Linnéuniversitetet
Kalmar Växjö

Assignment 1:
Language and Logic

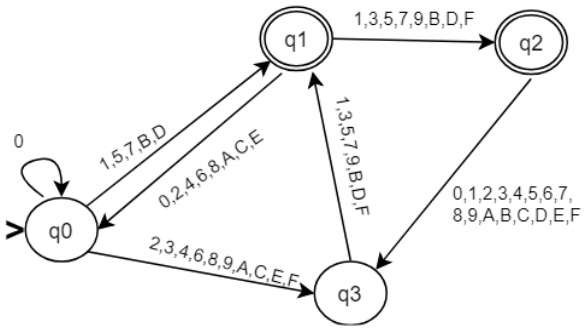
Ruth Dirnfeld
rd222dv@student.lnu.se

Faculty of Technology
Department of Computer Science

Table of contents

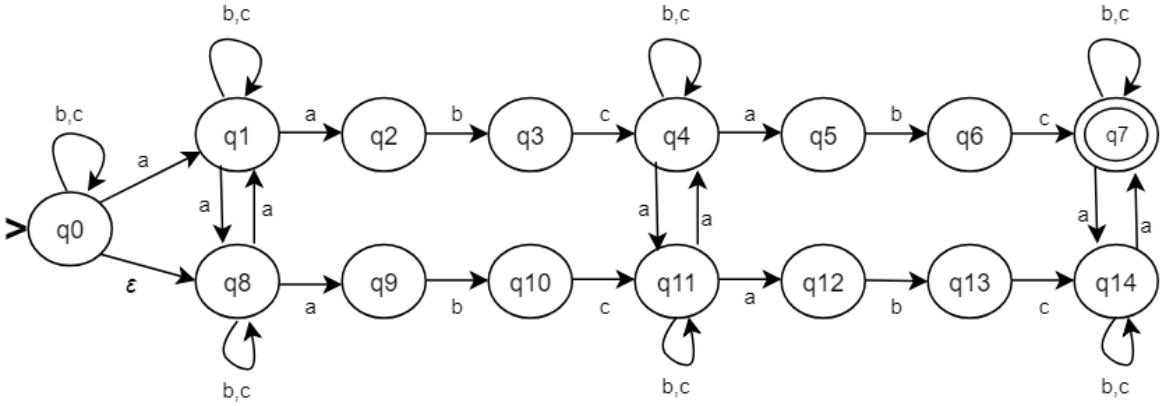
1. Designing an NFA for the following and conversion to regular expression:

1.1. All Hexadecimal numbers divisible by 6 with a remainder of 1 or 5.



1.2. All strings over {a,b,c} with at least two occurrences of abc and an odd number of a's.

In the following NFA we can see all strings over {a,b,c} with at least two occurrences of abc and an odd number of a's.



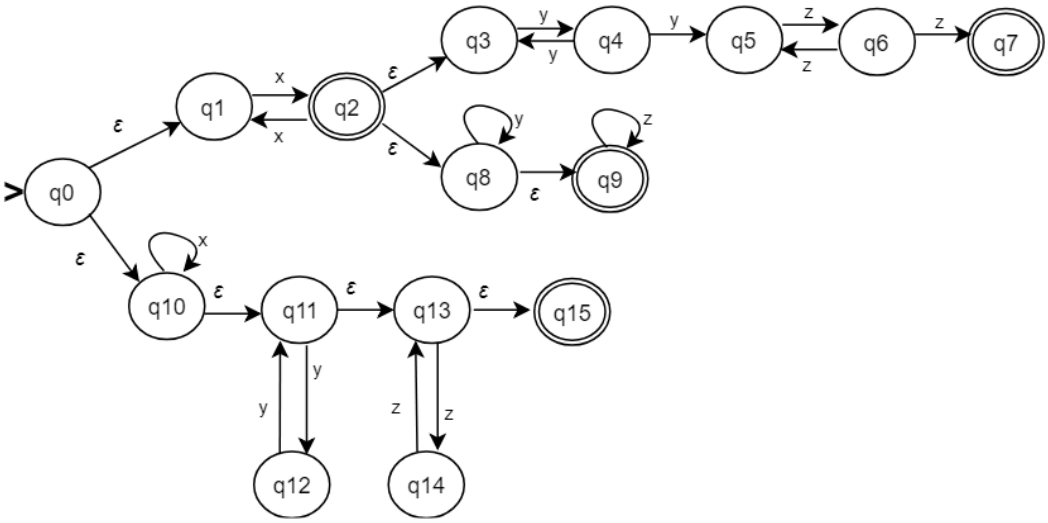
Regular Expression to the above NFA:

{abc}^2m {b,c}*a^(2n){abc}^2l{b,c}*a^(2k){abc}^2i
i,k,l,m,n >= 0
m+l+i >= 1

1.3. All strings over {x,y,z} where x occurs an odd number of times or y and z occur an even number of times.

In the following NFA we can see the three accepted states:

- IF x is odd THEN y & z are even (x = 1,3,5,.. + y&z = 0,2,4,..)
- IF x is odd THEN y & z are anything (x = 1,3,5,.. + y&z = 0,1,2,3,4,..)
- IF x is anything THEN y & z are even (x = 0,1,2,3,4,.. + y&z = 0,2,4,..)



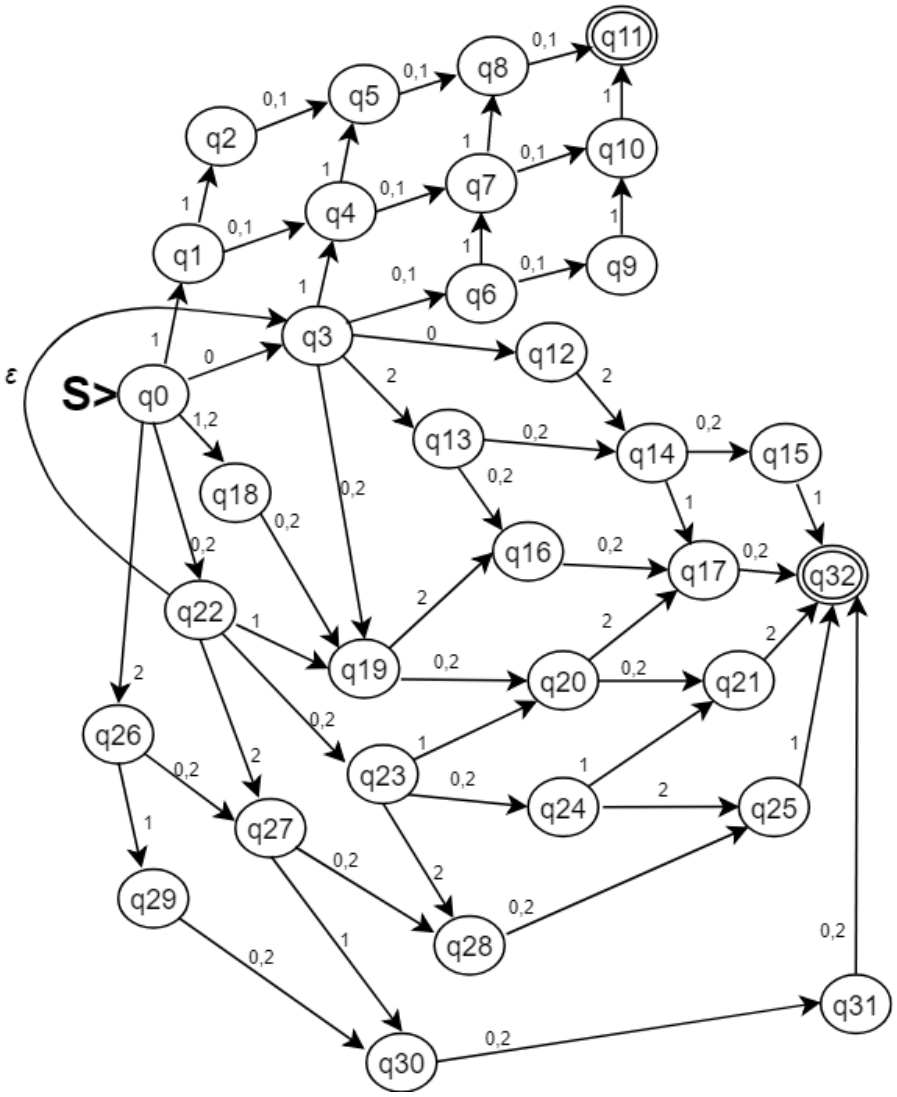
Regular Expression to the above NFA:

x^ny^mz^l

when n = odd -> m,l = even
when n = odd m,l = anything
when n = anything -> m,l = even

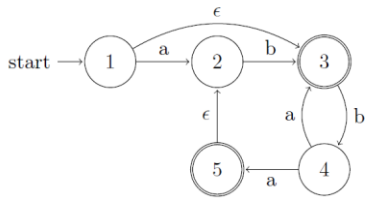
1.4. L={w 2 {0,1,2}*} where a string of exactly 5 symbols contains at least two 1's or at least one 2(but not both).

In the following NFA we can see all options of at least two 1's || at least one 2 || max one 1 and at least one 2.

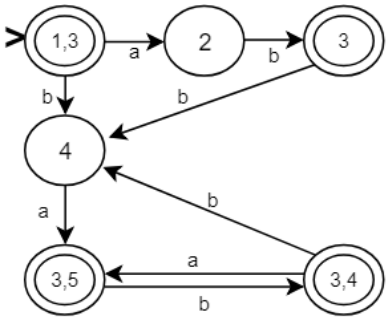

ε
S>
q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
q12
q13
q14
q15
q16
q17
q18
q19
q20
q21
q22
q23
q24
q25
q26
q27
q28
q29
q30
q31
q32
0,1
0
1
2
1,2
0,2

2. Converting NFAs to a DFAs

2.1.1. NFA



2.1.2. DFA

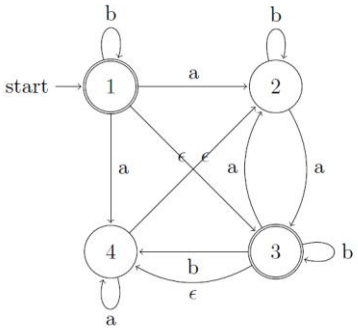


	a	b
>{1, 3}*	2	4
{2, 5}*	--	3
4	3, 5	--
{3, 5}*	--	3, 4
{3, 4}*	3, 5	4
2	--	3
3	--	4

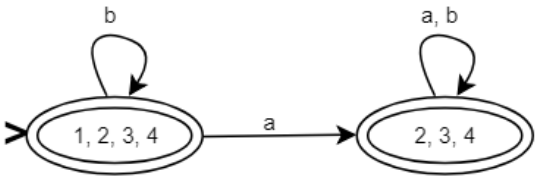
Regular Expression: {{ab}},{ba}*,{bab}*}*

With the above table we can see which states reach with a or b which other states. This table was later implemented into the above DFA.

2.2.1. NFA



2.2.2. DFA



	a	b
>{1, 2, 3, 4}*	2, 3, 4	1, 2, 3, 4
{2, 3, 4}*	2, 3, 4	2, 3, 4

Regular Expression: {b}a{a,b}

With the above table we can see which states reach with a or b which other states. This table was later implemented into the above DFA.

3. Prove that the following languages are or are not regular.

3.1. L is the language with alphabet {x,y}, number of x's is equal to the number of y's.

p >= 1
$w = x^p y^p$
w = abc
p = 7

a = xxx
b = xxxx
c = yyyyyyy

$ab^n c$ has to be part of L

$ab^2 c$ => xxx xxxxxxxx yyyyyyy
11 != 7

The Language with alphabet {x,y} is not regular, since the number of x's has to be equal to the number of y's and $ab^n c$ has to be part of L. In the above proof, we can see that the number of x's is not equal to the number of y's.

3.2. L is the language with alphabet {(,)} and balanced parentheses. i.e. (()) is accepted but ()) is not

p >= 1
$w = x^p y^p$
w = abc
p = 7

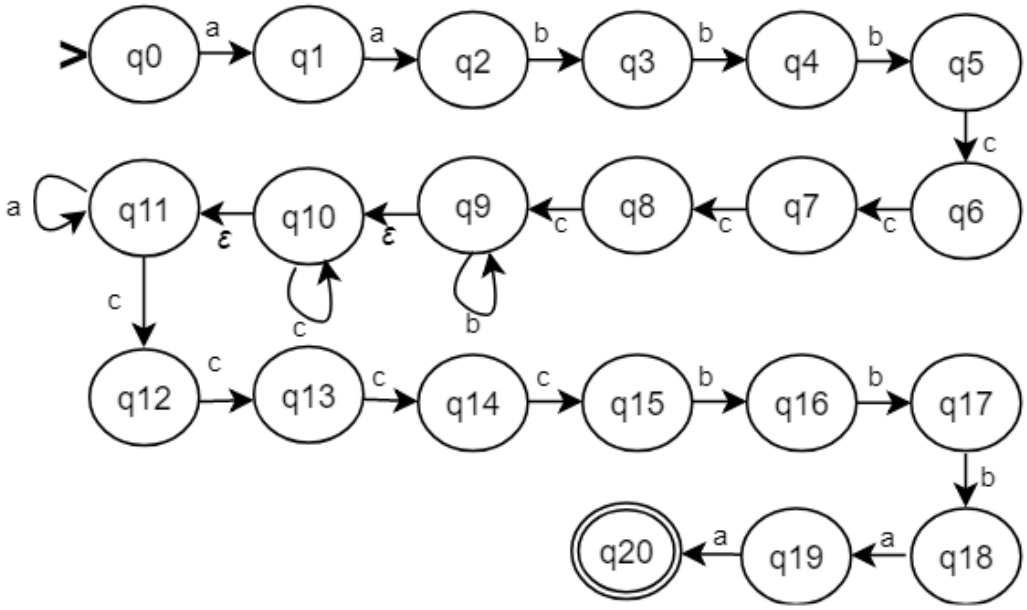
a = (((
b = ((((
c =)))))))

$ab^n c$ has to be part of L

$ab^2 c$ => ((((((((((( )))))))
11 != 7

The Language with alphabet {(,)} and balanced parentheses is not regular, since the number of (’s has to be equal to the number of)’s and ab^nc has to be part of L. In the above proof, we can see that the number of (’s is not equal to the number of)’s, further more in the above proof we can see that the result has not balanced parentheses.

3.3. L = { a2b3c4blcmanc4b3a2 | l,m,n >= 0 }



Since it is possible to make a finite automata, the Language:
L = { a2b3c4blcmanc4b3a2 | l,m,n >= 0 } is regular. Works for all L, since l,m,n only have to be >= 0 and not equal.

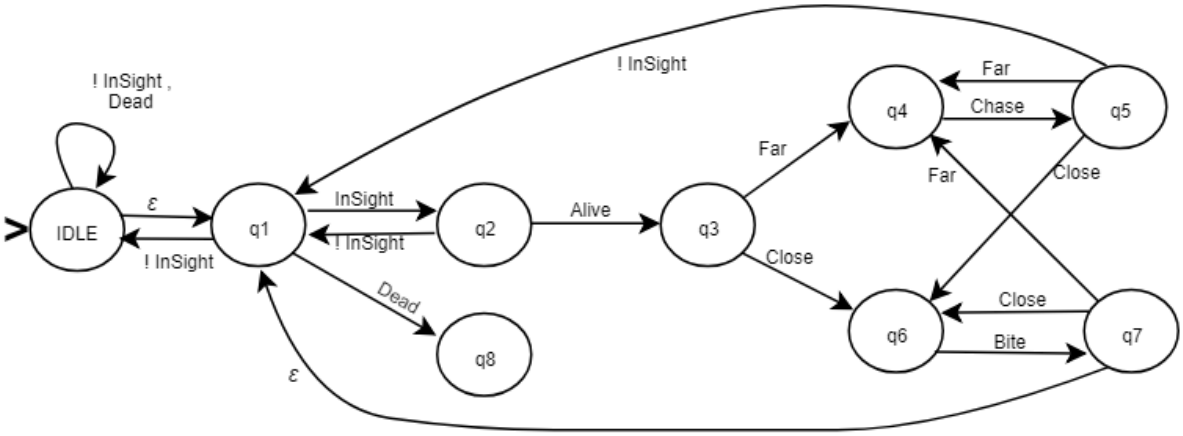
4. Game AI

4.1. Zombie

A zombie behaves in the following way:

- If the player is in sight and is alive but not close enough to bite it will chase the player.
  - ➔ If the player is Far, the zombie chases the player
  - ➔ If the zombie chases the player, he can be far and continue chasing || get Close to the player and bite him || loose the player out of sight, which brings the zombie to q1, where the player can be in sight or not in sight and continue
- If the player is in sight, alive and close enough for the zombie to bite, it will do so.
  - ➔ If the player is close and the zombie bites the player, he can stay close and continue biting || or the player can get Far where the Zombie would need to Chase the player etc. || the Player dies of all the biting || the Player get out of sight
- If the player is not in sight it will simply stay idle.

The process repeats until either the player or the zombie is dead.



4.2. Archer

The archer behaves a bit differently:

• If a player is in sight, alive and in range but not too close he/she will try to shoot the player.

➔ If the player is in range, the archer tries to shoot him || the player gets Far or Close || the archer shoots him and player get low HP and tries to escape || archer kills him with the shooting and the player dies.

• If a player is in sight, alive and not in range he/she will try to approach.

➔ If the player is Far, the archer tries to Approach him, so the player can stay Far and the Archer continues to Approach him || he can get Close or in Range || or the Player can disappear out of sight.

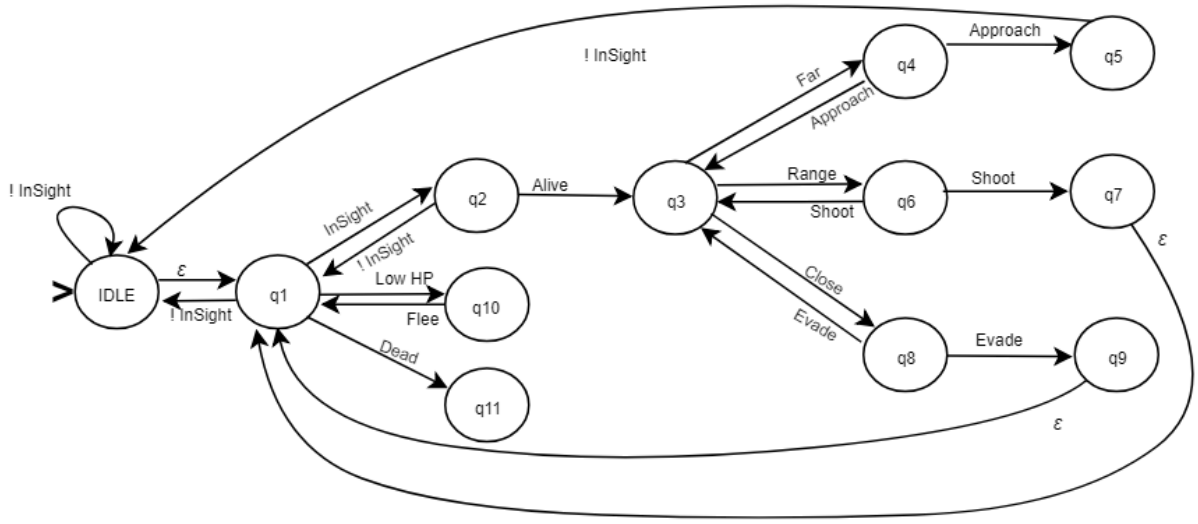
• If a player is too close he/she will try to evade

➔ If the player is close, the Archer tries to Evade, if he stays close he will always try to evade || he can get Far or in Range || he can get low HP and try to flee || he can die || get out of sight.

• If his/her hit points are low he/she will try to flee.

• If a player is not it sight he/she will stay idle as well.

This process again repeats until either the player or the archer is dead.



5. HTML Scrapping

5.3 Do you think it is possible to parse the whole HTML language with regular expressions?

When handling simple regular expressions, it is possible to run into conditions when we want to parse out HTML tags with a regular expression:
Something like <tag[^>]*>(.*?)</tag>

But there are certain problems with HTML and regular expressions. For example nested tags or tags that contains improperly escaped characters.
It most probably is possible to solve these problems with more and more regular expressions, which on the other hand would result in a rather complex regular expression.

Furthermore, regular expressions can match regular languages, but HTML is a context-free language and not a regular language.