Automata and Computability

Solutions to Exercises

Fall 2018

Alexis Maciel
Department of Computer Science
Clarkson University

Contents

Preface **v**

1 Introduction **1**

2 Finite Automata **3**
2.1 Turing Machines 3
2.2 Introduction to Finite Automata 3
2.3 More Examples 9
2.4 Formal Definition 13
2.5 Closure Properties 19

3 Nondeterministic Finite Automata **27**
3.1 Introduction 27
3.2 Formal Definition 29
3.3 Equivalence with DFA's 32
3.4 Closure Properties 36

4 Regular Expressions **41**
4.1 Introduction 41
4.2 Formal Definition 41

Preface

This document contains solutions to the exercises of the course notes *Automata and Computability*. These notes were written for the course CS345 *Automata Theory and Formal Languages* taught at Clarkson University. The course is also listed as MA345 and CS541. The solutions are organized according to the same chapters and sections as the notes.

Here's some advice. Whether you are studying these notes as a student in a course or in self-directed study, your goal should be to understand the material well enough that you can do the exercises on your own. Simply studying the solutions is not the best way to achieve this. It is much better to spend a reasonable amount of time and effort trying to do the exercises yourself before looking at the solutions.

If you can't do an exercise on your own, you should study the notes some more. If that doesn't work, seek help from another student or from your instructor. Look at the solutions only to check your answer once you think you know how to do an exercise.

If you needed help doing an exercise, try redoing the same exercise later on your own. And do additional exercises.

If your solution to an exercise is different from the solution in this document, take the time to figure out why. Did you make a mistake? Did you forget some-

thing? Did you discover another correct solution? If you're not sure, ask for help from another student or the instructor. If your solution turns out to be incorrect, fix it, after maybe getting some help, then try redoing the same exercise later on your own and do additional exercises.

Feedback on the notes and solutions is welcome. Please send comments to `alexis@clarkson.edu`.

Chapter 1

Introduction

There are no exercises in this chapter.

Chapter 2

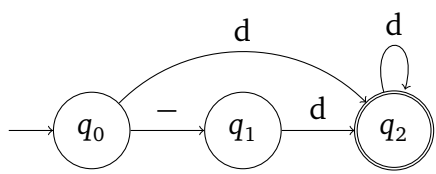
Finite Automata

2.1 Turing Machines

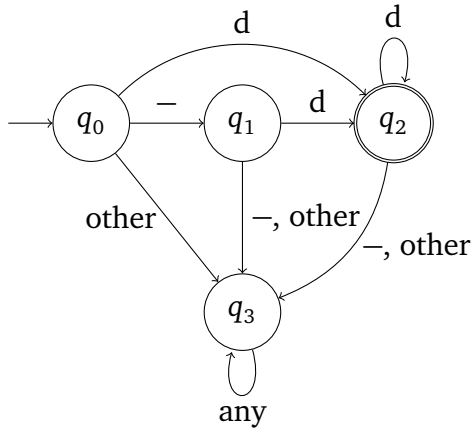
There are no exercises in this section.

2.2 Introduction to Finite Automata

2.2.3.

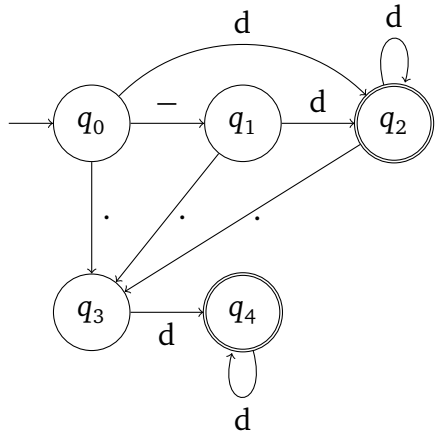


Missing edges go to a garbage state. In other words, the full DFA looks like this:



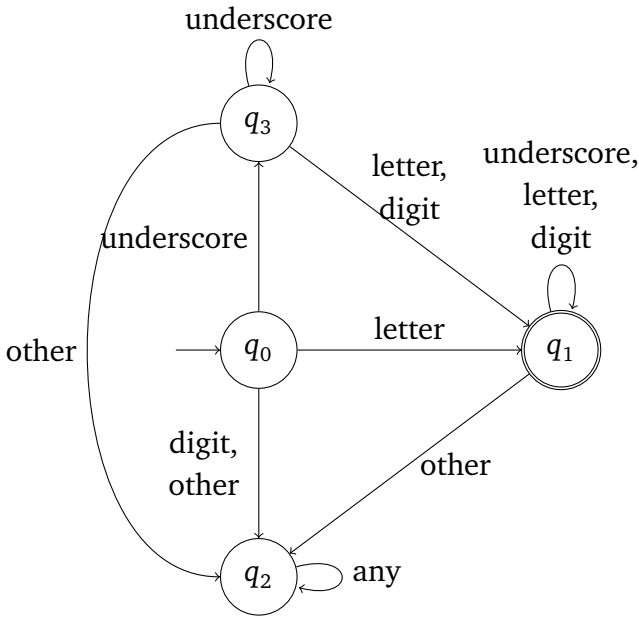
The transition label *other* means any character that's not a dash or a digit.

2.2.4.

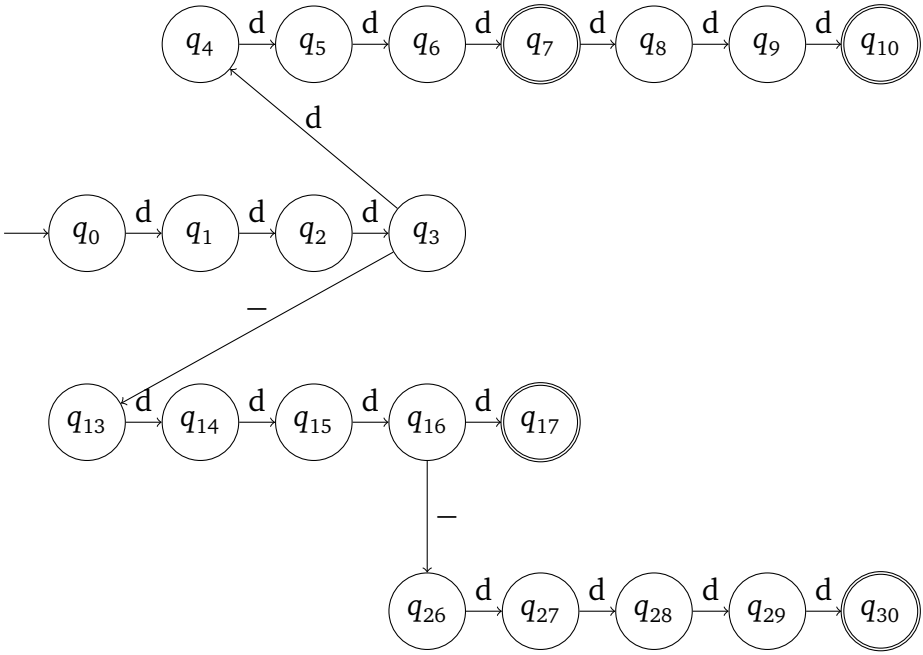


Missing edges go to a garbage state.

2.2.5.


underscore
q_3
letter,
digit
underscore,
letter,
digit
underscore
other
q_0
letter
q_1
digit,
other
other
q_2
any

2.2.6.


q_4
d
q_5
d
q_6
d
q_7
d
q_8
d
q_9
d
q_{10}
d
q_0
d
q_1
d
q_2
d
q_3
–
q_{13}
d
q_{14}
d
q_{15}
d
q_{16}
d
q_{17}
–
q_{26}
d
q_{27}
d
q_{28}
d
q_{29}
d
q_{30}

2.2.7.

```
starting_state() { return q0 }

is_accepting(q) { return true iff q is q1 }

next_state(q, c) {
    if (q is q0)
        if (c is underscore or letter)
            return q1
        else
            return q2
    else if (q is q1)
        if (c is underscore, letter or digit)
            return q1
        else
            return q2
    else // q is q2
        return q2
}
```

2.2.8. The following assumes that the garbage state is labeled q_9. In the pseudocode algorithm, states are stored as integers. This is more convenient here.

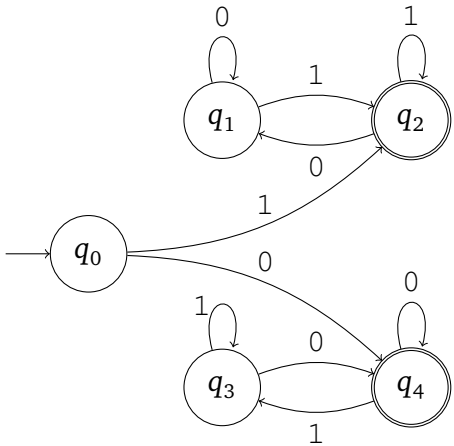
```
starting_state() { return 0 }

is_accepting(q) { return true iff q is 8 }
```

```
next_state(q, c) {
    if (q in {0, 1, 2} or {4, 5, 6, 7})
        if (c is digit)
            return q + 1
        else
            return 9
    else if (q is 3)
        if (c is digit)
            return 5
        else if (c is dash)
            return 4
        else
            return 9
    else if (q is 8 or 9)
        return 9
}
```

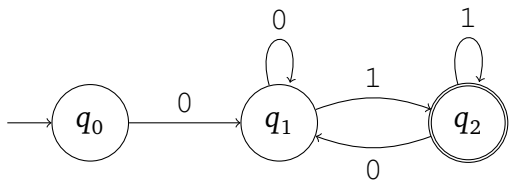
2.3 More Examples

2.3.5.

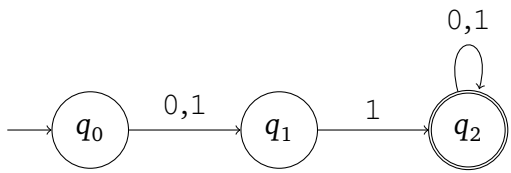


2.3.6. In all cases, missing edges go to a garbage state.

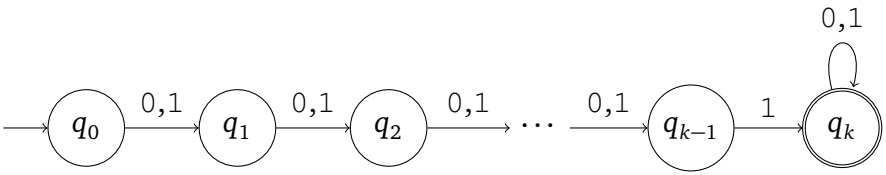
a)



b)

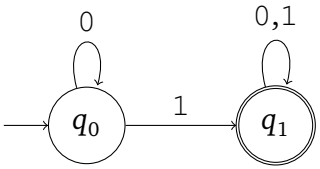


c)

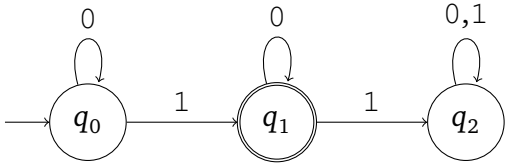


2.3.7. In all cases, missing edges go to a garbage state.

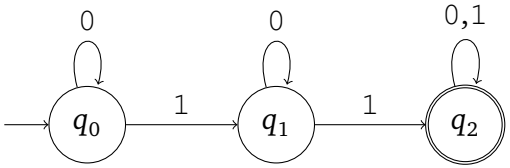
a)



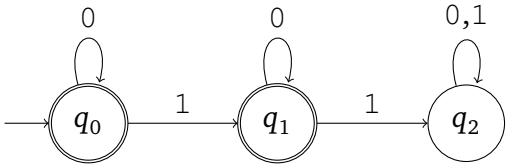
b)



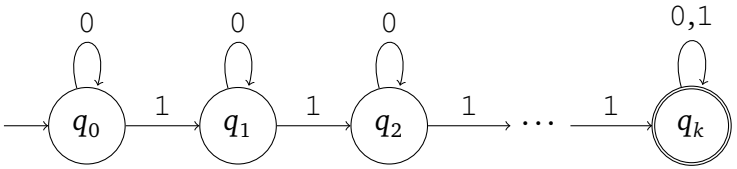
c)



d)

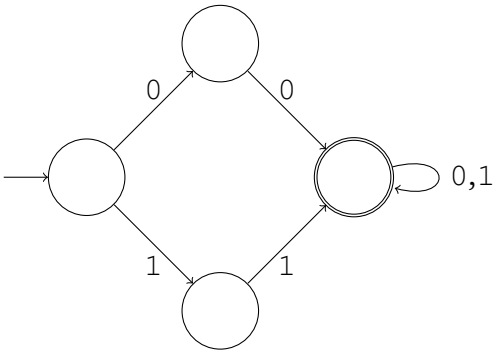


e)

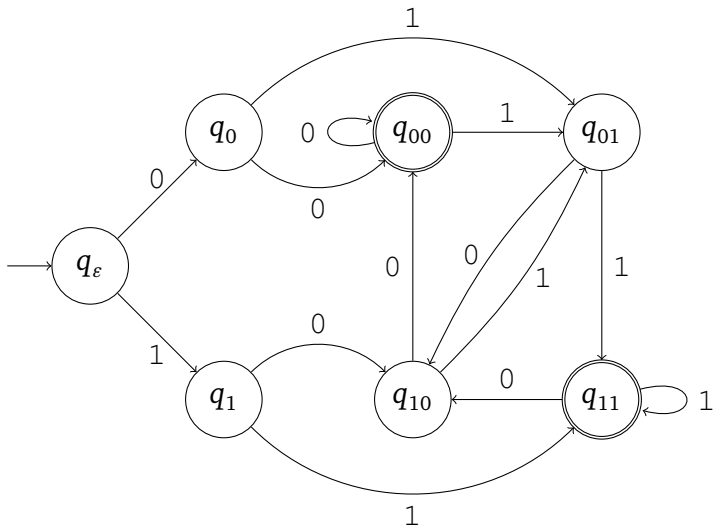


2.3.8. In all cases, missing edges go to a garbage state.

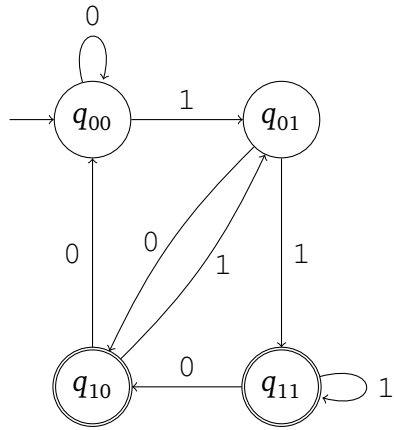
a)



b) The idea is for the DFA to remember the last two symbols it has seen.



c) Again, the idea is for the DFA to remember the last two symbols it has seen. We could simply change the accepting states of the previous DFA to $\{q_{10}, q_{11}\}$. But we can also simplify this DFA by assuming that strings of length less than two are preceded by `00`.



2.4 Formal Definition

2.4.5.

a) The DFA is $(\{q_0, q_1, q_2, \ldots, q_9\}, \Sigma, \delta, q_0, \{q_8\})$ where Σ is the set of all characters that appear on a standard keyboard and δ is defined as follows:

$$\delta(q_i, c) = \begin{cases} q_{i+1} & \text{if } i \notin \{3, 8, 9\} \text{ and } c \text{ is digit} \\ q_9 & \text{if } i \notin \{3, 8, 9\} \text{ and } c \text{ is not digit} \end{cases}$$

$$\delta(q_3, c) = \begin{cases} q_4 & \text{if } c \text{ is dash} \\ q_5 & \text{if } c \text{ is digit} \\ q_9 & \text{otherwise} \end{cases}$$

$$\delta(q_8, c) = q_9 \quad \text{for every } c$$

$$\delta(q_9, c) = q_9 \quad \text{for every } c$$

b) The DFA is $(\{q_0, q_1, q_2, q_3\}, \Sigma, \delta, q_0, \{q_2\})$ where Σ is the set of all characters that appear on a standard keyboard and δ is defined as follows:

$$\delta(q_0, c) = \begin{cases} q_1 & \text{if } c \text{ is dash} \\ q_2 & \text{if } c \text{ is digit} \\ q_3 & \text{otherwise} \end{cases}$$

$$\delta(q_i, c) = \begin{cases} q_2 & \text{if } i \in \{1, 2\} \text{ and } c \text{ is digit} \\ q_3 & \text{if } i \in \{1, 2\} \text{ and } c \text{ is not digit} \end{cases}$$

$$\delta(q_3, c) = q_3 \quad \text{for every } c$$

c) The DFA is $(\{q_0, q_1, q_2, \ldots, q_5\}, \Sigma, \delta, q_0, \{q_2, q_4\})$ where Σ is the set of all characters that appear on a standard keyboard and δ is defined as follows:

$$\delta(q_0, c) = \begin{cases} q_1 & \text{if } c \text{ is dash} \\ q_2 & \text{if } c \text{ is digit} \\ q_3 & \text{if } c \text{ is decimal point} \\ q_5 & \text{otherwise} \end{cases}$$

$$\delta(q_i, c) = \begin{cases} q_2 & \text{if } i \in \{1, 2\} \text{ and } c \text{ is digit} \\ q_3 & \text{if } i \in \{1, 2\} \text{ and } c \text{ is decimal point} \\ q_5 & \text{if } i \in \{1, 2\} \text{ and } c \text{ is not digit or decimal point} \end{cases}$$

$$\delta(q_i, c) = \begin{cases} q_4 & \text{if } i \in \{3, 4\} \text{ and } c \text{ is digit} \\ q_5 & \text{if } i \in \{3, 4\} \text{ and } c \text{ is not digit} \end{cases}$$

$$\delta(q_5, c) = q_5 \quad \text{for every } c$$

2.4.6. The idea is for the DFA to remember the last k symbols it has seen. But this is too difficult to draw clearly, so here's a formal description of the DFA: $(Q, \{0,1\}, \delta, q_0, F)$ where

$$\begin{aligned} Q &= \{q_w \mid w \in \{0,1\}^* \text{ and } w \text{ has length } k\} \\ q_0 &= q_{w_0} \quad \text{where } w_0 = 0^k \text{ (that is, a string of } k \text{ 0's)} \\ F &= \{q_w \in Q \mid w \text{ starts with a } 1\} \end{aligned}$$

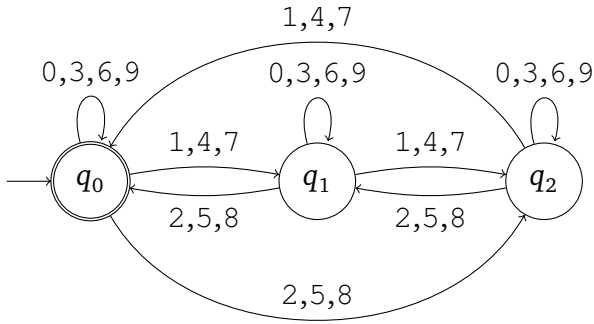
and δ is defined as follows:

$$\delta(q_{au}, b) = q_{ub}$$

where $a \in \Sigma$, u is a string of length $k-1$ and $b \in \Sigma$.

2.4.7.

a) The idea is for the DFA to store the value, modulo 3, of the portion of the number it has seen so far, and then update that value for every additional digit that is read. To update the value, the current value is multiplied by 10, the new digit is added and the result is reduced modulo 3.



(Note that this is exactly the same DFA we designed in an example of this section for the language of strings that have the property that the sum of their digits is a multiple of 3. This is because 10 mod 3 = 1 so that when we multiply the current value by 10 and reduce modulo 3, we are really just multiplying by 1. Which implies that the strategy we described above is equivalent to simply adding the digits of the number, modulo 3.)

b) We use the same strategy that was described in the first part, but this time, we reduce modulo k. Here's a formal description of the DFA: $(Q, \Sigma, \delta, q_0, F)$ where

$$Q = \{q_0, q_1, q_2, \ldots, q_{k-1}\}$$
$$\Sigma = \{0, 1, 2, \ldots, 9\}$$
$$F = \{q_0\}$$

and δ is defined as follows: for every $i \in Q$ and $c \in \Sigma$,

$$\delta(q_i, c) = q_j \quad \text{where } j = (i \cdot 10 + c) \bmod k.$$

2.4.8.

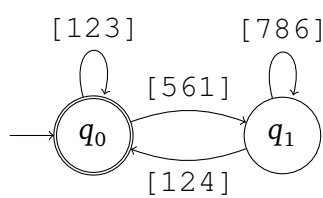
a) The idea is for the DFA to verify, for each input symbol, that the third digit is the sum of the first two plus any carry that was previously generated, as well as determine if a carry is generated. All that the DFA needs to remember is the value of the carry (0 or 1). The DFA accepts if no carry is generated when processing the last input symbol. Here's a formal description of the DFA, where state q_2 is a garbage state: $(Q, \Sigma, \delta, q_0, F)$ where

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{\texttt{[}abc\texttt{]} \mid a, b, c \in \{\texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{9}\}\} \\
F &= \{q_0\}
\end{aligned}
$$

and δ is defined as follows:

$$
\delta(q_d, \texttt{[}abc\texttt{]}) = \begin{cases} q_0 & \text{if } d \in \{0,1\} \text{ and } d + a + b = c \\ q_1 & \text{if } d \in \{0,1\},\ d + a + b \geq 10 \text{ and} \\ & \quad (d + a + b) \bmod 10 = c \\ q_2 & \text{otherwise} \end{cases}
$$

Here's a transition diagram of the DFA that shows only one of the 1,000 transitions that come out of each state.



b) Since the DFA is now reading the numbers from left to right, it can't compute the carries as it reads the numbers. So it will do the opposite: for each input symbol, the DFA will figure out what carry it needs from the rest of the numbers. For example, if the first symbol that the DFA sees is `[123]`, the DFA will know that there should be no carry generated from the rest of the numbers. But if the symbol is `[124]`, the DFA needs the rest of the number to generate a carry. And if a carry needs to be generated, the next symbol will have to be something like `[561]` but not `[358]`. The states of the DFA will be used to remember the carry that is needed from the rest of the numbers. The DFA will accept if no carry is needed for the first position of the numbers (which is given by the last symbol of the input string). Here's a formal description of the DFA, where state q_2 is a garbage state: $(Q, \Sigma, \delta, q_0, F)$ where

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{\texttt{[}abc\texttt{]} \mid a, b, c \in \{\texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{9}\}\} \\
F &= \{q_0\}
\end{aligned}
$$

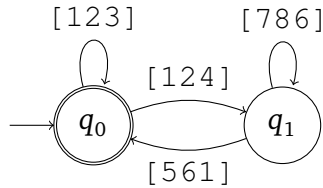
and δ is defined as follows:

$$
\delta(q_0, \texttt{[}abc\texttt{]}) = \begin{cases} q_d & \text{if } d \in \{0,1\} \text{ and } d + a + b = c \\ q_2 & \text{otherwise} \end{cases}
$$

$$
\delta(q_1, \texttt{[}abc\texttt{]}) = \begin{cases} q_d & \text{if } d \in \{0,1\},\ d + a + b \geq 10 \text{ and} \\ & \quad (d + a + b) \bmod 10 = c \\ q_2 & \text{otherwise} \end{cases}
$$

$$
\delta(q_2, \texttt{[}abc\texttt{]}) = q_2, \quad \text{for every } \texttt{[}abc\texttt{]} \in \Sigma
$$

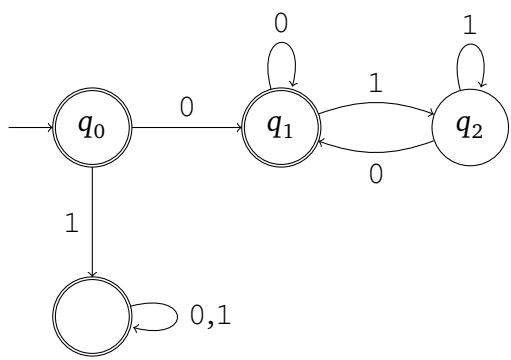
Here's a transition diagram of the DFA that shows only one of the 1,000 transitions that come out of each state.



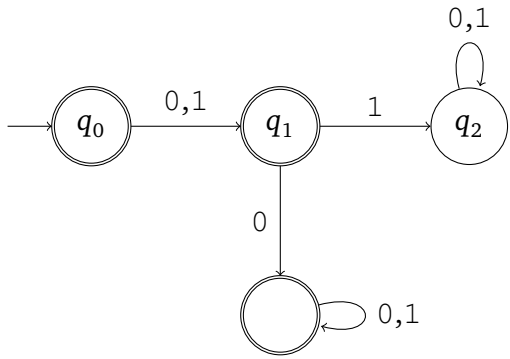
2.5 Closure Properties

2.5.3. In each case, all we have to do is switch the acceptance status of each state. But we need to remember to do it for the garbage states too.

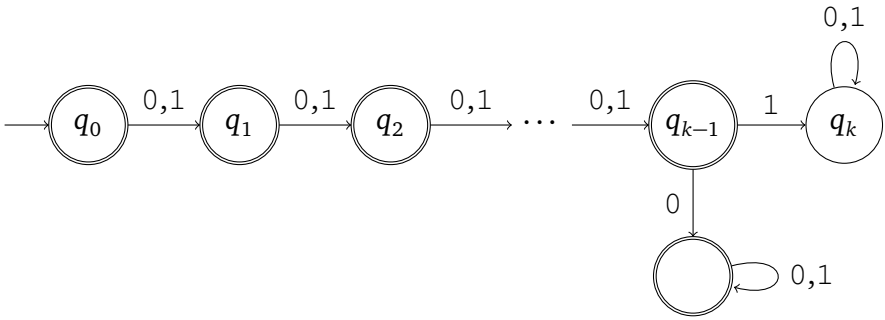
a)



b)

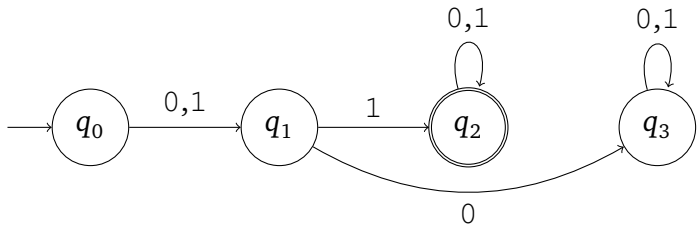


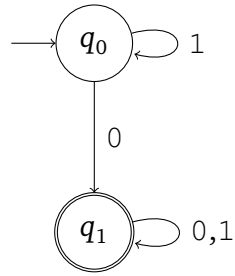
c)

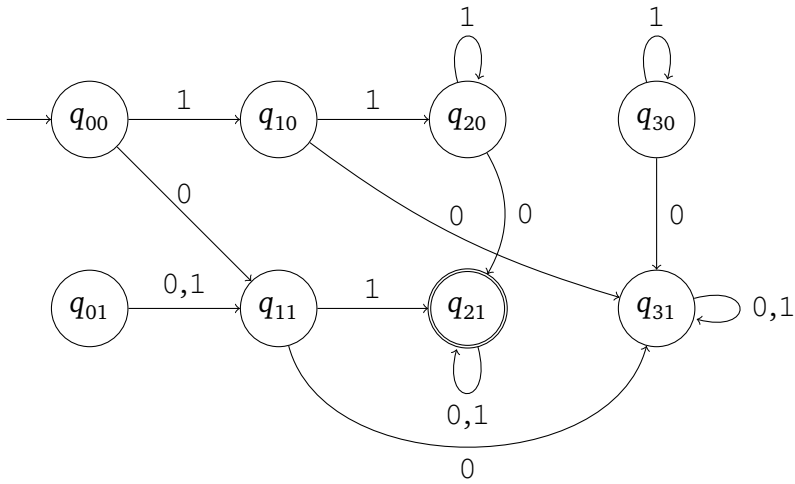


2.5.4. It is important to include in the pair construction the garbage states of the DFA's for the simpler languages. (This is actually not needed for intersections but it is critical for unions.) In each case, we give the DFA's for the two simpler languages followed by the DFA obtained by the pair construction.

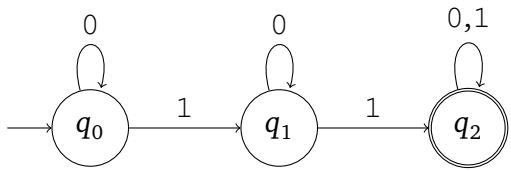
a)

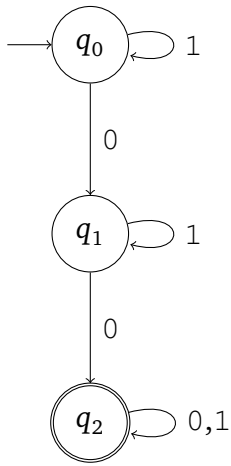

0,1
0,1
q_0
0,1
q_1
1
q_2
q_3
0

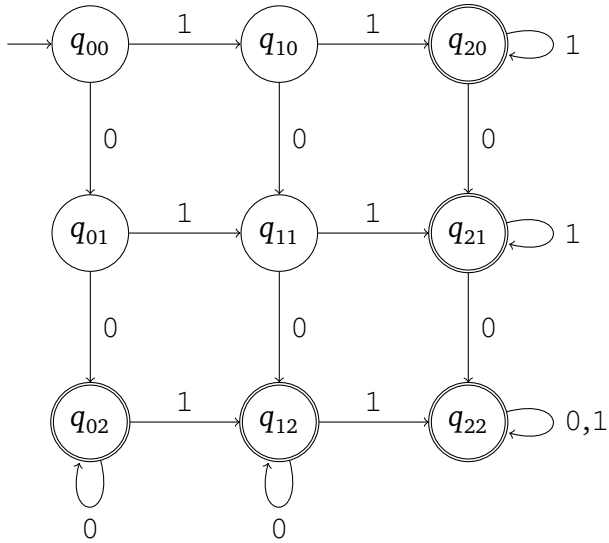

q_0
1
0
q_1
0,1


1
1
q_{00}
1
q_{10}
1
q_{20}
q_{30}
0
0
0
0
q_{01}
0,1
q_{11}
1
q_{21}
q_{31}
0,1
0,1
0

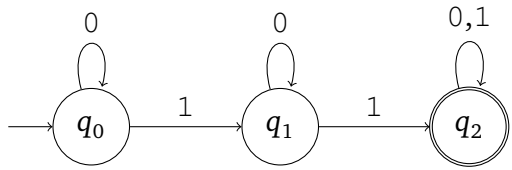
b)

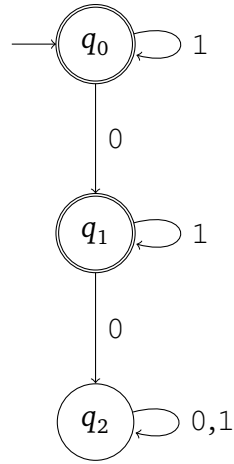


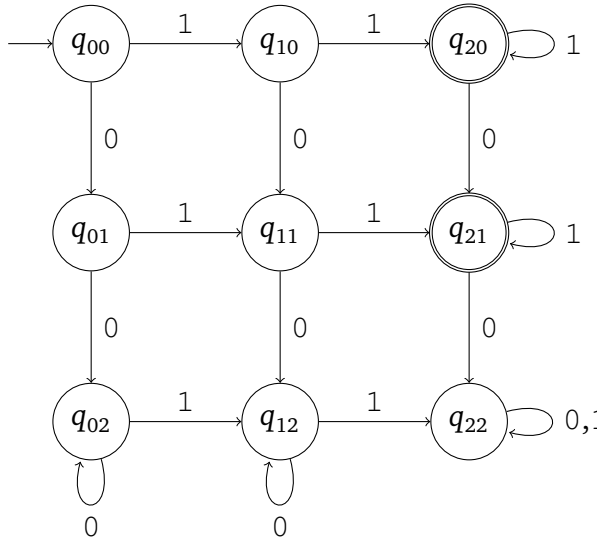




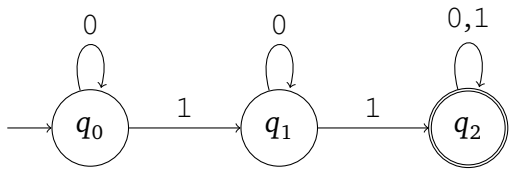
c)

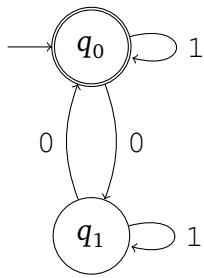


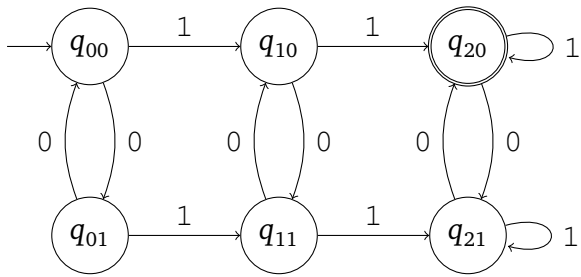

q_0
1
0
q_1
1
0
q_2
0,1


q_{00}
1
q_{10}
1
q_{20}
1
0
0
0
q_{01}
1
q_{11}
1
q_{21}
1
0
0
0
q_{02}
1
q_{12}
1
q_{22}
0,1
0
0

d)

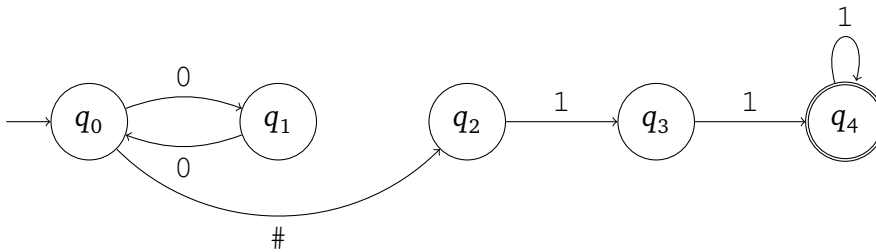

0
0
0,1
1
1
q_0
q_1
q_2


q_0
1
0
0
q_1
1

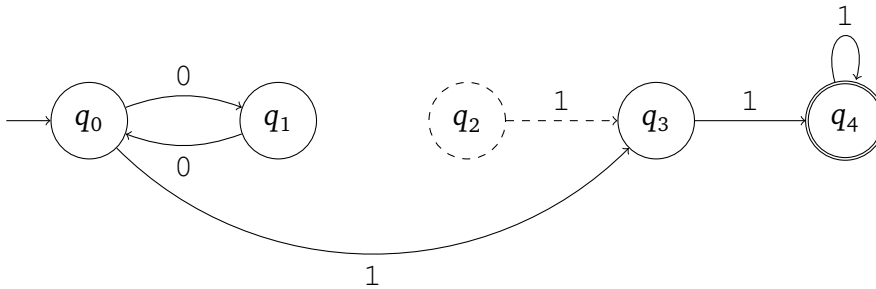

q_{00}
1
q_{10}
1
q_{20}
1
0
0
0
0
0
0
q_{01}
1
q_{11}
1
q_{21}
1

2.5.5. In both cases, missing edges go to a garbage state.

a)



b) The dashed state and edge could be deleted.



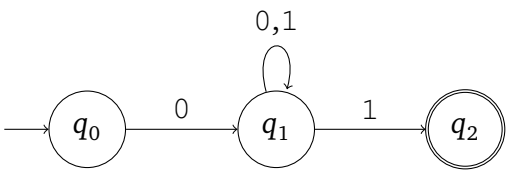
Chapter 3

Nondeterministic Finite Automata

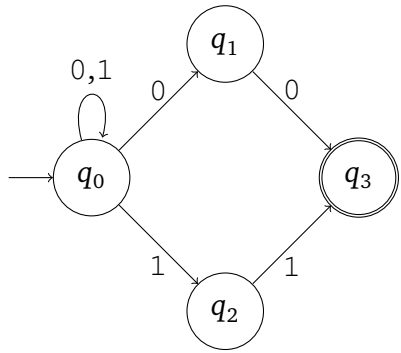
3.1 Introduction

3.1.3.

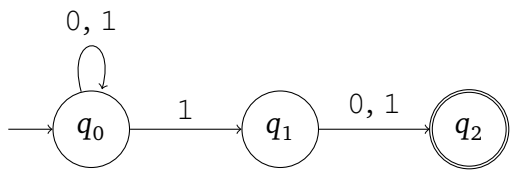
a)



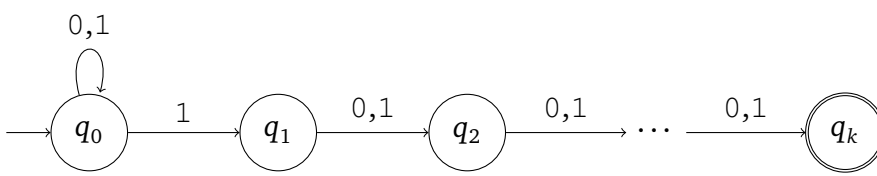
b)



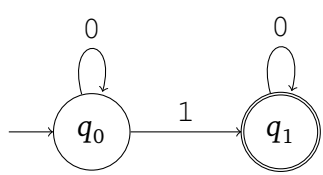
c)



d)



e)



3.2 Formal Definition

3.2.1.

a) The NFA is $(Q, \{0, 1\}, \delta, q_0, F)$ where

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$F = \{q_3\}$$

and δ is defined by the following table

δ	0	1	ε
q_0	q_0	q_0, q_1	$-$
q_1	q_2	q_2	$-$
q_2	q_3	q_3	$-$
q_3	$-$	$-$	$-$

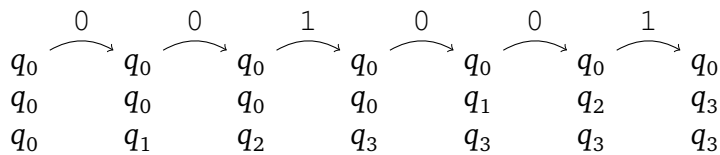
b) The NFA is $(Q, \{0, 1\}, \delta, q_0, F)$ where

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$F = \{q_3\}$$

and δ is defined by the following table:

δ	0	1	ε
q_0	q_1	q_0	—
q_1	q_2	—	q_0
q_2	—	q_3	q_1
q_3	q_3	q_3	—

3.2.2.



The NFA accepts because the last two sequences end in the accepting state.

3.2.3.

$$q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0$$

$$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0$$

$$q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0$$

$$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0$$

$$q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_3$$

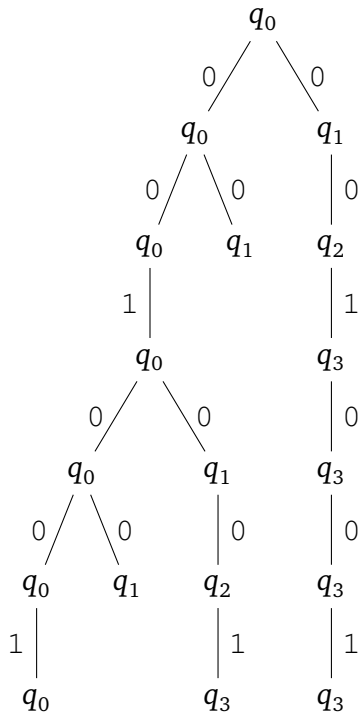
$$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_3$$

$$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_3 \xrightarrow{0} q_3 \xrightarrow{1} q_3$$

The NFA accepts because the last three sequences end in the accepting state.

3.3 Equivalence with DFA's

3.3.2.



3.3.3.

a)

δ'	0	1
q_0	q_1	$-$
q_1	q_1	q_1, q_2
q_2	$-$	$-$
q_1, q_2	q_1	q_1, q_2

The start state is $\{0\}$. The accepting state is $\{q_1, q_2\}$. (As usual, the symbol $-$ represents the state $\emptyset$. That state is a garbage state.)

b)

δ'	0	1
q_0	q_0, q_1	q_0, q_2
q_1	q_3	$-$
q_2	$-$	q_3
q_3	$-$	$-$
q_0, q_1	q_0, q_1, q_3	q_0, q_2
q_0, q_2	q_0, q_1	q_0, q_2, q_3
q_0, q_1, q_3	q_0, q_1, q_3	q_0, q_2
q_0, q_2, q_3	q_0, q_1	q_0, q_2, q_3

The start state is $\{q_0\}$. The accepting states are $\{q_0, q_1, q_3\}$ and

$\{q_0, q_2, q_3\}$.

c)

δ'	0	1
q_0	q_0	q_0, q_1
q_1	q_2	q_2
q_2	—	—
q_0, q_1	q_0, q_2	q_0, q_1, q_2
q_0, q_2	q_0	q_0, q_1
q_0, q_1, q_2	q_0, q_2	q_0, q_1, q_2

The start state is $\{q_0\}$. The accepting states are $\{q_0, q_2\}$ and $\{q_0, q_1, q_2\}$.

d)

δ'	0	1
q_0	q_0	q_1
q_1	q_1	—

The start state is $\{q_0\}$. The accepting state is $\{q_1\}$. (The given NFA was almost a DFA. All that was missing was a garbage state and that's precisely what the algorithm added.)

3.3.4.

a)

δ'	0	1
q_0	q_1	$-$
q_1	q_1	q_1, q_2
q_2	$-$	$-$
q_1, q_2	q_1	q_1, q_2

The start state is $E(\{q_0\}) = \{q_0\}$. The accepting state is $\{q_1, q_2\}$.

b)

δ'	0	1
q_0	q_0, q_1, q_2	q_0, q_1, q_2
q_1	q_3	$-$
q_2	$-$	q_3
q_3	$-$	$-$
q_0, q_1, q_2	q_0, q_1, q_2, q_3	q_0, q_1, q_2, q_3
q_0, q_1, q_2, q_3	q_0, q_1, q_2, q_3	q_0, q_1, q_2, q_3

The start state is $E(\{q_0\}) = \{q_0, q_1, q_2\}$. The accepting state is $\{q_0, q_1, q_2, q_3\}$.

3.4 Closure Properties

3.4.2. Suppose that $M_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$, for $i = 1, 2$. Without loss of generality, assume that Q_1 and Q_2 are disjoint. Then $N = (Q, \Sigma, \delta, q_0, F)$ where

$$Q = Q_1 \cup Q_2$$
$$q_0 = q_1$$
$$F = F_2$$

and δ is defined as follows:

$$\delta(q, \varepsilon) = \begin{cases} \{q_2\} & \text{if } q \in F_1 \\ \emptyset & \text{otherwise} \end{cases}$$
$$\delta(q, a) = \{\delta_i(q, a)\}, \quad \text{if } q \in Q_i \text{ and } a \in \Sigma.$$

3.4.3. Suppose that $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$. Let q_0 be a state not in Q_1. Then $N = (Q, \Sigma, \delta, q_0, F)$ where

$$Q = Q_1 \cup \{q_0\}$$
$$F = F_1 \cup \{q_0\}$$

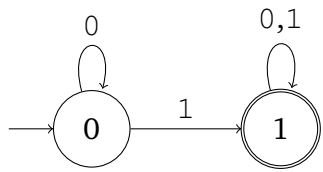
and δ is defined as follows:

$$\delta(q, \varepsilon) = \begin{cases} \{q_1\} & \text{if } q \in F_1 \cup \{q_0\} \\ \emptyset & \text{otherwise} \end{cases}$$
$$\delta(q, a) = \{\delta_1(q, a)\}, \quad \text{if } q \neq q_0 \text{ and } a \in \Sigma.$$

3.4.4.

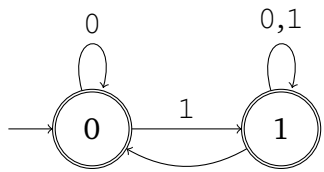
a) In the second to last paragraph of the proof, it is claimed that $w = x_1 \cdots x_{k+1}$, with each $x_i \in A$. It is true that $x_1, \ldots, x_k$ are all in A because they must lead from the start state to one of the original accepting states of M. But this is not true for x_{k+1}: that string could lead back to the start state instead of leading to one of the original accepting states of M. In that case, x_{k+1} wouldn't be in A and we wouldn't be able to conclude that w is in A^*.

b) Consider the following DFA for the language of strings that contain at least one `1`:



M

If we used this idea, we would get the following NFA:



N

This NFA accepts strings that contain only `0`'s. These strings are not in the language $L(M)^*$. Therefore, $L(N) \neq L(M)^*$.

3.4.5. One proof is to notice that $A^+ = AA^*$. Since the class of regular languages

is closed under star and concatenation, we also get closure under the plus operation.

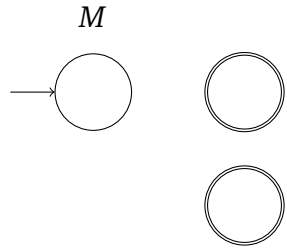
An alternative proof is to modify the construction that was used for the star operation. The only change is that a new start state should not be added. The argument that this construction works is almost the same as before. If $w \in A^+$, then $w = x_1 \cdots x_k$ with $k \geq 1$ and each $x_i \in A$. This implies that N can accept w by going through M k times, each time reading one x_i and then returning to the start state of M by using one of the new ε transitions (except after x_k).

Conversely, if w is accepted by N, then it must be that N uses the new ε "looping back" transitions k times, for some number $k \geq 0$, breaking w up into $x_1 \cdots x_{k+1}$, with each $x_i \in A$. This implies that $w \in A^+$. Therefore, $L(N) = A^+$.

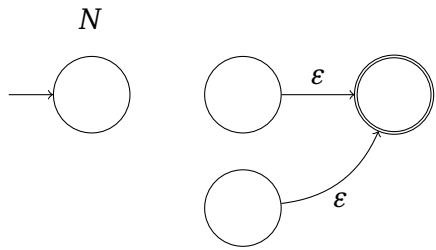
3.4.6. Suppose that L is regular and that it is recognized by a DFA M that doesn't have exactly one accepting state.

If M has no accepting states, then simply add one and make all the transitions leaving that state go back to itself. Since this new accepting state is unreachable from the start state, the new DFA still recognizes L (which happens to be the empty set).

Now suppose that M has more than one state. For example, it may look like this:



Then M can be turned into an equivalent NFA N with a single accepting state as follows:



That is, we add a new accepting state, an ε transition from each of the old accepting states to the new one, and we make the old accepting states non-accepting.

We can show that $L(N) = L(M)$ as follows. If w is accepted by M, then w leads to an old accepting, which implies that N can accept w by using one of the new ε transitions. If w is accepted by N, then the reading of w must finish with one of the new ε transitions. This implies that in M, w leads to one of the old accepting states, so w is accepted by M.

3.4.7. Suppose that L is recognized by a DFA M. Transform N into an equivalent NFA with a single accepting state. (The previous exercise says that this can be done.) Now reverse every transition in N: if a transition labeled a goes

from q_1 to q_2, make it go from q_2 to q_1. In addition, make the accepting state become the start state, and switch the accepting status of the new and old start states. Call the result N'.

We claim that N' recognizes $L^{\mathcal{R}}$. If $w = w_1 \cdots w_n$ is accepted by N', it must be that there is a path through N' labeled w. But then, this means that there was a path labeled $w_n \cdots w_1$ through N. Therefore, w is the reverse of a string in L, which means that $w \in L^{\mathcal{R}}$. It is easy to see that the reverse is also true.

Chapter 4

Regular Expressions

4.1 Introduction

4.1.5.

a) $(-\cup\varepsilon)DD^*$

b) $(-\cup\varepsilon)DD^*\cup(-\cup\varepsilon)D^*.DD^*$

c) $\_(\_\cup L\cup D)^*(L\cup D)(\_\cup L\cup D)^*\cup L(\_\cup L\cup D)^*$

d) $D^7\cup D^{10}\cup D^3{-}D^4\cup D^3{-}D^3{-}D^4$

4.2 Formal Definition

There are no exercises in this section.