JFLAP USER MANUAL AND EXERCISES

Written by student Tobias Fransson as a Manual for JFLAP simulator use in the course: Formal Languages, Automata and Theory of Computation (FABER)

Content

Introduction to JFLAP	2
1. Regular Languages and Finite State Automata	3
2. Context Free Languages and Pushdown Automata	14
3. Restriction Free Languages and Turing Machines	17
JFLAP exercises	18
1: Regular Languages and Finite State Automata	19
2: Context Free Languages and Push Down Automata	25
3: Restriction Free Languages and Turing Machines	27
Solutions to JFLAP exercises	29
1: Regular Languages and Finite State Automata	29
2: Context Free Languages and Push Down Automata	34
3: Restriction Free Languages and Turing Machines	39

JFLAP User Manual

For JFLAP version 7.0

Introduction

What is JFLAP?

JFLAP program makes it possible to create and simulate automata. Learning about automata with pen and paper can be difficult, time consuming and error-prone. With JFLAP we can create automata of different types and it is easy to change them as we want. JFLAP supports creation of DFA and NFA, Regular Expressions, PDA, Turing Machines, Grammars and more.

Setup

JFLAP is available from the homepage: (www.JFLAP.org). From there press "Get FLAP" and follow the instructions. You will notice that JFLAP have a .JAR extension. This means that you need Java to run JFLAP. With Java correctly installed you can simply select the program to run it. You can also use a command console run it from the files current directory with, *Java –jar JFLAP.jar.*

Using JFLAP

When you first start JFLAP you will see a small menu with a selection of eleven different automata and rule sets. Choosing one of them will open the editor where you create chosen type of automata. Usually you can create automata containing states and transitions but there is also creation of Grammar and Regular Expression which is made with a text editor.

Additional Resources

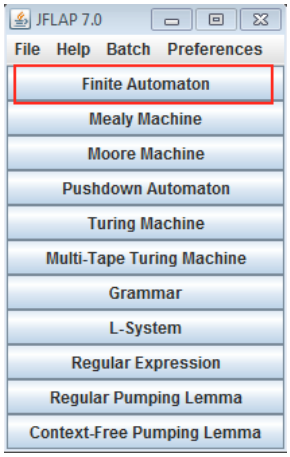
Recommended Reading: JFLAP - An Interactive Formal Languages and Automata Package
Rodger, Finley, ISBN: 0763738344

JFLAP assignments for JFLAP - An Interactive Formal Languages and Automata Package
http://www.cs.duke.edu/csed/jflap/jflapbook/files/

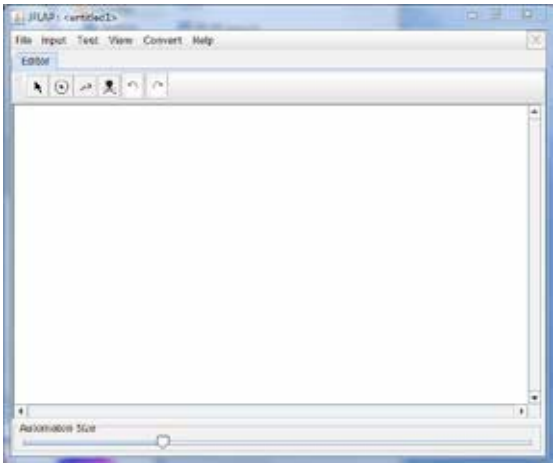
Getting Started With JFLAP, Colorado State University
http://www.cs.colostate.edu/~massey/Teaching/cs301/RestrictedAccess/JFLAP/
gettingstarted.html

1. FINITE AUTOMATA AND REGULAR LANGUAGES

The simplest type to begin with is Finite Automata, which is the first option from the selection menu. In JFLAP both DFA and NFA are created using Finite Automata.



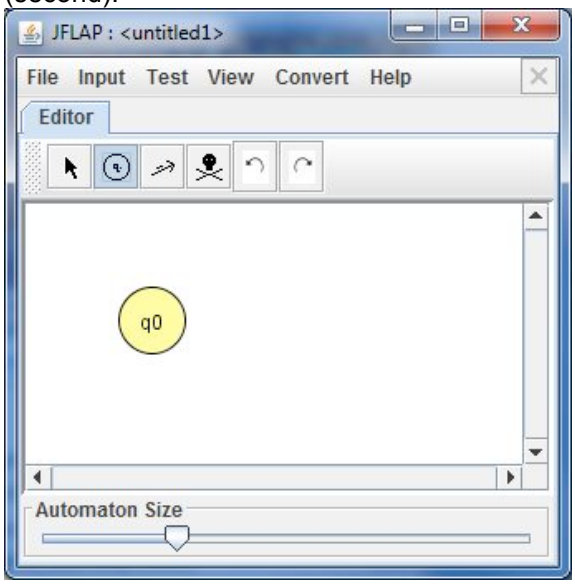
Now you should have an empty window in front of you. You will have a couple of tools and features at your disposal.

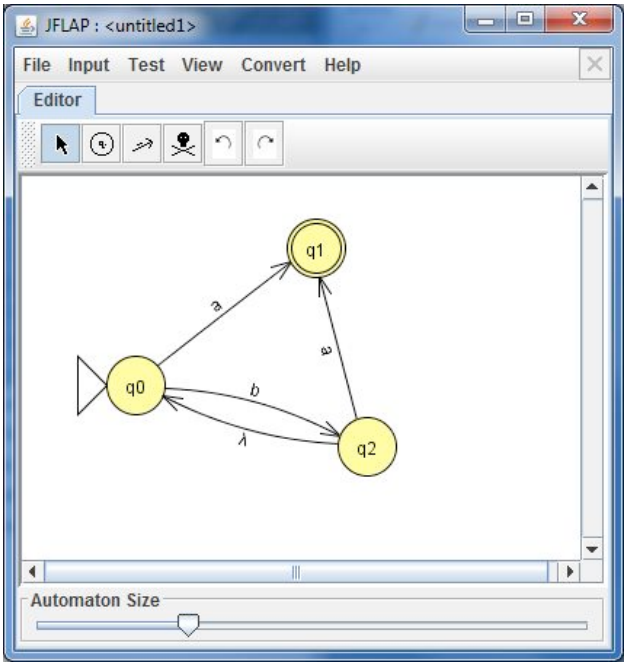


The toolbar contains six tools, which are used to edit automata.

Attribute Editor Tool, changes properties and position of existing states and transitions.
State Creator Tool, creates new states.
Transition Creator Tool, creates transitions.
Deletion Tool, deletes states and transitions.
Undo/Redo, changes the selected object prior to their history.

Creating an automaton is easy with the state and transition tools. Note that you need to change back to the Attribute Editor Tool (first) to change states. Let's try to add states with the State Creator Tool (second).

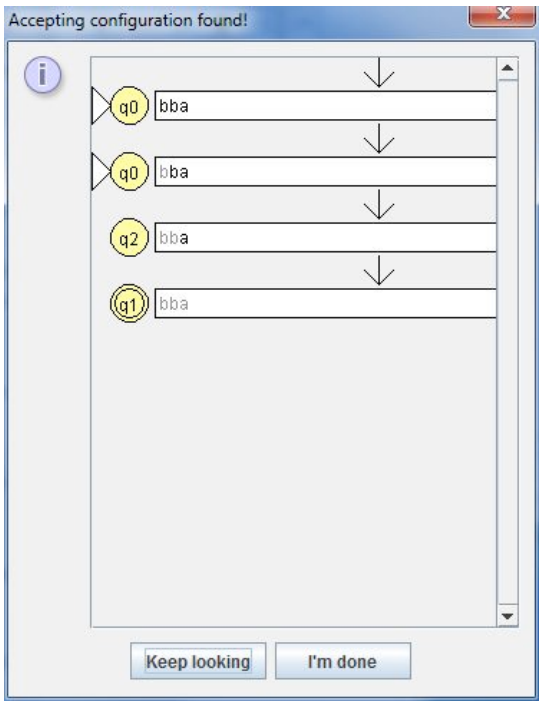




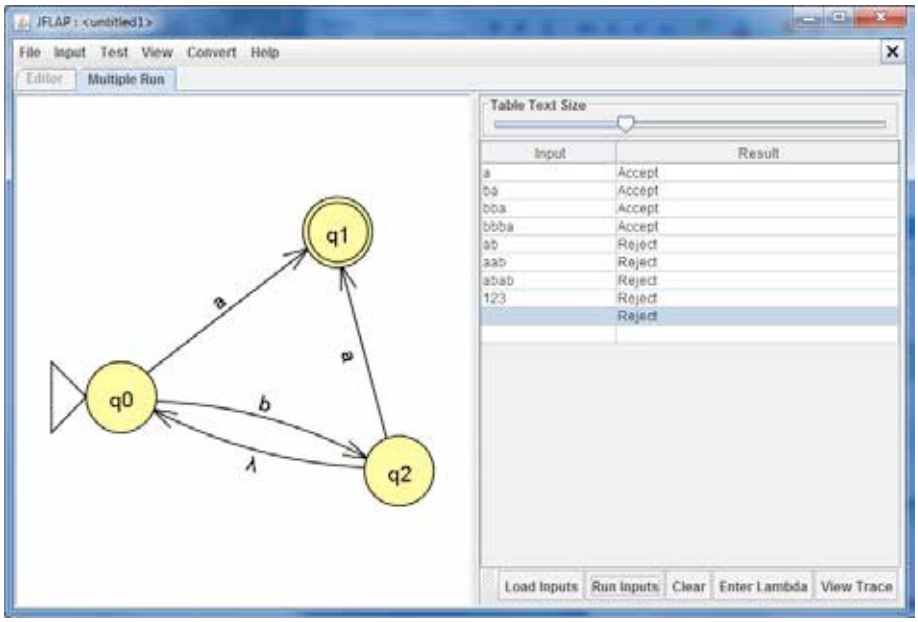
When adding states they automatically get a name assigned to them, which can be changed using the Attribute Editor Tool. Transitions are easily dragged between states with the Transition Creator Tool.

The automaton above only accepts strings containing b which end with a. To test this automaton you can use any of the available tools under the Input menu.

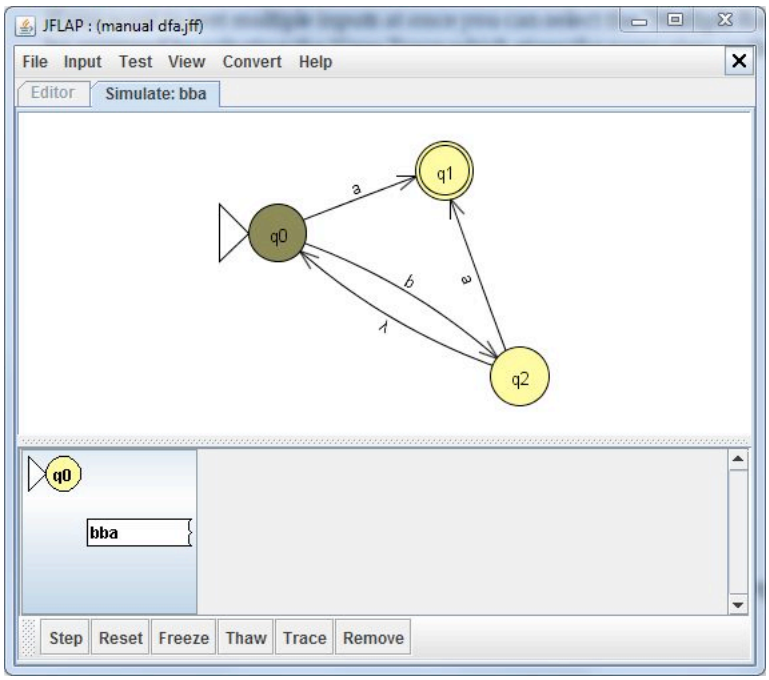
The fastest in this case is to use Input → Fast Run. A menu where you can set your input string pops up. Type the string “bba” and select “OK”. The program shows all the transitions that are done when consuming the input string.



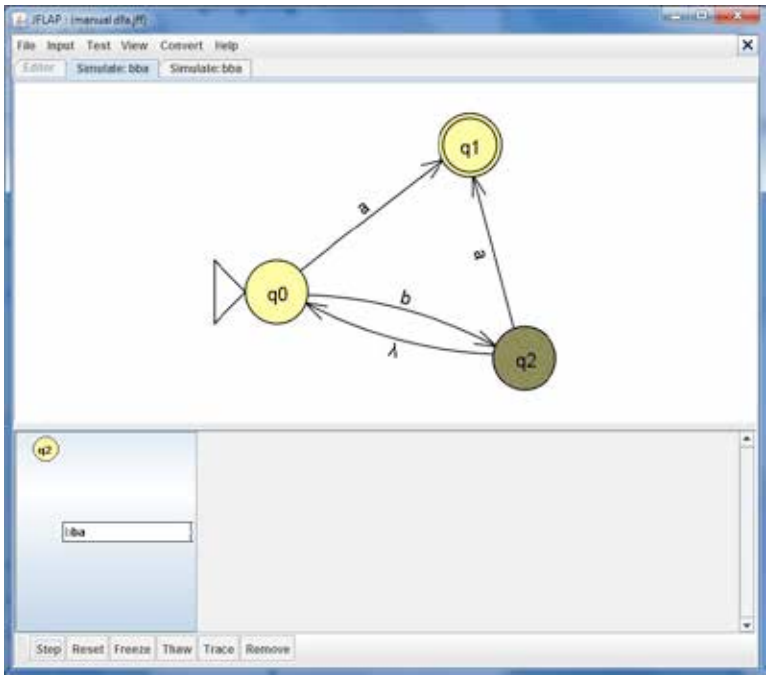
If you want to test multiple inputs at once you can select the “Multiple Run” option. If you wish to individually review single runs, it can be accessed by selecting the View “Trace”, which gives a view similar to the “Fast Run” option.



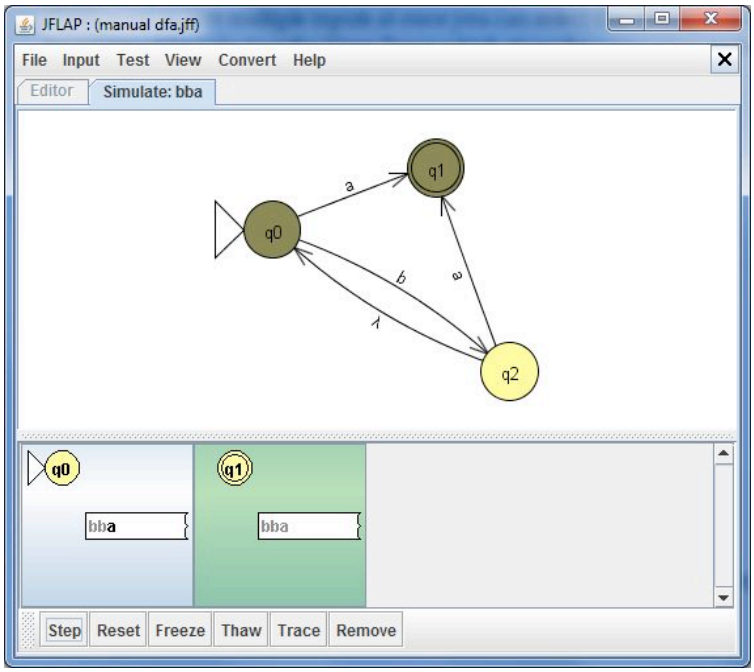
To debug the automata “Step by Closure” or “Step by State” can be selected.
For each step the automaton highlights the currently active state.
For “stepping” you press the “Step” button.



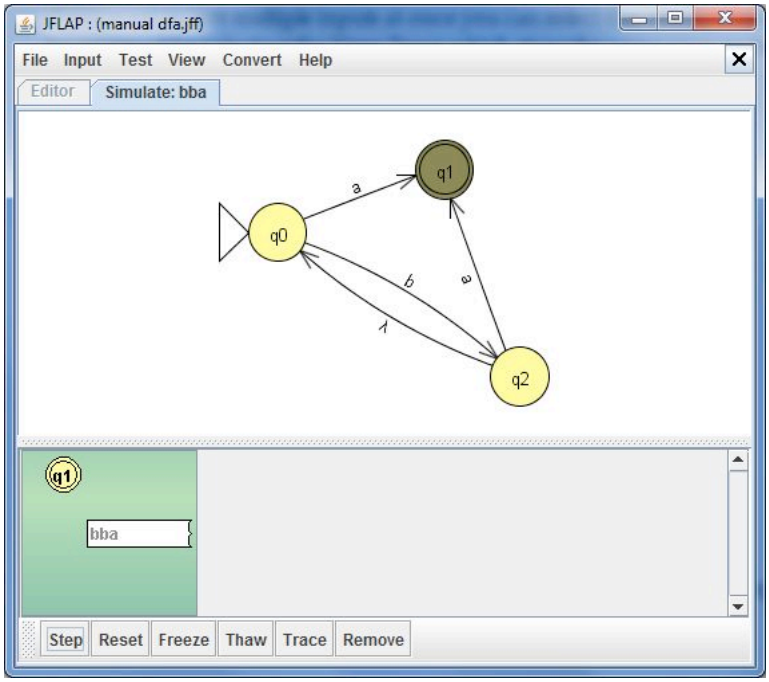
The first character consumed. Notice that the first letter is grayed out and the currently active state have changed.



Now the last character is about to be consumed. This step shows the transition between q0 and q1. If there happen to be multiple paths with the same character you will see them grayed out.



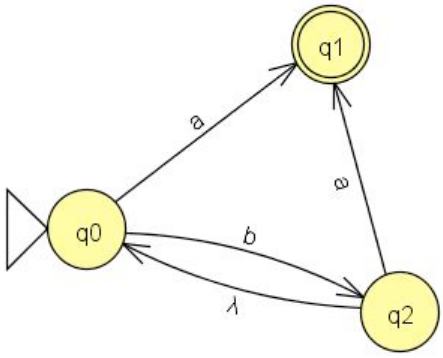
The simulator reached the final state.



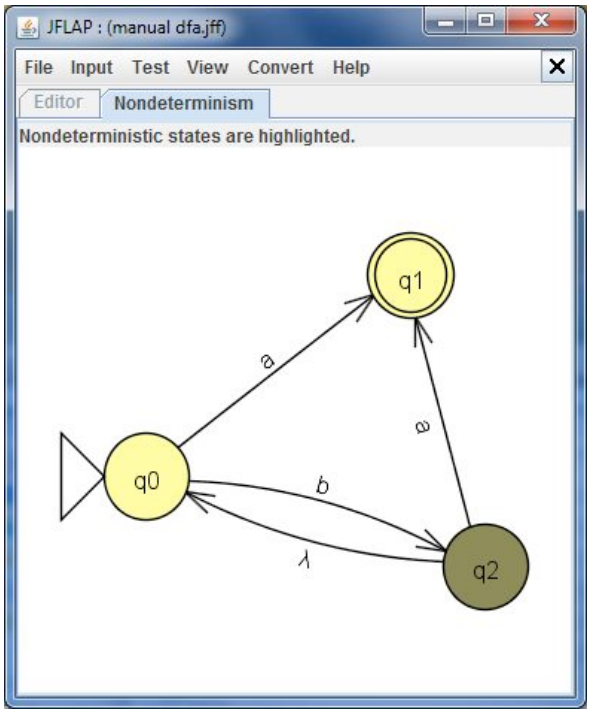
To restart the test you can select the reset button to start from the beginning or press the X button in the top right corner to go back to the editor.

NFA – Non-deterministic Finite Automaton

Finite Automaton is either a DFA or an NFA. Unlike DFA, NFA can have lambda transitions (transitions on an empty string) or several transitions from the same state with the same symbol (that is non-determinism as we do not know which of the possible paths will be the next). The automaton below is an NFA because of the lambda transition from the q2 state.
Creating an NFA or a DFA proceeds in the same way in JFLAP.



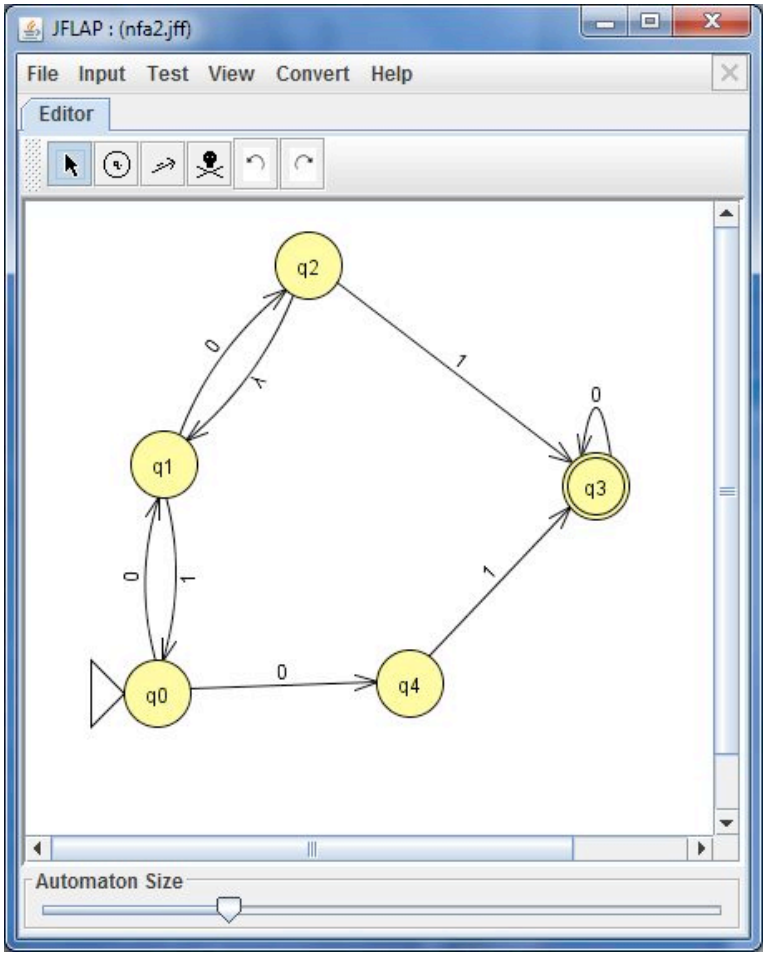
It is possible to let JFLAP determine if the automaton is a non-deterministic automaton. Select Test then “Highlight Non-Determinism”.



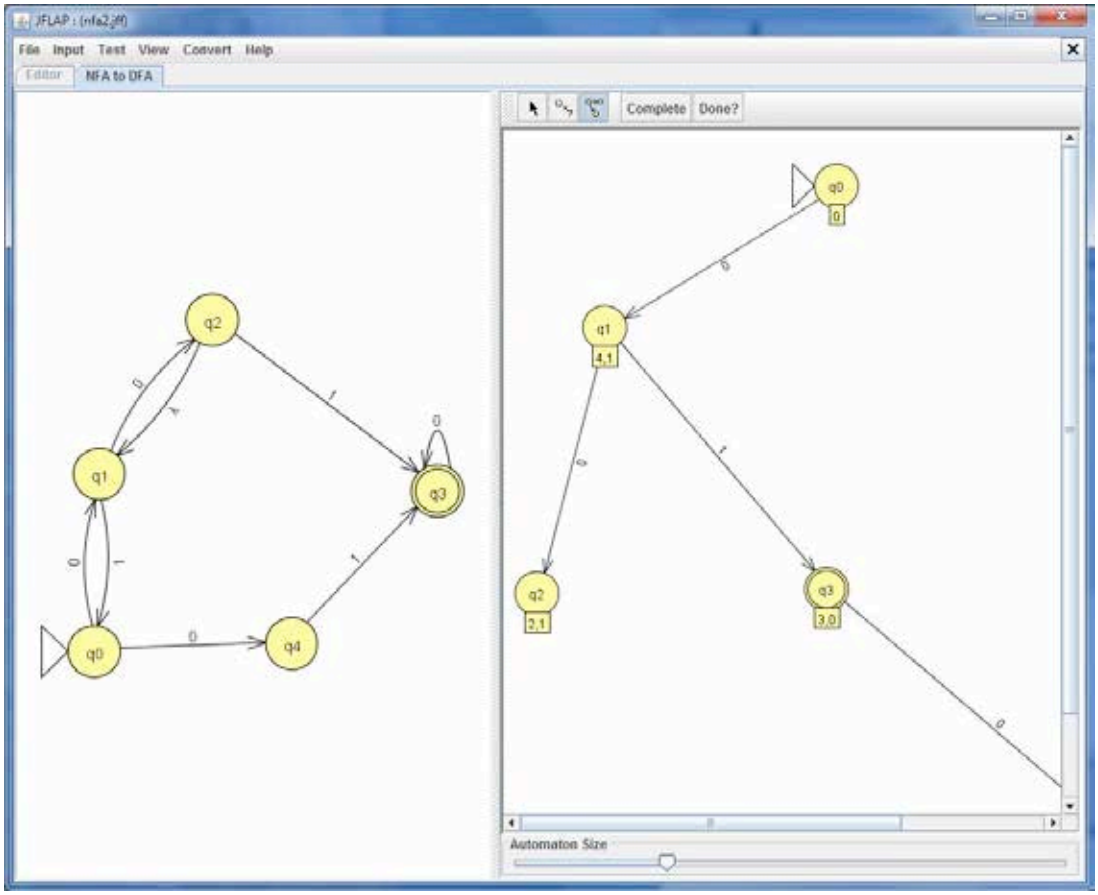
This simply shows that the q2 state is non-deterministic (because of lambda transition).

Converting NFA to DFA

As we have learned in the course, for each NFA there is a corresponding DFA. You can use JFLAP ton convert NFA to DFA. The following automaton can easily be changed to a DFA.

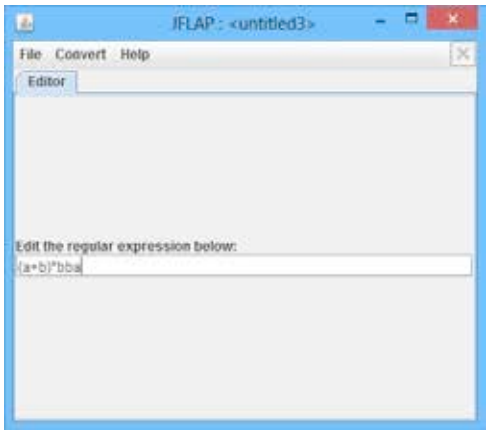


Create a NFA and then choose Convert → Convert to DFA. This will open the conversion view where you either let JFLAP do the work or try yourself to convert it. The left view is the original automaton and the right one is the new DFA. Use the state expander tool to expand the states until the DFA is complete. Using the *Complete* button will automatically create the whole DFA for you. The *Done?* button will tell if the DFA is done or not. Once the DFA is complete it will be exported to a new JFLAP window with your converted DFA.



REGULAR EXPRESSIONS

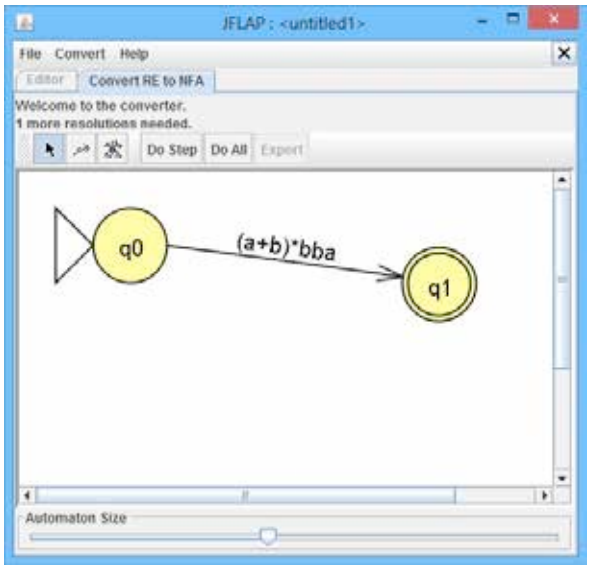
Regular Expressions can be typed into JFLAP to be converted to an NFA.



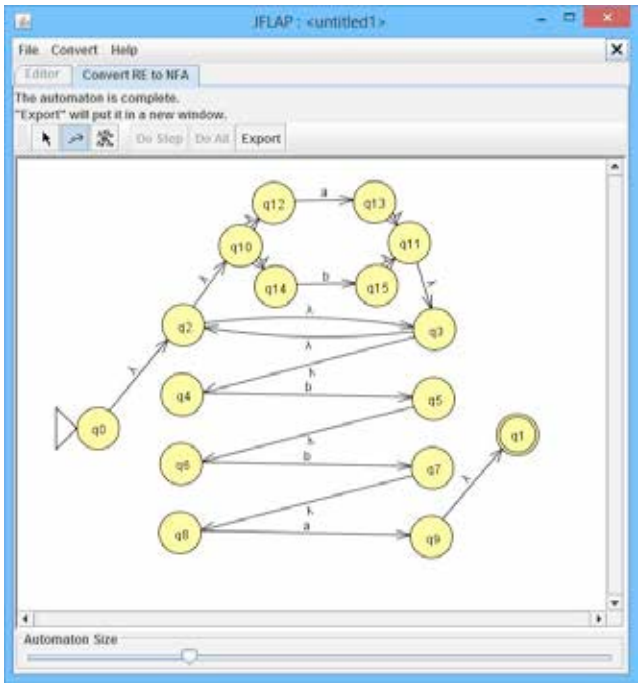
Choose Regular Expression in the main menu, then just type the expression in the textbox.
Definitions for Regular Expressions in JFLAP:

- * Kleene Star
- + Union
- ! Empty String

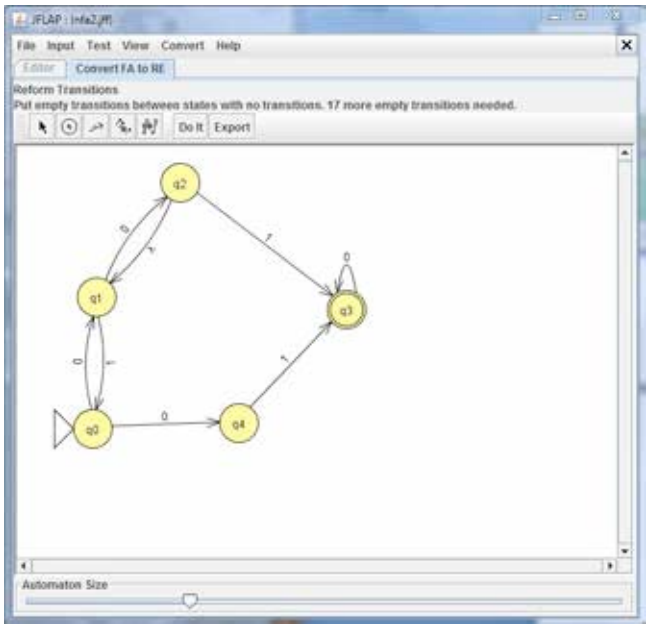
Correctly written expressions can then be converted to an NFA. To convert your expression select Convert → Convert to NFA. The conversion will begin with two states and a transition with your Regular Expression. With the *(D)e-expressionify Transition* tool you can break down the Regular Expression into smaller parts. Each transition will contain a sub expression. The next step is to link every rule with lambda transitions. Add new transition between states that should be connected with the Transition Tool. If you are unsure what to do you can select *Do Step* to automatically make the next step. If you want the NFA immediately *Do All* creates the whole NFA for you.



You can notice how the conversion differs depending on how the Regular Expression looks. For example the expression *a+b* results in a fork, were either ‘a’ or ‘b’ can be chosen.

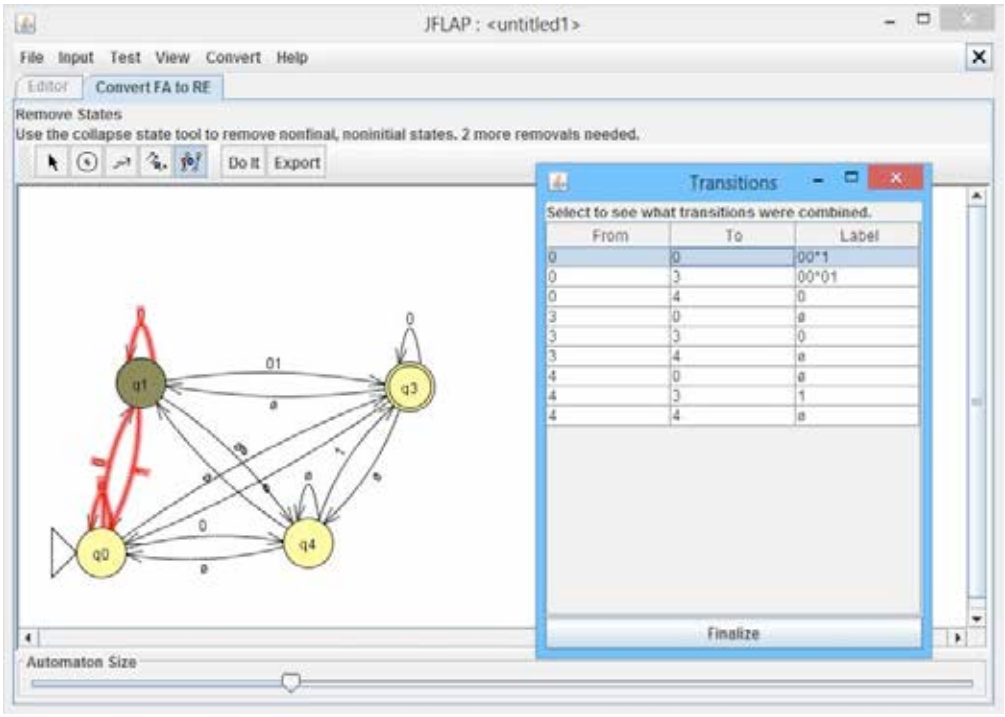


Converting FA to Regular Expression

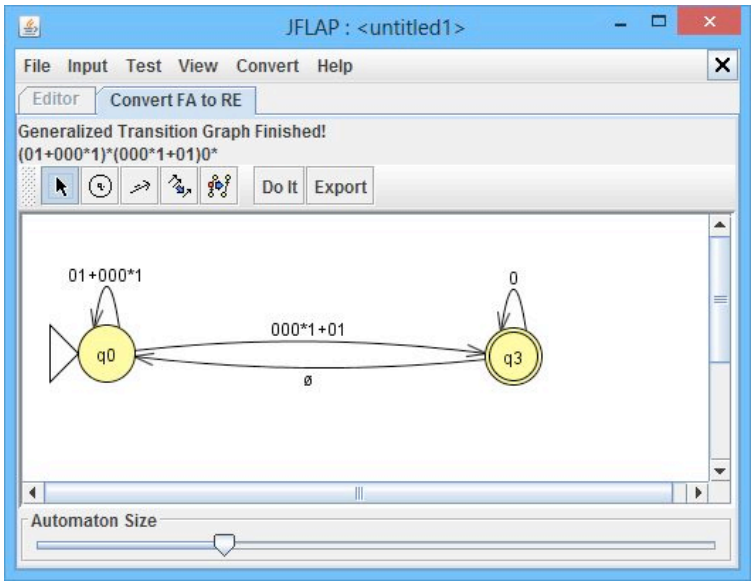


Follow the instructions above the toolbar. To make the conversion work, empty transitions must be added between states that have yet no transition. States that either is Initial or Final must be removed which you do with the collapse state tool. With the collapse tool you can use the table to inspect

combined transitions from that state. The state is removed with The Finalize button.



When all the necessary steps are made, the converted automaton contains the regular expression. You can also see the complete regular expression above the toolbar that can be exported using *Export*.

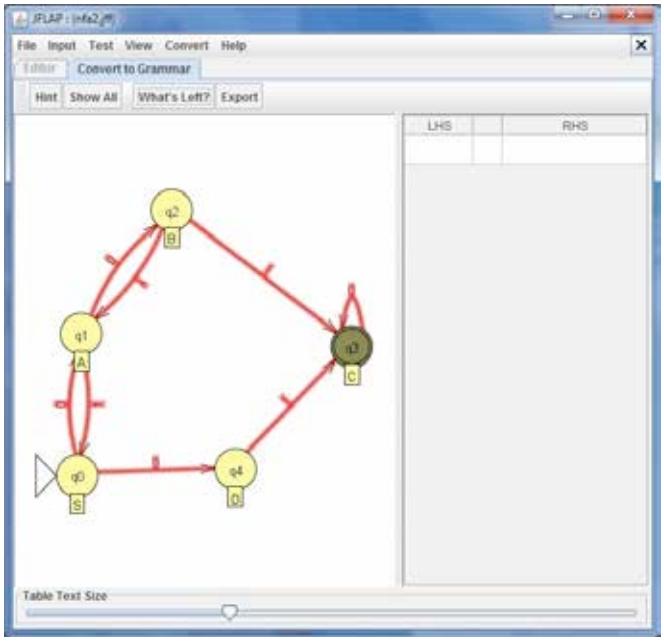


JFLAP is capable to convert the regular expression to an NFA again. If the original automaton is a DFA the result might differ because JFLAP add a lot of lambda transitions. You might need to convert further to a minimized DFA to get your automata back.

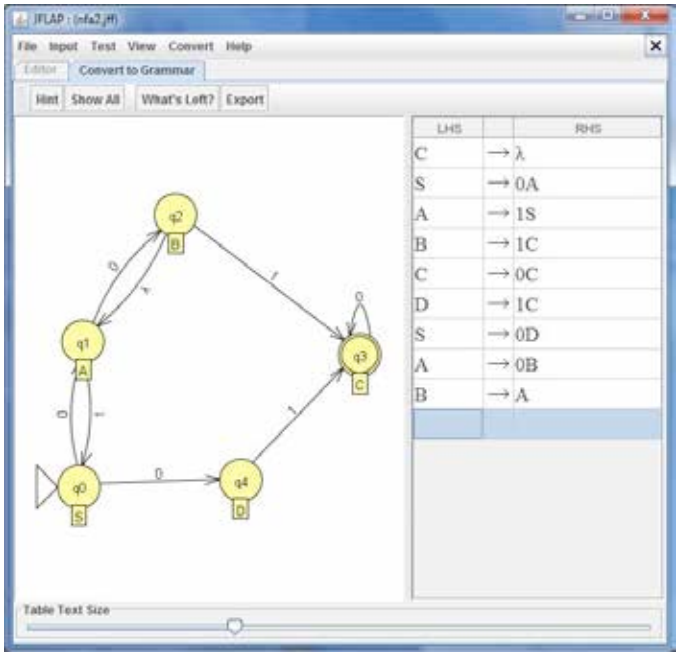
Converting FA to a Grammar

When using a Finite Automaton select Convert → Convert to Grammar. The conversion view will contain your automata on the left and the grammar on the right. You are free to edit the grammar yourself or let JFLAP more or less do the work.

The "*What's Left?"* option will show which transition that not have been used in the grammar yet. JFLAP automatically puts labels to states to tell which symbols they represent in the grammar.



As mentioned you can either edit the right side grammar table or click on states to automatically reveal the grammar for each step. The *Hint* reveals which state you should select next. *Show All* automatically creates the grammar for you.

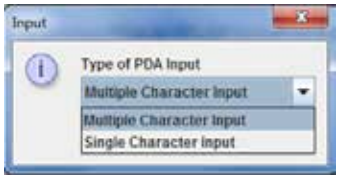


Once the grammar is complete you can select *Export* to open a new JFLAP window with your new Grammar, don't forgot to save if you want your grammar saved.

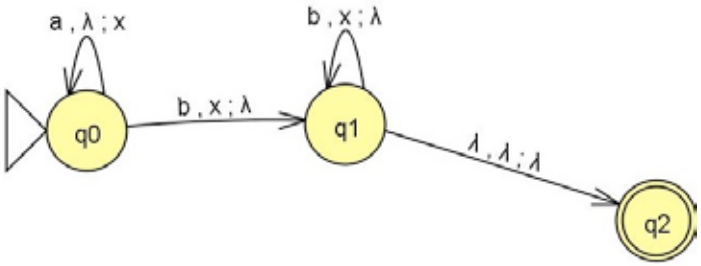
2. CONTEXT FREE GRAMMARS AND PUSHDOWN AUTOMATA

Pushdown Automata

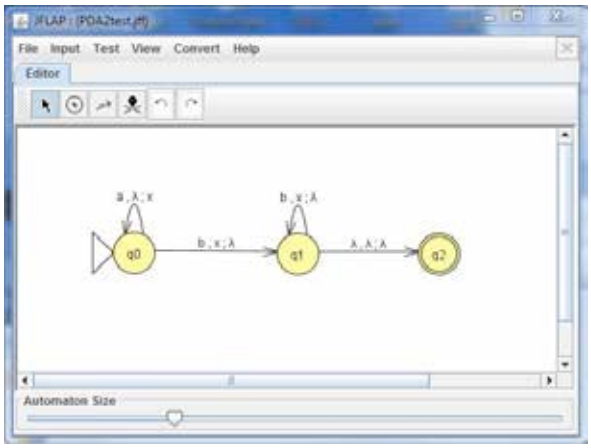
Creating PDA in JFLAP is just as easy as creating FA but there are some differences. First you select "Pushdown Automaton" in the selection menu which also shows up when selecting "new" in File.



First the program asks if you want to use either Multiple or Single Character Input which means how many characters may be consumed at each step.

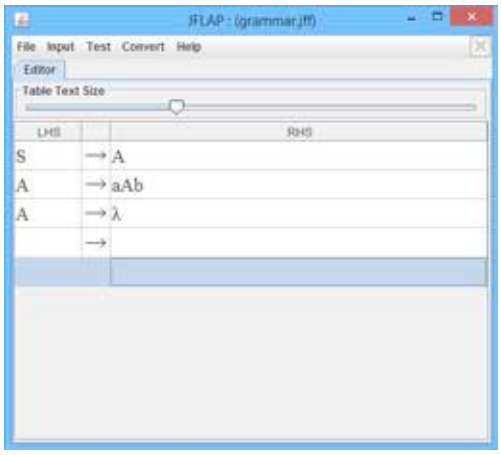


The editor looks exactly like editing Finite Automata. This PDA accepts the language L = $\{a^n b^n\}$ where n > 0. The PDA is tested exactly like FA's. Transitions are different where they now consist of three variables. First is the input, second pops from the stack and the third variable is push.
Try to debug the PDA with the step by state feature. You will notice that you have a stack that is added and removed from during each step.



Context-free Grammar

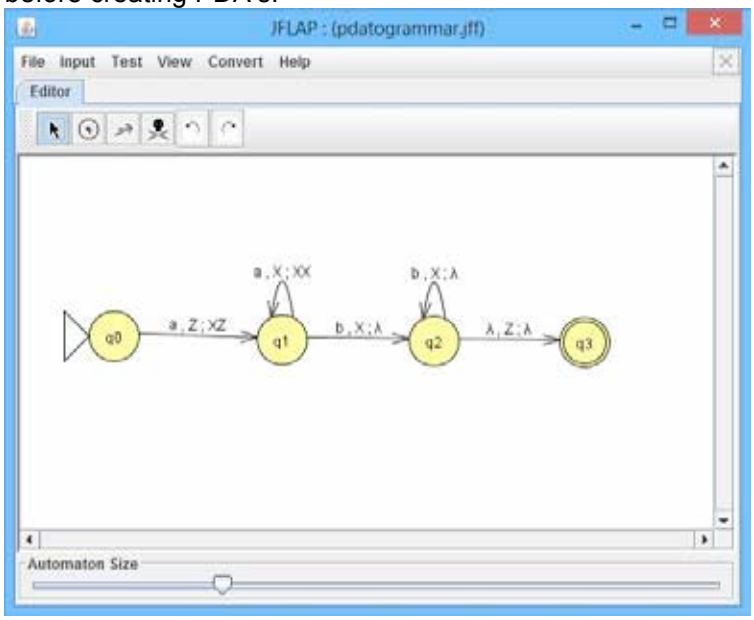
Grammar is created using a table in JFLAP. If you choose grammar as a new project in JFLAP you will have a table where you can edit the grammar. Here we have a simple grammar with the language $a^n b^n$ where $n \geq 0$.



Create and edit a grammar is simple. Create symbols by adding them to the LHS column and rules under RHS then press enter. The arrow between the symbols and rules is automatically created. Lambda is done by leaving the rule empty then press enter and you will see the lambda symbol there. Grammar can be tested by using any of the features under Input. One is Brute Force Parse which can test single strings each time.

Convert Pushdown Automata to a Context-free Grammar

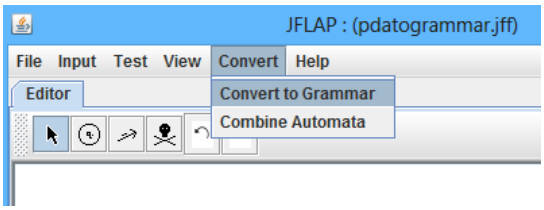
Conversion with PDA is very similar to converting FA, but there are differences that are good to know before creating PDA's.



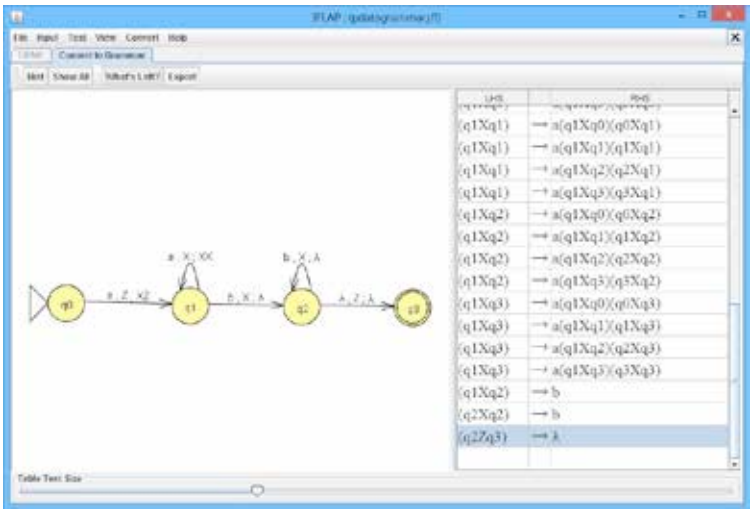
To convert PDA to CFG (Context-Free Grammar) in JFLAP, a couple of conditions must be met:

- For each transition, pop 1 symbol and push either 0 or 2 symbols.
- There must be only one final state with transitions that pop Z off the stack.

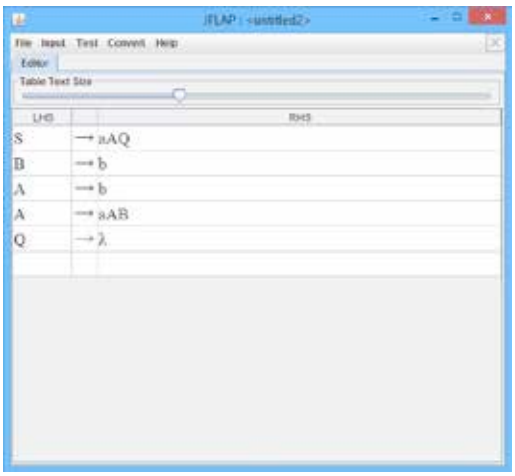
Trying to convert a PDA that does not follow this will result in an error message that shows which transitions should be corrected. To start converting, select Convert → Convert To Grammar.



This is the conversion screen, on the left is the PDA and on the right is the grammar. You can either fill the grammar yourself or select each transition to fill out the grammar. Note that the grammar will be filled with a lot of useless rules, which happens when using brute force.



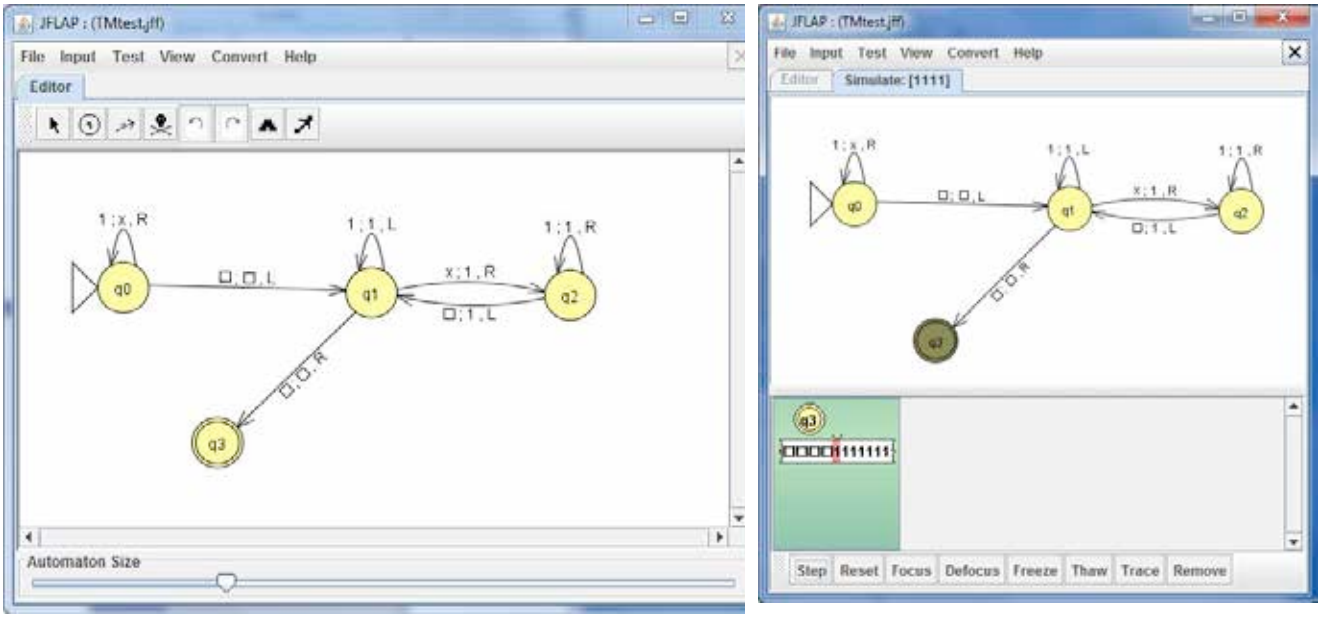
When all rules are set from every transition, you can choose to select Export to transfer the Grammar to a new JFLAP window. JFLAP can trim down the useless rules into a readable grammar.



3. RESTRICTION FREE LANGUAGES AND TURING MACHINES

Turing Machine

A Turing machine uses a tape that can be written to and removed from by the head. For every transition there are three variables to set. First one is what value is expected to be under the head. Second is if the first is correct what should be written under the head. Third and last is the direction the head should take, the head can go left (L), right (R) or stay (S). In JFLAP □ is the blank symbol. Create this Turing machine below, which duplicates the number of symbol 1 on the tape.



When testing your Turing machine you can see the tape. Stepping trough will update the currently active state and the tape content. In this example the tape starts with the string “1111”, where the TM steps trough and changes each 1 to an x. To know how many extra 1 to write, x will be used as a symbol for each extra 1 to be written. The TM stops once the computation is done and the tape should contain the result with the head in its initial position.

JFLAP EXERCISES

A few words of introduction about JFLAP.

JFLAP is an automata simulator that supports DFA/NFA, PDA, Turing machines and more.
JFLAP also supports Regular Expressions and Grammar which can be converted to an automaton and back. JFLAP has been used worldwide on many automata theory courses and is still under development.

Links to manuals and resources.

If you have problems use the provided manual *JFLAP User Manual.*
JFLAP's home page also contains a very thorough tutorial of everything the program can do.
JFLAP Home Webpage: www.jflap.org
Recommended reading is: *JFLAP - An Interactive Formal*
Languages and Automata Package by Susan H. Rodger and Thomas W. Finley
ISBN: 9780763738341
Here are as well files for this book: http://www.cs.duke.edu/csed/jflap/jflapbook/files/

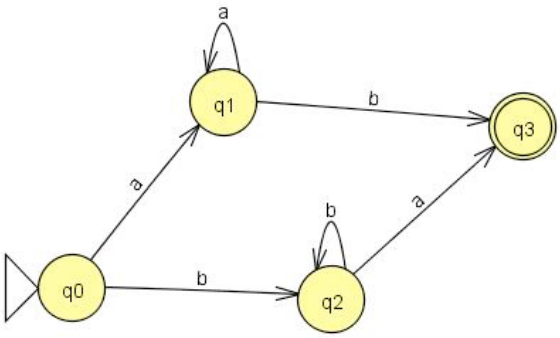
Lab Instructions

When you are done with each task, save your work in JFLAP as .jff files and call them ass1_1.jff after each assignment. Also a Word document should be provided if there are any questions asked. Save everything in a compressed RAR file with the name laborationX_yourname.rar.

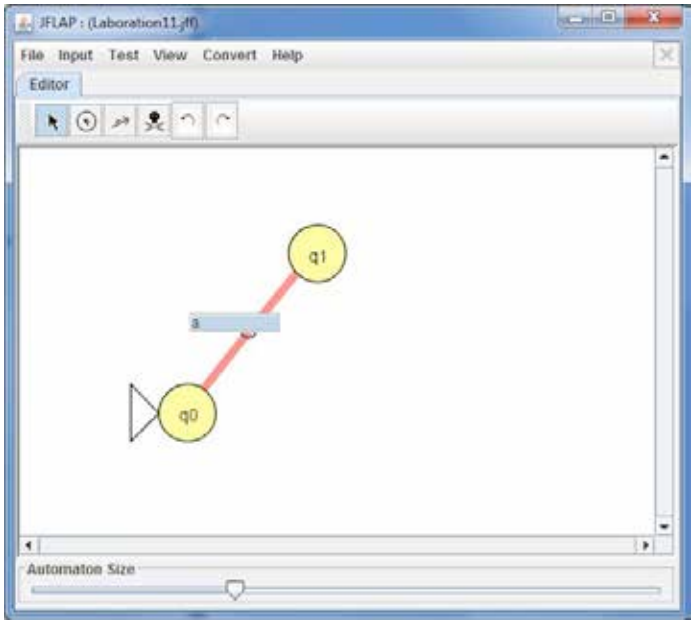
1. REGULAR LANGUAGES AND FINITE STATE AUTOMATA

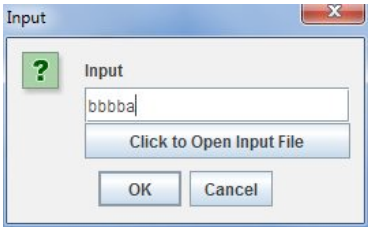
Assignment 1.1

Create this Deterministic Finite Automaton. First start JFLAP and select “Finite Automaton” as a new automaton. Use the toolbar to drag and drop states and transitions.

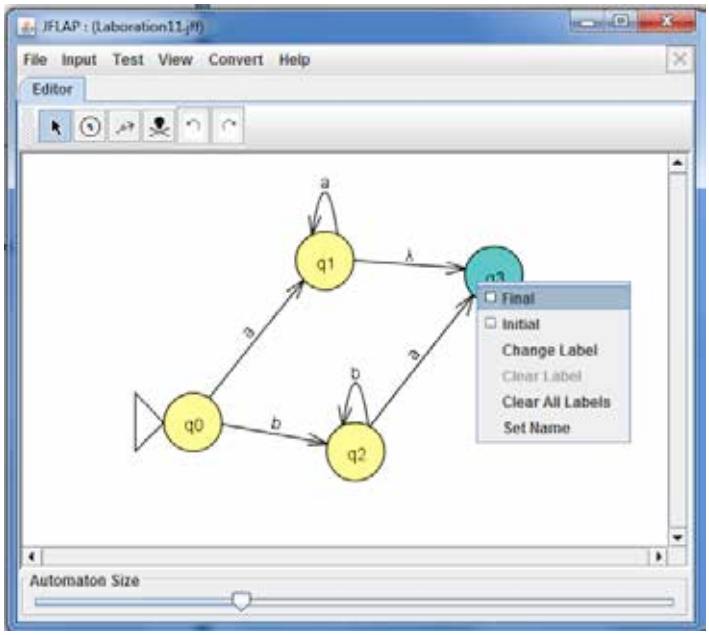


Select the transitions with the selection tool. Then type the input character for each transition.

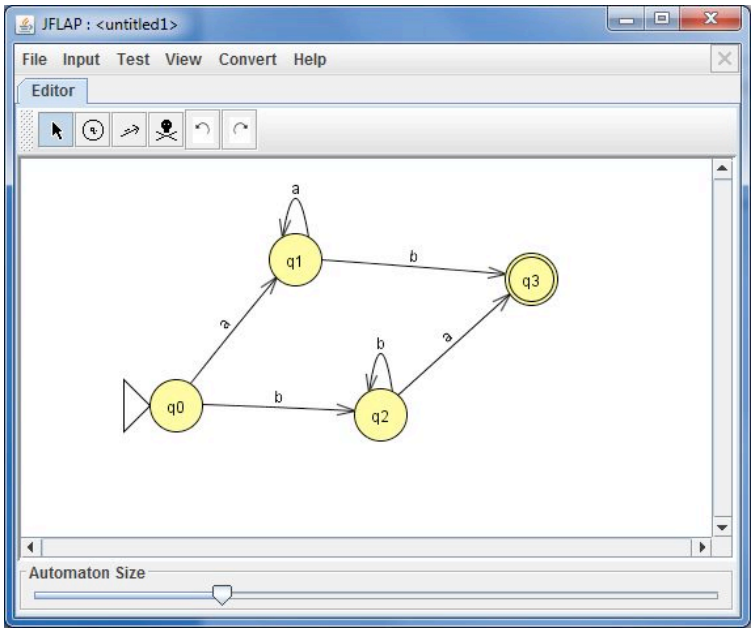




Remember that initial and final states must be set to make the automaton run. State q0 is set as the initial state which means the starting state. State q3 will be the final state which is a state that should be active once the string is accepted and consumed. There can be more than one final state.



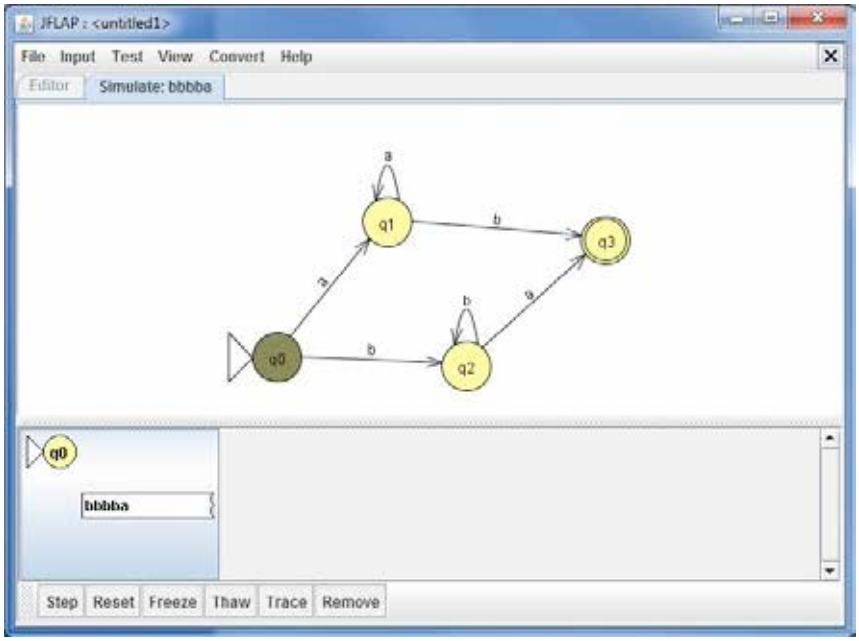
When you are done, the automaton should look like this.



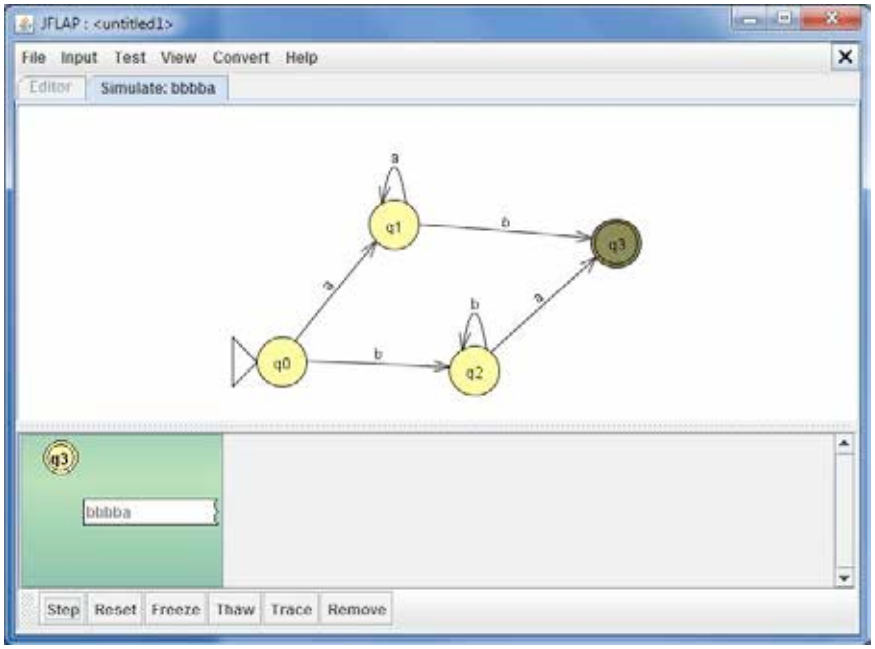
To run this DFA select *Input* → *Step By State*, then choose an input string to run the automata with.

You will see JFLAP's simulation view where you can debug the automaton while it consumes the input

string. Use the Step button to step trough the string. Here the automaton have moved the active state and have consumed all the b characters.

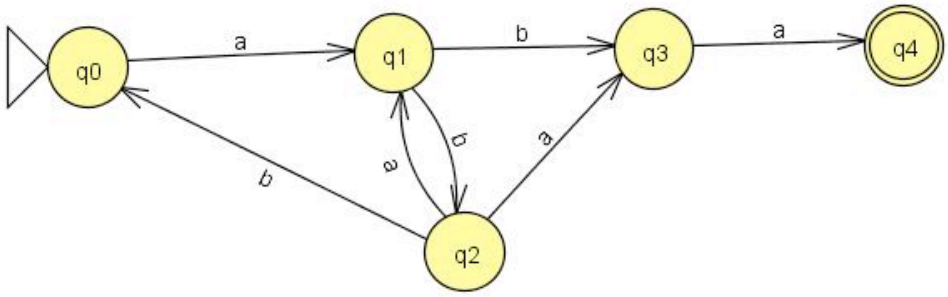


Once the whole string is consumed the automata should park at the final state and the stack turns green. The string is accepted.



Try to use Input → Multiple Run instead. Which strings are accepted and which ones are rejected?

Assignment 1.2
Create the following NFA:



a) Write down 4 accepted strings and 4 strings that are not accepted.

b) Which of the following regular languages over the alphabet Σ = {a,b} are accepted by the automaton?

1) L = {aba},

2) L = {aba*a},

3) L = {a(ba)*ba},

4) L = {a(bba)*baa}

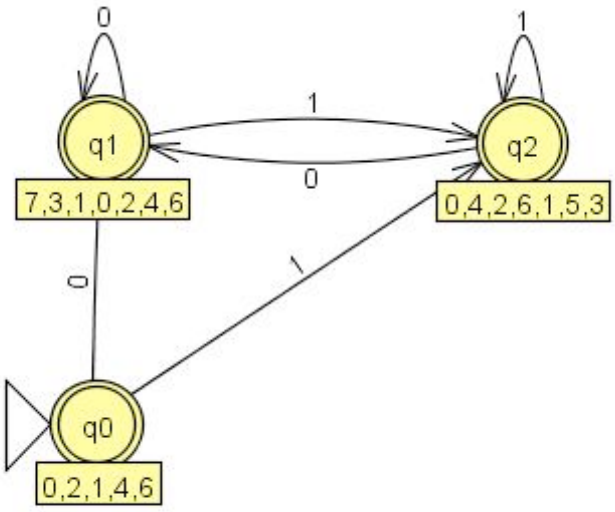
c) Construct a regular expression using JFLAP. Use Convert→Convert FA to RE.

d) Construct a Grammar using JFLAP, use Convert→Convert To Grammar.

e) Try to convert the automaton to a DFA. Once you are done JFLAP should export the result to a new JFLAP window. Save the resulting DFA as a new JFLAP file.

Assignment 1.3

JFLAP supports creation of FA from a regular expression. First you need to select “Regular Expression” in the selection menu. Just type the expression in the box, and then select ”Convert to FA” where you can step trough the conversion until the automaton can be exported to a new window. If the automaton is correct it should be possible to convert it back to a regular expression again. Use these regular expressions to be converted to a Finite Automaton.
Note that JFLAP creates a lot of lambda transitions in this part. The result can be improved by converting the resulting NFA to a DFA and then minimize the automaton.



Example, DFA created from the Regular Expression (1+0)*

Convert the following regular expressions to FA:

a) a(a+b)*b

b) 1((1+0)1)*

c) a(a+b)*bb(a+b)

d) ab(a+ba)*b*

Assignment 1.4

Create Regular Grammar and tell which language the grammar generates.
Then convert Regular Grammar to a Finite Automaton, try to do the FA yourself!
Then save the result.

a) $S \rightarrow aB$
$B \rightarrow bB$
$B \rightarrow \lambda$

b) $S \rightarrow bA$
$A \rightarrow abB|baS$
$B \rightarrow bba$

c) $S \rightarrow 10A$
$A \rightarrow 01B$
$A \rightarrow 11B$
$B \rightarrow 1S|10A$
$B \rightarrow 11$

Each of these Finite Automata can easily be converted back to a Grammar which you are free to try for yourself.

Assignment 1.5

a) Use this Regular Language to make a Nondeterministic Finite Automata.

   Σ = {a,b,c} L = {aawbbwcc | w = {a,b,c}* | |w| is even }

   After the NFA is made, find the Grammar for this NFA!

b) Now create a new FA(NFA or DFA) with this language:

   Σ = {a,b} L = { ab(a(ba)*)* }

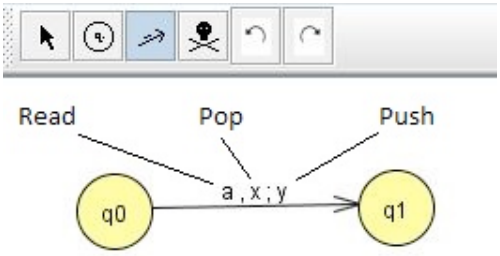
   What is the minimal DFA for this language?

Save both the automaton and grammar. Languages should be saved as a Word document.

2. CONTEXT FREE LANGUAGES AND PUSHDOWN AUTOMATA

Assignment 2.1

Pushdown Automaton (PDA) is an automaton that also handles a stack. Unlike regular Finite Automata PDA's have a Pop and a Push symbol which handles the stack during execution.

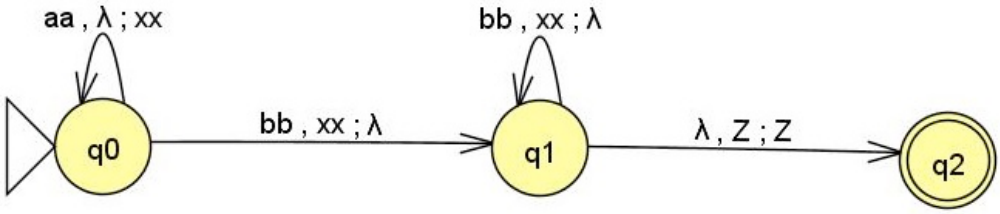


Create a PDA that accepts strings that contains the language L = $\{a^x cb^{2x} \mid \text{where } x \geq 0\}$ using the alphabet $\sum = \{a,b,c\}$.

Assignment 2.2

Create each PDA with at least five test results with the following languages over alphabet: $\Sigma = \{a,b\}$

a) Create the PDA from the Figure.



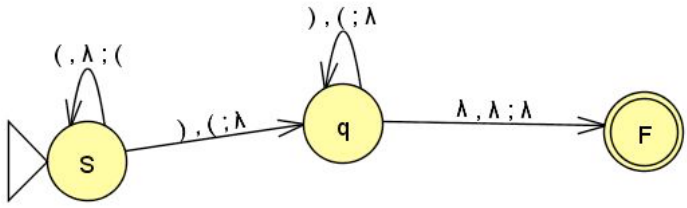
b) L = $\{a^n b^n \mid \text{where } n > 0\}$

c) L = $\{a^n b^n c^n \mid \text{where } n > 0\}$

d) L = $\{W_1 c W_2 \mid W_1, W_2 \varepsilon (a,b)^* \text{ where } W_1 \neq W_2\}$

Assignment 2.3

Consider the following Pushdown Automaton.



a) What language does this PDA accept?

b) Extend this automata to accept simple expressions like "((1+0) +1)". That is expressions with 1 and 0 and the + operator.

Assignment 2.4

Construct PDA that accepts the following grammars.

a) $S \rightarrow aBB$
$A \rightarrow cc$
$B \rightarrow bB \mid A$
$B \rightarrow bb$
$B \rightarrow \lambda$

b) $S \rightarrow aABC$
$A \rightarrow aa \mid aaA$
$B \rightarrow bbB \mid bc$
$C \rightarrow ccC \mid c$

c) $S \rightarrow AB$
$A \rightarrow aa \mid \lambda$
$B \rightarrow bbB \mid bb$

Assignment 2.5

Consider the following language L = { $a^x b^{x2} a$ | where x > 0} using the alphabet Σ = { a,b }.

a) Create the Grammar for this language.

b) What type of grammar do you get? You can use JFLAP to determine which type you have via the Test menu.

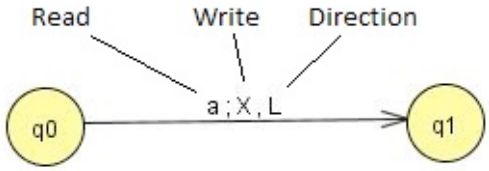
c) Create a PDA from this Grammar.

Save both the Grammar and the PDA in separate JFLAP files.

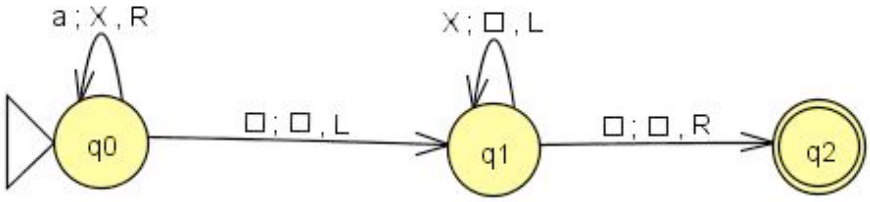
3. RESTRICTION FREE LANGUAGES AND TURING MACHINES

Assignment 3.1

Turing machines are more powerful than PDA and JFLAP gets very useful when creating these. Every transition has three symbols that you have to specify. Read, Write and Direction which tells the head what to do next. In JFLAP the blank symbol is represented as □.



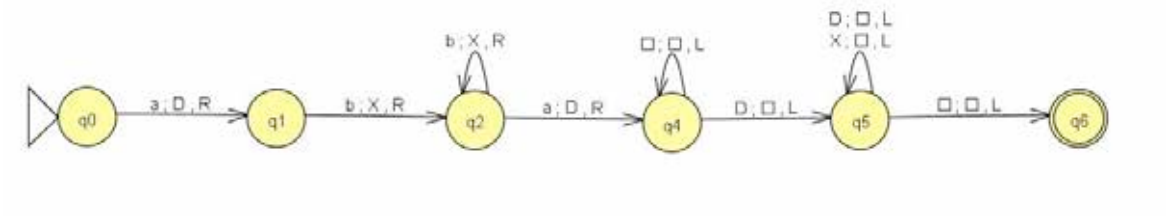
Now create the following TM in JFLAP.



a) What happens in this TM?
b) Why is it important to use a tape?
c) Show some input using JFLAPs transducer.
d) As you would notice empty tapes is accepted, how should you prevent that from happening? Save your project in a new file and create a TM that fixes that.

Assignment 3.2

Create the following Turing machine



Describe what this Turing machine does, recreate it and test run in JFLAP.
What language does this Turing machine accept?

Assignment 3.3

Construct a Turing machine that accepts the following languages.

a) L = $\{a^n b^n \mid \text{where } n > 0\}$, $\Sigma = \{a,b\}$

b) L = $\{a^n b^n c^n \mid \text{where } n > 0\}$, $\Sigma = \{a,b,c\}$

c) TM that doubles the number of a in a string of a's. (Ex. aaa should be aaaaaa.)

Assignment 3.4

Create a Turing machine that adds two binary strings into one single binary.
The operands should be binary strings with a plus operator in-between (Example is that 1010+1001 on the tape should produce the result 10011 on the tape).

To get you started the program can check each digit on both strings from left to right, and check what the result should be from these. Symbols can also be set (like 1010+1001B) at the end of the right operand where the program can write the answer.

Assignment 3.5

Create a Turing machine that accepts the language L = $\{a^m b^n \mid m > n > 0\}$.

SOLUTIONS

1. Regular Languages and Finite State Automata

Assignment 2

a)

Accepted Strings:	Rejected Strings:
aba	bbbaa
abaa	ababab
ababa	aaaaaa
abbaba	bbbbb

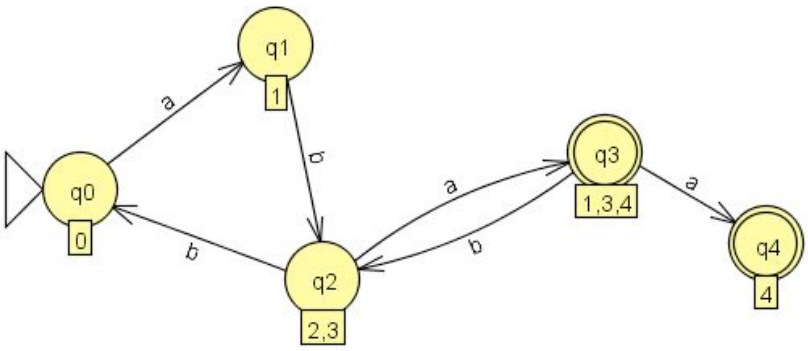
b) 1) Yes 2) No 3) Yes 4) Yes

c) (ab(ab)*b)*(ab+ab(ab)*(a+ab))a

d)

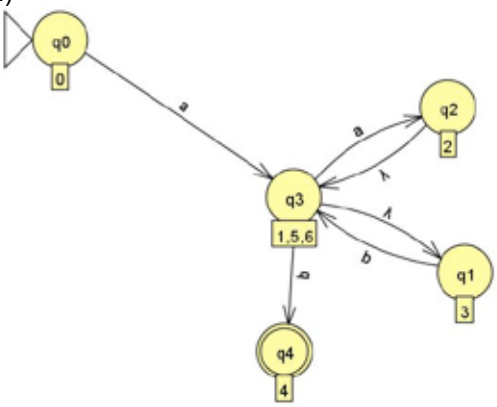
$$S \rightarrow aA$$
$$D \rightarrow \lambda$$
$$B \rightarrow bS$$
$$B \rightarrow aA$$
$$A \rightarrow bB$$
$$A \rightarrow bC$$
$$B \rightarrow aC$$
$$C \rightarrow aD$$

e)

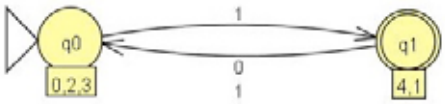


Assignment 3

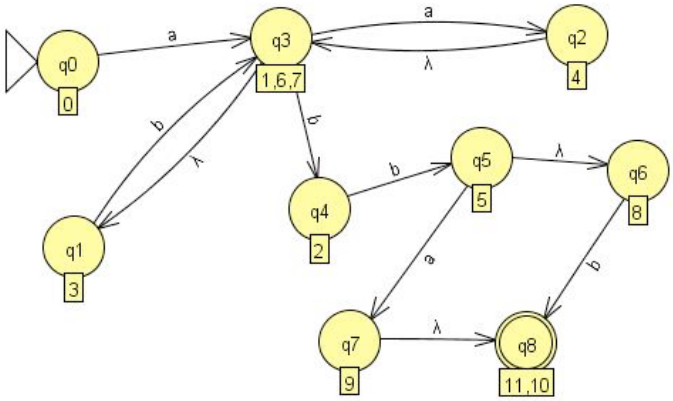
a)



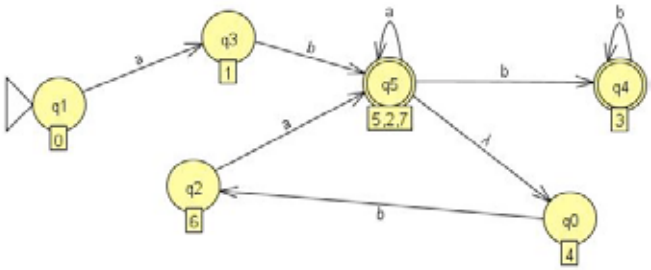
b)



c)

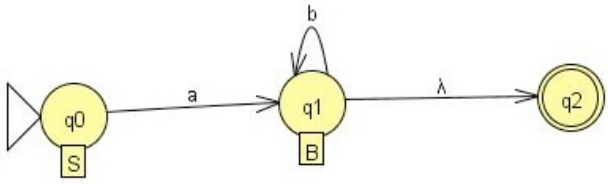


d)

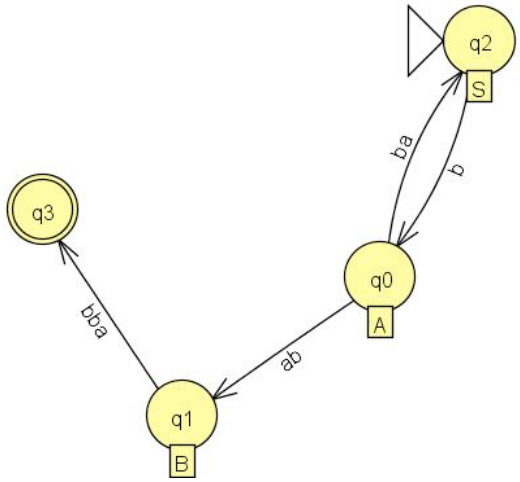

q1
0
a
q3
1
b
a
q5
5,2,7
b
q4
3
b
a
q2
6
λ
b
q0
4

Assignment 4

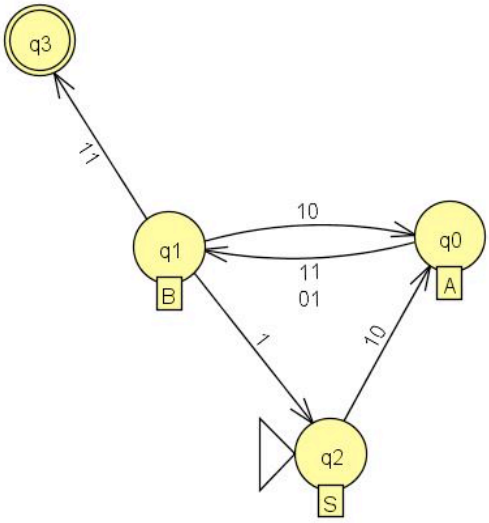
a)



b)

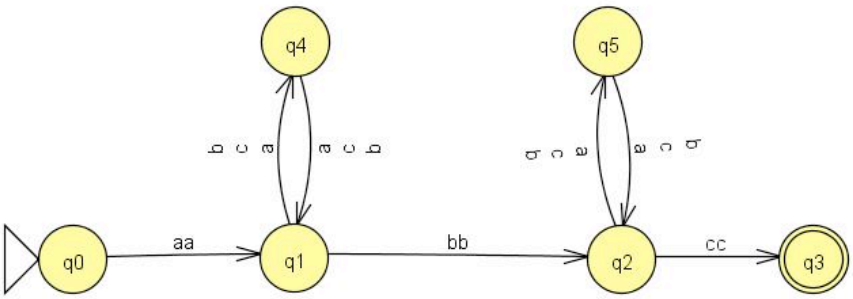


c)



Assignment 5

This is the NFA for the language, *Z = {a,b,c} L = {aaWbbWcc | W = {a,b,c}* | W is even }.*



Grammar for the language, *Z = {a,b,c} L = {aaWbbWcc | W = {a,b,c}* | W is even }.*

$S \rightarrow aaA$

$C \rightarrow \lambda$

$B \rightarrow ccC \mid cE$

$E \rightarrow cB$

$B \rightarrow bE$

$E \rightarrow bB \mid aB$

$B \rightarrow aE$

$D \rightarrow cA$

$A \rightarrow cD$

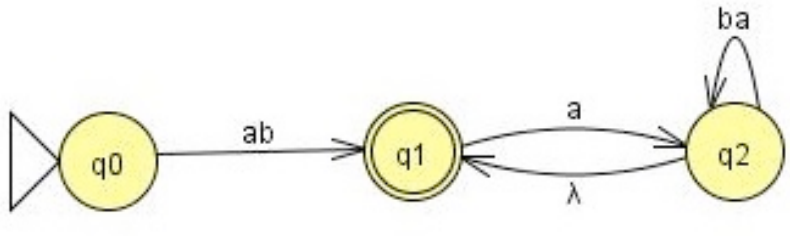
$D \rightarrow bA$

$A \rightarrow bD \mid aD$

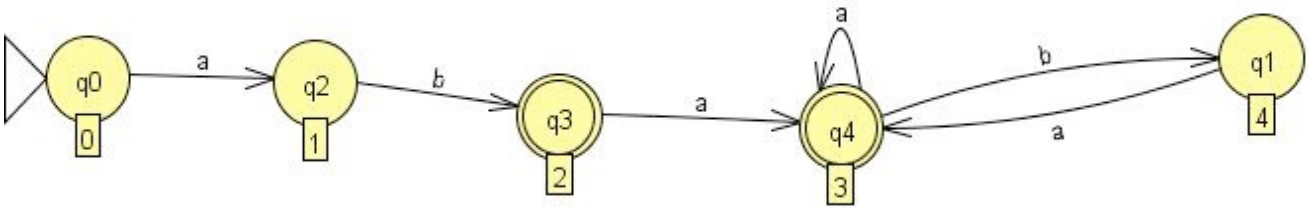
$D \rightarrow aA$

$A \rightarrow bbB$

Here you are free to create either a DFA or NFA. This is an example of a NFA.

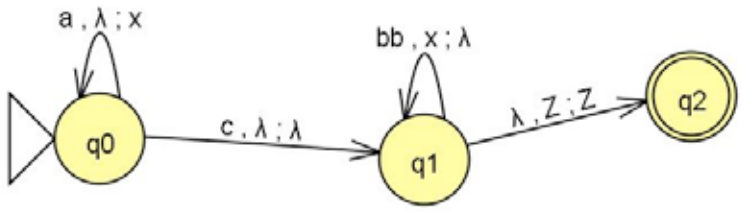


With that NFA, JFLAP can easily convert the NFA to a DFA which is slightly larger and can be converted to a minimized DFA.



2. Context-free Languages and Push Down Automata

Assignment 1

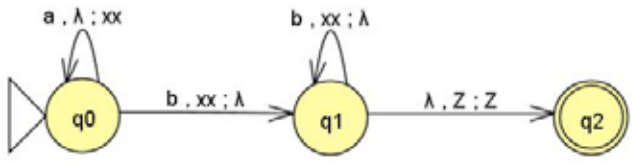


Assignment 2

a) PDA in assignment 1 should be equal to the one in the assignment.

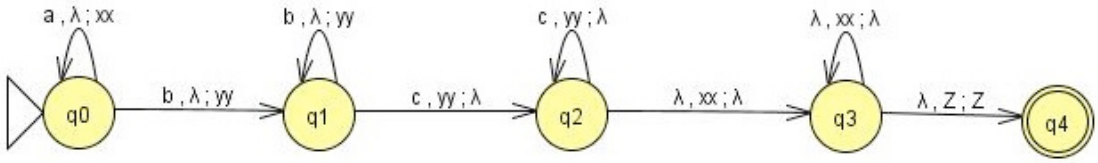
Input	Result
aabb	Accept
aaabbb	Reject
aaaabbbb	Accept
aaaaaabbbbbb	Accept
aaabbbb	Reject
ba	Reject
	Reject
aaaabbb	Reject

b) The other PDA is very similar just change the transitions aa and bb to a and b.



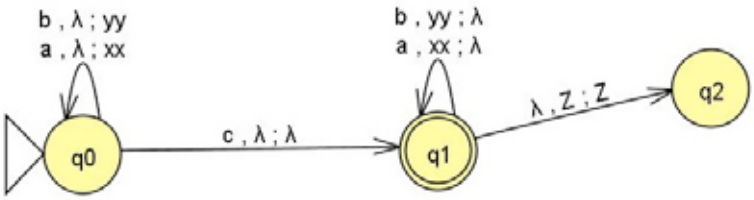
Input	Result
ab	Accept
aabb	Accept
aaabbb	Accept
	Reject
ba	Reject

c)



Input	Result
aabbcc	Accept
aaabbbccc	Accept
aaaabbbbcccc	Accept
aaaaaabbbbbbcccccc	Accept
aaabbbbccccc	Reject
bac	Reject
	Reject
aaaabbbcc	Reject

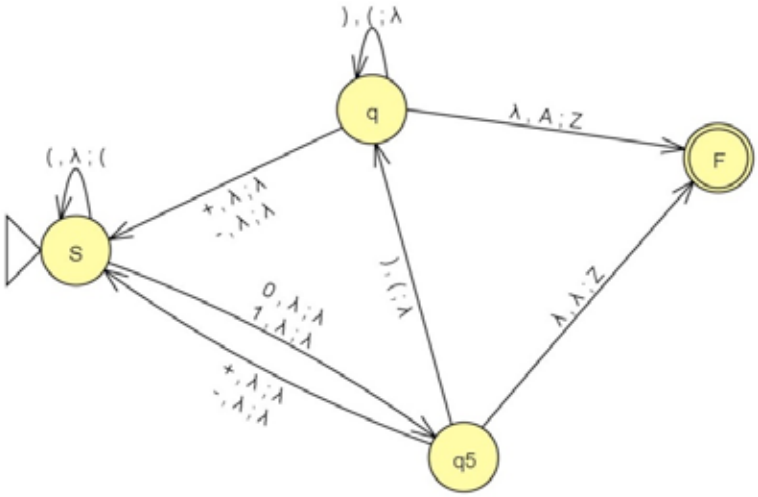
d)



Input	Result
abcab	Reject
abcba	Accept
aca	Accept
acb	Reject
abbbacabbba	Accept
abacbab	Reject

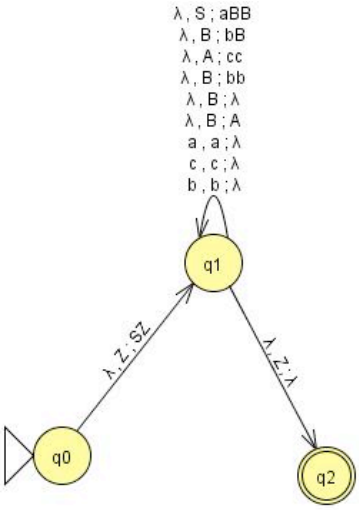
Assignment 3

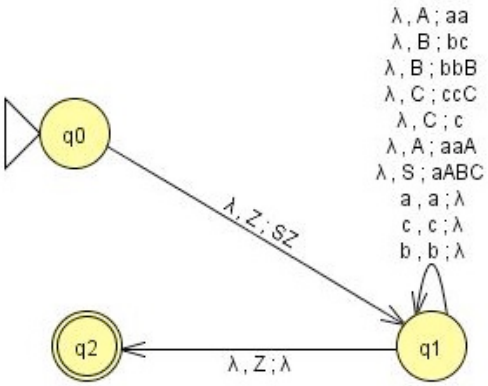
This is the PDA that accepts simple expressions.

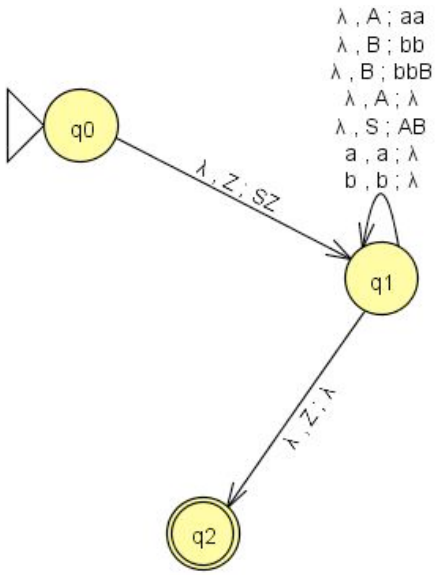


Input	Result
(1+0)	Accept
((1+1)+(1-0))	Accept
((1+1))	Accept
1+0	Accept
1+0+1-1	Accept
	Reject

Assignment 4





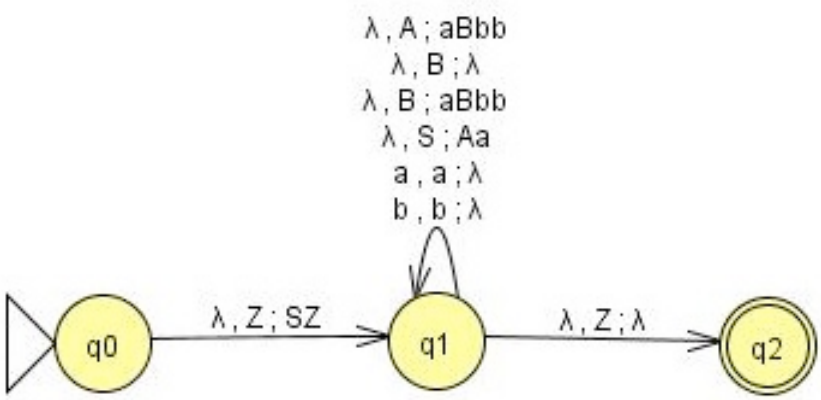


Assignment 5

The Grammar:

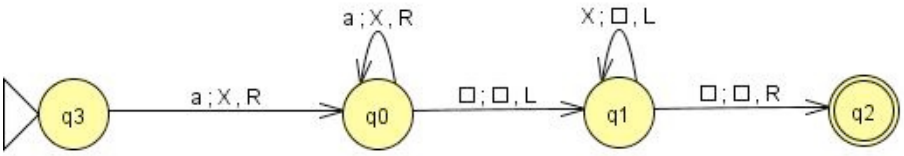
$$S \rightarrow Aa$$
$$A \rightarrow aBbb$$
$$B \rightarrow aBbb$$
$$B \rightarrow \lambda$$

is a Context Free Grammar with the following PDA:



3. Restriction Free Languages and Turing Machines

Assignment 1

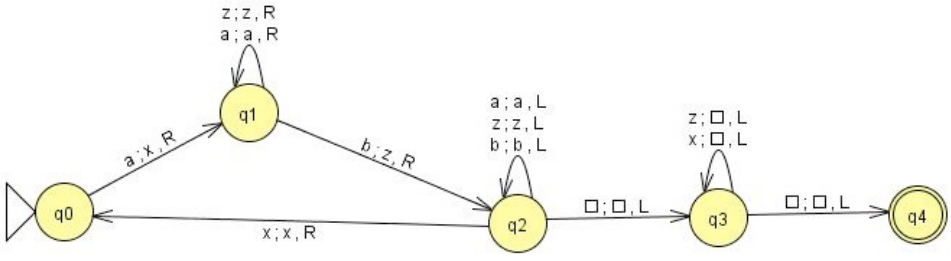


Assignment 2

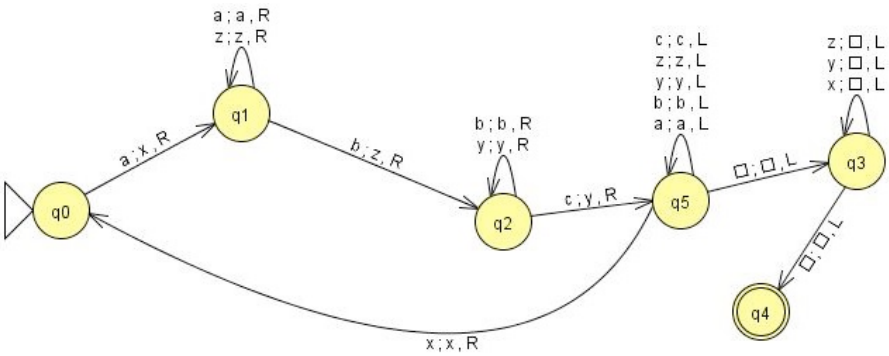
This Turing machine only accepts strings that start and end with a and one or more *b* in between. These gets replaced with the symbol D and b with X while the tape is going to the right. So the string *abbba* would become DXXXD. After this the tape is turned back to the left these symbols are read and determined if the string is accepted of not. Until the final state is reached the tape is also cleared which is done in JFLAP with the square symbol. Adding symbols makes it easy to detect patterns in strings and compute the string properly.
This is the language for the created Turing machine.
L = {aW$_1$a | W$_1$ = b+}, Σ = (a, b)

Assignment 3

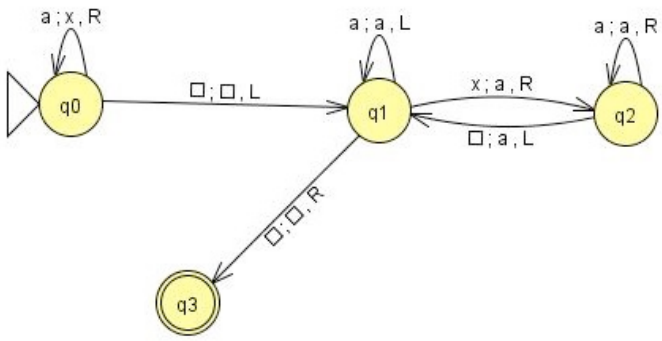
a)



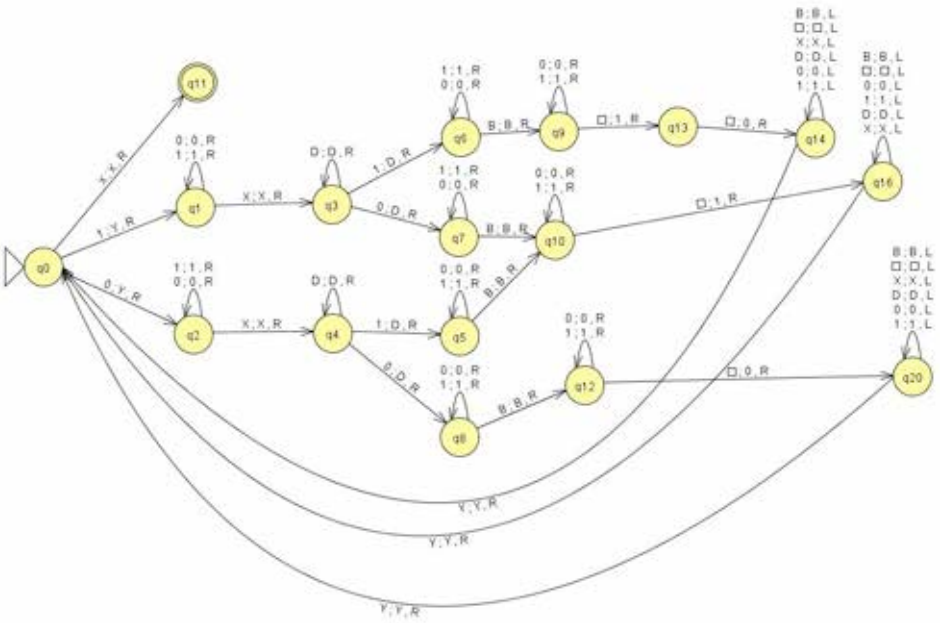
b)



c)



Assignment 4



Assignment 5