COMP 442 /6421 Compiler Design

Tutorial 1

Instructor:	Dr. Joey Paquet	paquet@cse.concordia.ca
TAs:	Hamed Jafarpour	hamed.jafarpour@concordia.ca
	Dhaval Patel	p_haval@encs.concordia.ca

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.

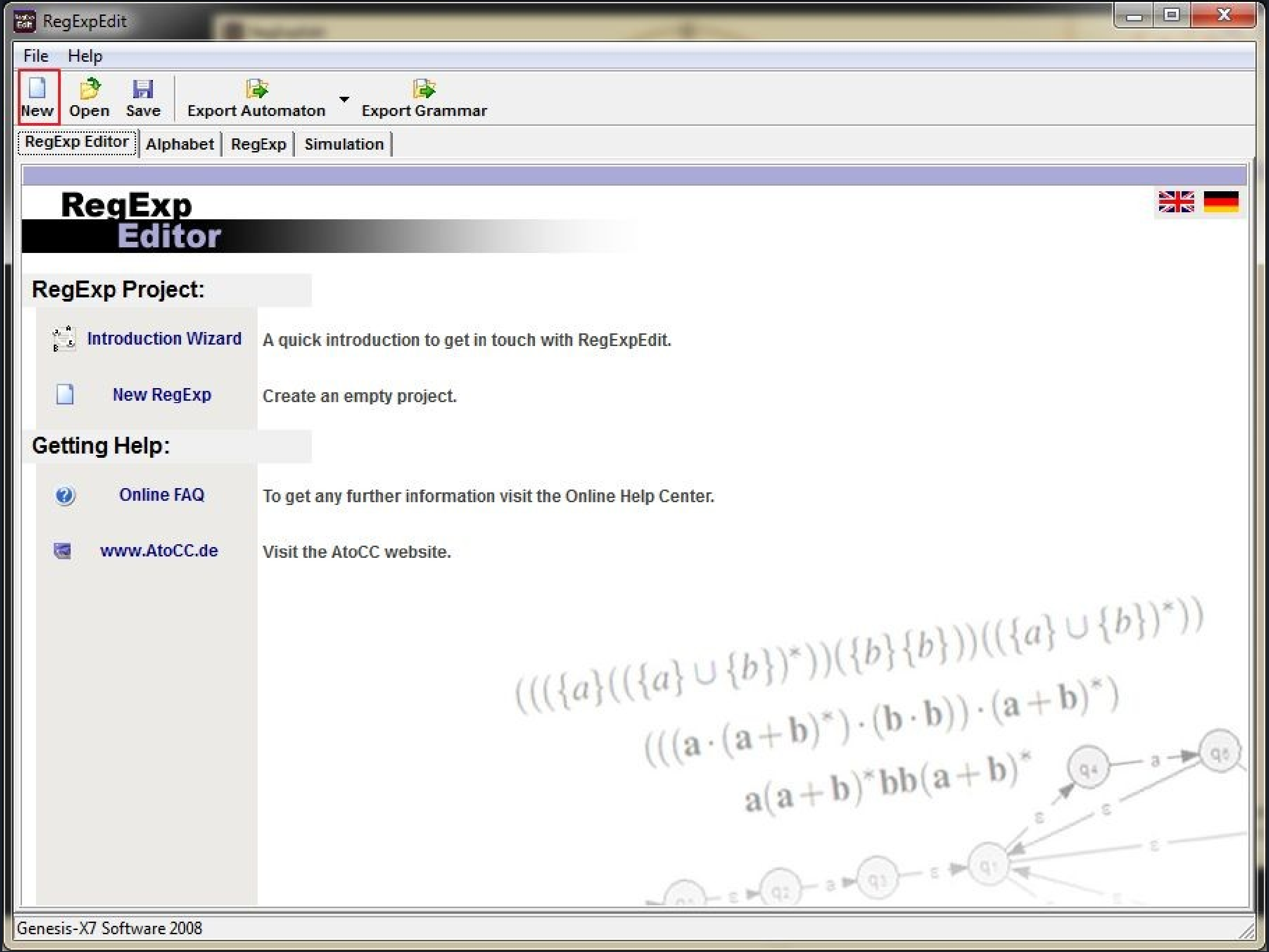
AtoCC consists of 6 components: AutoEdit, AutoEdit Workbook, kfG Edit, TDiag, VCC and SchemeEdit.

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.



RegExp Project:

Introduction Wizard — A quick introduction to get in touch with RegExpEdit.

New RegExp — Create an empty project.

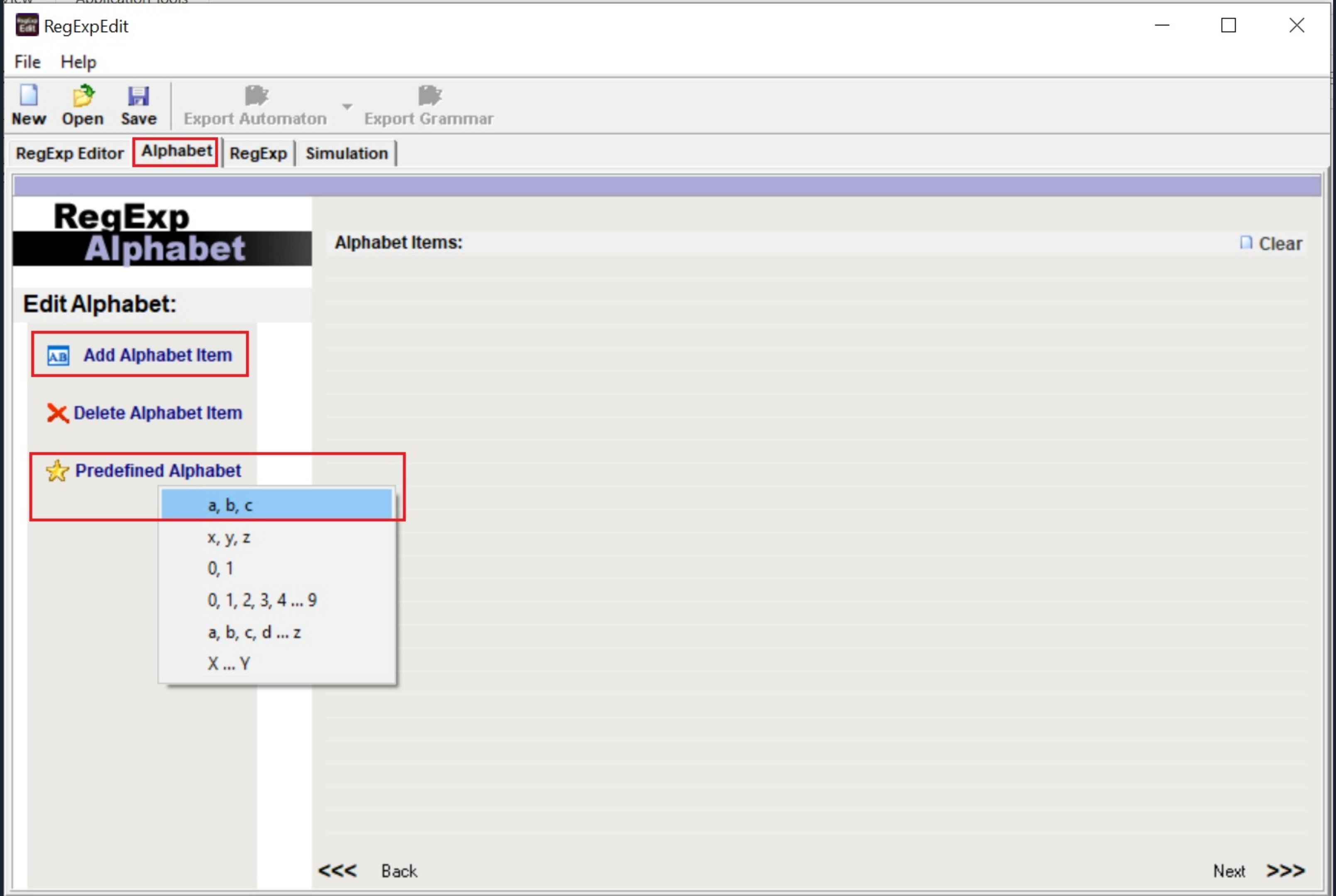
Getting Help:

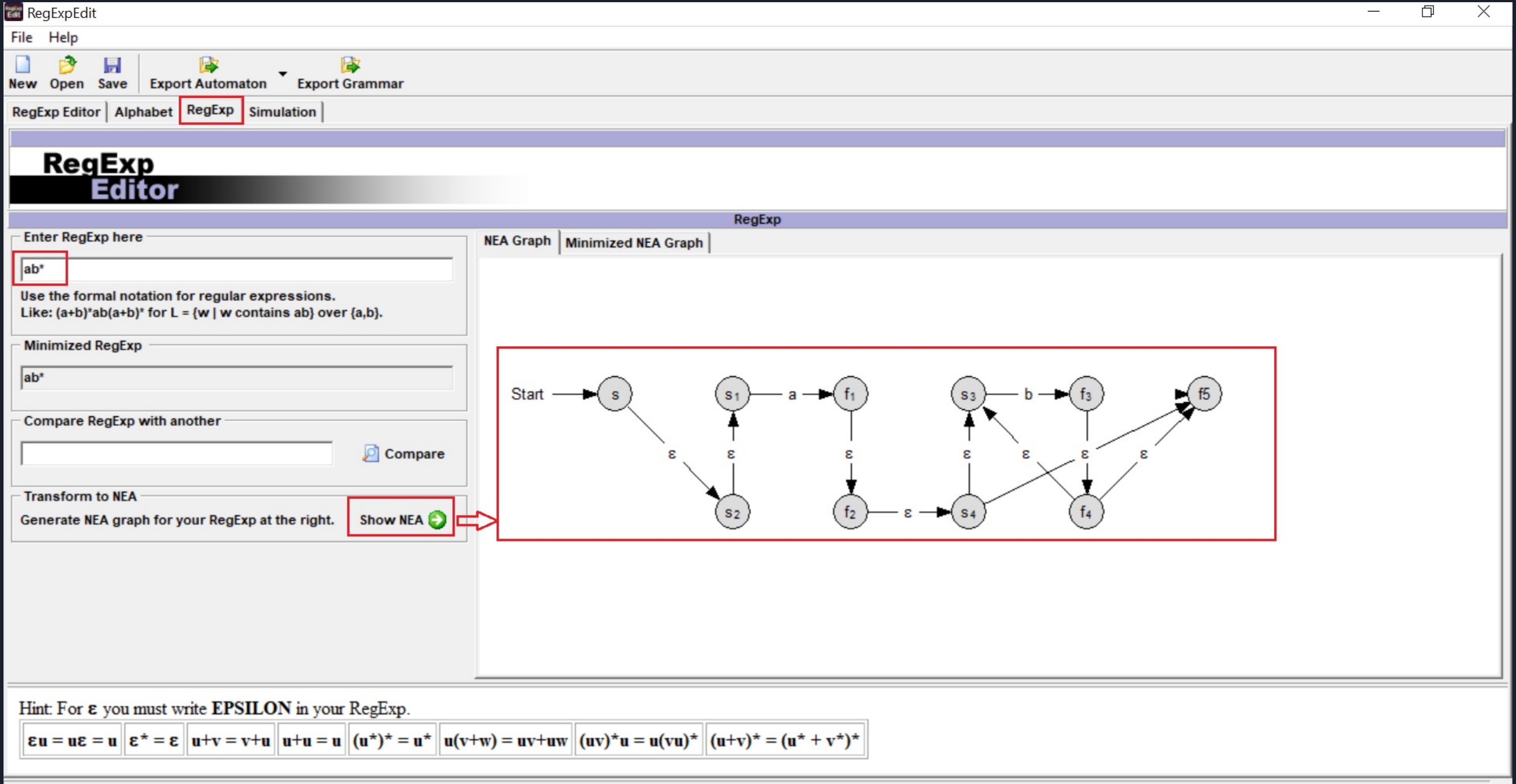
Online FAQ — To get any further information visit the Online Help Center.

www.AtoCC.de — Visit the AtoCC website.

$((((\{a\}((\{a\} \cup \{b\})^*))(\{b\}\{b\}))((\{a\} \cup \{b\})^*))$
$(((a \cdot (a+b)^*) \cdot (b \cdot b)) \cdot (a+b)^*)$
$a(a+b)^*bb(a+b)^*$
q4
q6
q1
q2
q3
a
ε

Genesis-X7 Software 2008


RegExpEdit
File
Help
New
Open
Save
Export Automaton
Export Grammar
RegExp Editor
Alphabet
RegExp
Simulation
RegExp
Alphabet
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
Alphabet Items:
Clear
<<< Back
Next >>>



Enter RegExp here

ab*

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

Minimized RegExp

ab*

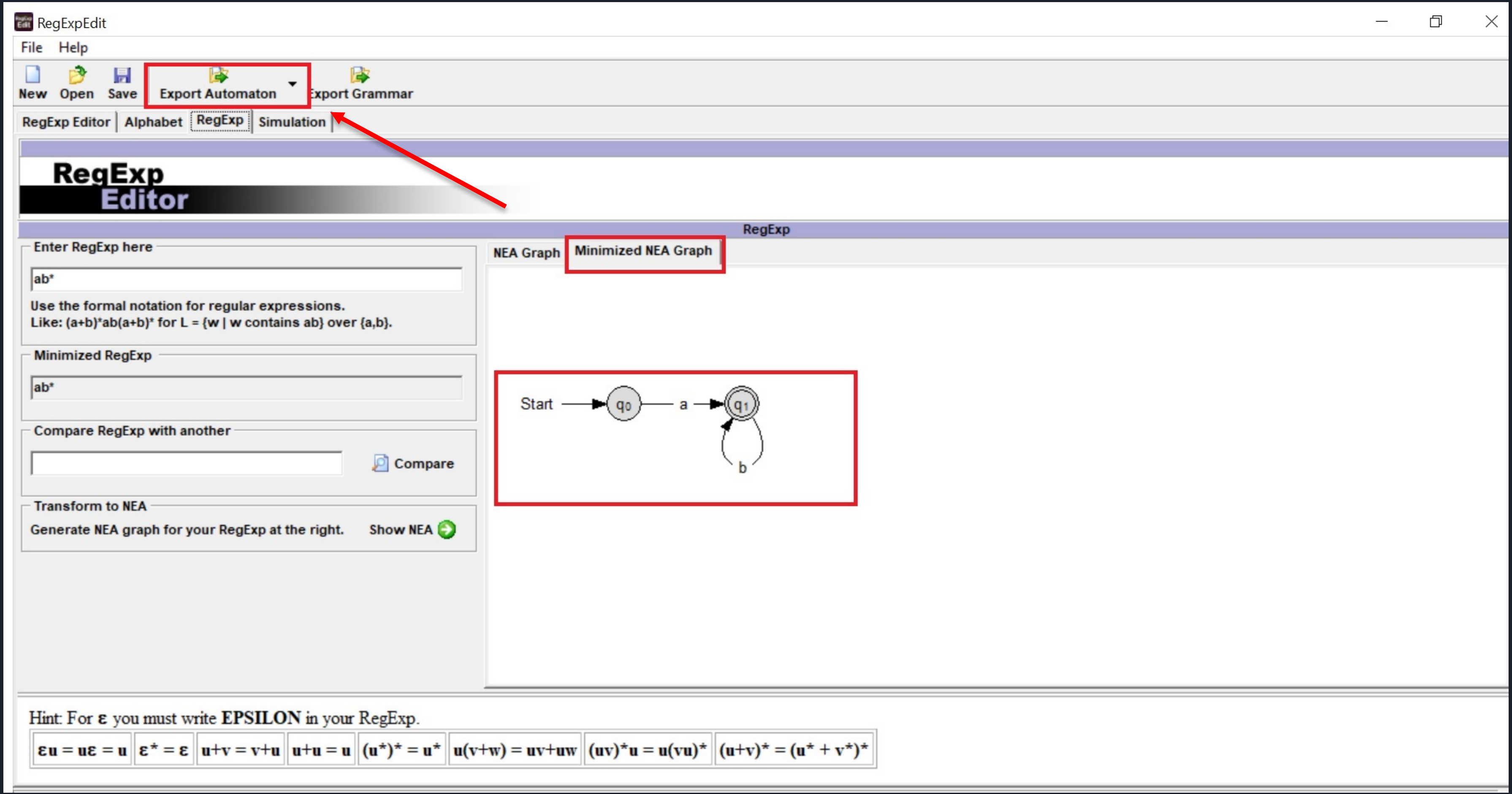
Compare RegExp with another

Transform to NEA

Generate NEA graph for your RegExp at the right.

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

εu = uε = u	ε* = ε	u+v = v+u	u+u = u	(u*)* = u*	u(v+w) = uv+uw	(uv)*u = u(vu)*	(u+v)* = (u* + v*)*



Enter RegExp here

ab*

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

Minimized RegExp

ab*

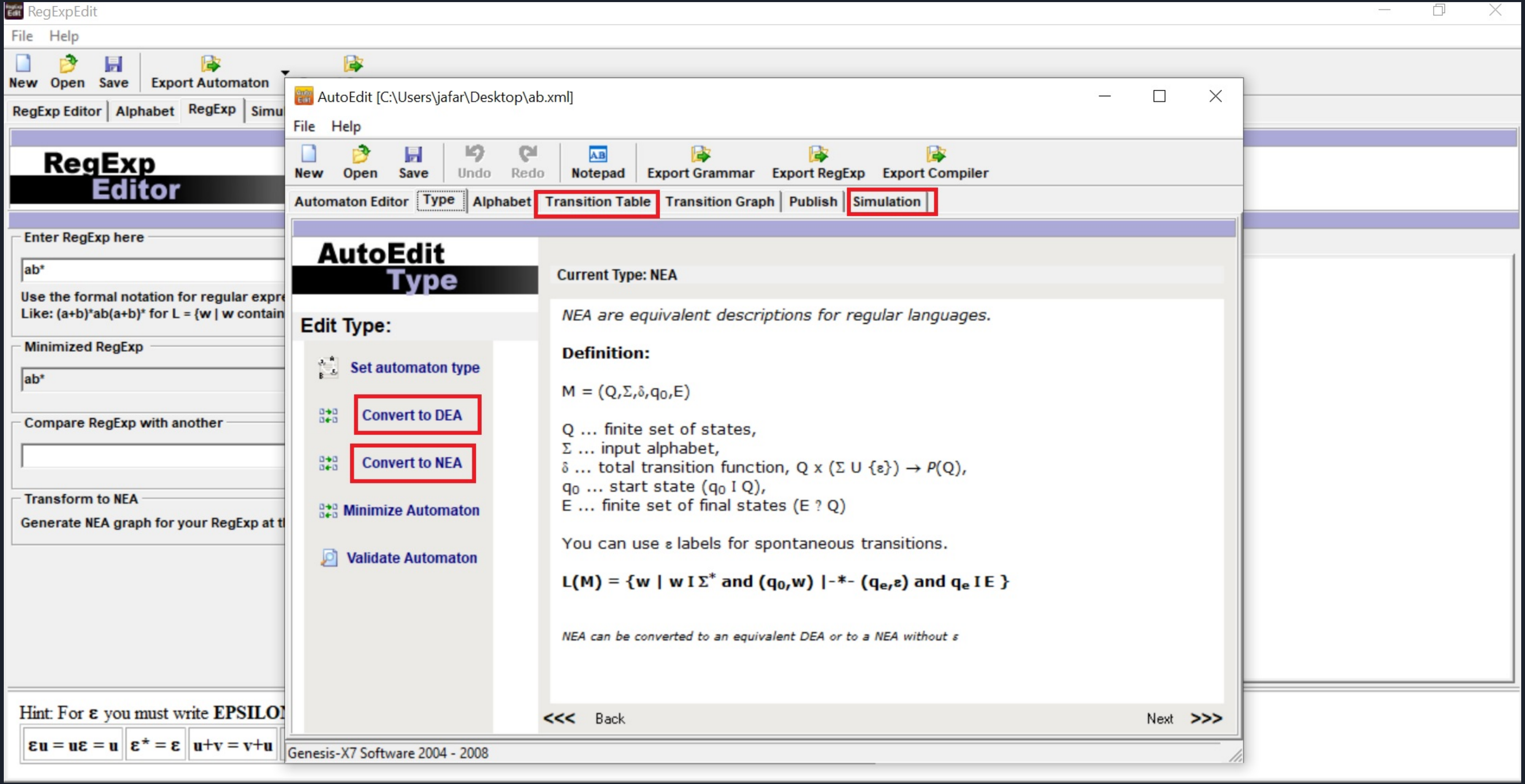
Compare RegExp with another

Transform to NEA

Generate NEA graph for your RegExp at the right.

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

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



Enter RegExp here

ab*

Use the formal notation for regular expre
Like: (a+b)*ab(a+b)* for L = {w | w contain

Minimized RegExp

ab*

Compare RegExp with another

Transform to NEA

Generate NEA graph for your RegExp at t

Hint: For ε you must write EPSILO

εu = uε = u | ε* = ε | u+v = v+u

Current Type: NEA

NEA are equivalent descriptions for regular languages.

Definition:

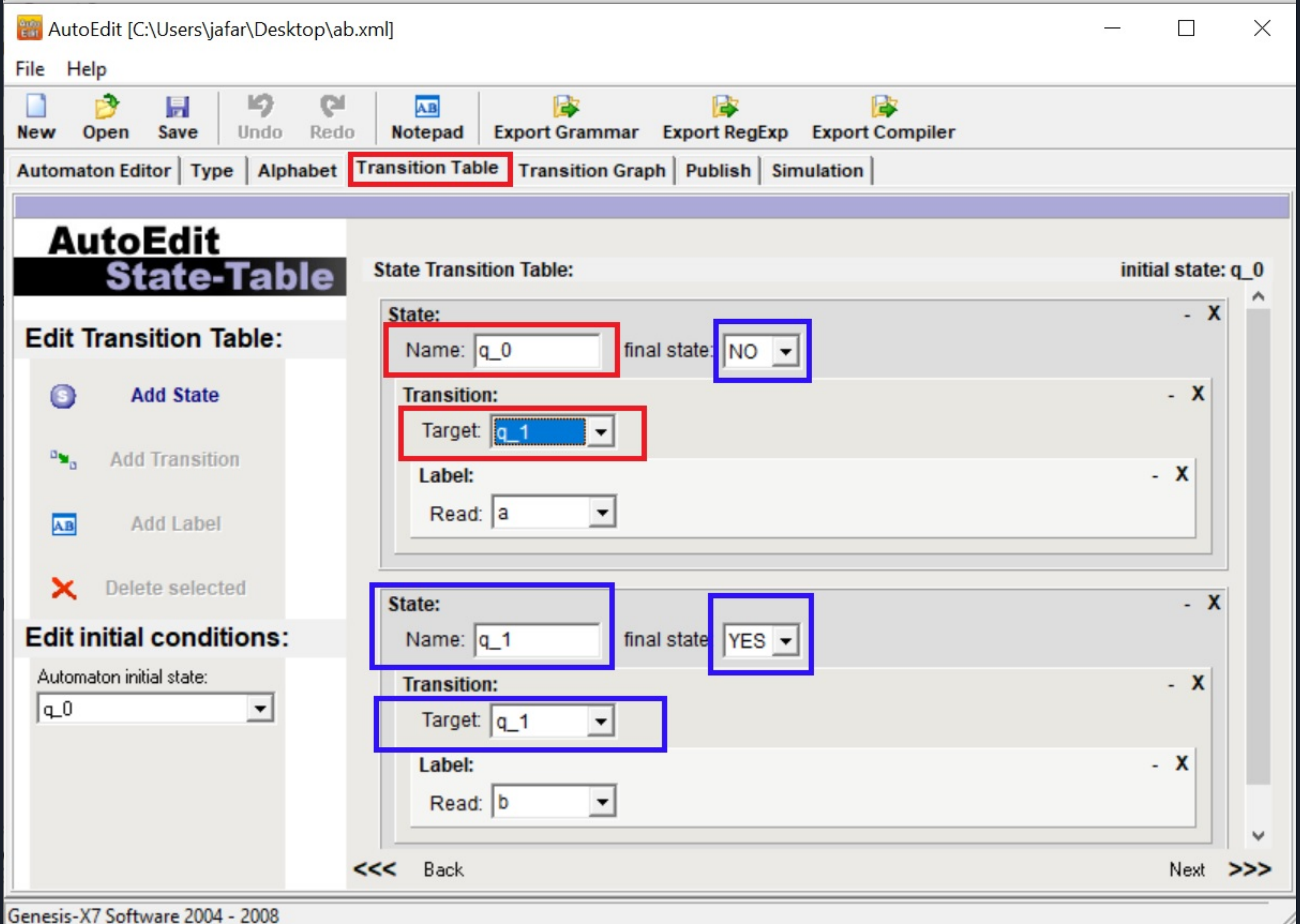
M = (Q,Σ,δ,q_0,E)

Q ... finite set of states,
Σ ... input alphabet,
δ ... total transition function, Q x (Σ U {ε}) → *P*(Q),
q_0 ... start state (q_0 I Q),
E ... finite set of final states (E ? Q)

You can use ε labels for spontaneous transitions.

L(M) = {w | w I Σ^* and (q_0,w) |-*- (q_e,ε) and q_e I E }

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

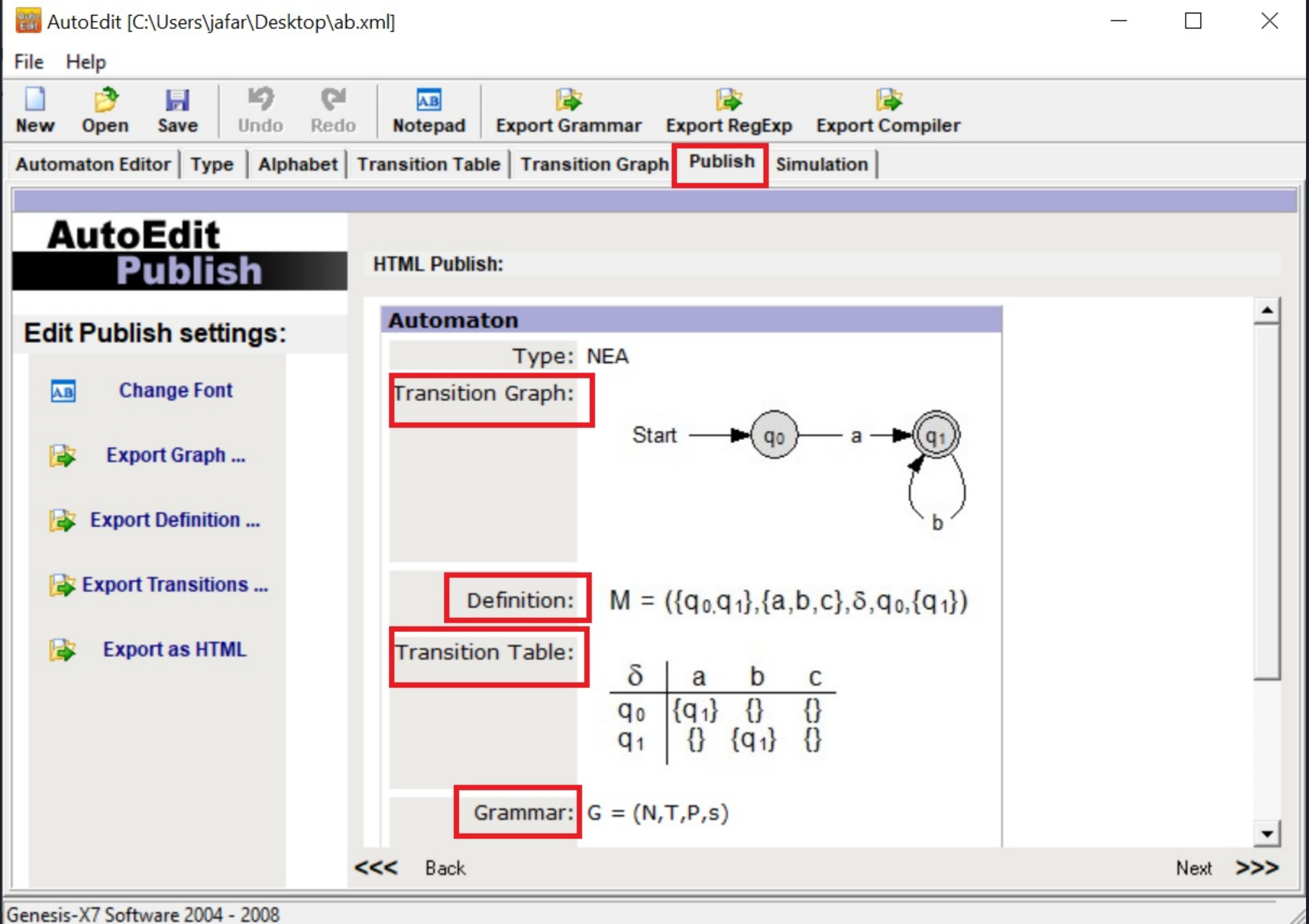


Automaton initial state: q_0

State Transition Table: initial state: q_0

State:
Name: q_0 final state: NO
Transition:
Target: q_1
Label:
Read: a

State:
Name: q_1 final state: YES
Transition:
Target: q_1
Label:
Read: b



Edit Publish settings:

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

HTML Publish:

Automaton

Type: NEA

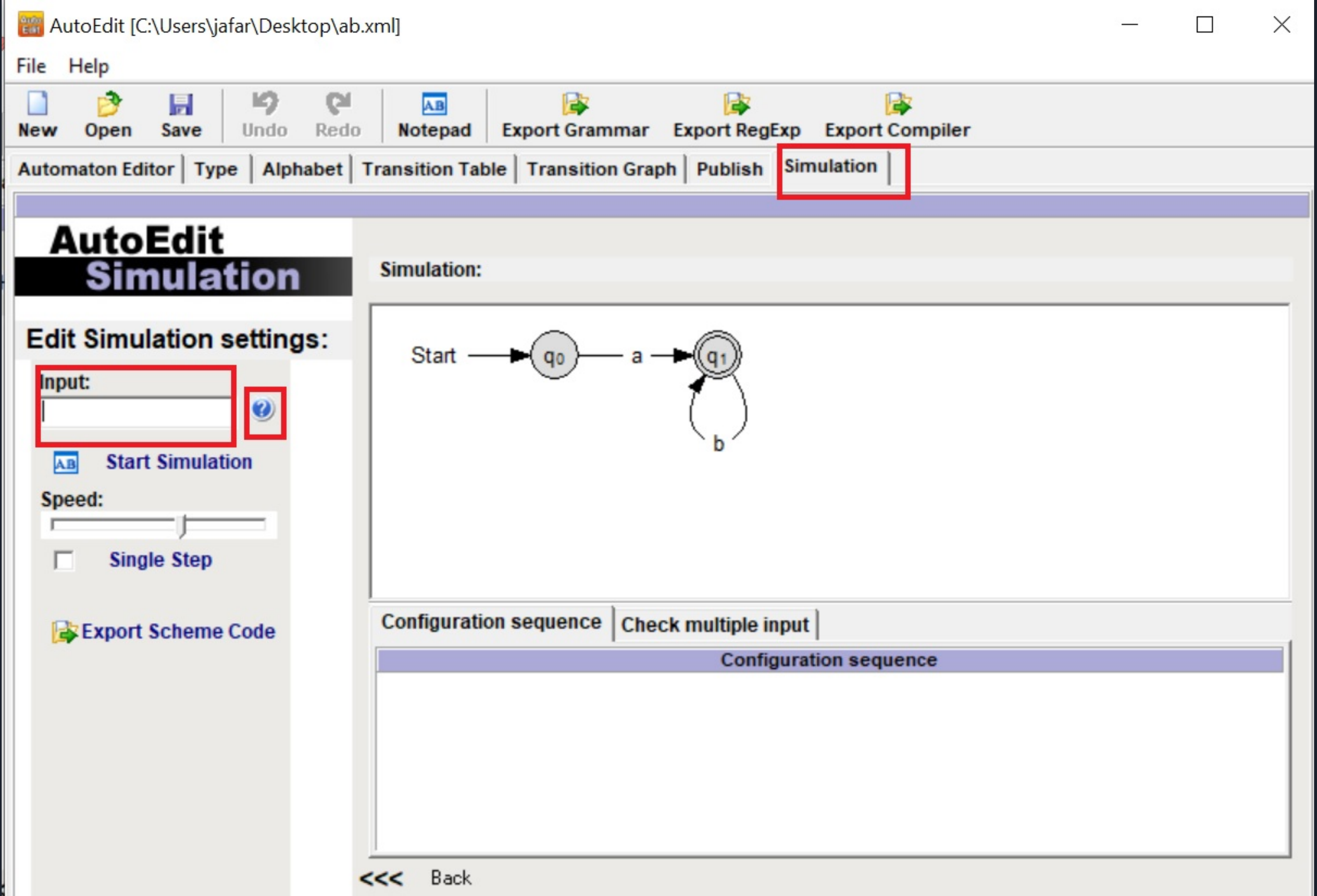
Transition Graph:

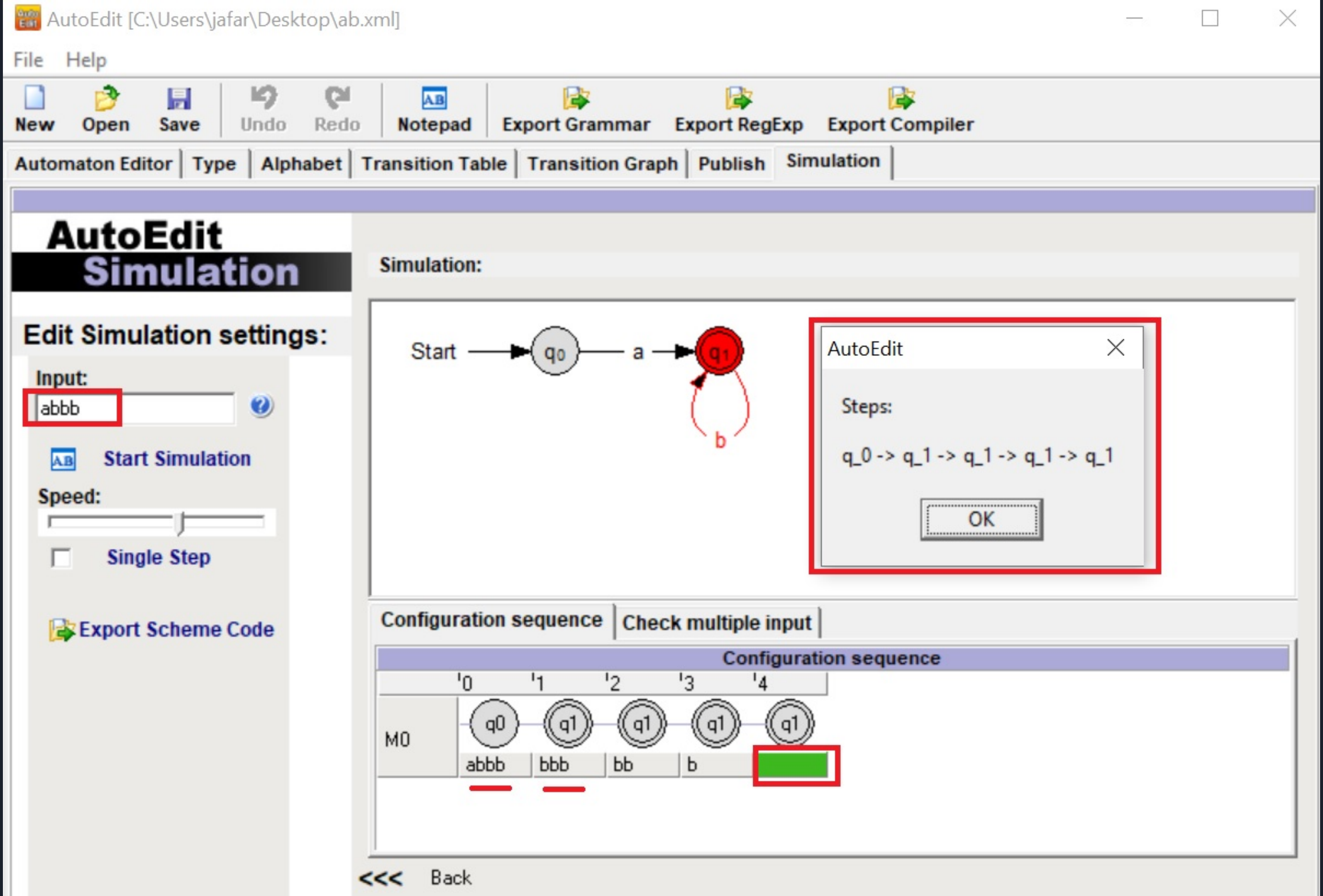
Definition: $M = (\{q_0, q_1\}, \{a,b,c\}, \delta, q_0, \{q_1\})$

Transition Table:

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

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


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
Simulation
Edit Simulation settings:
Input:
Start Simulation
Speed:
Single Step
Export Scheme Code
Simulation:
Start
q0
a
q1
b
Configuration sequence
Check multiple input
Configuration sequence
<<<
Back



Input: abbb

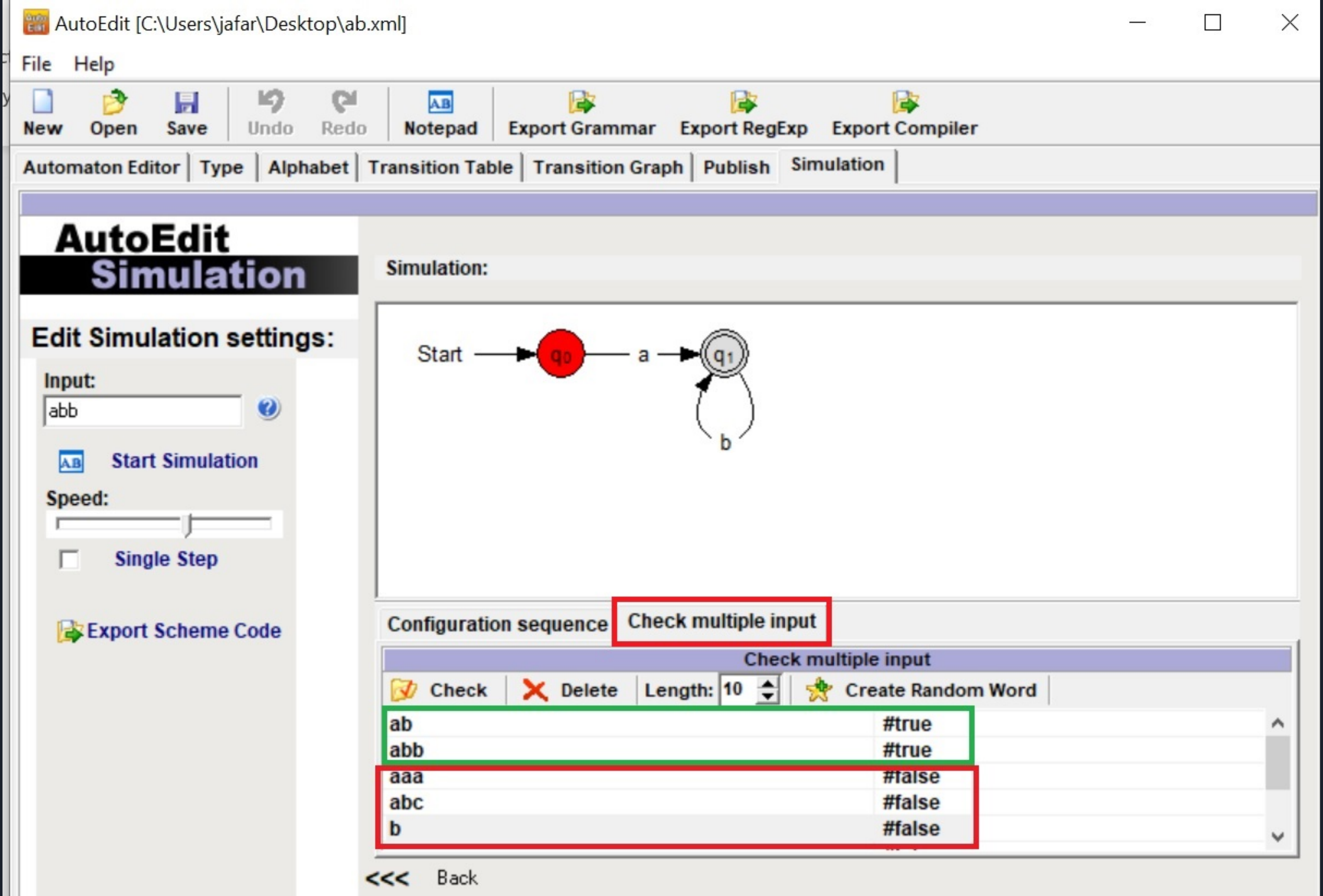
□ Single Step

Steps:

q_0 -> q_1 -> q_1 -> q_1 -> q_1

Configuration sequence

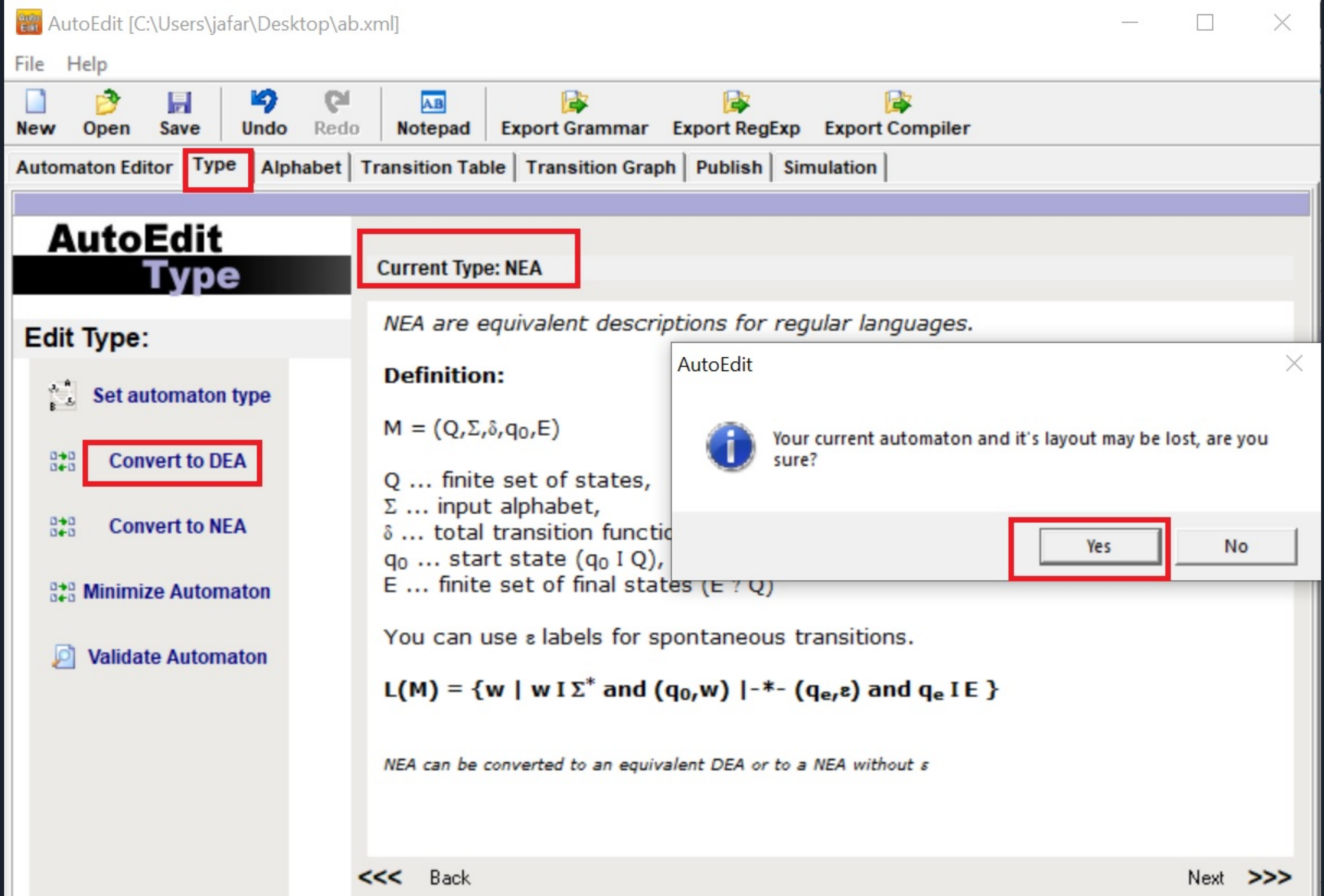
	'0	'1	'2	'3	'4
M0	q0	q1	q1	q1	q1
	abbb	bbb	bb	b	🟩



Input: abb

Length: 10

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



Current Type: NEA

NEA are equivalent descriptions for regular languages.

Definition:

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

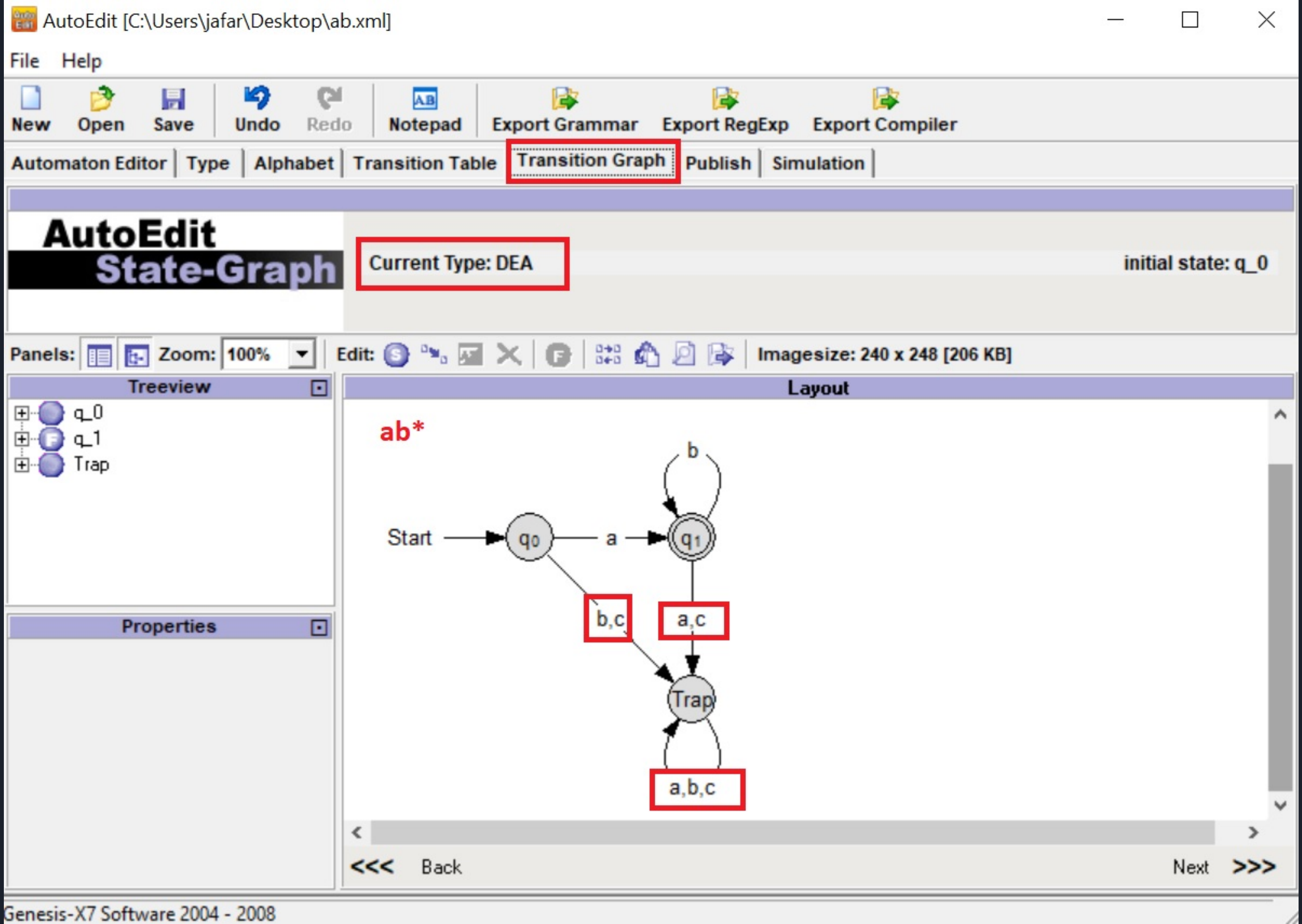
Q … finite set of states,
Σ … input alphabet,
δ … total transition functic
q_0 … start state (q_0 I Q),
E … finite set of final states (E ? Q)

You can use ε labels for spontaneous transitions.

$L(M) = \{w \mid w \text{ I } \Sigma^* \text{ and } (q_0,w) \text{ |-*- } (q_e,\varepsilon) \text{ and } q_e \text{ I } E \}$

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

Your current automaton and it's layout may be lost, are you sure?

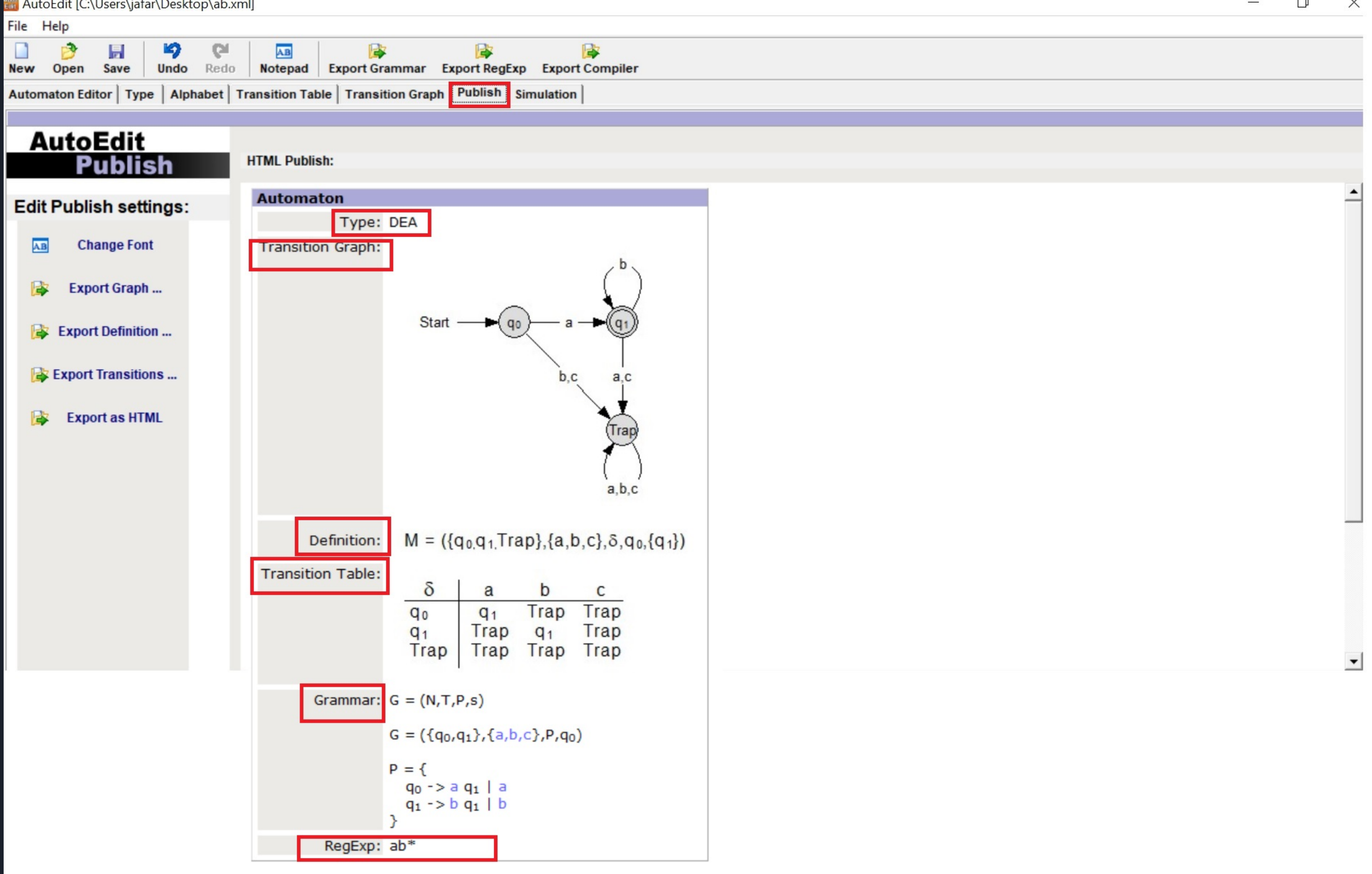


Current Type: DEA

initial state: q_0

Zoom: 100%

Imagesize: 240 x 248 [206 KB]



HTML Publish:

Automaton

Type: DEA

Transition Graph:

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

Transition Table:

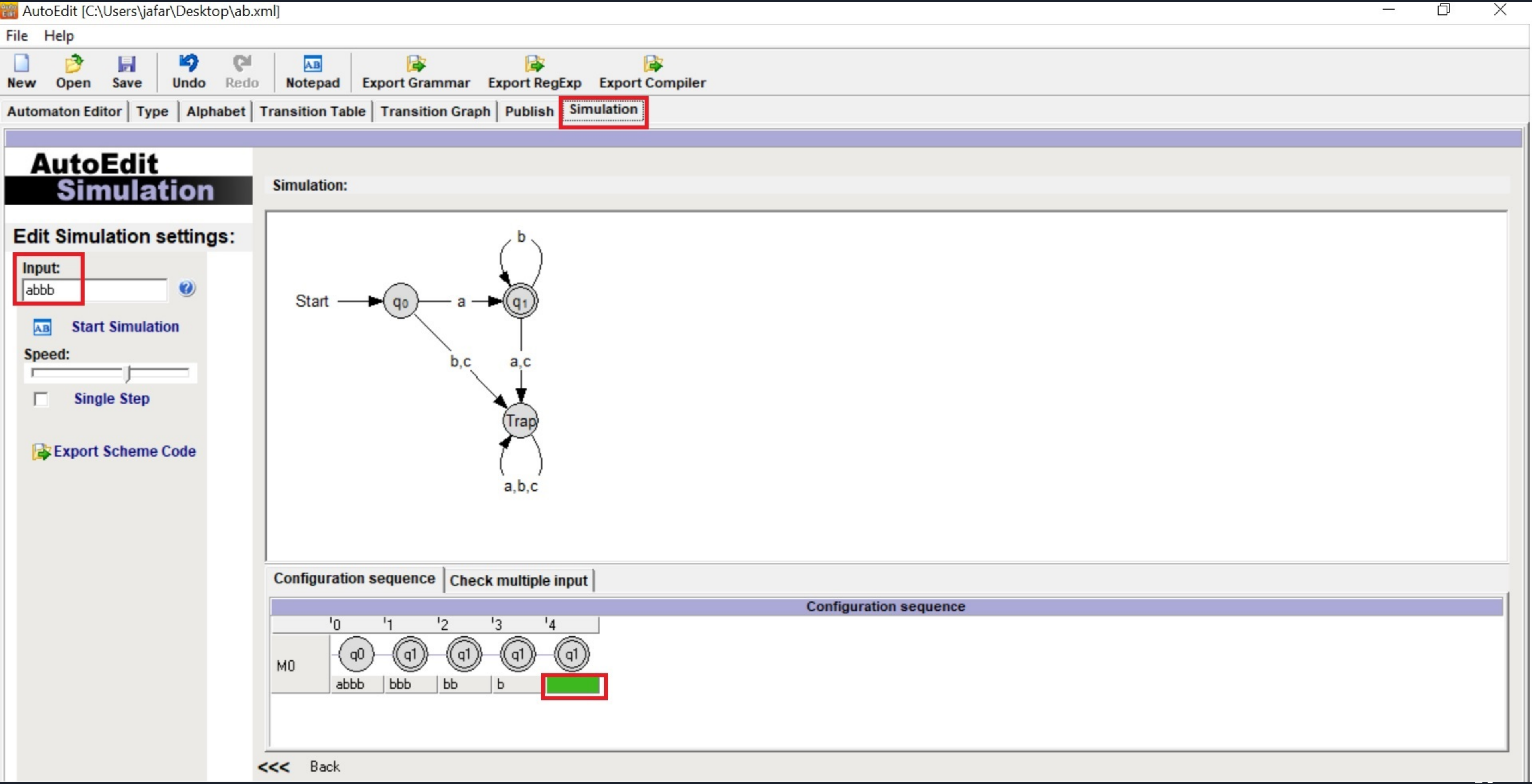
δ	a	b	c
q_0	q_1	Trap	Trap
q_1	Trap	q_1	Trap
Trap	Trap	Trap	Trap

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

$G = (\{q_0,q_1\},\{a,b,c\},P,q_0)$

```
P = {
   q0 -> a q1 | a
   q1 -> b q1 | b
}
```

RegExp: ab*



Edit Simulation settings:

Input: abbb

☐ Single Step

Configuration sequence

	0	1	2	3	4
M0	q0 abbb	q1 bbb	q1 bb	q1 b	q1 🟩

JFLAP?

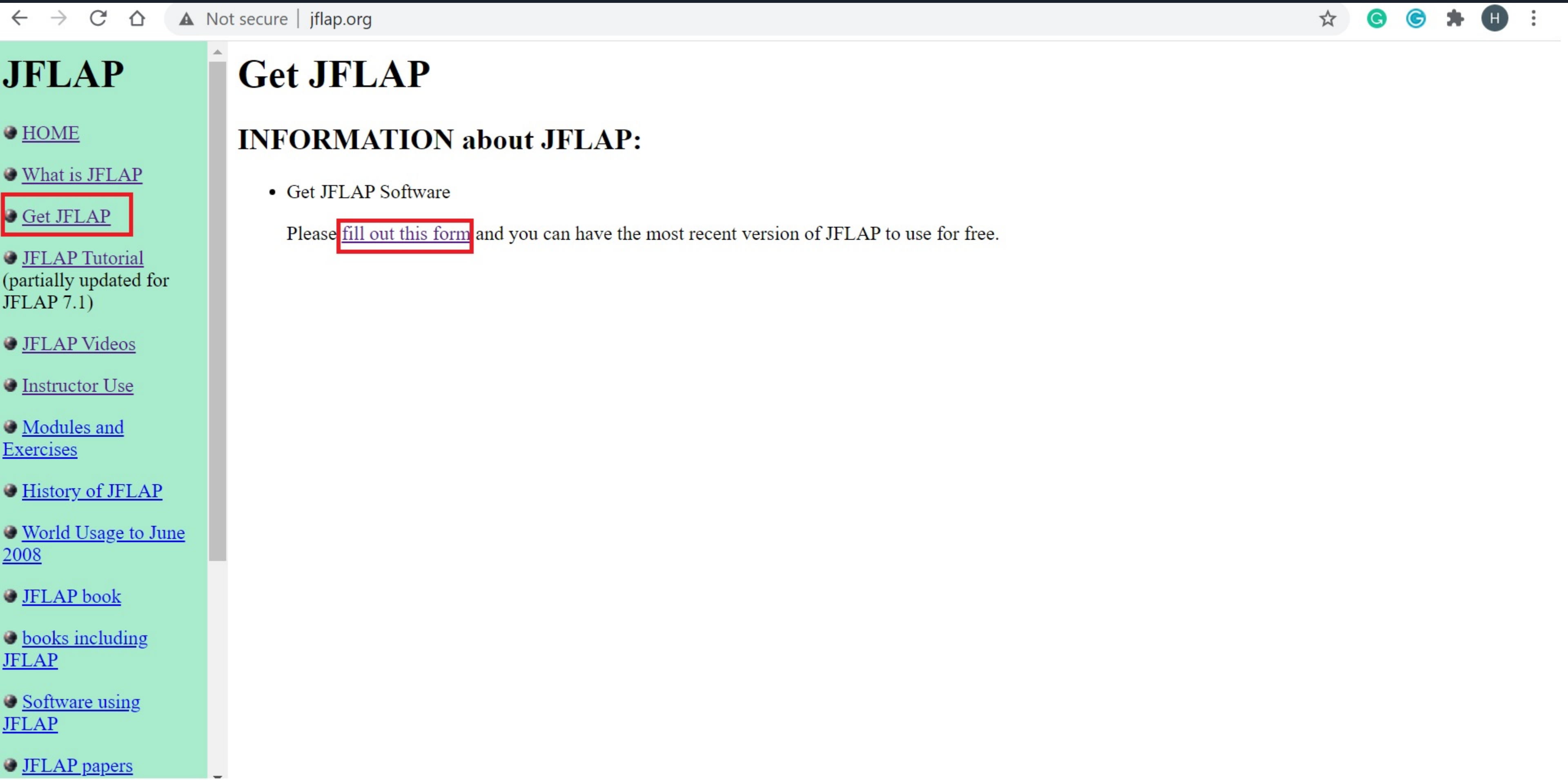
What Is JFLAP?

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

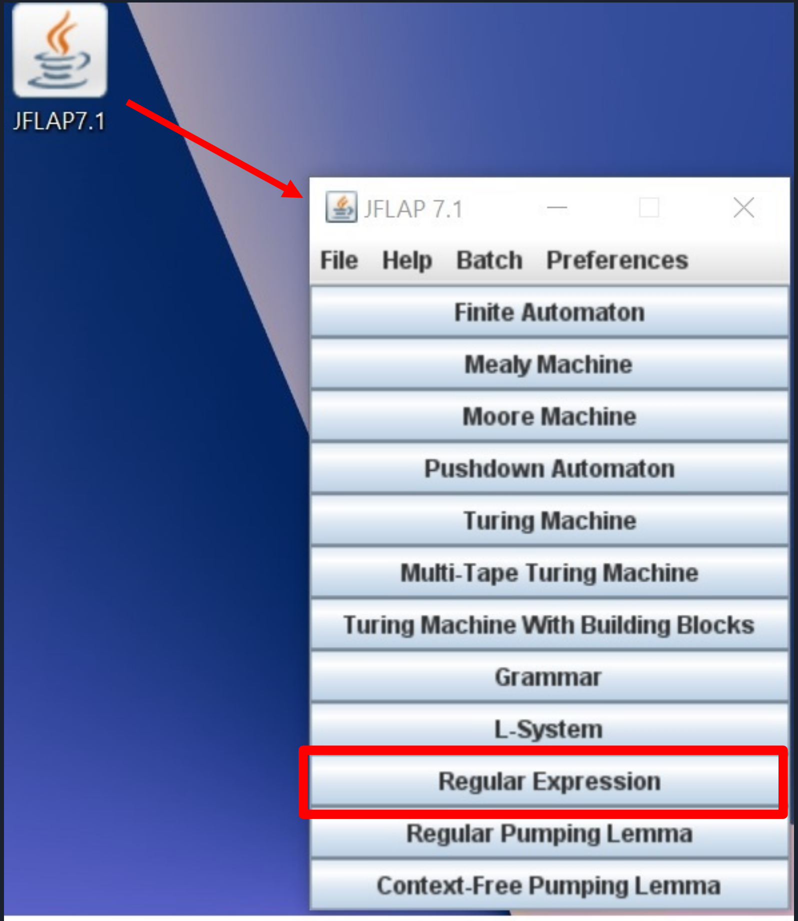
- HOME
- What is JFLAP
- Get JFLAP
- JFLAP Tutorial (partially updated for JFLAP 7.1)
- JFLAP Videos
- Instructor Use
- Modules and Exercises
- History of JFLAP
- World Usage to June 2008
- JFLAP book
- books including JFLAP
- Software using JFLAP
- JFLAP papers

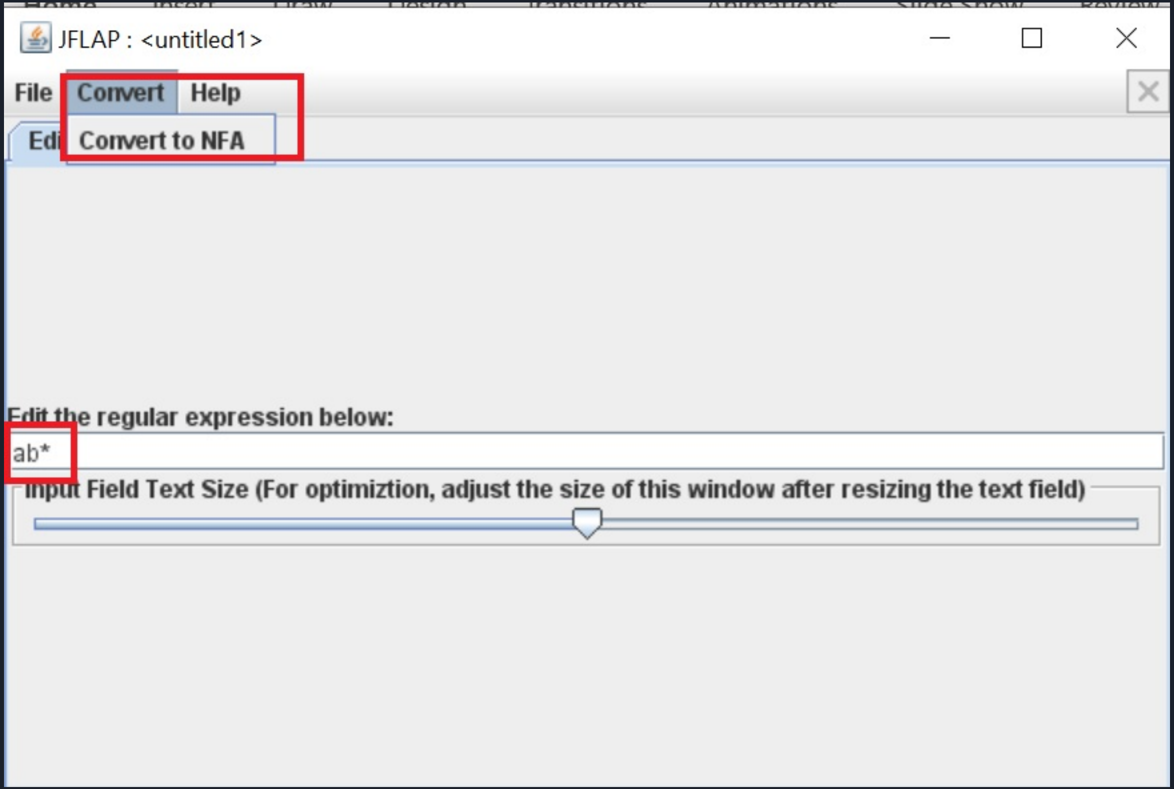
Get JFLAP

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.

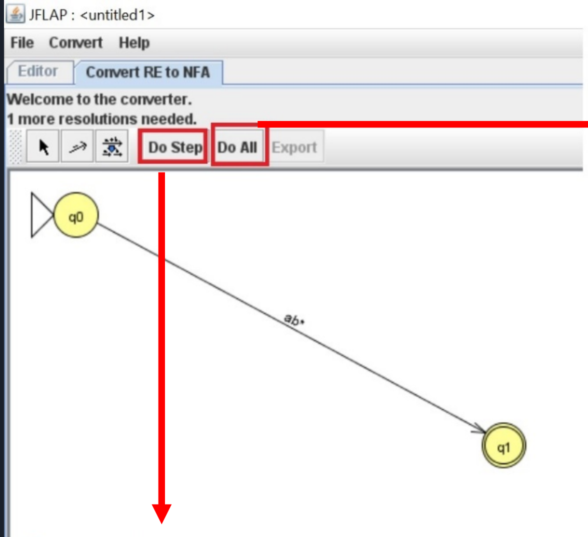

JFLAP7.1
JFLAP 7.1
File
Help
Batch
Preferences
Finite Automaton
Mealy Machine
Moore Machine
Pushdown Automaton
Turing Machine
Multi-Tape Turing Machine
Turing Machine With Building Blocks
Grammar
L-System
Regular Expression
Regular Pumping Lemma
Context-Free Pumping Lemma



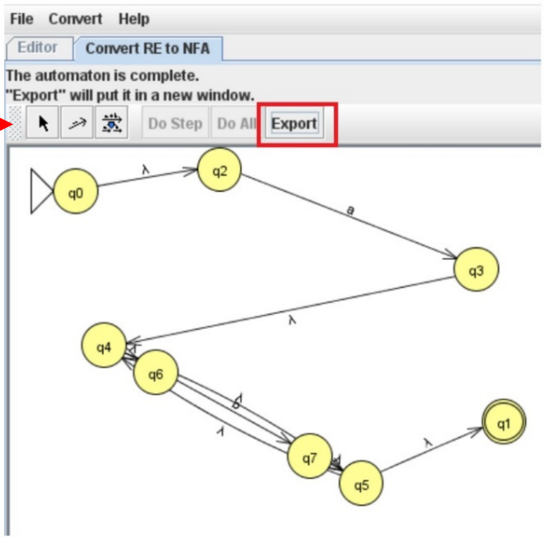
Edit the regular expression below:

ab*

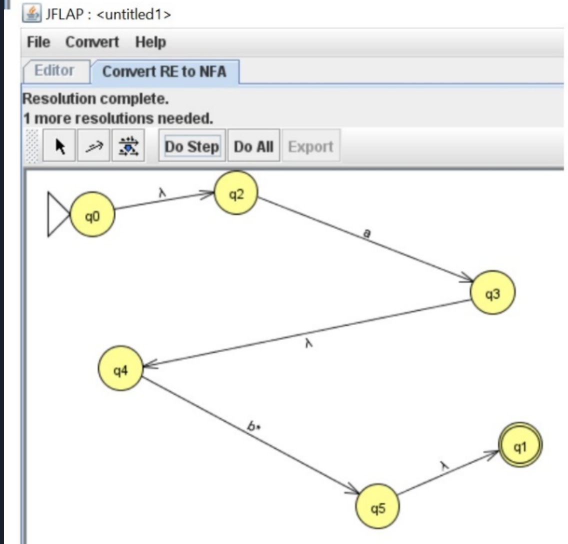
Input Field Text Size (For optimiztion, adjust the size of this window after resizing the text field)



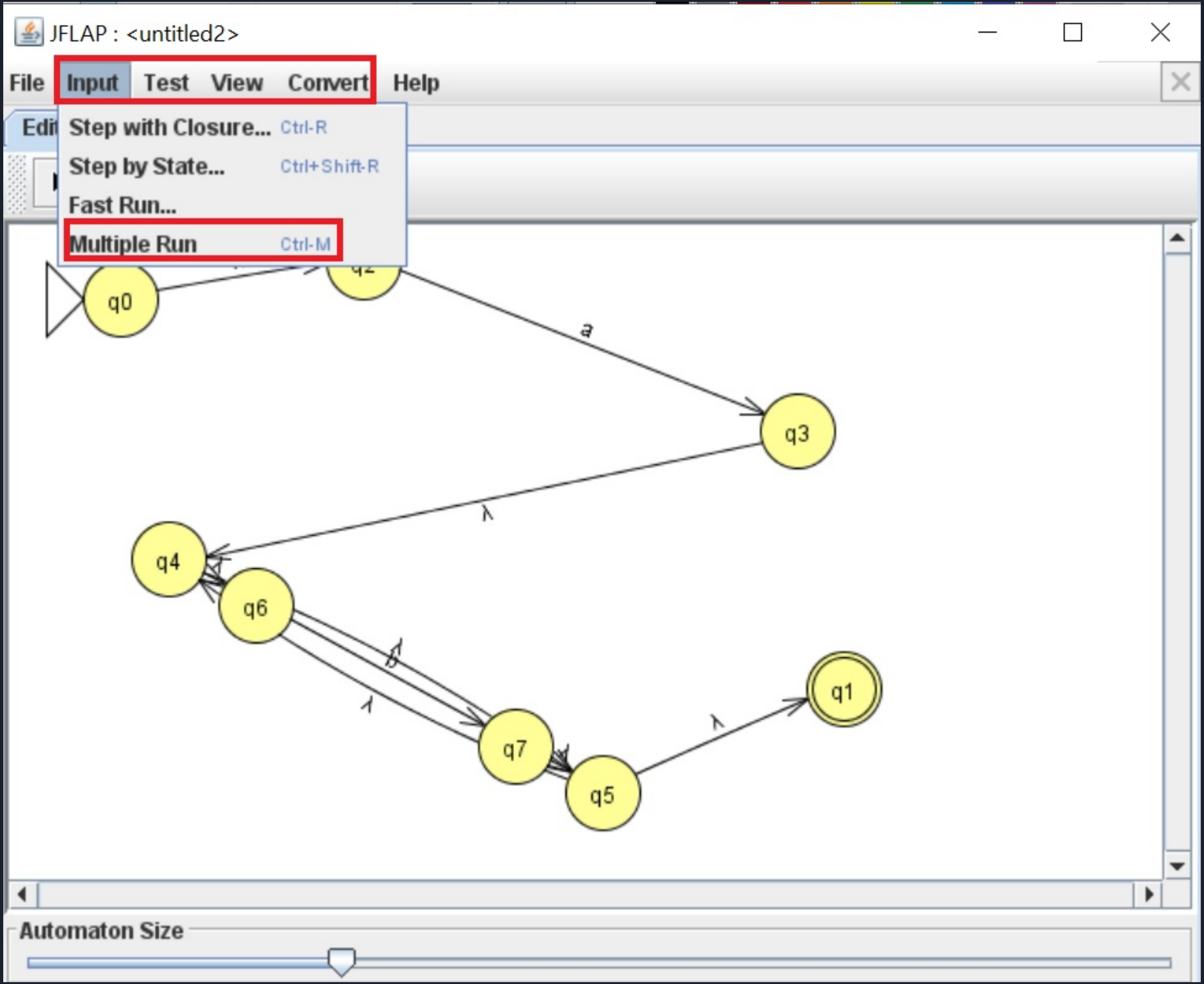
Welcome to the converter.
1 more resolutions needed.

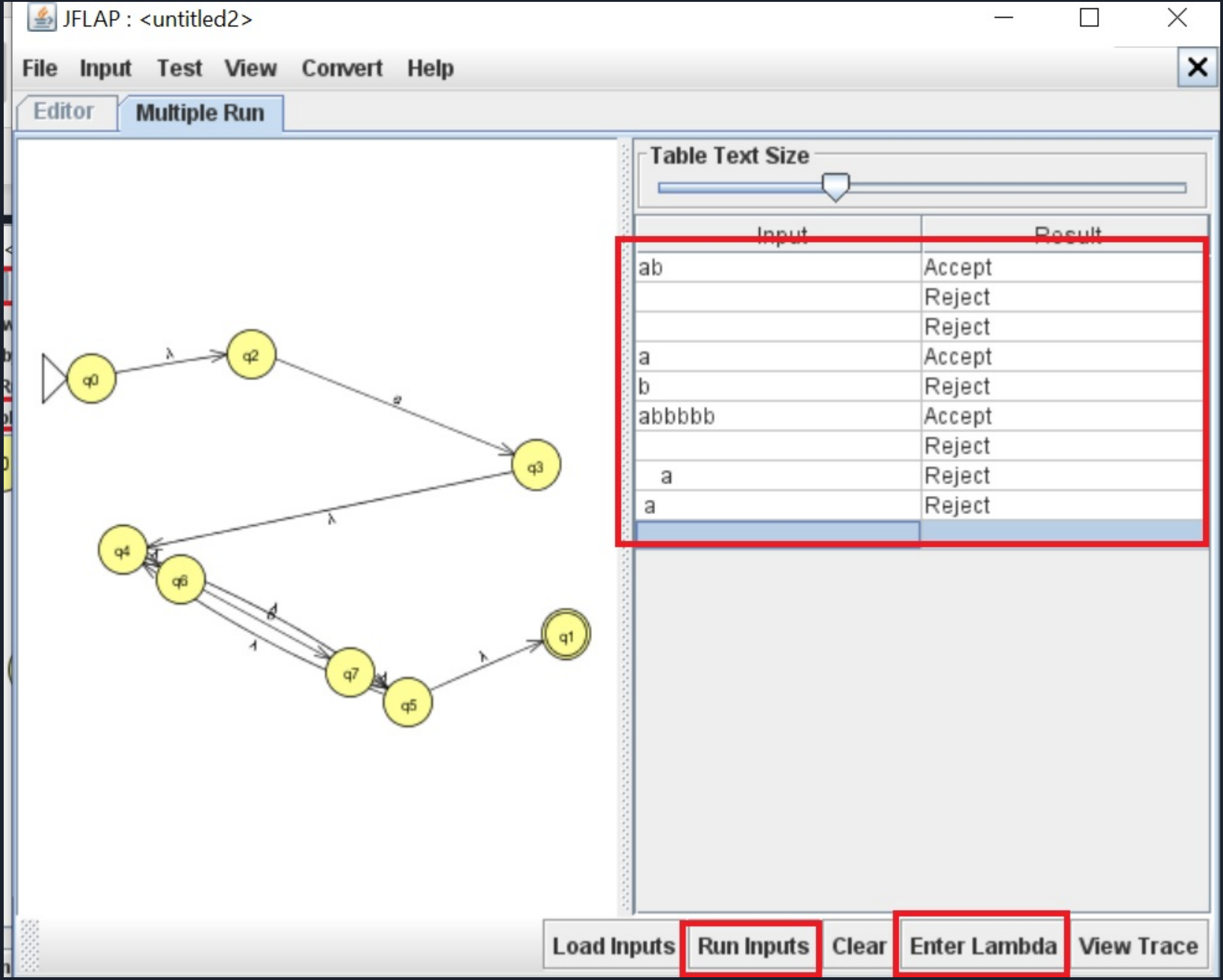


The automaton is complete.
"Export" will put it in a new window.

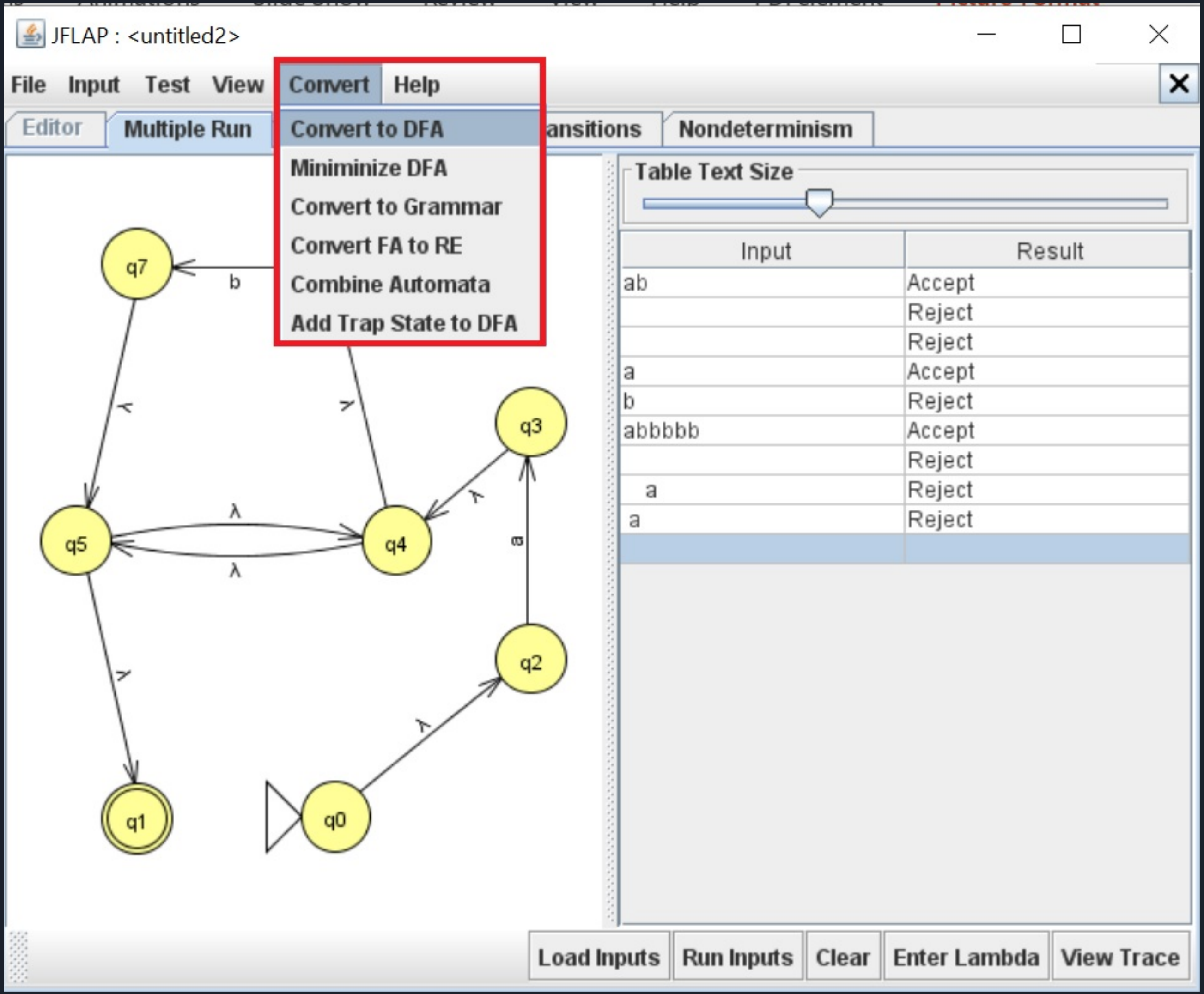


Resolution complete.
1 more resolutions needed.

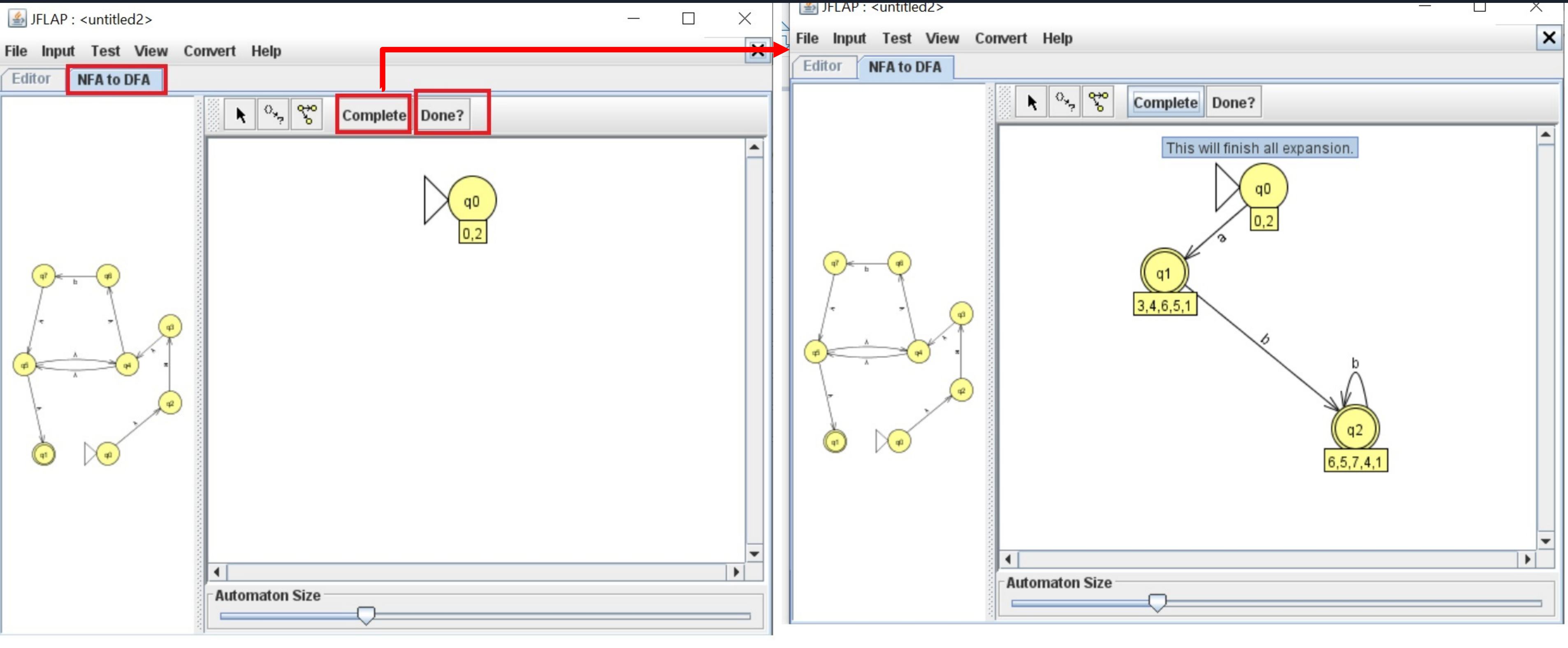

JFLAP : <untitled2>
File
Input
Test
View
Convert
Help
Edit
Step with Closure... Ctrl-R
Step by State... Ctrl+Shift-R
Fast Run...
Multiple Run Ctrl-M
q0
q2
a
q3
λ
q4
q6
λ
λ
q7
q5
λ
q1
Automaton Size



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



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



This will finish all expansion.

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!