)^*$

8. Fig 6 showed Kleene's recognizer for $r_1 + r_2$. Here's a proposed streamlined version which merges the two starting states instead of putting in a new start. Show why it doesn't work.



SECTION 5.6 KLEENE'S THEOREM PART II

Part II of Kleene's theorem says that the set recognized by a FSM is regular.
First note that the FSM's in Figs 1 and 2 recognize λ and ϕ respectively. So to keep part II true, with no exceptions, in Section 5.5 we had to (artificially) define λ and ϕ to be regular sets.

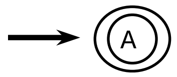


FIG 1 Recognizes the set λ

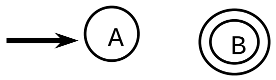


FIG 2 Recognizes the set ϕ

In this section. I'll give a method for actually finding a regular expression for the set recognized by a FSM which proves part II. First some preliminaries.

factoring/combining rule

Let A, B, C, D be sets of strings.
Remember that

AB = set of strings of the form ab where a is in A and b is in B
A + B = A union B = set of strings in either A or B
0A = set of strings of the form 0a where a is a string in A
A1 = set of strings of the form a1 where a is a string in A
$A\lambda = A$
$A\phi = \phi$

Then here are some algebra rules.

AB + AC = A(B + C)
AB + CB = (A + C)B
(A + B)(C + D) = AC + BC + AD + BD

For example, 01A + 0*A can combine to (01 + 0*)A

why the algebra rules work

I'll check on one of them. Let

A = {p,q,r}, B = {a,b}, C = {v}

Then
AB = {pa, pb, qa, qb, ra, rb}
AC = {pv, qv, rv}
B + C = {a, b, v}

Now you can see that A(B+C) and AB+AC are both {pa, pb, pv, qa, qb, qv, ra, rb, rv}.
So A(B + C) = AB + AC

the P*Q solving rule

Let P and Q be sets of strings. Suppose P does not contain λ.

(1) The equation X = PX + Q has the unique solution $X = P^*Q$

(2) (special case of (1) where Q is the null set)

The equation X = PX has the unique solution $X = P^*\phi = \phi$

footnote

If P contains λ then X = P*Q is still a solution to X = PX + Q but it may not be the *only* solution. For example look at the equation

$$X = 0^*X + 1$$

In this case, P = 0*, which *does* contains λ, and Q = 1. One solution is

$$X = P^*Q = 0^{**}1 = 0^*1.$$

but it isn't unique. Some other solutions are

$$X = 0^*1 + 0^*11,$$
$$X = 0^*1 + 0^*101$$

and more generally

$$X = 0^*1 + 0^*w$$

where w is any word. But don't worry about it since this won't happen in any of the equations you have to solve; you'll always find that P does *not* contain λ.

half proof of the P*Q rule

I'll show that X = P*Q *is* a solution of X = PX + Q, whether or not P contains λ. The proof that X = P*Q is the *unique* solution is much harder so I'm leaving it out.

Saying that P*Q is a solution to the equation X = PX + Q means that if we take P*Q, multiply in front by (concatenate with) P and add (take union with) Q, you are back where you started; i.e., back to P*Q. Let's keep track to see.

P*Q	Contains Q strings with P strings in front (as many P strings as you like) It also contains plain Q strings (using λ from P*).
PP*Q	Not much different. Nothing new is created by putting *more* P strings in front. But something might be lost, namely all the plain Q strings. If P contains λ then you don't lose the plain Q strings. If P doesn't contain λ then you do lose the Q strings.
PP*Q + Q	In case the Q strings *were* lost, put them back in. Now we've got Q strings with P strings in front *and* plain Q strings.

QED

solving the state equations of a FSM to find the regular expression for the set recognized (this proves Kleene's theorem part II)

I'll illustrate the procedure with the FSM in Fig 3.

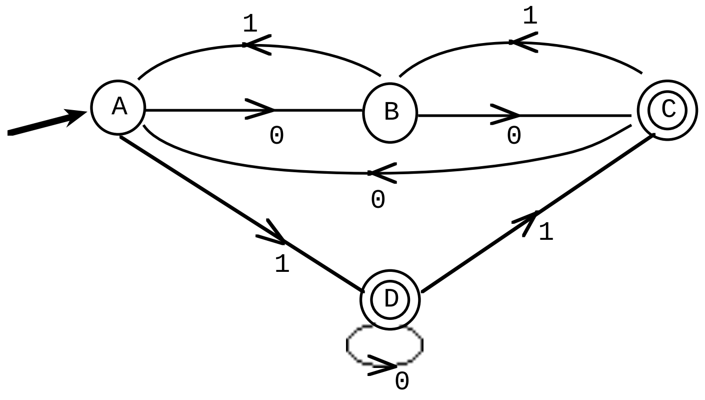


FIG 3

Let A be the set of words which end up in an accepting state starting from state A
Let B be the set of words which end up in an accepting state starting from state B
Let C be the set of words which end up in an accepting state starting from state C
Let D be the set of words which end up in an accepting state starting from state D

I want to find the set A since the FSM in Fig 3 has starting state A. But I'll use the sets B,C,D along the way.

Look at an arbitrary string in the set A, i.e., a string that would end up in an accepting state starting from state A.

If it begins with 0 and you start in state A then the next state is B and the *rest of the word* leads to an accepting state *from B*. So the rest of the word is in the set B and the original string is in 0B.

If it begins with 1 and you start in state A then the next state is D and the rest of the word leads to an accepting state from D. So the rest of the word is in the set D and the original string is in 1D.

Each string in A begins with either 0 or 1 so the strings in the set A are either in 0B or in 1D. So

(3) $A = 0B + 1D$

Similarly

(4) $B = 0C + 1A$

(5) $C = 0A + 1B + \lambda$

(6) $D = 0D + 1C + \lambda$

warning
Don't leave out the λ's in (5) and (6).
They are there because C and D are accepting states.
The λ in (5) reflects the fact that λ leads from C to an accepting state since C itself is accepting.
See problem 11 for what happens if you do leave out λ's

The equations in (3)−(6) are called the *state equations* for the FSM in Fig 3. We want to solve for A since the starting state in Fig 3 is A. To do this, use substitution, factoring and the solving rule in (1). Here is one solution.

I'm going to substitute for B and D in (3) until (3) looks like

$$A = (0,1 \text{ stuff})A + 0,1 \text{ stuff}$$

and then use the P^*Q rule to solve for A.

I'll begin by solving (6) for D using the P^*Q rule with $P = 0$, $Q = 1C + \lambda$:

$$D = 0^*(1C + \lambda)$$

Then substitute in (3):

(7) $A = 0B + 10^*(1C + \lambda)$
$= 0B + 10^*1C + 10^*$ (*Note* $10^*\lambda = 10^*$)

Substitute the B value from (4) into (5):

$$C = 0A + 1(0C + 1A) + \lambda$$

Collect terms:

$$C = 10C + 0A + 11A + \lambda$$

And use the P^*Q rule to solve for C:

(8) $C = (10)^*(0A + 11A + \lambda)$

Substitute the C value from (8) into (4).

(9) $B = 0(10)^*(0A + 11A + \lambda) + 1A$

Substitute (8) and (9) into (7):

$$A = 0\left[0(10)^*(0A + 11A + \lambda) + 1A\right] + 10^*1\left[(10)^*(0A + 11A + \lambda)\right] + 10^*$$

And collect terms:

$$(10) \qquad A = \left[00(10)^*(0+11) + 01 + 10^*1(10)^*(0+11)\right]A + 00(10)^* + 10^*1(10)^* + 10^*$$

Use the P^*Q rule to solve for A to get the final answer

$$A = \left[00(10)^*(0+11) + 01 + 10^*1(10)^*(0+11)\right]^*\left[00(10)^* + 10^*1(10)^* + 10^*\right]$$

If you do the substitutions in a different order you may get a different expression for A. The expression is not unique.

As a partial check you can trace some strings in A to see that they do reach an accepting state. The beginning of a string in the set A comes from the closure of the set

$$\left\{\ 00(10)^*(0 + 11),\quad 01,\quad 10^*1(10)^*(0 + 11)\ \right\}$$

In other words a string in A begins with blocks of

01's
$00(10)^*(0 + 11)$'s
$10^*1(10)^*(0 + 11)$'s

so here's a typical beginning:

$$(01)^n\ \ 00(10)^m(0 \text{ or } 11)\ \ 10^j1(10)^k\ (0 \text{ or } 11)$$

Then the typical string can continue with a word in

$$00(10)^* \ \text{ or }\ 10^*1(10)^* \text{ or } 10^*$$

The table in Fig 4 traces the route of this typical string from set A (allowing for three possible string endings) and shows that it *does* end up in an accepting state starting from state A.
(This is at best a partial check; it tries to show that the strings in A *are* recognized but it doesn't check that A includes *all* strings recognized.)

in		$(01)^n$	00	$(10)^m$	0 or 11	10^j	1	$(10)^k$	0 or 11
next	A	A	C	C	A	D	C	C	A

00	$(10)^*$
C	C

10^*1	$(10)^*$
C	C

1	0^*
D	D

FIG 4

warning
1. Don't forget that the state equation corresponding to an *accepting* state has a λ term
2. Solve for the right letter, the one that was the starting state in the FSM.

difference of two sets

If A and B are sets then A − B is the set of things in A that are not also in B. Fig 5 shows some pictures of A − B in various circumstances.

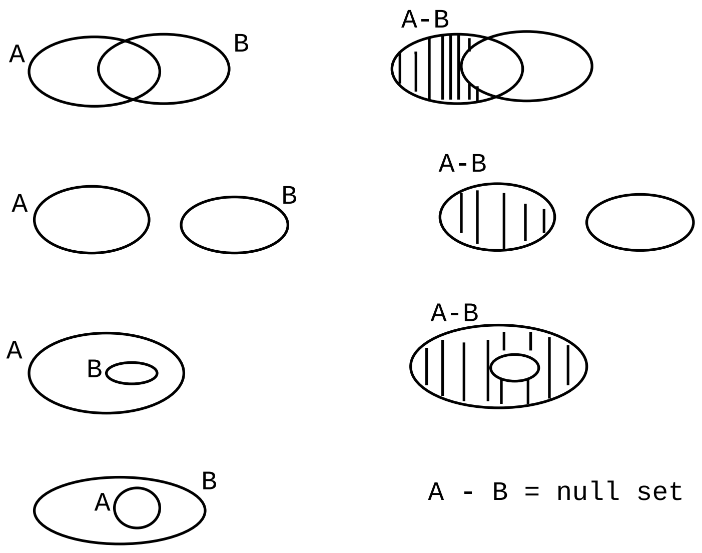


FIG 5

corollary of Kleene's theorem: unions, intersections, complements, differences of regular sets

Let r_1 and r_2 be regular sets of strings using some fixed alphabet. Then the following sets are also regular:

$$r_1 \cup r_2, \quad \overline{r_1}, \quad r_1 \cap r_2, \quad r_1 - r_2$$

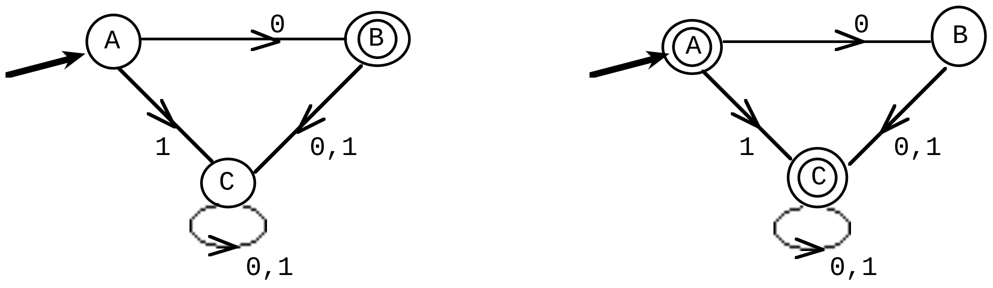
proof that the union of regular sets is regular

$r_1 \cup r_2$ is the same as $r_1 + r_2$ and by definition of regular sets if r_1 and r_2 are regular then so is $r_1 + r_2$

proof that the complement of regular sets is regular

By Kleene's theorem part I there is a FSM recognizing the regular set r_1. Find a *deterministic* version (can always do that by Section 5.3) so that every string follows exactly one track; the words in r_1 lead to an accepting state and the words in $\overline{r}_1$ lead to a non-accepting state. Change every accepting state to non-accepting and every non-accepting state to accepting. The new FSM recognize $\overline{r}_1$. Therefore by Kleene's theorem part II, $\overline{r}_1$ is regular.

As an example, Fig 6 shows a DFSM recognizing 0 and the new machine recognizing everything (including λ) except 0.



Recognizes 0 FIG 6 Recognizes $\overline{0}$

proof that the intersection of regular sets is regular

Write the interesection in terms of complements and unions:

$$\overline{r_1 \cap r_2} = \overline{r_1} \cup \overline{r_2} \qquad \text{(DeMorgan's law of set theory)}$$

$$r_1 \cap r_2 = \overline{\overline{r_1 \cap r_2}} = \overline{\overline{r_1} \cup \overline{r_2}}$$

But as already proved about complements and unions, if r_1 and r_2 are regular then so are $\overline{r_1}$ and $\overline{r_2}$ and so is $\overline{r_1} \cup \overline{r_2}$ and so is $\overline{\overline{r_1} \cup \overline{r_2}}$. So $r_1 \cap r_2$ is regular.

proof that the difference of regular sets is regular

See problem 7.

summary of how to show that a set is regular

You now have three ways to show that a set, r, of strings is regular.

1. Find a reg expression for r (then the set is regular by definition).
2. Find a FSM recognizer for r (then the set is reg by Kleene part II).
3. Show that r is the complement, union, intersection or difference of regular sets (then r is regular by the corollary above).

How do you know which one to use? There's no absolute rule.

You might want to use method 3 if the strings in r must have *two* patterns (then r is an intersection) or if the strings in r must *not* have a certain pattern (then r is a complement) or if the strings in r have one pattern but must avoid another (then r is a difference) or if the strings in r have a choice of patterns (then r is a union/sum).

example 1

Let
A = set of strings ending in 10 *and* containing at least two 1's
B = set of strings containing at least two 1's but *not* ending in 10
C = set of strings *not* ending in 10
D = set of strings ending in 10 *or* containing at least two 1's

Show that they are regular sets.

solution

The strings in A must have two patterns. So I'll think of A as an intersection. In particular let

E = set of strings ending in 10
F = set of strings containing at least two 1's

Then A = E $\cap$ F.
E is regular since E = $(0+1)^*10$.
Alternatively, E is regular since Fig 7 shows a FSM recognizer for E.
And F is regular since F = $(0 + 1)^*1(0 + 1)^*1(0 + 1)^*$.
Alternatively, F is regular since Fig 8 is a FSM recognizer for F.
So A is regular since the intersection of regular sets is reg.

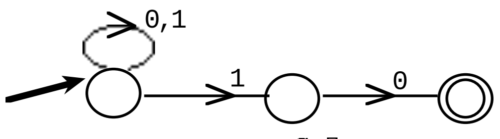


FIG 7

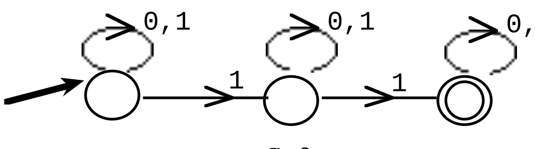


FIG 8

The strings in B must have one pattern but avoid a second. With the E and F as above, B = F − E.
E and F are regular sets, so B is also reg.

The strings in C must avoid a pattern. In particular, C = $\overline{E}$. We already know that E is regular, so C is also reg.

The strings in D have a choice of patterns. In particular, D = E + F (the union of E and F). E and F are regular, so D is also reg. QED

clarification

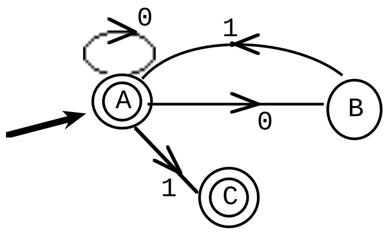
A regular expression can't contain a minus sign or a complement sign. The set $\overline{(0+1)^*10}$ is regular because $(0+1)^*10$ is reg and the complement of a regular set is regular. But the expression $\overline{(0+1)^*10}$ itself is not a regular expression for the set.

Similarly, the set $(01)^* - 0101$ is a regular set because the difference of two regular sets is regular but the expression $(01)^* - 011$ itself is not a regular expression for the set.

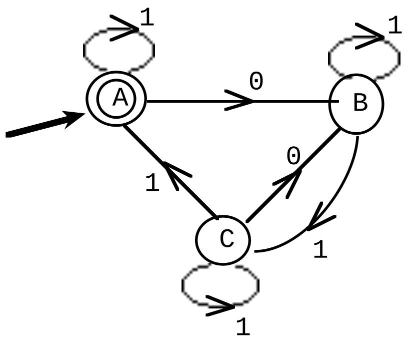
PROBLEMS FOR SECTION 5.6

1. For the following FSMs it may be possible to find a regular expression by inspection for the set of strings recognized. But use state equations for practice (and why not try checking your answer a little).

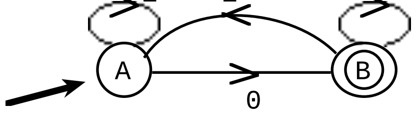
(a)



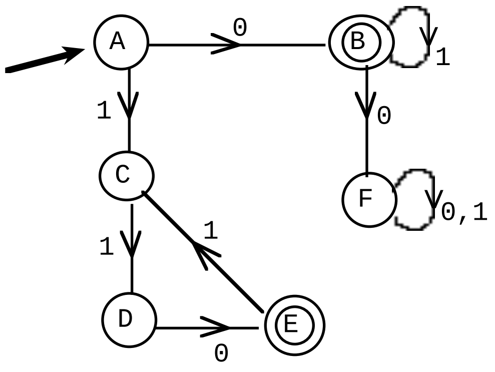
(b)



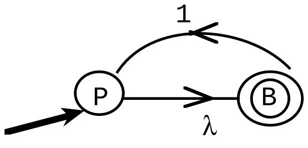
(c)



(d)



(e)



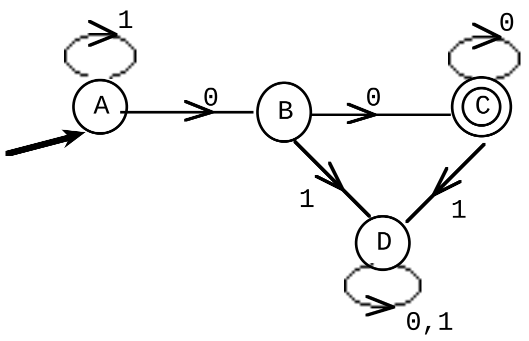
2. Look at the state equations

$$A = 0A + 0B + 1C + \lambda$$
$$B = 1A + 0B$$
$$C = 0C + 1B$$

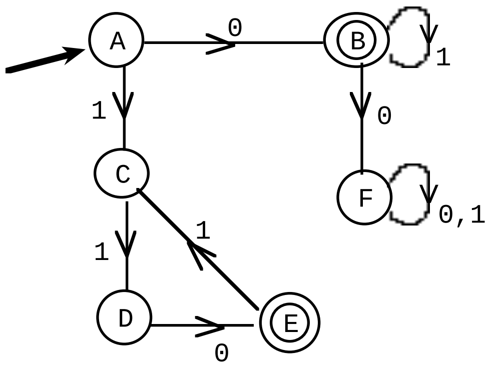
Can you tell which is the starting state? which are the accepting states ?

3. Solving state equations is foolproof and is the guarantee that the set recognized by a FSM is regular but sometimes you can get a regular expression for the set recognized by a FSM by inspection. Try it for these FSMs.

(a)

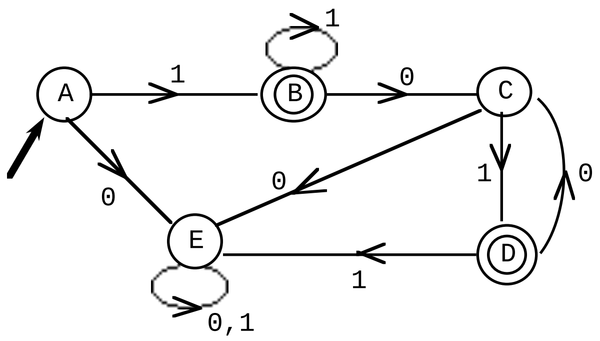


(b)

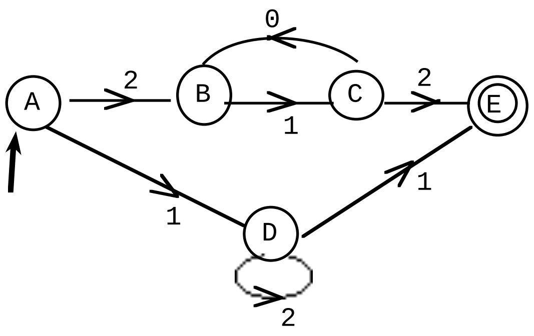


(c) Same as (b) but make the starting state accepting.

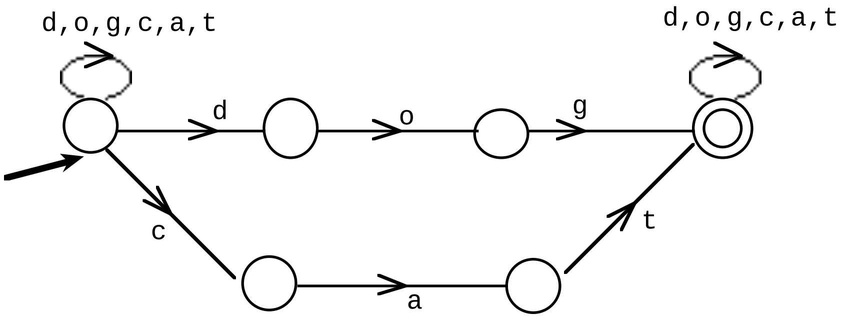
(d)



(e) (the alphabet is 0,1,2)



4. By inspection, find a regular expression for the set recognized by the FSM below and describe the set in words (the alphabet is d,o,g,c,a,t).



5. Let the alphabet be a,b,c,d. Show that the following are regular sets.

r_1 = set of strings with at least one occurrence of abc

r_2 = set of strings with at least two occurrences of abc

r_3 = set of strings with at least three occurrences of abc

r_4 = set of strings with no occurrences of abc

r_5 = set of strings with exactly one occurrence of abc

r_6 = set of strings with exactly two occurrences of abc

6. Show that the following are regular sets.

(a) r = set of strings containing 001 and containing an even number of 1's
(b) s = set of strings containing 001 and not containing 0000
(c) t = set of strings not ending in 0010

7. I already proved that complements and intersections of reg sets are reg.
Now give prove that the difference of reg sets is reg by writing the difference in terms of intersections and complements.

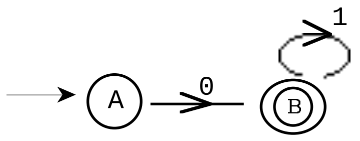
8. Let r be a set of strings and let s be the set of reversed strings.
For example if adc is in r then cda is in s.
Show that if r is regular then s is regular.

9. (a) What theorem does the Kleene construction stuff in the preceding section prove.
(b) What theorem does the state equation stuff in this section prove.

10. Let r be the set of strings containing *no* occurrences of 1101.

Then $\overline{r}$ is the set of strings containing at least one occurrence of 1101 and it is regular because it has the reg expression $(0 + 1)^*\ 1101\ (0 + 1)^*$.
So r is regular. But I don't see how to get a regular expression for r by inspection.
Outline some steps that could be implemented to get it (don't actually do the implementing) (too tedious).

11. By inspection, the set of strings recognized by the FSM in the diagram below is 01^* but we're going to look at state equations anyway for practice.



The correct state equations are

(1) $A = 0B$
(2) $B = 1B + \lambda$

Suppose you make a mistake and leave out the λ in the second equation so that your (incorrect) state equations are

(3) $A = 0B$
(4) $B = 1B$

(a) Predict (just think!) the answer you should get if you solved the wrong state equations in (3) and (4) for A.
(b) Now solve the wrong equations and see what you actually do get.
(c) Solve the *correct* state equations in (1) and (2)

SECTION 5.7 SOME NON-REGULAR SETS (SETS THAT NO FSM CAN RECOGNIZE)

example 1

Let A be the set of strings of the form $0^n 1^n$, $n \geq 0$. In other words,

$$A = \{\lambda, 01, 0^2 1^2, 0^3 1^3, \ldots\},$$

the set of strings consisting of any number of 0's (including none of them) followed by an equal number of 1's. I'll show that the set is not regular.

This will be a proof by contradiction)

Suppose the set is regular. (I'm hoping to get a contradiction.)

Then it has a FSM recognizer (by Kleene part I).

Let N be the number of states in the FSM.

Look at the string $0^N 1^N$. The string is in A so it is recognized by the FSM. So there is a track (Fig 1) leading to an accepting state. In the 0^N part of the table there are N+1 next-state boxes, one for each of the 0's plus a starting state. But there are only N states. More boxes than states. So the boxes in the 0 part of the track must have a repeated state, say Z. So the track includes a loop (Fig 2).

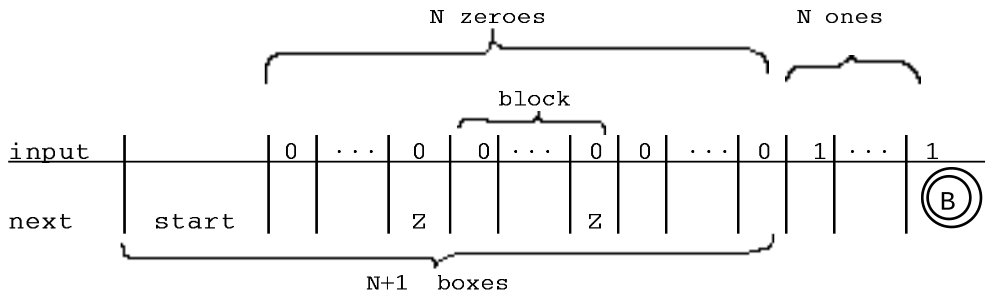


FIG 1

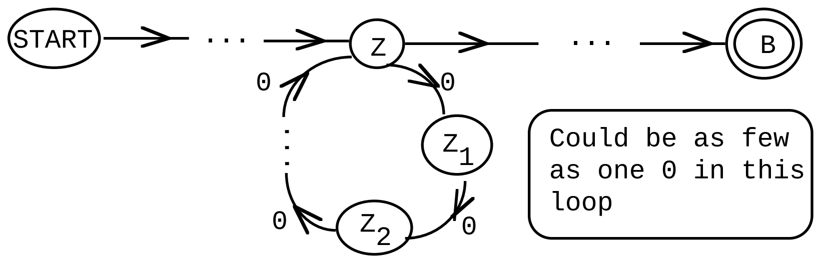


FIG 2

The block of 0's in the loop might contain as few as one 0 if the repeated states are in adjacent boxes in Fig 1. Or it might contain as many as all N of the 0's.

Make a new string by deleting the block of 0's from the old string. The new string still reaches B (same track but without the loop) so it is still accepted.

But the new string has fewer 0's than 1's so it should *not* be accepted. Contradiction!

So A couldn't have been regular after all. QED

example 1 repeated

Instead of *deleting* the block in Fig 1 you could get a new string by *duplicating* the block (duplicating the block once so that it *appears* twice is called pumping the block twice) to get the new string

$$\underbrace{0 \ \ldots\ 0 \text{ block block } 0 \ \ldots\ 0}_{\text{more than N zeroes}} \quad \underbrace{1 \ \ldots\ 1}_{\text{N ones}}$$

The new string still reaches B (on a track that goes around the loop an extra time) so it's accepted. But the new string isn't of the form $0^n 1^n$ since it has more 0's than 1's. So it shouldn't be accepted. Contradiction! So the set is not regular.

example 2 (trickier)

Let A be the set of strings of the form 0^p where p is a prime, i.e.,

$$A = \{0^2, 0^3, 0^5, 0^7, 0^{11}, \ldots\}.$$

I'll show that A is not regular.
Suppose A is regular. (I want to get a contradiction.)
Then it has a FSM recognizer.
Let N be the number of states in the FSM.

Let q be a prime $\geq$ N. Look at the string 0^q. It's in A so it's recognized by the FSM. Fig 3 shows a track leading to an accepting state.

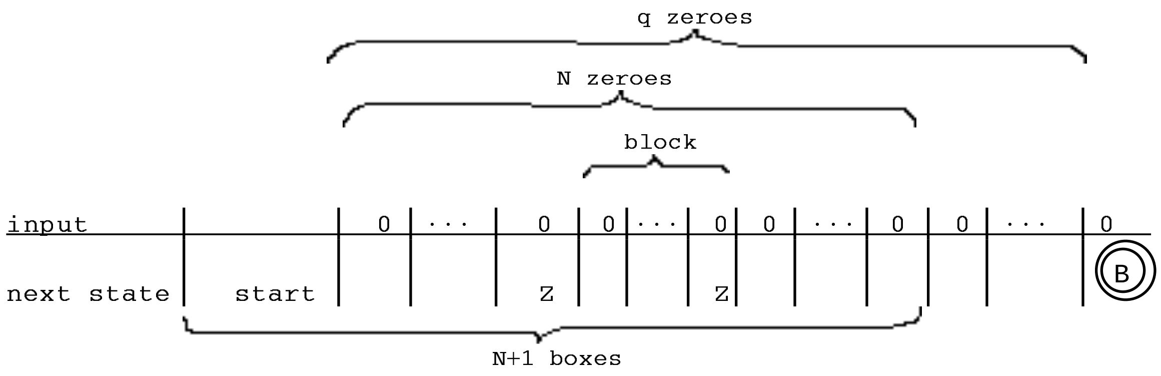


FIG 3

Look at the first N zeroes. There are N+1 corresponding state-boxes in Fig 3, one for each 0 and one for the starting state. More boxes than states. So within the first N zeroes there must be a repeated state, say Z
(Fig 3), i.e., the track contains a loop.
Look at the block of 0's in Fig 3 between repeated states. Let the number of 0's in the block be called k. All you know about k is that it's between 1 and N.
Make a new string by duplicating the block q times. The new string still reaches the accepting state B (same track but go around the loop q extra times) so it's accepted. But the new word contains qk *extra* 0's for a total of q + qk zeroes. But

$$q + qk = q(1 + k) \text{ where } 1 + k \geq 2$$

so q + qk factors into two integers neither of which is 1. So q + qk is *not* prime and the new word *shouldn't* be recognized.
Contradiction!.
So the set is not regular.

Note that in this example, it is not enough to duplicate the block once. It had to be done q times to get the contradiction. See problem #3.

example 3

Let A be the set of palindromes (strings which read the same backwards as forwards, e.g., 000, 11011, Madam I'm Adam, Able was I ere I saw Elba).
I'll show that A is not regular.
Suppose it is regular.
Then it has a FSM recognizer, say with N states.

first try (unsuccessful) Look at the palindrome 0^N. Fig 4 shows a track leading to an accepting state. There are N zeroes and N+1 next-state boxes. More boxes than states. So there must be a repeated state, say Z, i.e., the track contains a loop.

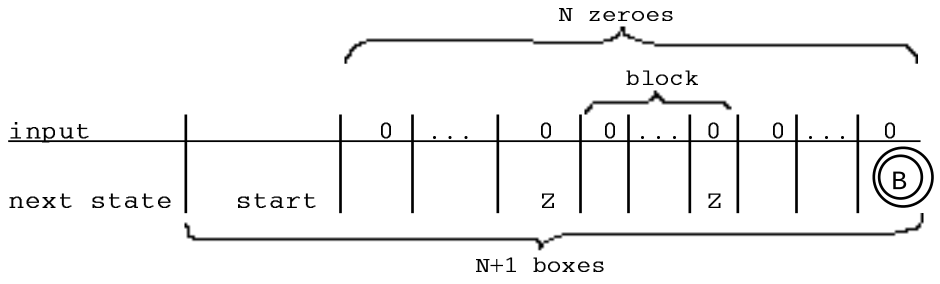


FIG 4

Look at the block of 0's corresponding to the loop. Make a new string by deleting the block or duplicating the block any number of times. The new word still reaches the accepting state B (same track but either skip the loop or go around it more than once) so it's accepted. But this *isn't* a contradiction since the new word is still a palindrome and *should* be accepted. So we have no conclusion (we do *not* have the conclusion that the set is regular).

second try (still unsuccessful) Look at the string $0^{N/2}$ 1 $0^{N/2}$ where N is even (if this works I'll have to come back and consider what to do if N is odd).
Since it is a palindrome, there is a track in the FSM leading to an accepting state (Fig 5). There are N+1 symbols and N+2 next-state boxes. More boxes than states. So there must be a repeated state, say Z.
So the track contains a loop.

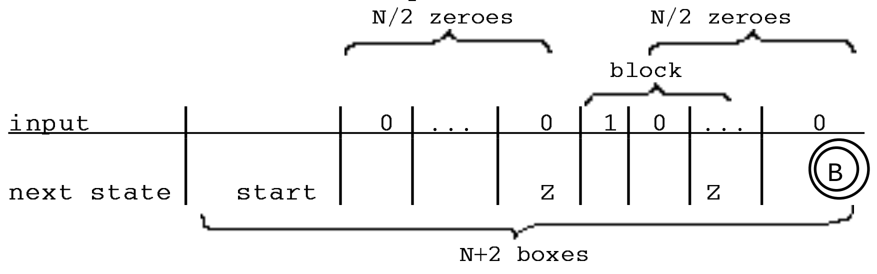


FIG 5

Look at the block corresponding to the loop. It might contain just 0's. It might contain just the 1. It might contains 0's and the 1. Make a new string by deleting the block (or duplicating the block). The new string still reaches B (on a track that skips the loop or goes around it twice) so it's accepted. But *this is not necessarily a contradiction.* If the block consists just of the single 1 then the new string is still a palindrome and deserves to be accepted.
No conclusion about whether the set of palindromes is regular or not.

third try (successful) Look at the string $0^N 1\, 0^N$.
Since it is a palindrome, there is a track in the FSM leading to an accepting state (Fig 6). Within the first N zeroes there are N+1 next-state boxes. More boxes than states so there must be a repeated state, say Z. So the track contains a loop.

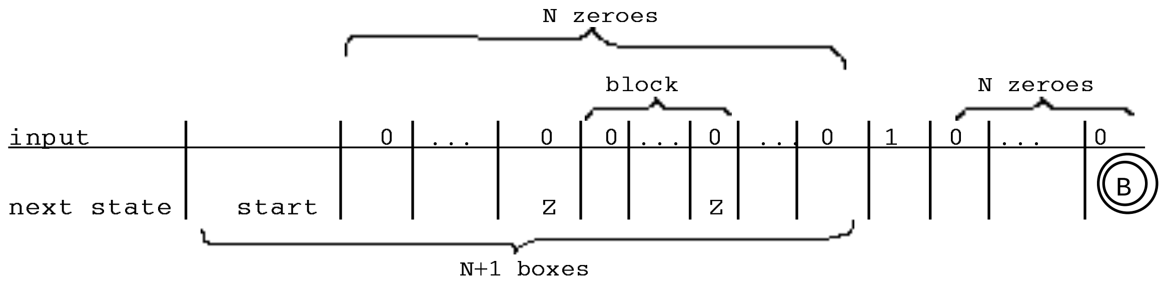


FIG 6

Look at the block of 0's in Fig 6 corresponding to the loop. Make a new string by deleting the block (or duplicating the block). The new string still reaches B (on a track that skips the loop or goes around it twice) so it's accepted. But it isn't a palindrome anymore so it shouldn't be accepted. Contradiction!

So the set of palindromes isn't regular.

outline of the proof you must give to show that a set r is *not* regular

Follow this script as closely as you can.

In particular, you must include first three lines.

Suppose r is regular
Then it has a FSM recognizer.
Let N be the number of states in the FSM.

footnote You need the "Let" because this is a definition of N.

Look at the particular string ··· in r
[Choose a suitable particular string in r to play with; its length must be at least N to be of any use.]

The string has a track leading to an accepting state
There must be a repeated state along the track, i.e., a loop [EXPLAIN WHY].
Make a new string by deleting or duplicating [you decide which] the block of symbols between the repeated states.
The new string still has a track leading to an accepting state but [if you set things up right] the new string isn't in r anymore (doesn't have the "pattern" any more) so it shouldn't be accepted.
Contradiction!
So the set r can't be regular.

warning

1. Not every string in r may help get the contradiction.
In example 3, 0^N didn't help but $0^N 1 0^N$ did.

2. You might have to choose a specific amount of duplication to get the contradiction. In example 2 it took q duplications where q was a prime ≥ N.

3. You must work with a *specific* (not vague, not general, not abstract) string in the set and it must have length ≥ N to be useful.

example 4

Let A be the set of binary strings where the number of 0's is a multiple of 3. For example, A contains 111, 10100, 0^9, 010^{11},

Is A regular?

first try (unsuccessful) I'll try to show that A is not regular.
Suppose A is regular.
Then it has a FSM recognizer.
Let N be the number of states.
Look at the string 0^{3N}.
It has a track leading to an accepting state.
In the table in Fig 6 there are 3N zeroes and 3N+1 next-state boxes. More boxes than states so there must be a repeated state, say Z. So the track contains a loop.

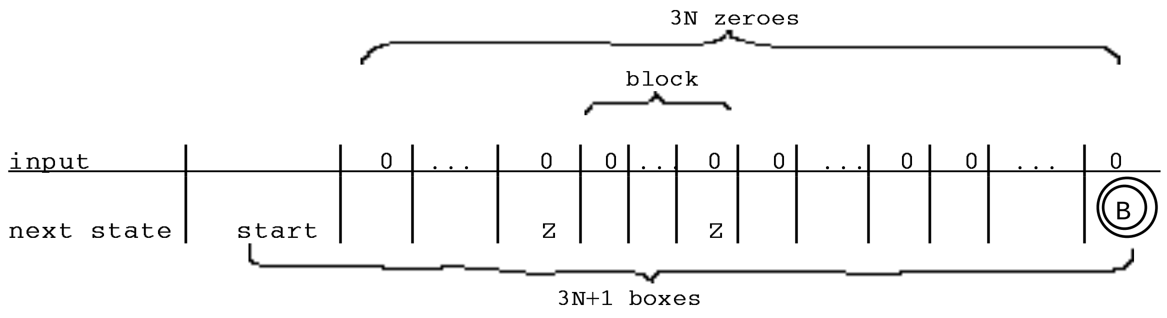


FIG 6

Make a new string by deleting or duplicating the block of 0's in the loop. The new string reaches B (on the same track but without the loop or with extra trips around the loop) so it is still recognized.
But is this a contradiction?
No if the number of 0's in the block happens to be a multiple of 3 in which case the number of 0's in the new string is still a multiple of 3 and the new string deserves to be recognized.
Yes if the number of 0's in the block happens to be 2 or 17 or 5 etc in which case the number of 0's in the new string is no longer a multiple of 3.

But only a *guaranteed* contradiction is useful and we don't have that.
So we have no conclusion about whether A is regular or not.

warning 1 It is of no value to say "I would get a contradiction *if* ... "
You must be able to say "I definitely got a contradiction".

warning 2 When you don't get the contradiction you are looking for it does not mean that the set is regular. It means that you have to start all over again.

second try (WRONG)
Continue as in the first try but delete only *one of the 0's in the block*.
The number of 0's in the new string no longer is a multiple of 3. Contradiction.
So the set is not regular. WRONG WRONG
Here's why the argument is wrong.
To get the contradiction you must know that the number of 0's in the new string is not a multiple of 3 (so far so good, you have that). *And* you must know that the new string reaches an accepting state. You don't know that here. Since you have not deleted *all* the zeros in the block you do *not* have a track that just skips the loop and still reaches B. (You actually have a track that stops one state short of B. That state might be acc also but you can't count on it.)

warning 3 You have to delete or duplicate the entire block corresponding to a loop to have a chance.

third try (successful)
A is regular since it has the regular expression 000(000)*.

fourth try (also successful)
A is regular since it is recognized by the FSM in Fig 7.

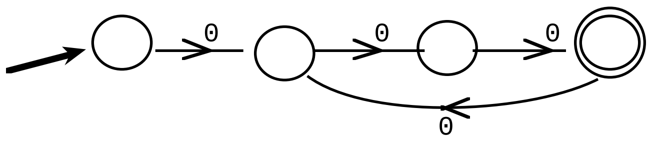


FIG 7

PROBLEMS FOR SECTION 5.7

1. Look at the set of strings of the form $0^n\ 1^m$ where m > n, i.e., strings of 0's followed by a larger number of 1's. Show that it isn't regular.

2. Look at the set of binary strings where the second half of the string is a repeat of the first half (e.g., 110 110, 10011 10011, 00 00). (The strings must have an even number of symbols to begin with or they don't even have halves.)
Here's the beginning of a proof to show that the set is not regular.
Suppose it is regular.
Then it has a FSM recognizer.
Let N be the number of states.
Continue the argument using the following strings.

(a) $10^N 10^N$

(b) $\underbrace{10\ \ldots\ 0}_{N/2 \text{ symbols}}\ \ \underbrace{10\ \ldots\ 0}_{N/2 \text{ symbols}}$ (this assumes N is even)

3. Look at example 2 (set of strings with a prime number of 0's). I got a new string by duplicating the block in Fig 3 q times to eventually get a contradiction. Why so complicated? Why can't you just delete the block or duplicate it once? What was so special about duplicating it q times.

4. Let A be the set of binary strings beginning with 0 where every occurrence of 0 is followed immediately by a block of at least three 1's.
Decide if the set is regular or non-regular and prove that you're right.

5. Let A be the set of binary strings not containing any occurrences of 010
Decide if the set is regular or not regular and prove that you're right.

6. Let A be the set of strings of 0's where the number of 0's is a perfect square, i.e., A = {0, 0^4, 0^9, 0^{16}, ...}
Decide if the set is regular or non-regular and prove that you're right.

7. Let r be the set of binary strings with a 1 in the "middle"
Right in the middle if the string has an odd number of symbols and in one of the two middle spots if the string has an even number of symbols.
For example, r contains

$001\ \boxed{1}\ 111$

$0110\ \boxed{1}0\ 1000$

$111\ \boxed{1}\ 111$ etc

Let's begin a proof to show that r is not regular.
Suppose it is regular.
Then it has a FSM recognizer.
Let N be the number of states.

(a) What's wrong with the following way to continue the argument.

Look at a string of the form

$$\underbrace{\ldots}_{N \text{ symbols}}\ 1\ \underbrace{\ldots}_{N \text{ symbols}}$$

The string has a track to an accepting state.
There is a loop within the first N symbols.
Duplicate the block of symbols corresponding to the loop, say ten times for good measure.
The new word is still accepted but the 1 is now pushed way out of the middle so it shouldn't be accepted. Contradiction.
So the set is not regular.

(b) Find a correct way to finish the argument.

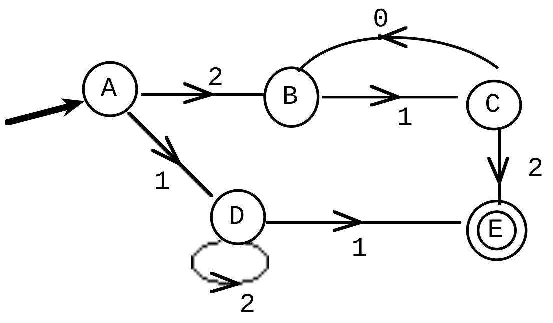
8. Show that the following set of strings is not regular (the alphabet is 0–9).

```
1
12
123
1234
12345
123456
1234567
12345678
123456789    and now it gets more interesting
123456789 10
123456789 10 11
123456789 10 11 12
123456789 10 11 12 13
etc.
```

REVIEW PROBLEMS FOR CHAPTER 5

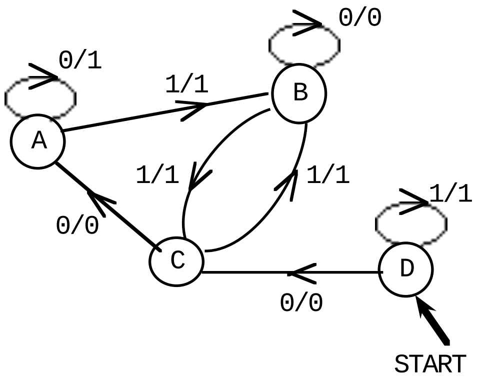
1. The alphabet is a, b, c. Find a FSM recognizer (nondeterministic is fine, but without λ moves) for the strings containing at least one occurrence of bb or at least one occurrence of cc.

2. Solve state equations to find the set of strings recognized by the following FSM. The alphabet here is 0, 1, 2.

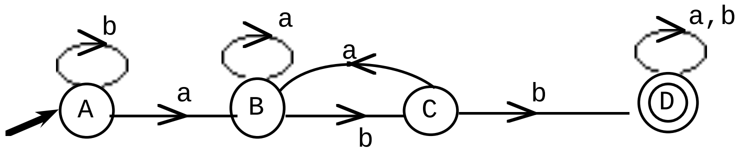


3. Let r be the set of strings with at most two 1's (the alphabet is 0,1). See if you can find three different ways to show that r is regular.

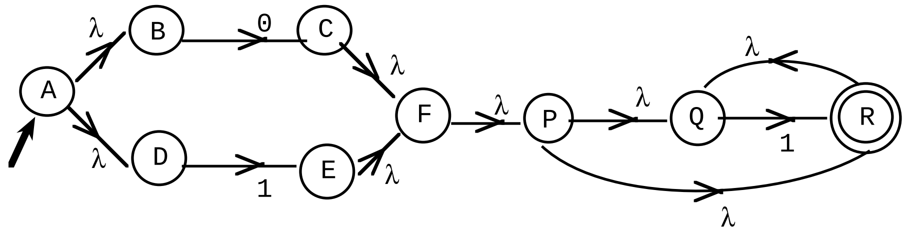
4. Here is a FSM with outputs. Using the standard method to find an equivalent FSM (i.e., one which recognizes the same strings) with accepting states instead of outputs.



5. Solving state equations is a foolproof way to get a regular expression for the set recognized by a FSM and is the guarantee that the set recognized by a FSM is regular. But sometimes you can get a regular expression for the set recognized by a FSM by inspection. Try it for this FSM.

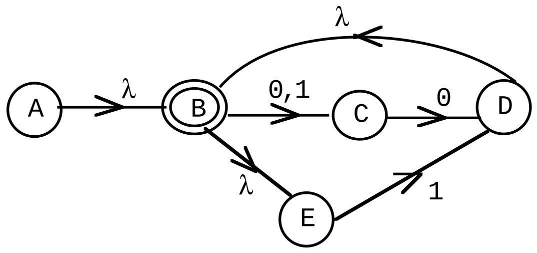


6. Here's a NFSM



(a) By inspection, describe the set of strings recognized.
(b) Find an equivalent DFSM using the standard method.
(c) Find a simpler DFSM by inspection.

7. Keep track of all next states to see if 10 is accepted by this FSM



8. Find a FSM recognizers for $((00+1)^*)^*$ using the Kleene construction method and then find simpler one by inspection.

9. Find a FSM recognizer and a regular expression for the set of strings containing no consecutive 1's.

10. Look at the set of strings of the form $0^3 1^m$ where $m \geq 4$, i.e., strings starting with three 0's followed by at least four 1's.

Here's the beginning of a proof trying to show that the set isn't regular.
Suppose the set is regular.
Then it has a FSM recognizer.
Let N be the number of states.
Look at the string $000\ 111\ 1^N$.

Can you finish the proof.

11. Simplify $00^*(0+1)^*1$ if possible