

COMP 442 / 6421

Compiler Design

Tutorial 1

Instructor: Dr. Joey Paquet
TAs: Hamed Jafarpour
Vashisht Marhwal

paquet@cse.concordia.ca
hamed.jafarpour@concordia.ca
vmarhwal97@gmail.com



Outline

- **AtoCC**

- **JFLAP**



AtoCC

AtoCC is a learning environment helps the learner in theoretical computer science (automaton theory, formal languages) and its application in compiler design.

AtoCC consists of 7 components: AutoEdit, AutoEdit Workbook, RegExpEdit, kfG Edit, TDiag, VCC and SchemeEdit.

Further information on the architecture of AtoCC can be found in:

<http://www.atocc.de/cgi-bin/atocc/site.cgi?lang=de&site=main>

Note: Students will need to use it only for learning. The learning environments are not available whenever and wherever.



AtoCC --- RegExp Edit

It is a powerful tool that we can use to generate DFA from regular expression and validate your work. In the following slides you will find screenshots on how to use this tool in order to create a DFA from a regular expression that should conform to the lexical specification of the language.

RegExpEdit

File Help

New Open Save Export Automaton Export Grammar

RegExp Editor Alphabet RegExp Simulation

RegExp Editor

RegExp Project:

- Introduction Wizard** A quick introduction to get in touch with RegExpEdit.
- New RegExp** Create an empty project.

Getting Help:

- Online FAQ** To get any further information visit the Online Help Center.
- www.AtoCC.de** Visit the AtoCC website.

$$(((\{a\}(\{a\} \cup \{b\})^*))(\{b\}\{b\}))((\{a\} \cup \{b\})^*))$$

$$(((a \cdot (a+b)^*) \cdot (b \cdot b)) \cdot (a+b)^*)$$

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

Genesis-X7 Software 2008

RegEx Alphabet

Alphabet Items:

Clear

Edit Alphabet:

Add Alphabet Item

Delete Alphabet Item

Predefined Alphabet

a, b, c

x, y, z

0, 1

0, 1, 2, 3, 4 ... 9

a, b, c, d ... z

X ... Y

RegExEdit

File Help

New Open Save Export Automaton Export Grammar

RegEx Editor Alphabet **RegExp** Simulation

RegExp Editor

Enter RegExp here

ab*

Use the formal notation for regular expressions.
Like: $(a+b)^*ab(a+b)^*$ for $L = \{w \mid w \text{ contains } ab\}$ over $\{a,b\}$.

Minimized RegExp

ab*

Compare RegExp with another

Compare

Transform to NEA

Generate NEA graph for your RegExp at the right. **Show NEA** →

RegExp

NEA Graph Minimized NEA Graph

The NEA graph consists of the following nodes and transitions:

- Start node s has an ϵ -transition to node s_1 and another ϵ -transition to node s_2 .
- Node s_1 has an ϵ -transition to node s_2 .
- Node s_1 has a transition labeled a to node f_1 .
- Node f_1 has an ϵ -transition to node f_2 .
- Node f_2 has an ϵ -transition to node s_4 .
- Node s_3 has an ϵ -transition to node s_4 .
- Node s_3 has a transition labeled b to node f_3 .
- Node f_3 has an ϵ -transition to node f_4 .
- Node s_4 has an ϵ -transition to node f_4 .
- Node f_4 has an ϵ -transition to node f_5 .
- Node f_5 is the final state, indicated by a double arrow.

Hint: For ϵ you must write **EPSILON** in your RegExp.

$\epsilon u = u \epsilon = u$ $\epsilon^* = \epsilon$ $u+v = v+u$ $u+u = u$ $(u^*)^* = u^*$ $u(v+w) = uv+uw$ $(uv)^*u = u(vu)^*$ $(u+v)^* = (u^* + v^*)^*$

RegExpEdit

File Help

New Open Save **Export Automaton** Export Grammar

RegExp Editor | Alphabet | **RegExp** | Simulation

RegExp Editor

Enter RegExp here

ab*

Use the formal notation for regular expressions.
Like: $(a+b)^*ab(a+b)^*$ for $L = \{w \mid w \text{ contains } ab\}$ over $\{a,b\}$.

Minimized RegExp

ab*

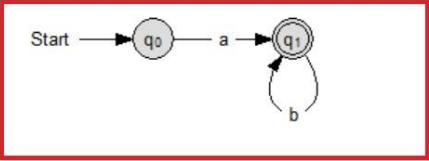
Compare RegExp with another

Compare

Transform to NEA

Generate NEA graph for your RegExp at the right. Show NEA

NEA Graph **Minimized NEA Graph**



The diagram shows a Non-deterministic Automaton (NEA) with two states: q_0 and q_1 . q_0 is the start state, indicated by an arrow labeled "Start". q_1 is the final state, indicated by a double circle. There is a transition from q_0 to q_1 labeled 'a'. There is a self-loop transition on q_1 labeled 'b'.

Hint: For ϵ you must write **EPSILON** in your RegExp.

$\epsilon u = u \epsilon = u$ $\epsilon^* = \epsilon$ $u+v = v+u$ $u+u = u$ $(u^*)^* = u^*$ $u(v+w) = uv+uw$ $(uv)^*u = u(vu)^*$ $(u+v)^* = (u^* + v^*)^*$

RegExpEdit

File Help

New Open Save Export Automaton

RegExp Editor | Alphabet | RegExp | Simu

RegExp Editor

Enter RegExp here

ab*

Use the formal notation for regular expressions
Like: $(a+b)^*ab(a+b)^*$ for $L = \{w \mid w \text{ contains } ab\}$

Minimized RegExp

ab*

Compare RegExp with another

Transform to NEA
Generate NEA graph for your RegExp at t

Hint: For ϵ you must write EPSILON

$\epsilon u = u\epsilon = u$ $\epsilon^* = \epsilon$ $u+v = v+u$

Genesis-X7 Software 2004 - 2008

AutoEdit [C:\Users\jafar\Desktop\ab.xml]

File Help

New Open Save Undo Redo Notepad Export Grammar Export RegExp Export Compiler

Automaton Editor | Type | Alphabet | **Transition Table** | Transition Graph | Publish | **Simulation**

AutoEdit Type

Current Type: NEA

NEA are equivalent descriptions for regular languages.

Definition:

$M = (Q, \Sigma, \delta, q_0, E)$

Q ... finite set of states,
 Σ ... input alphabet,
 δ ... total transition function, $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$,
 q_0 ... start state ($q_0 \in Q$),
 E ... finite set of final states ($E \subseteq Q$)

You can use ϵ labels for spontaneous transitions.

$L(M) = \{w \mid w \in \Sigma^* \text{ and } (q_0, w) \vdash^* (q_e, \epsilon) \text{ and } q_e \in E\}$

NEA can be converted to an equivalent DEA or to a NEA without ϵ

<<< Back Next >>>

File Help















Automaton Editor | Type | Alphabet | **Transition Table** | Transition Graph | Publish | Simulation

AutoEdit State-Table

Edit Transition Table:

-  Add State
-  Add Transition
-  Add Label
-  Delete selected

Edit initial conditions:

Automaton initial state:

State Transition Table:

initial state: q_0

State:

Name: final state:

- X

Transition:

Target:

- X

Label:

Read:

- X

State:

Name: final state:

- X

Transition:

Target:

- X

Label:









Read:

- X

<<< Back

Next >>>

File Help

 New
  Open
  Save
  Undo
  Redo
  Notepad
  Export Grammar
  Export RegExp
  Export Compiler

Automaton Editor | Type | Alphabet | Transition Table | Transition Graph | **Publish** | Simulation

AutoEdit

Publish

Edit Publish settings:

 Change Font

 Export Graph ...

 Export Definition ...

 Export Transitions ...

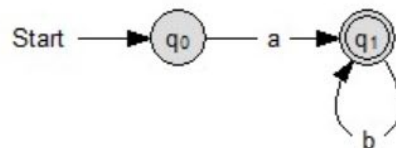
 Export as HTML

HTML Publish:

Automaton

Type: NEA

Transition Graph:



Definition:

$$M = (\{q_0, q_1\}, \{a, b, c\}, \delta, q_0, \{q_1\})$$

Transition Table:

δ	a	b	c
q_0	$\{q_1\}$	$\{\}$	$\{\}$
q_1	$\{\}$	$\{q_1\}$	$\{\}$

Grammar:

$$G = (N, T, P, s)$$

<<< Back

Next >>>

File Help

New Open Save Undo Redo Notepad Export Grammar Export RegExp Export Compiler

Automaton Editor | Type | Alphabet | Transition Table | Transition Graph | Publish | **Simulation**

AutoEdit

Simulation

Edit Simulation settings:

input:



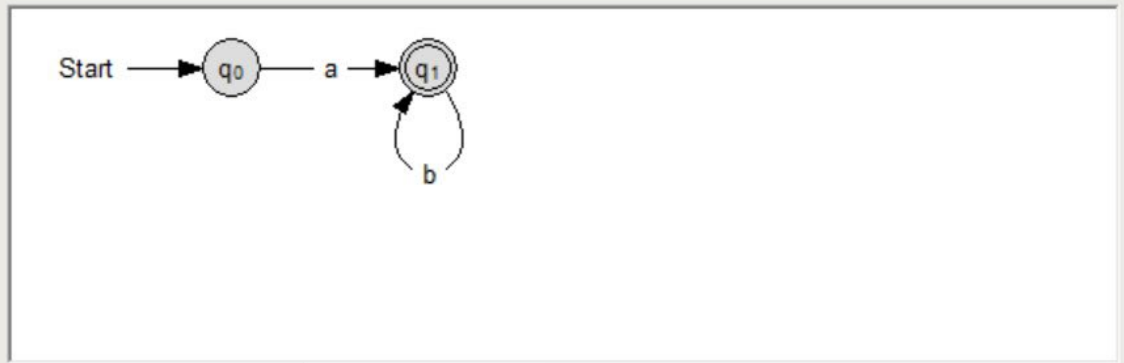
AB Start Simulation

Speed:

Single Step

Export Export Scheme Code

Simulation:



Configuration sequence | Check multiple input

Configuration sequence

<<< Back

File Help

New Open Save Undo Redo Notepad Export Grammar Export RegExp Export Compiler

Automaton Editor | Type | Alphabet | Transition Table | Transition Graph | Publish | Simulation

AutoEdit Simulation

Edit Simulation settings:

Input:

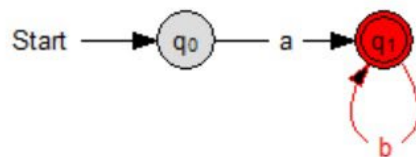
abbb

 Start Simulation

Speed:


 Single Step

Simulation:



AutoEdit

×

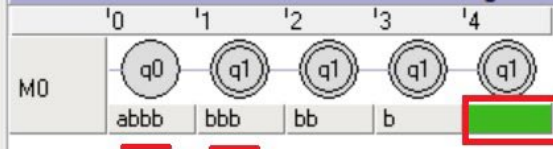
Steps:

q₀ -> q₁ -> q₁ -> q₁ -> q₁

OK

Configuration sequence | Check multiple input

Configuration sequence



<<< Back

File Help



New



Open



Save



Undo



Redo



Notepad



Export Grammar



Export RegExp



Export Compiler

Automaton Editor

Type

Alphabet

Transition Table

Transition Graph

Publish

Simulation

AutoEdit Simulation

Edit Simulation settings:

Input:



Start Simulation

Speed:

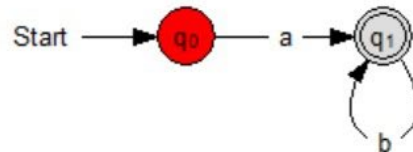


Single Step



Export Scheme Code

Simulation:



Configuration sequence

Check multiple input

Check multiple input



Check



Delete

Length: 10



Create Random Word

ab	#true
abb	#true
aaa	#false
abc	#false
b	#false

<<< Back

File Help

New Open Save Undo Redo Notepad Export Grammar Export RegExp Export Compiler

Automaton Editor **Type** Alphabet Transition Table Transition Graph Publish Simulation


AutoEdit Type


Current Type: NEA

Edit Type:

 Set automaton type

 **Convert to DEA**

 Convert to NEA

 Minimize Automaton

 Validate Automaton

NEA are equivalent descriptions for regular languages.

Definition:

$$M = (Q, \Sigma, \delta, q_0, E)$$

Q ... finite set of states,

Σ ... input alphabet,

δ ... total transition function

q_0 ... start state ($q_0 \in Q$),

E ... finite set of final states ($E \subseteq Q$)

You can use ϵ labels for spontaneous transitions.

$$L(M) = \{w \mid w \in \Sigma^* \text{ and } (q_0, w) \xrightarrow{*} (q_e, \epsilon) \text{ and } q_e \in E\}$$

NEA can be converted to an equivalent DEA or to a NEA without ϵ

AutoEdit



Your current automaton and its layout may be lost, are you sure?

Yes

No

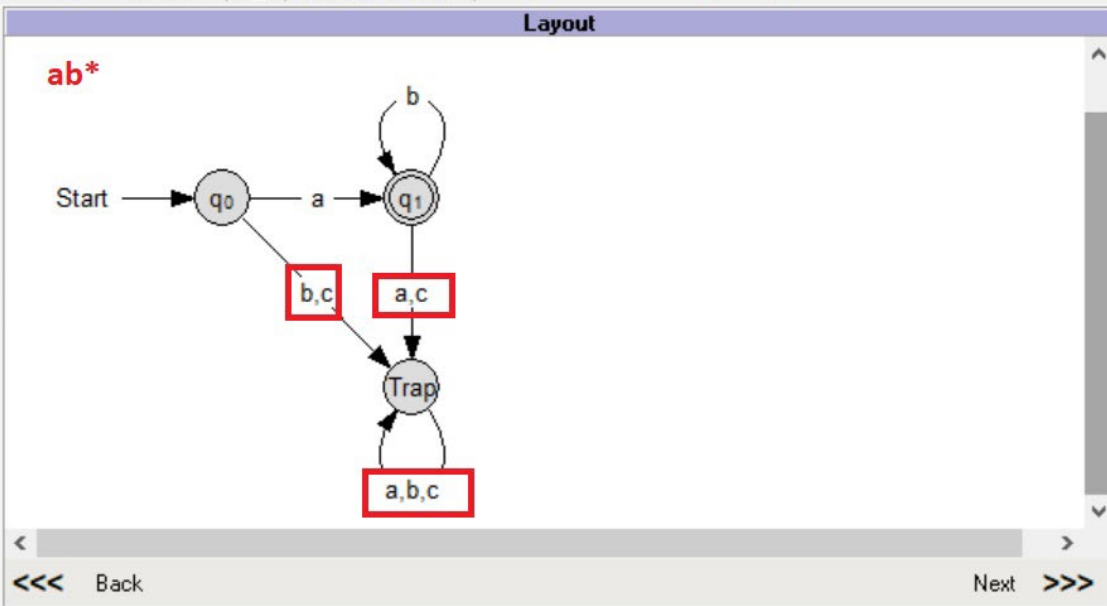
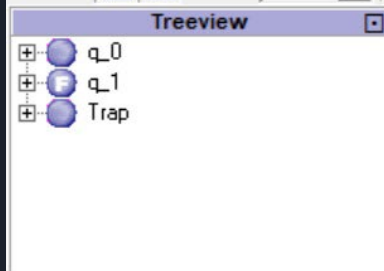
<<< Back

Next >>>

AutoEdit
State-Graph

Current Type: DEA

initial state: q_0



AutoEdit Publish

Edit Publish settings:

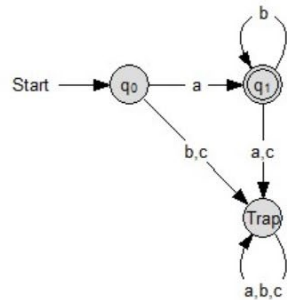
-  Change Font
-  Export Graph ...
-  Export Definition ...
-  Export Transitions ...
-  Export as HTML

HTML Publish:

Automaton

Type: DEA

Transition Graph:


 Definition: $M = (\{q_0, q_1, \text{Trap}\}, \{a, b, c\}, \delta, q_0, \{q_1\})$

Transition Table:

δ	a	b	c
q ₀	q ₁	Trap	Trap
q ₁	Trap	q ₁	Trap
Trap	Trap	Trap	Trap

Grammar: $G = (N, T, P, s)$
 $G = (\{q_0, q_1\}, \{a, b, c\}, P, q_0)$
 $P = \{$
 $q_0 \rightarrow a q_1 \mid a$
 $q_1 \rightarrow b q_1 \mid b$
 $\}$
RegExp: ab^*

AutoEdit Simulation

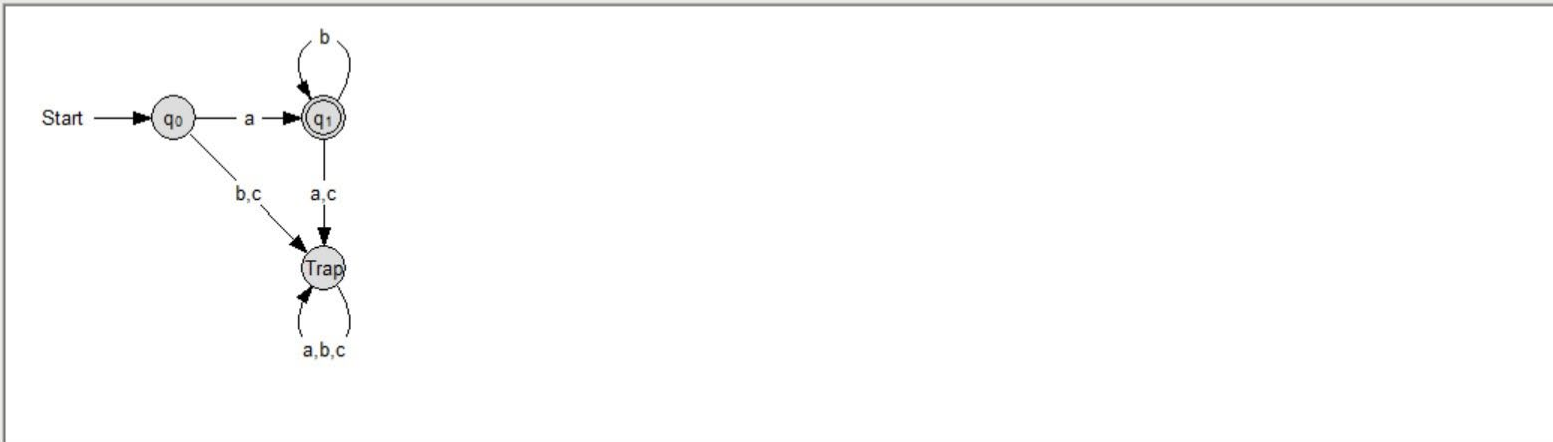
Edit Simulation settings:

Input:
abbb

Speed:

Single Step

Simulation:



Configuration sequence | Check multiple input |

Configuration sequence					
	'0	'1	'2	'3	'4
M0	q0	q1	q1	q1	q1
	abbb	bbb	bb	b	



JFLAP?

What Is JFLAP?

JFLAP is a package which can be used as an aid in learning the basic concepts of Formal Languages and Automata Theory. Some properties of the JFLAP:

- Regular languages – create
 - DFA
 - NFA
 - regular grammar
 - regular expression

- Regular languages – conversions
 - NFA -> DFA -> Minimal DFA
 - NFA <-> regular expression
 - NFA <-> regular grammar

Note: For more information about JFLAP visit the bellow link:

<http://www.jflap.org/>

JFLAP

- [HOME](#)
- [What is JFLAP](#)
- [Get JFLAP](#)
- [JFLAP Tutorial](#)
(partially updated for JFLAP 7.1)
- [JFLAP Videos](#)
- [Instructor Use](#)
- [Modules and Exercises](#)
- [History of JFLAP](#)
- [World Usage to June 2008](#)
- [JFLAP book](#)
- [books including JFLAP](#)
- [Software using JFLAP](#)
- [JFLAP papers](#)

Get JFLAP

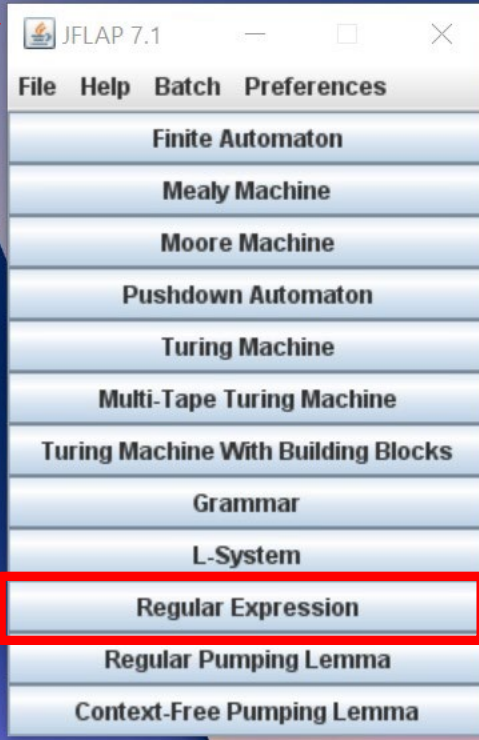
INFORMATION about JFLAP:

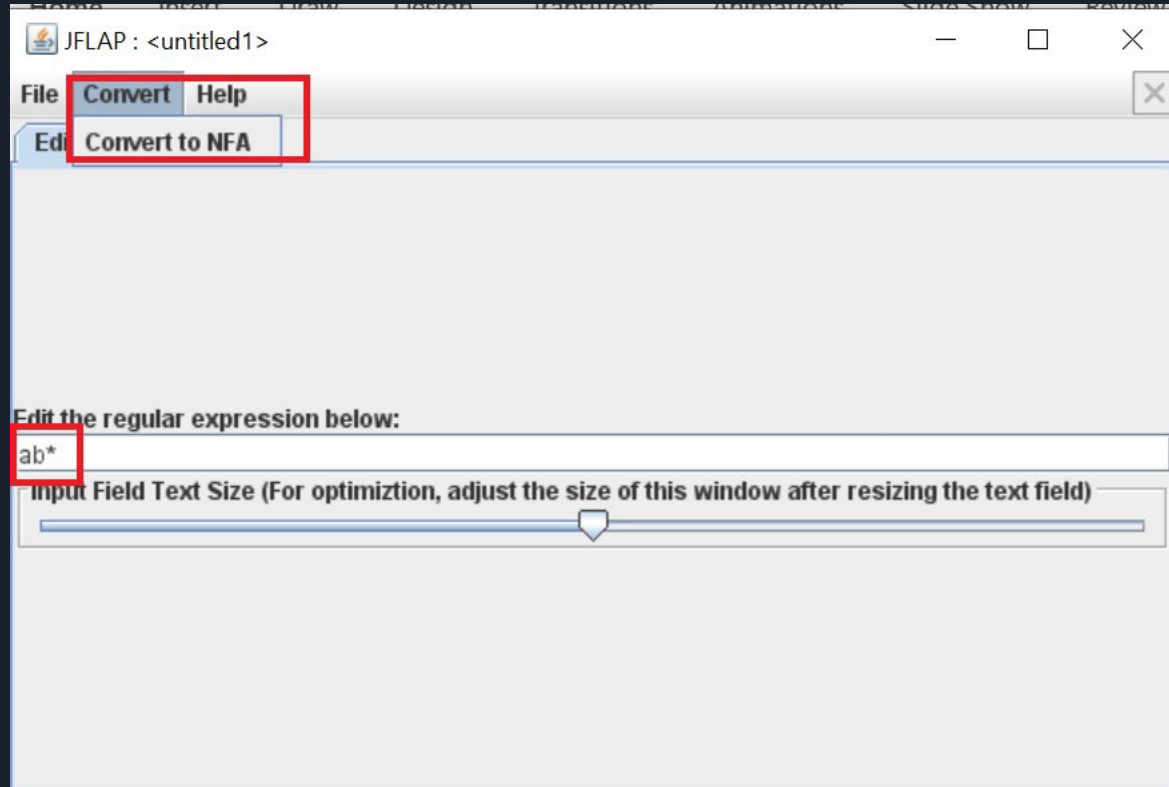
- Get JFLAP Software

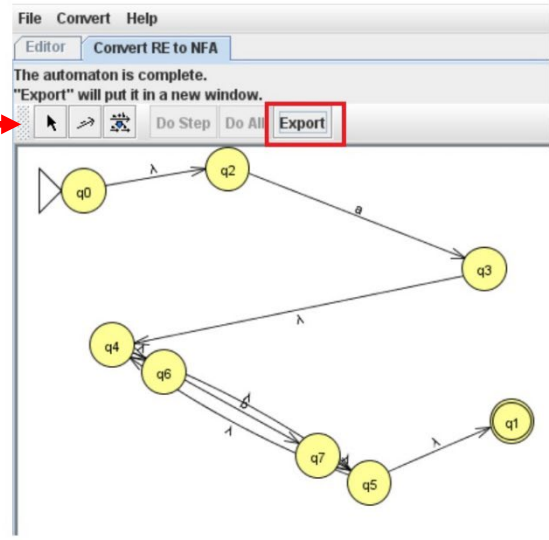
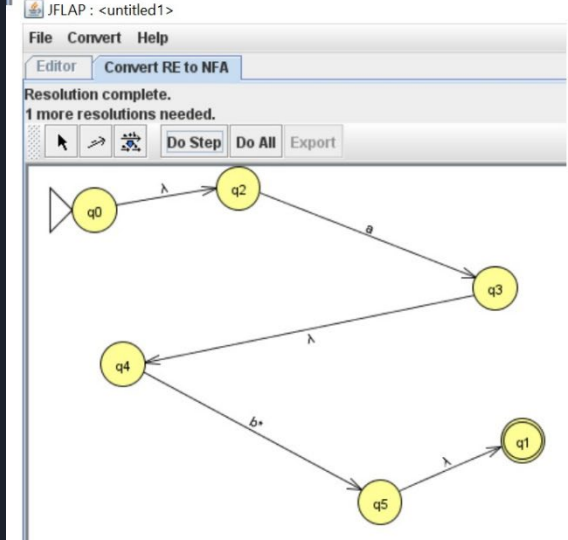
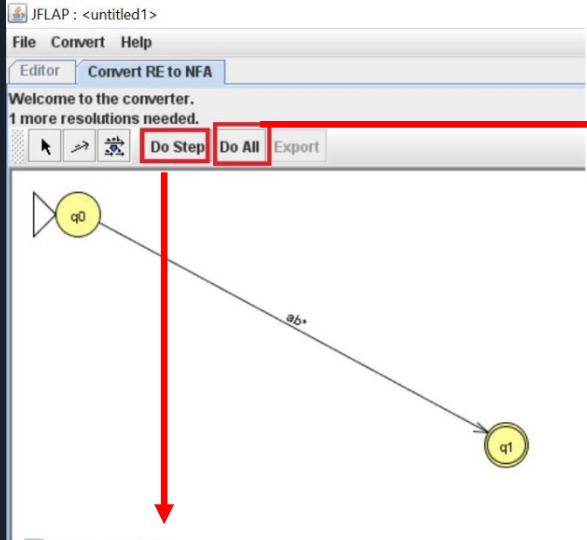
Please [fill out this form](#) and you can have the most recent version of JFLAP to use for free.



JFLAP7.1







JFLAP : <untitled2>

File **Input** Test View Convert Help

Step with Closure... Ctrl-R
Step by State... Ctrl+Shift-R
Fast Run...
Multiple Run Ctrl-M

```
graph LR; q0((q0)) -- a --> q2((q2)); q2 -- λ --> q3((q3)); q3 -- λ --> q4((q4)); q4 -- λ --> q6((q6)); q6 -- λ --> q7((q7)); q7 -- λ --> q5((q5)); q5 -- λ --> q1(((q1))); style q0 fill:#ffff00,stroke:#000,stroke-width:1px; style q1 fill:#ffff00,stroke:#000,stroke-width:2px; style q2 fill:#ffff00,stroke:#000,stroke-width:1px; style q3 fill:#ffff00,stroke:#000,stroke-width:1px; style q4 fill:#ffff00,stroke:#000,stroke-width:1px; style q5 fill:#ffff00,stroke:#000,stroke-width:1px; style q6 fill:#ffff00,stroke:#000,stroke-width:1px; style q7 fill:#ffff00,stroke:#000,stroke-width:1px;
```

Automaton Size

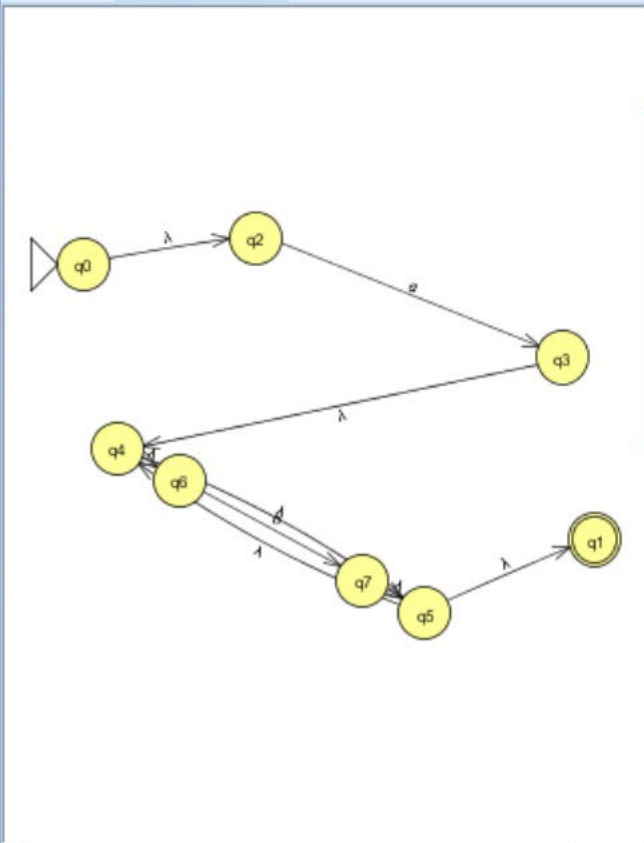


Table Text Size



Input	Result
ab	Accept
	Reject
	Reject
a	Accept
b	Reject
abbbbb	Accept
	Reject
a	Reject
a	Reject

JFLAP : <untitled2>

File Input Test View **Convert** Help

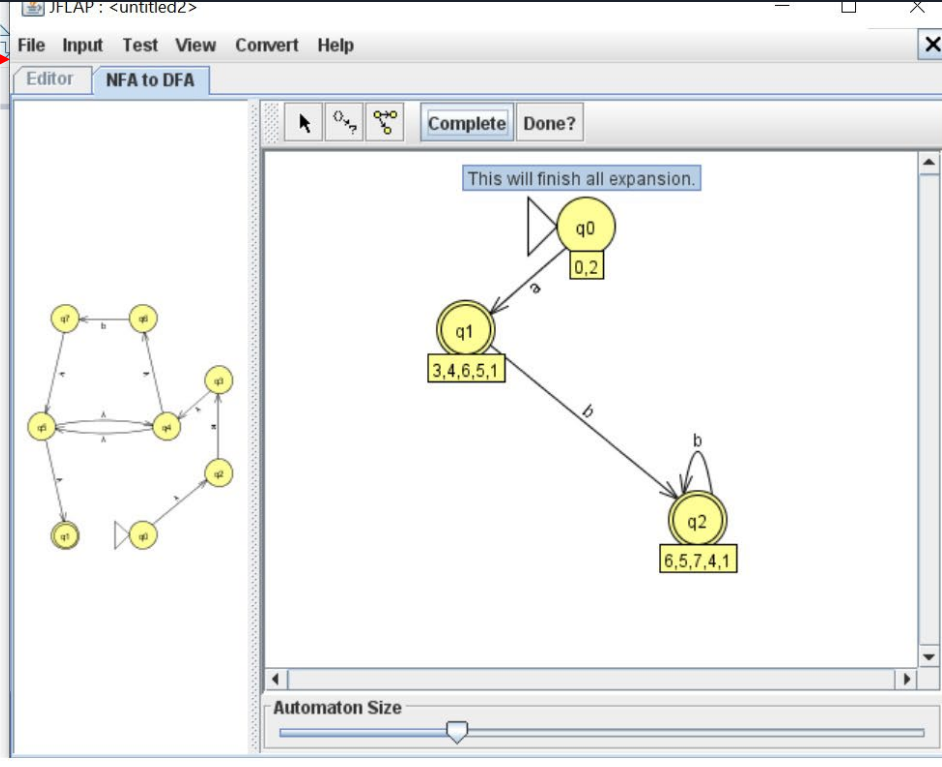
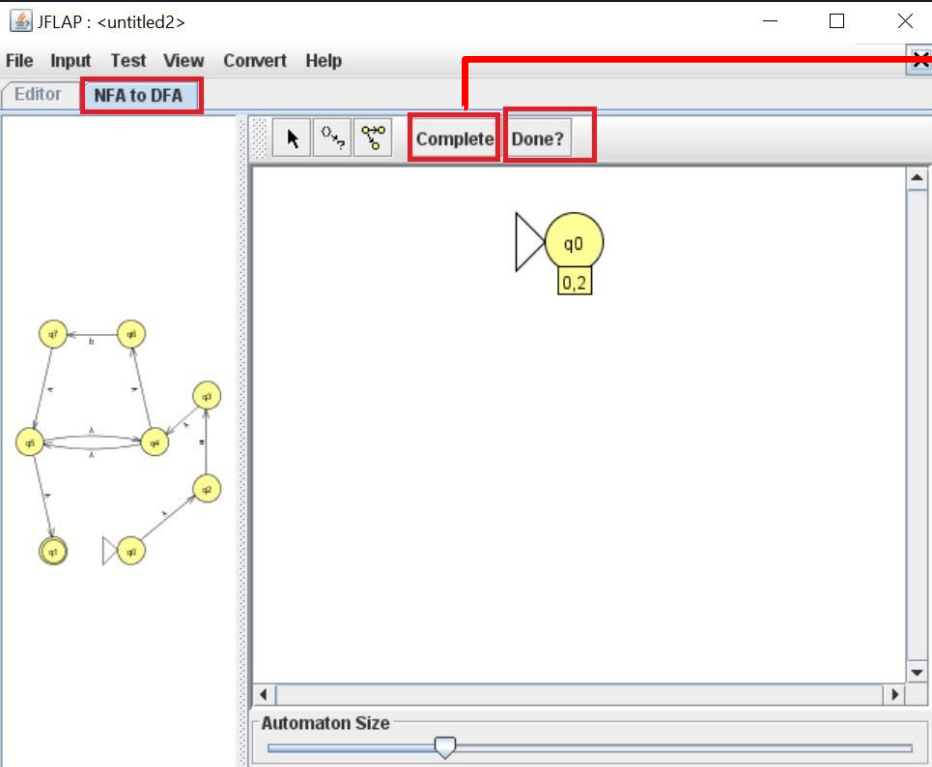
Editor Multiple Run **Transitions** Nondeterminism

Convert to DFA
 Minimize DFA
 Convert to Grammar
 Convert FA to RE
 Combine Automata
 Add Trap State to DFA

Table Text Size

Input	Result
ab	Accept
	Reject
	Reject
a	Accept
b	Reject
abbbbb	Accept
	Reject
a	Reject
a	Reject

Load Inputs Run Inputs Clear Enter Lambda View Trace





Implementation of lexical analyzer

Two ways to implement the lexical analyzer:

1. Table driven (but constructing a transition table by hand is not an easy job)
2. Handwritten (it requires you to be very careful considering all the possible situations)

Notes:

- It is your choice to pick one of the methods to implement and your choice will not affect the prospective assignments.
- The output of the Scanner is the stream of tokens which can be accessed when the nextToken() method being called.
- You are not allowed to use any tool like Lex can generate a Scanner automatically.

Thanks!

