Fontys University of Applied Sciences
School for ICT (FHICT)

Course Notes

Automata and Logic Engineering 2

1.0 version

Eindhoven, November 2020

Versioning

Version	Date	Changes
1.0	November 2020	Concept

Abstract

This document describes the assignments for the course Automata and Logic Engineering 2 (ALE2) (3 ECTS).

Contents

Acronyms

ALE2 Automata and Logic Engineering 2. 1

DFA deterministic finite state automata. 4, 5, 7–9, 13–18

FSA finite state automata. 3, 4

NFA non-deterministic finite state automata. 4, 5, 10–14, 17, 18

PDA push-down automata. 19–21, 23

RE regular expression. 10–12

UML Unified Modeling Language. 1

Chapter 1

Introduction

Automata and Logic Engineering 2 (ALE2) is a project based course of 6 weekly assignments. The assignments can be done in any modern object oriented language (we advise C# and Java). The assignments differ in difficulty. The table below gives a global indication (1 = relatively easy; 4 = relatively difficult), together with an advised week planning.

assignment	difficulty	week
1	1	1
2	2	2
3	2	3
4	3	4
5	3	5
6	2	6

Besides the specific contents of the assignments, the following general aspects of software engineering are assessed:

- UML modelling
- refactoring (in particular when your initial UML modelling was not that optimal)
- testing (module tests and system tests)
- code analysis (coverage, complexity)
- user interface design

Grading

The course is project based; to successfully pass the course the final grade should be greater or equal to 6. Apart from the assignments, further software engineering practices are graded accordingly:

- thorough testing
- good software design (classes, inheritance, interfaces, SOLID principles, design patterns)
- clear documentation
- spectacular new features

Chapter 2

Preliminaries

Strings

A non-empty finite set is called an *alphabet* and its elements are called *letters* or *symbols*.

Definition 2.1. A collection of words in an alphabet is called a *language*.
Let A the alphabet containing the following symbols: $A = \{a, b, c, d\}$
A language of A can then be:
$L_1 = \{aa, ab, bc, cd, ad\}$
$L_2 = \{a, b, bc, abc, acd\}$
where *words* are a, ab, abc, etc.

Definition 2.2. An empty sequence of symbols is called *empty word* and it is annotated as ε.

Note that the empty word is always considered a word in the language.

Example 2.1. Let $A_1 = \{x, y, z\}$ be a three-letter alphabet with symbols x, y and z. Then a possible language of A_1 can be:
$L_1 = \{x, y, z, xx, zz\}$
$L_2 = \{xy, yz, zzz, xxwwz\}$

Example 2.2. Let $A_2 = \{w, o\}$ be a two-letter alphabet with symbols w and o. Then a possible language of A_2 can be:
$L_1 = \{o, w, wow, woooow, wwwowww\}$

For further reading refer to the textbook by Hopcroft, Motwani and Ullman, *Introduction to Automata Theory, Languages and Computation, (2nd ed.)* (Hopcroft, Motwani & Ullman, 2001).

Chapter 3

Finite state automata

Languages can also be described by *graphs*.

Example 3.1. Let $A = \{a, b\}$ be a two-letter alphabet with symbols a and b. Then let $L = \{a, aa, ab\}$ a language of A.

To build a graph for L:

– start with the lined arrow

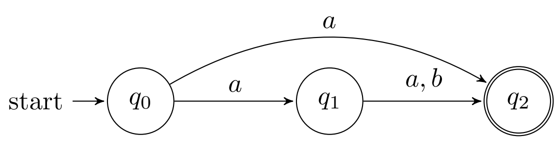
– you may follow any path in the direction of the arrows

– make a string of the labels of the arrows that you pass

– if your path ends in the node with the double border then the string belongs to the language described by this graph

– if the path does not end in a double bordered node, the string does not belong to that language

Note we will refer to graph nodes as *states* for the reminder of this document.



Above is the graph corresponding to language L of alphabet A, with accepted words a, aa and ab. This graph is called a *state diagram* and corresponds to a finite state automata (FSA).

Definition 3.1. A finite state automata (FSA) consists out of:

– alphabet: a finite set A of inputs (symbols)

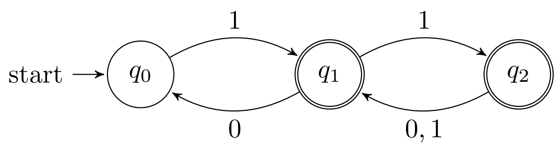
– vertices: a finite set S of states

– one initial (start) state

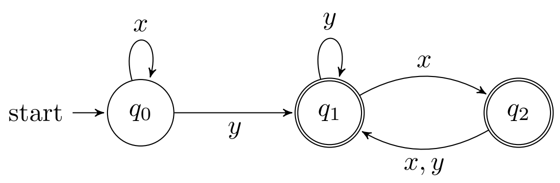
– accepting states: a subset Y of S of end-states (there can be more ending states) to indicate an accepted word

– next state function (transition): $\tau : A(l) \times S \to S$ for $\forall\ l \in A \cup \{\varepsilon\}$

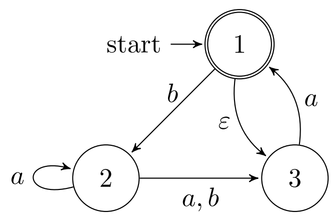
Example 3.2. Given the FSA below with alphabet $A = \{0, 1\}$ with letters 0 and 1, then, possible accepted words for the A language are: $1, 10, 100, 101, 1000000$, etc.



Example 3.3. Given the FSA below with alphabet $A = \{x, y\}$ with letters x and y, then, possible accepted words for the A language are: $x, y, xy, yy, xyy, yyy, yyxx, yyyyyyy$, etc.



Example 3.4. Given the FSA below with alphabet $A = \{a, b\}$ with letters a and b, then, possible accepted words for the A language are: $a, baa, baba$, etc.



Deterministic vs. non-deterministic finite automata

There is a stricter automata called deterministic finite state automata (DFA), while for the rest of the FSA we will refer to as non-deterministic finite state automata (NFA) for the remainder of the document. Unlike DFA, the NFA can have any number of outgoing transitions for a given symbol, including no transitions. Additionally, transitions are allowed that do not consume input, e.i. there are silent transitions (empty word transitions).

Definition 3.2. A deterministic finite state automata (DFA) consists out of:

- alphabet: a finite set A of inputs (symbols)
- vertices: a finite set S of states
- one initial (start) state
- accepting states: a subset Y of S of end-states (there can be more ending states) to indicate an accepted word
- next state function (transition): $\tau : A(l) \times S \to S$ for $\exists!\ l \in A$, meaning that for each state a transition takes place exactly once for each alphabet symbol, excluding the empty word ε

Definition 3.3. A non-deterministic finite state automata (NFA) consists out of:

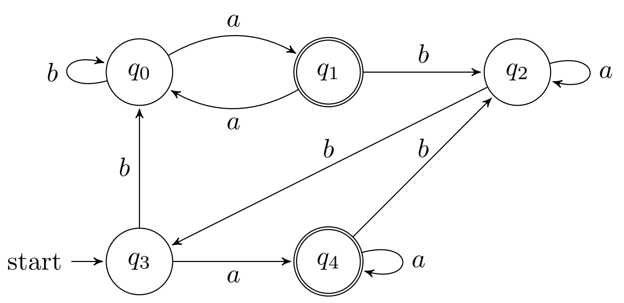
- alphabet: a finite set A of inputs (symbols)
- vertices: a finite set S of states

– one initial (start) state

– accepting states: a subset Y of S of end-states (there can be more ending states) to indicate an accepted word

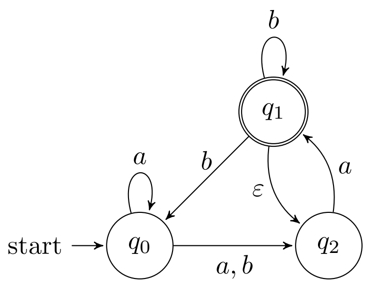
– next state function (transition): $\tau : A(l) \times S \to S$ for $\forall\ l \in A \cup \{\varepsilon\}$ meaning that for each state a transition takes with any alphabet symbol, including the empty word ε

Example 3.5. Deterministic finite state automata (DFA)
For $A = \{a, b\}$ a two-letter alphabet with symbols a and b, suppose graph:



This graph has is a DFA because each state has exactly one transition for each alphabet symbol and does not include the empty word ε.

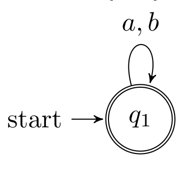
Example 3.6. Non-deterministic finite state automata (NFA)
For $A = \{a, b\}$ a two-letter alphabet with symbols a and b, suppose graph:



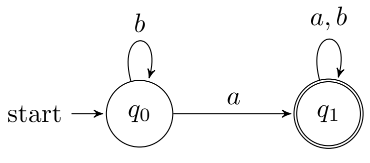
This graph has is a NFA because there are states that have transitions with the same symbols and includes the empty word ε.

Exercise 3.1. Indicate whether these automata are DFA or NFA:

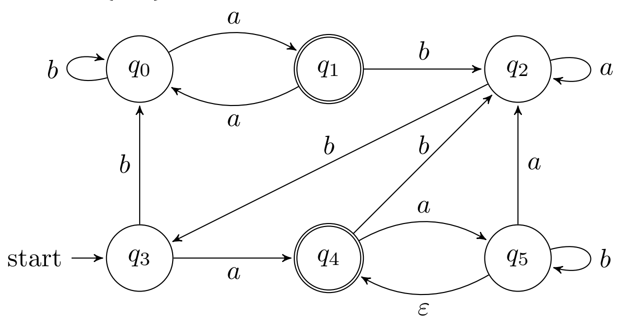
(a) For $A = \{a, b\}$ a two-letter alphabet with symbols a and b, suppose graph:



(b) For $A = \{a, b, c\}$ a three-letter alphabet with symbols a, b and c, suppose graph:

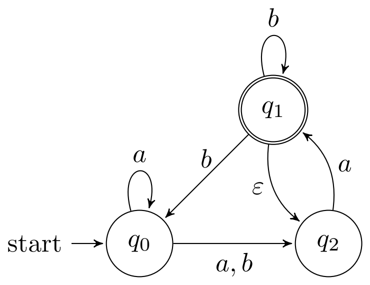


(c) For $A = \{a, b\}$ a two-letter alphabet with symbols a and b, suppose graph:



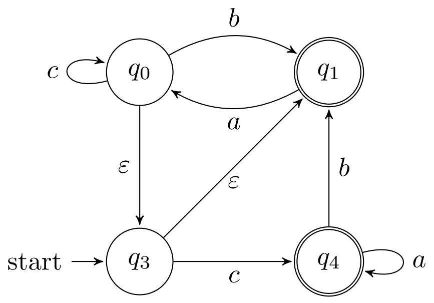
Exercise 3.2. Indicate whether the stated words are accepted by the automata:

(a) For $A = \{a, b\}$ a two-letter alphabet with symbols a and b, suppose graph:



Accepted words: *a*, *ba*, *aaa*, *babb*, *bba*

(b) For $A = \{a, b, c\}$ a three-letter alphabet with symbols a, b and c, suppose graph:



Accepted words: *a*, *ac*, *ab*, *aba*, *acccc*

3.1 Input file format for FSA

A line starting with '#' contains comments that may be ignored. Use this comment to give a short description of your constructed automaton. The following lines appear in the input: '*alphabet:*', '*states:*', '*final:*' and '*transitions:*'. Empty lines are allowed as well, for better readability.

'*alphabet:*' is followed by a row of lowercase characters, the alphabet. The symbols are concatenated without spaces or other separators. Note: the epsilon symbol is not part of the alphabet, so it should not be listed here.

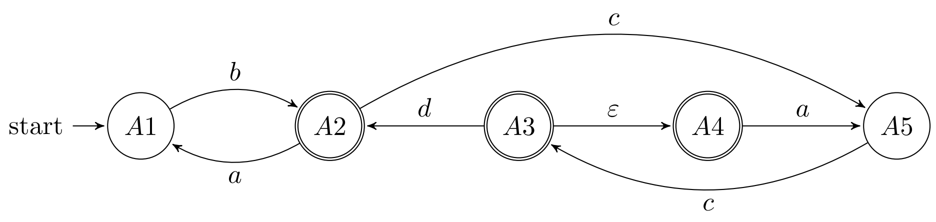
'*states:*' is followed by a list of strings, separated by commas. The first listed state is the initial state of the automaton.

'*final:*' is followed by a list of states (which must appear in the earlier mentioned state list), separated by commas. Those are the possible end states. Note: maybe there are no end states!

'*transitions:*' is at a separate line. The next lines contain each one name of an state, a comma, an alphabet symbol (or symbol '_' for an epsilon transition), the symbols '–>' and then a name of state The meaning of such a line should be evident. The alphabet symbol and the states must have been listed in the corresponding lines of the file. After the last transition, a separate line contains '*end.*'.

Example 3.7. As a summary, the input file (.txt) looks like, along with the corresponding graphical representation:

```
# comments

alphabet: abcd
states: A1,A2,A3,A4,A5
final: A2,A3,A4
transitions:
A1,b –> A2
A2,a –> A1
A2,c –> A5
A3,_ –> A4
A3,d –> A2
A4,a –> A5
A5,c –> A3
end.
```



Note that symbol '_' indicates an epsilon transition (empty word). It is not obliged, but it is perhaps convenient to use lower case letters for the alphabet and capitals for the states.

3.2 Input file format for FSA with test vectors

It is extremely convenient to add test cases of accepted/not accepted words by your automata. In the same input file, test vectors can be added.

'*dfa:*' indicates if the described automaton is a DFA ('*y*') or not ('*n*'). Please note that the example automaton of the previous paragraph is indeed not a DFA. See Assignment 1.

'*finite:*' indicatesif the automaton generates a finite number of words ('*y*') or not ('*n*'). Please note that the language of the example automaton is indeed not finite. See Assignment 4.

Each line after '*words:*' contains a word with an indication if it is accepted by the automaton ('*y*') or not ('*n*'). Please note that words like '*bababababab*' and '*bcc*' indeed belong to the language of the example automaton and that word '*ba*' does not belong to that language. The last word is followed by '*end.*', on a separate line. See Assignment 2.

Example 3.8. As a summary, the input file (.txt) looks like, extended from 3.7:

```
# comments

alphabet: abcd
states: A1,A2,A3,A4,A5
final: A2,A3,A4
transitions:
A1,b –> A2
A2,a –> A1
A2,c –> A5
A3,_ –> A4
A3,d –> A2
A4,a –> A5
A5,c –> A3
end.

dfa:n
finite:n
words:
b,y
a,n
ba,n
bababababab,y
bcc,y
bda,y
bcca,y.
```

3.3 Assignment 1

Write a program that reads a finite automaton (according to the input format as described above) and which indicates if it is a DFA or not. Test weather the '*dfa*' vector in the input file matches. Note that sometimes the test vector is wrong and you need to indicate this in your program (e.i. show a test result in the user interface).

Show a picture of the automaton (see Appendix A).

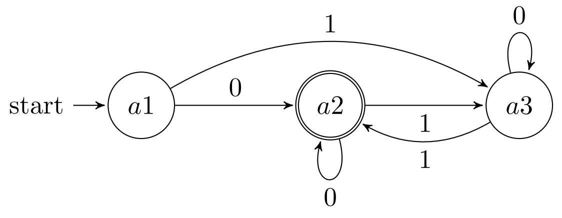
Example 3.9. Let input file:

```
# simple dfa

alphabet: 01
states: a1,a2,a3
final: a2
transitions:
a1,0 –> a2
a1,1 –> a3
a2,0 –> a2
a2,1 –> a3
a3,0 –> a3
a3,1 –> a2
end.
```

```
(continue)

dfa:y
finite:n
words:
0,y
01,n
10,n
00000000,y
00110011,n
101,y
1010,y.
```

You need to read the file and build the automaton. Then you need to check if it satisfies all the condition of a DFA, according to definition 3.2.

The automaton graph will like so:



This graph has is a DFA because each state has exactly one transition for each alphabet symbol and does not include the empty word.

3.4 Assignment 2

Extend the program such that it indicates if a given string is accepted by the automaton or not. You must be able to read the strings from the input file as well as from the user interface (text field where user types a word) and decide whether it is accepted or not by the automaton.

Example 3.10. For the input file given above, words like $0, 101$ and 1010 are accepted. Note that in the file, the test vector 00110011 is mentioned as not accepted, although it is actually accepted by the automaton; you need to indicate this in your program (e.i. show a test result in the user interface).

Chapter 4

Regular expressions

Apart from graphs, we can represent a language as a *regular expression.*

Definition 4.1. A regular expression (RE) an alphabet A is a string of letters from A with the following symbols: . (concatenation), | (choice) and * (repetition).

Note that we can leave out the . (concatenation) from the RE.

Example 4.1. Let $A = \{a, b\}$
Examples of RE r and its corresponding language $L(r)$:

$r = a$, then $L(r) = \{a\}$
$r = a.b$, then $L(r) = \{ab\}$
$r = a|b$, then $L(r) = \{a, b\}$
$r = a^*$, then $L(r)$ consists of all powers of a including empty word ε
$r = aa^*$, then $L(r)$ consists of positive powers of a (concatenation symbol . is left out: $aa^* \equiv a.a^*$)
$r = a|b^*$, then $L(r) = \{\varepsilon, a, b, bb, bbb, ...\}$
$r = (a|b)^*$, then $L(r) = \{a, b\}^*$

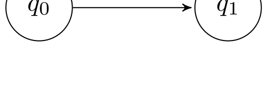
Note that ε is also a RE, where for $r = \varepsilon$, then $L(r) = \{\emptyset\}$.

NFA from regular expression

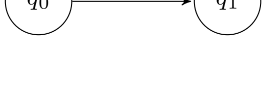
Thompson's construction algorithm is a method of transforming a regular expression (RE) into an equivalent non-deterministic finite state automata (NFA). More information about how the algorithm is implemented read (Wikipedia, 2020).

The graphical representation of RE sub-expressions is defined below:

Definition 4.2. Graphical representation of ε:



Definition 4.3. Graphical representation of symbol a of the input alphabet:



Definition 4.4. Graphical representation of . (concatenation) over two symbols a and b of the input alphabet:

$$\begin{tikzcd}
q_0 \arrow[r, "a"] & q_1 \arrow[r, "\varepsilon"] & q_2 \arrow[r, "b"] & q_3
\end{tikzcd}$$

Definition 4.5. Graphical representation of | (choice) over two symbols a and b of the input alphabet:

$$\begin{tikzcd}
 & q_2 \arrow[r, "a"] & q_4 \arrow[dr, "\varepsilon"] & \\
q_0 \arrow[ur, "\varepsilon"] \arrow[r, "\varepsilon"] & q_1 \arrow[r, "b"] & q_3 \arrow[r, "\varepsilon"] & q_5
\end{tikzcd}$$

Definition 4.6. Graphical representation of * (repetition) over symbol a of the input alphabet:

$$\begin{tikzcd}
q_0 \arrow[r, "\varepsilon"] \arrow[rrr, bend left, "\varepsilon"] & q_1 \arrow[r, "a"] & q_2 \arrow[r, "\varepsilon"] & q_3 \arrow[ll, bend left, "\varepsilon"]
\end{tikzcd}$$

Example 4.2. Let $r =^* (a|b)$, then the resulting NFA:

$$\begin{tikzcd}
 & & & q_2 \arrow[r, "a"] & q_4 \arrow[dr, "\varepsilon"] & & \\
\text{start} \arrow[r] & q_6 \arrow[r, "\varepsilon"] \arrow[rrrrr, bend right, "\varepsilon"] & q_0 \arrow[ur, "\varepsilon"] \arrow[r, "\varepsilon"] & q_1 \arrow[r, "b"] & q_3 \arrow[r, "\varepsilon"] & q_5 \arrow[r, "\varepsilon"] & q_7 \arrow[lllll, bend right, "\varepsilon"]
\end{tikzcd}$$

Exercise 4.1. Convert to NFA the following REs r:

(a) $r = .(^*(a),^*(b))$

(b) $r = (a.b).(c.d)$

(c) $r =^* (^*(a))$

(d) $r =^* ((a|b)|c)|(e.f)$

4.1 Input format for regular expressions

The RE format for your program should be in the prefix notation, shown in the table below:

regular expression	prefix
ε	_
a	a
$a.b$	$.(a,b)$
$a\vert b$	$\vert(a,b)$
$*$	$(a)^*$

Example 4.3. Let $A = a, b$ Examples of RE r and corresponding prefix notation:

$r = a^*$, then $^*(a)$
$r = a|b^*$, then $|(a,^*(b))$
$r = (a|(a.b))^*$, then $^*(|(a, .(a,b)))$
$r = b.((a|b)|(a.b))^*$, then $.(b,^*(|(|(a,b), .(a,b))))$

4.2 Assignment 3

Extend your program such that a NFA is constructed from a given RE with the Thompson's construction. This NFA must be written to a file, see section 3.1 for the file format.

Example 4.4. Let input RE $r = |(.(a,b),^*(c))$ for letters a, b and c. Then, the program output should be a text file containing a NFA corresponding to r, shown below.

```
# from regular expression |(.(a,b),*(c))

alphabet: abc
states: q1,q2,q3,q4,q5,q6,q7,q8
final: q2
transitions:
q1,_ --> q3
q1,_ --> q6
q3,a --> q4
q4,b --> q5
q5,_ --> q2
q6,_ --> q10
q6,c --> q7
q7,_ --> q8
q8,_ --> q6
q8,_ --> q2
end.
```

Convert NFA to DFA

We can convert a non-deterministic finite state automata (NFA) to deterministic finite state automata (DFA) using the Powerset construction (Wikipedia, 2019).

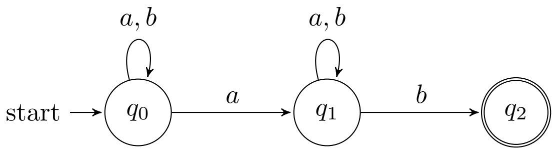
Definition 4.7. Let set $V = \{a, b, c\}$, then the powerset of V is the collection of all the subsets $P(V) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{ab\}, \{ac\}, \{bc\}, \{abc\}\}$

Definition 4.8. Consider the following non-deterministic finite state automata (NFA):

- alphabet: $A = \{a, b\}$
- states: $S = \{q0, q1, q2\}$
- initial state: $S_0 = \{q0\}$
- accepting states: $Y = \{q2\}$, where $Y \subseteq S$

A resulting deterministic finite state automata (DFA) of the above stated NFA:

- alphabet: A
- possible states: $T \subseteq P(S)$, where $P(S) = \{\emptyset, \{q0\}, ..., \{q0, q1\}, \{q0, q2\}, ..., \{q0, q1, q2\}\}$
- initial state: S_0
- accepting states: all states in T which have states in Y



We summarize state transitions in a table. We start with the initial state $S_0 = \{q0\}$ and for each letter in A we note the reachable state(s). From state $q0$ there is a transition with a to state $q0$ and state $q1$, and with b to state $q0$: As there are more reachable states with a, we group them into one set and we consider this a new state $\{q0, q1\}$:

	a	b
$q0$	$\{q0, q1\}$	$q0$

We then consider the new reachable states (in this case just state $\{q0, q1\}$) to further determine the next reachable states. From the set $\{q0, q1\}$ we can conclude that state $q0$ and $q1$ are reachable with a and state $q0$, $q1$ and $q2$ are reachable with b. Grouping the reachable states, we get $\{q0, q1\}$ and $\{q0, q1, q2\}$:

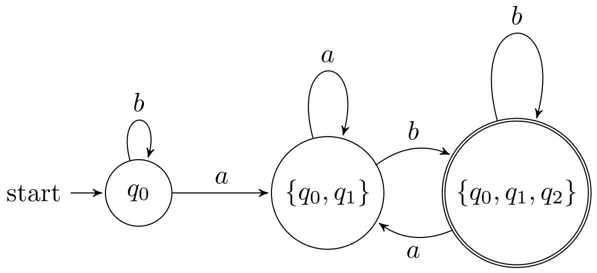
	a	b
$q0$	$\{q0, q1\}$	$q0$
$\{q0, q1\}$	$\{q0, q1\}$	$\{q0, q1, q2\}$

As reachable states $q0$ and $\{q0, q1\}$ have already been checked, the new reachable state $\{q0, q1, q2\}$ is further considered. From the set $\{q0, q1, q2\}$ we can conclude that state $q0$ and $q1$ are reachable with a and state $q0$, $q1$ and $q2$ are reachable with b. Grouping the reachable states, we get $\{q0, q1\}$ and $\{q0, q1, q2\}$:

	a	b
$q0$	$\{q0, q1\}$	$q0$
$\{q0, q1\}$	$\{q0, q1\}$	$\{q0, q1, q2\}$
$\{q0, q1, q2\}$	$\{q0, q1\}$	$\{q0, q1, q2\}$

Once all new reachable states have been checked, we can define the states and final state(s) for the resulting DFA: $T = \{q0, \{q0, q1\}, \{q0, q1, q2\}\}$ with final state $\{q0, q1, q2\}$.

Reading the final table, we come up with the following DFA:

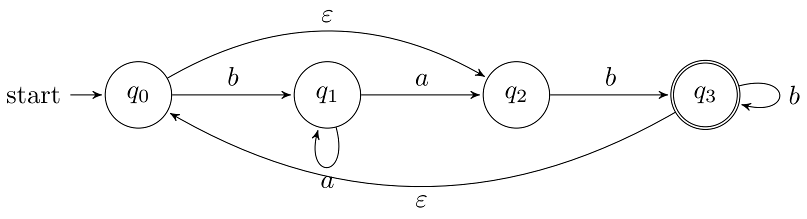


Definition 4.9. For a NFA with n states, there are at most 2^n states in your resulting DFA.

Note that for NFAs that contain the empty word ε, you need to construct the *epsilon closures* before you summarize reachable states.

Definition 4.10. Epsilon closure (ε-closure) for a given state is a set of states which can be reached from the states with ε moves including the state itself.

Let following NFA with $A = \{a, b\}$ with two ε transitions between $q0, q2$ and $q3, q0$:



Let ε-closures:

ε-closure$(q0) = \{q0, q2\}$

ε-closure$(q1) = \{q1\}$

ε-closure$(q2) = \{q2\}$

ε-closure$(q3) = \{q0, q2, q3\}$

If the ε-closure includes other states but the state itself, you need to make sure you will include the other states when you summarize the reachable states.

Since the ε-closure$(q0)$ includes state $q2$ as well, we start with this state instead. From the set $\{q0, q2\}$ we can conclude that there are no reachable states with a and state $q0, q1, q2$ and $q3$

are reachable with b. Note that with b we are able to reach all the states due to the ε-closures. Specifically, with $q0$ we reach $q1$ and with $q2$ we reach $q3$ which has ε-closure($q3$) $= \{q0, q2, q3\}$ that further extends with the ε-closure($q0$) $= \{q0, q2\}$.

	a	b
$\{q0, q2\}$	$\emptyset$	$\{q0, q1, q2, q3\}$

We consider the new reachable state $\{q0, q1, q2, q3\}$ to further determine the next reachable states. From the set $\{q0, q1, q2, q3\}$ we reach with a state $q1$ and $q2$ and with b state $q0, q1, q2$ and $q3$. Grouping the reachable states, we get $\{q1, q2\}$ and $\{q0, q1, q2, q3\}$:

	a	b
$\{q0, q2\}$	$\emptyset$	$\{q0, q1, q2, q3\}$
$\{q0, q1, q2, q3\}$	$\{q1, q2\}$	$\{q0, q1, q2, q3\}$

We consider the new reachable state $\{q1, q2\}$ to further determine the next reachable states. From the set $\{q1, q2\}$ we reach with a state $q1$ and $q2$ and with b state $q0, q2$ and $q3$:

	a	b
$\{q0, q2\}$	$\emptyset$	$\{q0, q1, q2, q3\}$
$\{q0, q1, q2, q3\}$	$\{q1, q2\}$	$\{q0, q1, q2, q3\}$
$\{q1, q2\}$	$\{q1, q2\}$	$\{q0, q2, q3\}$

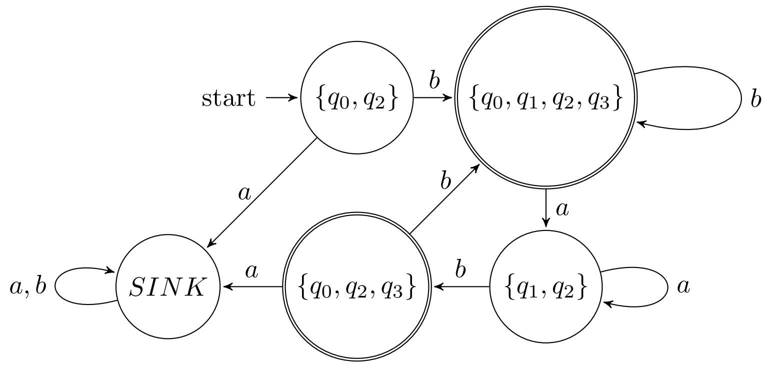
We consider the new reachable state $\{q0, q2, q3\}$ to further determine the next reachable states. From the set $\{q0, q2, q3\}$ we do not reach with a any state but with b state $q0, q1, q2$ and $q3$:

	a	b
$\{q0, q2\}$	$\emptyset$	$\{q0, q1, q2, q3\}$
$\{q0, q1, q2, q3\}$	$\{q1, q2\}$	$\{q0, q1, q2, q3\}$
$\{q1, q2\}$	$\{q1, q2\}$	$\{q0, q2, q3\}$
$\{q0, q2, q3\}$	$\emptyset$	$\{q0, q1, q2, q3\}$

Finally, as all reachable states have been checked, if there is an $\emptyset$ state, we need to consider it as well, to satisfy the DFA. This state is also referred to as $SINK$.

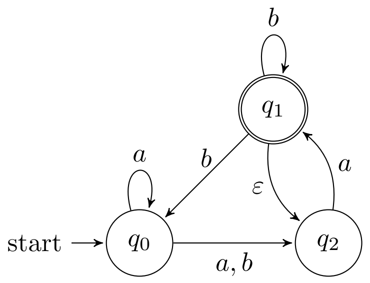
	a	b
$\{q0, q2\}$	$\emptyset$	$\{q0, q1, q2, q3\}$
$\{q0, q1, q2, q3\}$	$\{q1, q2\}$	$\{q0, q1, q2, q3\}$
$\{q1, q2\}$	$\{q1, q2\}$	$\{q0, q2, q3\}$
$\{q0, q2, q3\}$	$\emptyset$	$\{q0, q1, q2, q3\}$
$\emptyset$	$\emptyset$	$\emptyset$

Once all new reachable states have been checked, we can define the states and final state(s) for the resulting DFA: $T = \{\{q0, q2\}, \{q1, q2\}, \{q0, q2, q3\}, \{q0, q1, q2, q3\}, \emptyset\}$ with final state $\{q0, q2, q3\}$ and $\{q0, q1, q2, q3\}$.

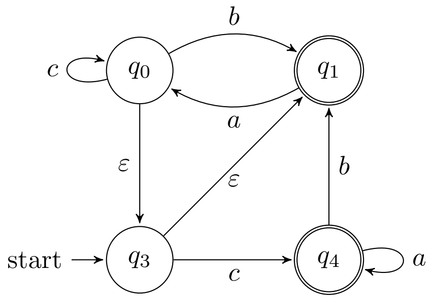


Exercise 4.2. Convert the following NFAs to DFAs:

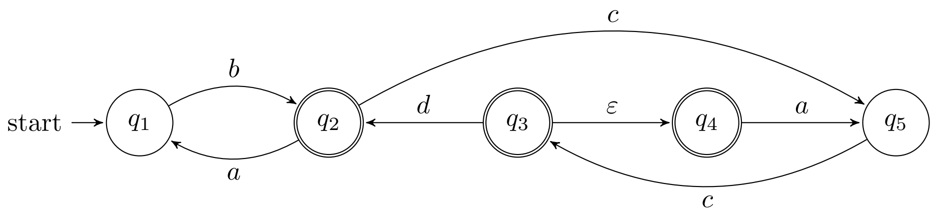
(a) Suppose $A = \{a, b\}$ a two-letter alphabet with symbols a and b:



(b) Suppose $A = \{a, b, c\}$ a three-letter alphabet with symbols a, b and c:



(c) Suppose $A = \{a, b, c, d, e\}$ a five-letter alphabet with symbols a, b, c, d and e:



4.3 Assignments 4

Extend your program such that a NFA is converted into a DFA with the Powerset construction (Wikipedia, 2019). This DFA must be written to a file, see section 3.1 for the file format. The test vectors of the original automaton need to be written to the new file as well.

Example 4.5. Let NFA input .txt file:

```
alphabet: abc
states: q0,q1,q2
final: q1
transitions:
q0,a –> q1
q1,b –> q2
q2,a –> q0
q2,b –> q1
end.
dfa:n
finite:n
words:
abb,y
ends.
```

Your program should convert this NFA to a DFA accordingly and output a .txt file:

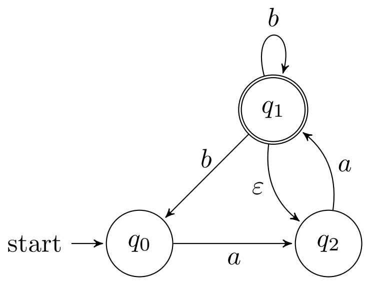
```
# nfa to dfa

alphabet: abc
states: q0,q1,q2,SINK
final: q1
transitions:
q0,a –> q1
q0,b –> SINK
q0,c –> SINK
q1,a –> SINK
q1,b –> q2
q1,c –> SINK
q2,a –> q0
q2,b –> q1
q2,c –> SINK
SINK,a –> SINK
SINK,b –> SINK
SINK,c –> SINK
end.

dfa:y
finite:n
words:
abb,y
ends.
```

4.4 Assignment 5

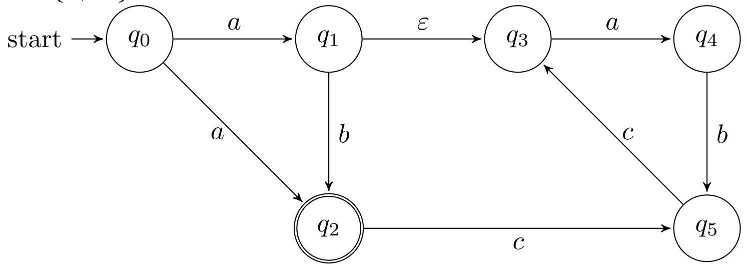
Extend your program such that it indicates if the number of generated words by the automaton is finite or not. If it is finite, list the words as well.

Example 4.6. Suppose the following automata:

(a) $L = \{\infty\}$



(b) $L = \{a, ab\}$



Chapter 5

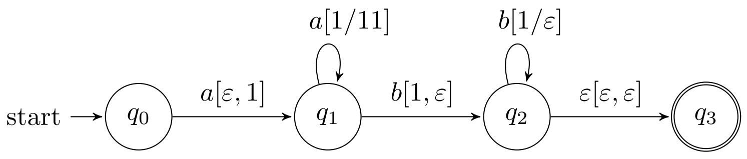
Push-down automata

There are other types of automata, namely automata with memory in the form of a stack, so-called push-down automata (PDA).

Definition 5.1. A push-down automata (PDA) consists out of:

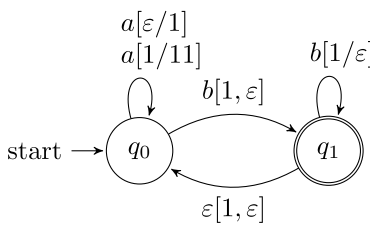
- alphabet: a finite set A of inputs (letters)
- stack alphabet: a finite set Δ of stack inputs (letters)
- empty stack: $\Delta^* = \{\emptyset\}$
- vertices: a finite set S of states
- one initial (start) state
- accepting states: a subset Y of S of end-states (there can be more ending states) to indicate an accepted word
- next state function (transition): $\tau : A(l) \times S \times \Delta^* \to S$ for $\forall\ l \in A \cup \{\varepsilon\}$ meaning that for each state a transition takes with any alphabet letter, including the empty word ε and $\Delta^* = \Delta \cup \{\emptyset\}$, Δ^* denoting the content of the stack.

The language $L(P)$, called the language accepted by the PDA requires the input string (word) to be processed completely and be accepted by a final state, but the stack has to be empty.

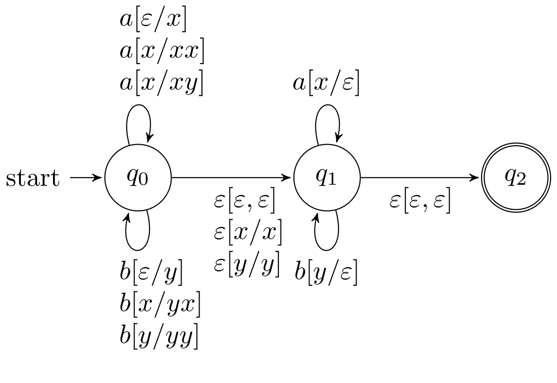


Let us look at the example of a push-down automaton above. We indicate the initial state $q0$ as usual. Initially, the stack is assumed to be empty. When we write $q \xrightarrow{a[\varepsilon,1]} q'$ this means that the machine, when it is in state q and ε (or no letter, thus, stack is empty) is the top element of the stack, can consume the input letter a, replace the top element ε by the string 1 (e.i. push 1 to the stack) and thereby move to state q'. In state $q0$, we can input a symbol a, thereby replacing the empty stack by the stack containing a single data element 1 while moving from state $q0$ to state $q1$. When we write $q \xrightarrow{a[1,11]} q'$ this means that the machine, when it is in state q and 1 is the top element of the stack, can consume the input letter a, replace the top element 1 by the string 11 (e.i. push 1 to the stack) and thereby move to state q'. When we write $q \xrightarrow{b[1,\varepsilon]} q'$ this means that the machine, when it is in state q and 1 is the top element of the stack, can consume the input letter a, replace the top element 1 by the empty string ε (e.i. pop the top element from the stack) and thereby move to state q'. In the state $q1$, we can either input letter a again, replacing the top element 1 by twice the item 1 indicated by 11 (thus effectively adding an item 1 on top), and remain in state $q1$, or alternatively input the letter b, popping the top item, which is an item 1, by putting nothing in its place as indicated by the string ε, and going to state $q2$. There, we can read in a b repeatedly, popping 1's, but we must terminate and go to final state $q3$ if (and only if) the stack has become empty. When we write $q \xrightarrow{\varepsilon[\varepsilon,\varepsilon]} q'$ this means that the machine, when it is in state q and ε (e.i. empty string) is the top element of the stack, move to state q' and the stack does not get modified.

Example 5.1. Given the PDA below with alphabet $A = \{a, b\}$ with letters a and b and stack alphabet $\Delta = \{1\}$ with letter 1, then, possible accepted words are: $ab, aabb, aaabbb$, etc.

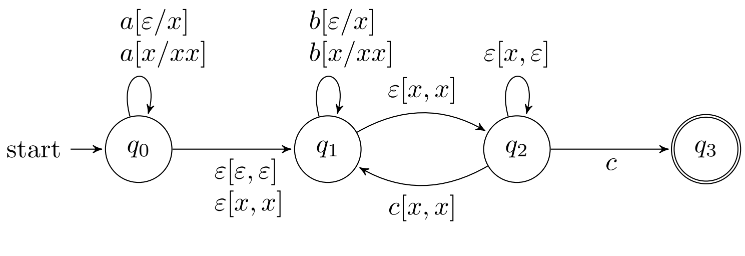


Example 5.2. Given the PDA below with alphabet $A = \{a, b\}$ with letters a and b and stack alphabet $\Delta = \{x, y\}$ with letters x and y, then, possible accepted words are: $\varepsilon, aa, bb, abab, aabbba$, etc.



Note that you may have multiple letters in the stack alphabet.

Example 5.3. Given the PDA below with alphabet $A = \{a, b\}$ with letters a and b and stack alphabet $\Delta = \{x, y\}$ with letters x and y, then, possible accepted words are: $a, abc, aacc$, etc.



Note that you may have transitions where the stack movement is missing, then you may consume the letter of the transition and the stack does not get modified.

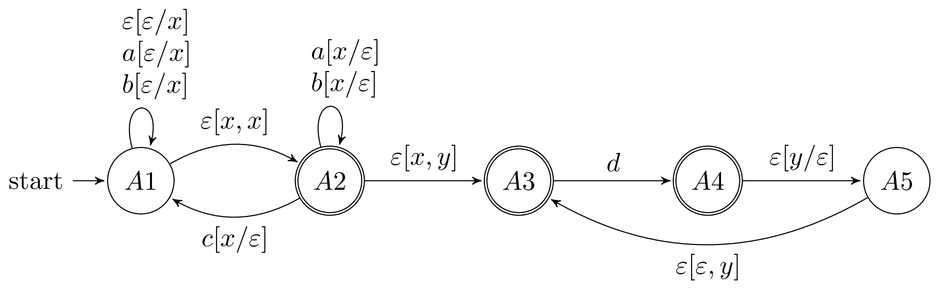
5.1 Input file format for PDA

The input format for PDA differs in three aspects:

- After the '*alphabet:*' line there is a line with '*stack:*'. This line contains the symbols that can by pushed onto the stack.
- A transition looks like: $A1, a[x, y]$ –> $A2$, where x is a stack symbol that is removed from the stack, and y is a stack symbol that is pushed on the stack.
- Test vectors '*dfa:*' and '*finite:*' are obsolete, you can ignore them

Example 5.4. As a summary, the input file (.txt) looks like, along with the corresponding graphical representation:

```
# comments

alphabet: abcd
stack: xy
states: A1,A2,A3,A4,A5
final: A2,A3,A4
transitions:
A1,a[_,x] –> A1
A1,b[_,x] –> A1
A1,_[_,x] –> A1
A1,_[_,_] –> A2
A2,a[x,_] –> A2
A2,b[x,_] –> A2
A2,c[x,_] –> A1
A2,_[x,y] –> A3
A3,d –> A4
A4,_[y,_] –> A5
A5,_[_,y] –> A3
end.

words:
ad,y
bd,y
abd,y
bad,y
a,n
ba,n
end.
```



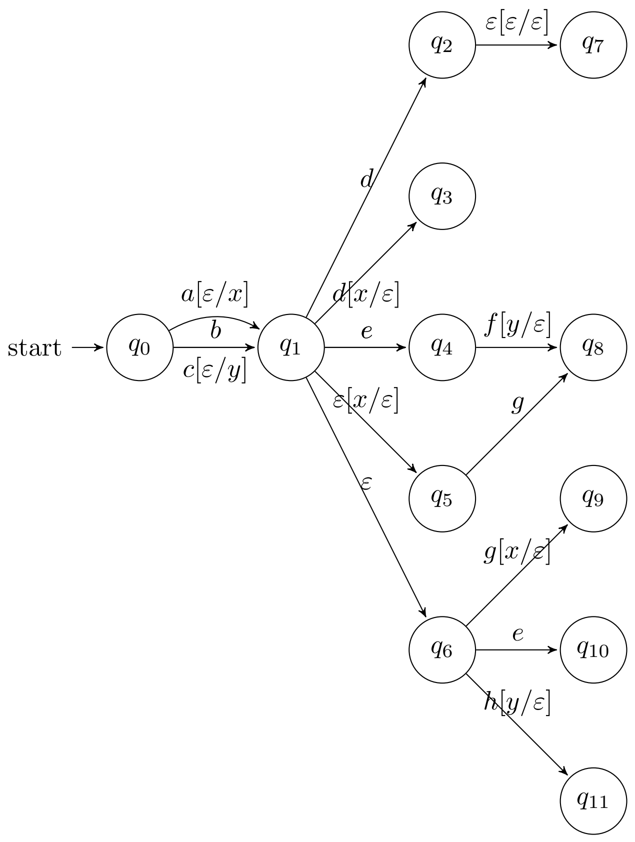
5.2 Transition priorities

It might occur that at a given moment several transitions are possible, but the following precedence of rules enforces a strict deterministic behavior:

1. transition whose [symbol + pop stack] matches the [current input symbol + current top stack] (if the stack is not empty)
2. transition with epsilon pop stack whose symbol matches the current input symbol
3. transition with epsilon symbol whose pop stack matches the current top stack (if the stack is not empty)
4. transition with epsilon symbol and epsilon pop stack

Having multiple transitions from the same state with the same priority is not allowed, since this would interfere with the determinism.

Let us look at the example of a push-down automaton below.



– for input word ad, rule 1 is applied when handling input character d to reach final state $q3$ (so: not state $q7$ by following rule 2)

– for input word be, rule 2 is applied when handling input character e (note that the stack is empty) to reach state $q4$ (so: not state q_{10} by following rule 4)

– for input word cef, rule 2 is applied when handling input character e (note that the stack is not empty)

- for input word ag, rule 3 is applied when handling input character g to reach final state q_8 (so: not state q_9 by following rule 4)
- for input word b, rule 4 is applied to reach final state q_6
- for input word ch, rule 4 is applied when handling input character h (note that the stack is not empty) to reach final state q_{11}

5.3 Assignment 6

Extend your program such that a PDA specification can be read, and that it can test if the PDA accepts a given string. Note that a word only belongs to the language when the PDA has reached a final state and when the stack is empty.

References

Hopcroft, J., Motwani, R. & Ullman, J. (2001). *Introduction to Automata Theory, Languages, and Computation (2nd Edition)* (Vol. 32). doi: 10.1145/568438.568455 2

Wikipedia. (2019). *Powerset construction.* 13, 17

Wikipedia. (2020). *Thompson's construction.* 10

Appendix A

Graphical representation of an automaton with GraphViz

To add a graphical picture of your automaton, please follow the following steps:

1. install GraphViz
2. adapt your $PATH (Linux) or %Path% (Windows) environment variable
3. in your application:

   3.1. generate a text file (e.g. abc.dot), similar to this:

   ```
   digraph myAutomaton {
   rankdir=LR;
   ”” [shape=none]
   ”A” [shape=doublecircle]
   ”B” [shape=doublecircle]
   ”C” [shape=circle]
   ”SINK” [shape=circle]
   ”” -> ”C”
   ”A” -> ”A” [label=”a”]
   ”B” -> ”SINK” [label=”a [x/ε]”]
   ”C” -> ”B” [label=”b [y/z]”]
   ”C” -> ”A” [label=”ε”]
   ”C” -> ”A” [label=”d”]
   ”C” -> ”C” [label=”a”]
   }
   ```

   3.2. start the GraphViz executable with the command line: dot -Tpng -oabc.png abc.dot

   3.3. (wait until this executable is finished)

   3.4. show a picture (e.g. abc.png), for example in a PictureBox

In C#, the steps 3.2 - 3.4 could look like:

```
WriteDot("abc.dot");
// your method to write an automaton into a dot-format file
Process dot = new Process();
dot.StartInfo.FileName = "dot.exe";
dot.StartInfo.Arguments = "-Tpng -oabc.png abc.dot";
dot.Start();
```

```
dot.WaitForExit();
myPictureBox.ImageLocation = "abc.png";
```

In Java, the steps could look like:

```
String[] cmd = { "dot.exe", "-Tpng", "-oabc.png", "abc.dot" };
Process p = Runtime.getRuntime().exec(cmd);
p.waitFor();
File file = new File("abc.png");
Image image = new Image(file.toURL().toString());
myWidget.setImage(image);
```