Regular Expressions and Converting an RE to a DFA$_{JP}$

- Prerequisite knowledge:
  Deterministic Finite Automata
  Nondeterministic Finite Automata
  Conversion of NFA to DFA
  Regular Languages
  Set Theory
  JFLAP Tutorial

Description of Regular Expressions

Regular expressions provide a relatively compact representation for *regular languages.*

Definition of Regular Expressions

Regular expressions are made up of sets of strings and operations over those sets. Formally, regular expressions are defined recursively as follows.

Given a finite alphabet Σ, the following are defined as regular expressions:

- the “empty set” { } is a set containing no strings, denoted by □
- the “empty string” { ε } is a set containing only the empty string, denoted by ε
- “literal characters” { α } a set for every α □ Σ, denoted by α

Given regular expressions R and S, the following operations over them produce regular expressions:

- “union” is the set union of R and S, denoted by R | S
- “concatenation” includes sequences of two regular expressions, denoted by RS
- *“*Kleene star” is a sequence of zero or more instances of a regular expression, denoted by R* and by S*

JFLAP Notation

JFLAP uses a slightly different notation. When entering regular expressions, the empty string is represented by the character “!” which is commonly known as “exclamation mark”, “exclamation point”, or “bang”. The union operator is represented by the character “+” which is commonly known as “plus sign” or “plus”.

Examples of Regular Operators

Union: R | S denotes the set union of sets described by R and S. For example, if R = {"a", "c"} and S = {"b", "d"}, then R | S = {"a", "c", "b", "d"}.

Concatenation: RS denotes the set of strings obtained by concatenating a string in R and a string in S. For example, if R = {"a", "c"} and S = {"b", "d"}, then RS = {"ab", "ad", "cb", "cd"}.

Kleene star: R* denotes the set that contains ε and all strings formed by concatenating any finite number of strings from R. For example, if R = {"a", "c"}, then R* = {ε, "a", "c", "aa", "ac", "ca", "cc", "aaa", "aac", "aca", "acc", "caa", ... }.

Precedence

Parentheses have highest precedence overall. Of the RE operators, Kleene star has highest precedence, then concatenation, and finally union.

Example: RE for a Regular Language

Consider the regular language L over the alphabet Σ = {a, b, c} comprised of (1) all strings that begin with "a" followed by an arbitrary length sequence made up of of "a" and "b" symbols and (2) the string "c".

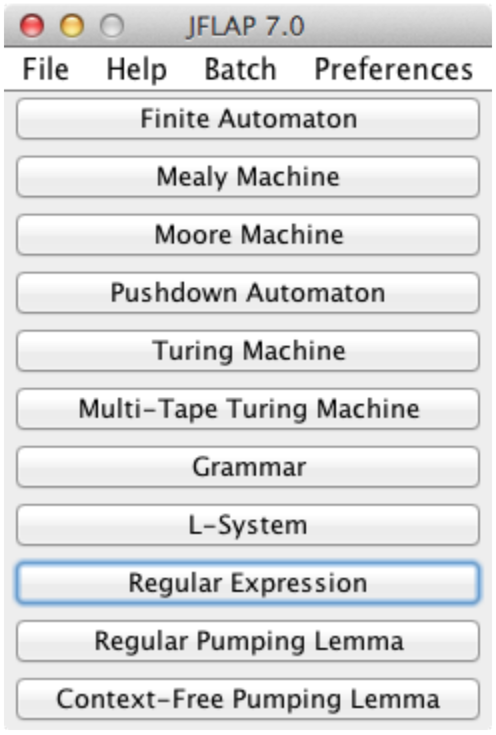
Since L is being described as the union of (1) and (2), let's first look at defining each of those components of L, then apply the union operator.

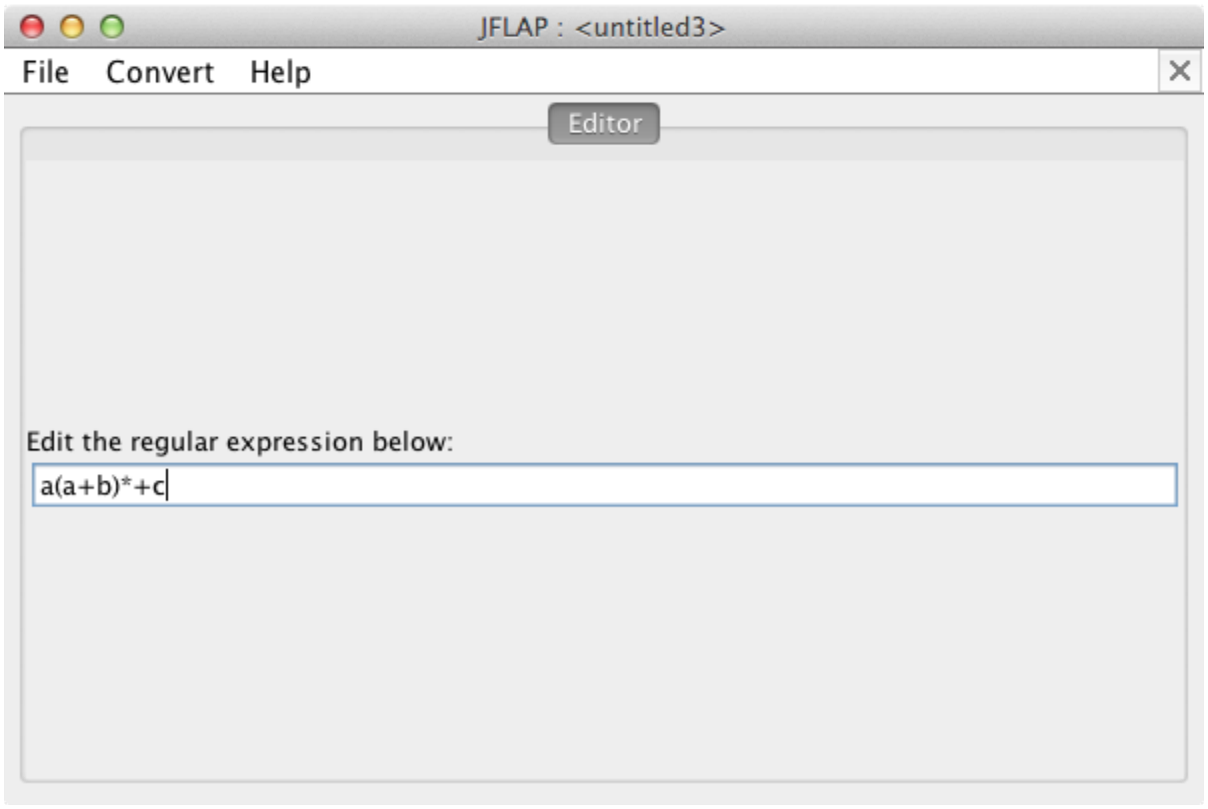
Strings beginning with "a" that is followed by another string can be represented as the concatenation of "a" with another string. The concatenation operator is simply ordered contiguous placement. The regular expression could thus begin with "a" followed by the set of arbitrary length strings made up of "a" and "b" symbols. The latter may be defined as the Kleene star of the union of "a" and "b", whose operator symbol is the vertical bar, and written as: **(a|b)***. Thus this first set of strings may be represented by the regular expression **a(a|b)***

Creating the union of this set with the set consisting of the string "**c**" only, we can produce the regular expression: **a(a|b)*|c**

Note that operator precedence renders a unique interpretation of this regular expression.

Entering this into JFLAP is simply a matter of typing the sequence of characters into the Regular Expression editor. (See: `regex_abc.jff`) Recall that JFLAP uses + as its union operator and ! to represent the empty string.





Edit the regular expression below:

a(a+b)*+c

Examples of strings in this language include: “a” “c” “ab” “aa” “aab” “aba” “abb” “aaaba”

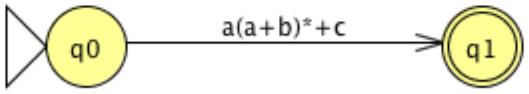
Examples of strings not in this language include: ε, "ac", “ca”, “b”, “ba”, “abc”, “bac”, “baa”

At present, JFLAP does not provide a means to directly assess which strings are and are not in the language represented by a specific RE. However, JFLAP does support converting a RE into an NFA which can then be used to test acceptance and rejection of specific strings.

Example: Converting an RE to a NFA by Inspection

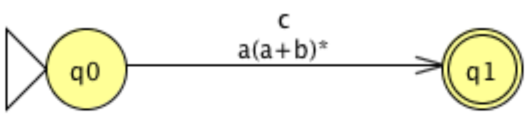
Again consider the regular language L over the alphabet Σ = {a, b, c} comprised of the string “c” and all strings that begin with “a” followed by an arbitrary length sequence made up of of “a” and “b” symbols, which we just saw could be represented by the RE **a(a|b)*|c**. Here is an example of converting that RE to an equivalent NFA using an informal approach. Note that this example was done using the Finite Automaton feature of JFLAP rather than the Regular Expression feature. As such, JFLAP did not interpret the transition labels as regular expressions!

1. Start by creating a two-state machine with transition labeled using the RE.

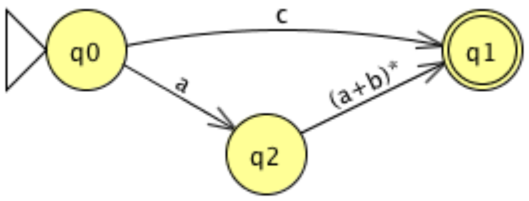


Recall that this is for illustrative purposes only and that JFLAP will not interpret the label on the transition as a regular expression.

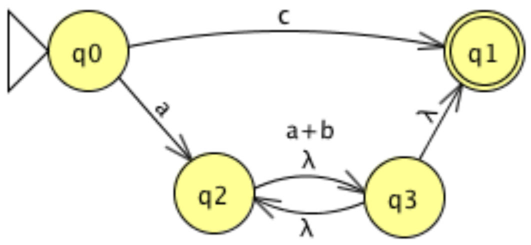
2. Noticing that this is the union of a(a+b)* and c, which means either can be chosen, split the transition into two parallel transitions representing that union.



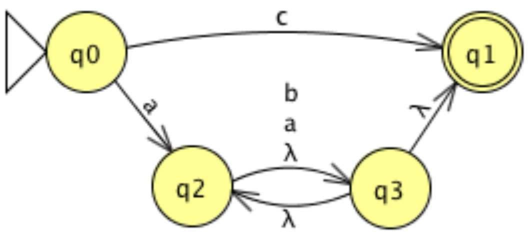
3. Since a(a+b)* represents the concatenation of a with (a+b)*, we can implement that transition by a sequence.



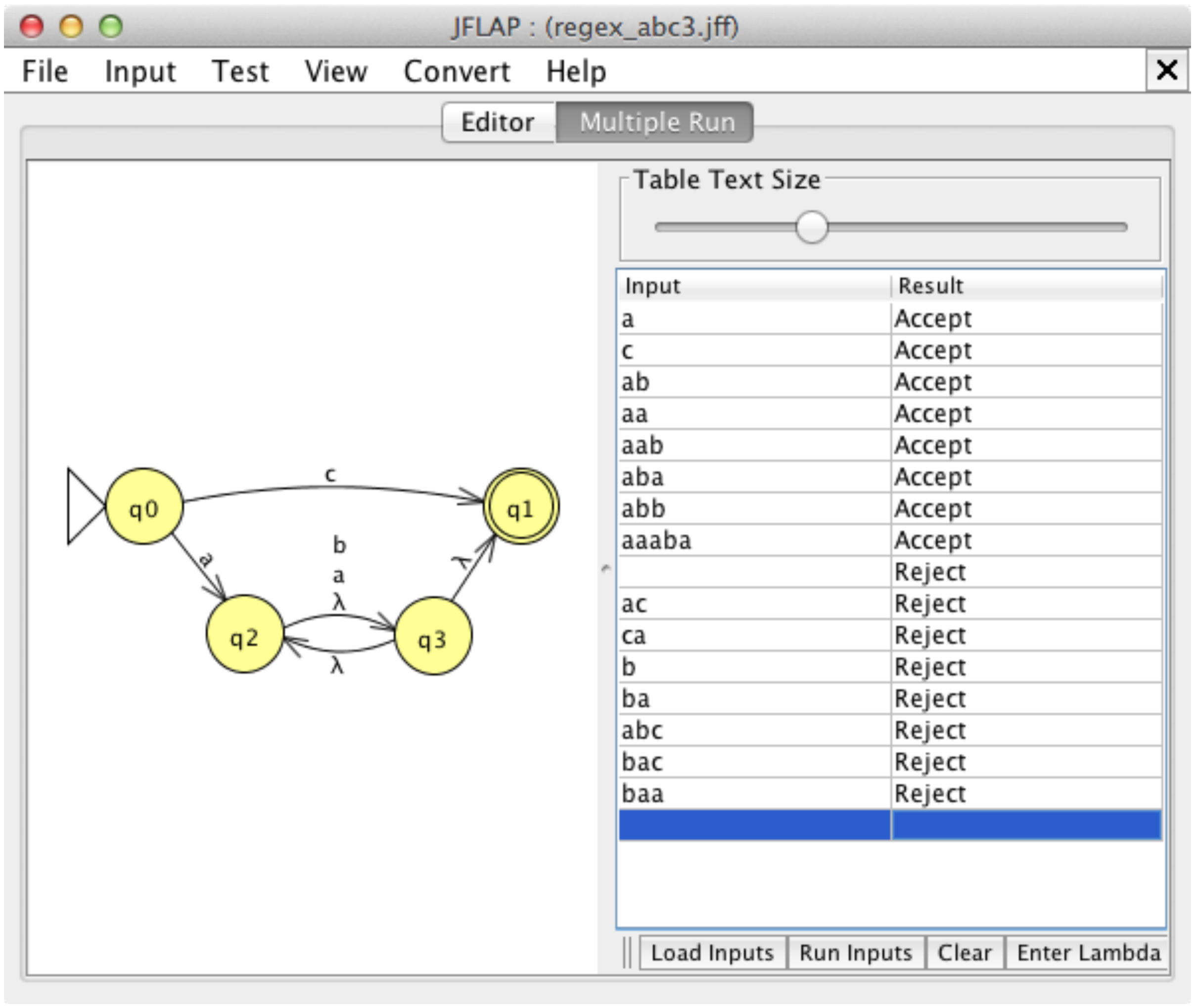
4. The Kleene star indicates zero or more repetitions of (a+b), which we can show using ε-transitions.



5. The remaining union operation, a+b, can again be decomposed into parallel transitions.

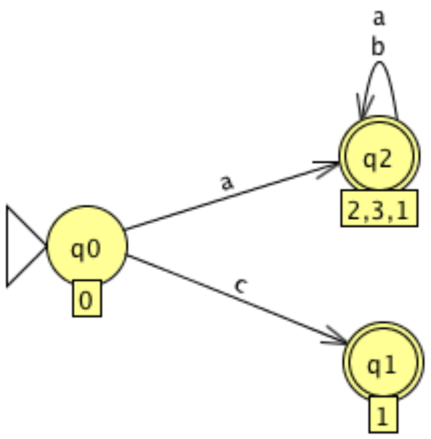


6. Since all transitions are valid, using only symbols from Σ*, this is a valid NFA and can be run to test acceptance and rejection of candidate strings.



Input	Result
a	Accept
c	Accept
ab	Accept
aa	Accept
aab	Accept
aba	Accept
abb	Accept
aaaba	Accept
	Reject
ac	Reject
ca	Reject
b	Reject
ba	Reject
abc	Reject
bac	Reject
baa	Reject

7. We can now also apply the known NFA to DFA conversion process.



For simple cases, such as this example, it may be feasible to proceed intuitively from RE to NFA. In general, however, it may not be obvious how best to proceed and since there are an infinite number of equivalent NFA, this heuristic approach is not guaranteed to halt with a solution. Fortunately, there is a formal process that guarantees finding an equivalent NFA and DFA for any given RE.

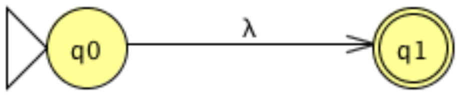
Description of Formal RE DFA Conversion Process

For any regular language L there exists one or more regular expressions (RE) as well as one or more deterministic finite automata (DFA) that represent L. The following algorithm facilitates the conversion from a RE for language L to a DFA for language L by first converting the RE to a nondeterministic finite automaton (NDA) then using converting the NFA into an equivalent DFA (as described in unit *"Explain algorithm and Convert NFA to DFA"*).

The conversion of a regular expression to a nondeterministic finite automaton proceeds by considering each case in the definition provided previously.

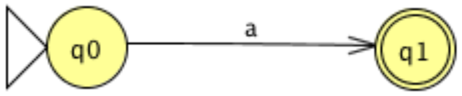
- The language comprised of the empty string ε is represented by a two-state NFA with transition for ε.

ε

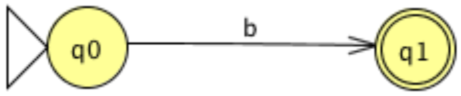


- The language comprised of a literal character α is represented by a two-state NFA with transition for α.

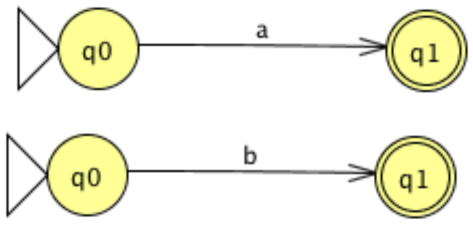
a



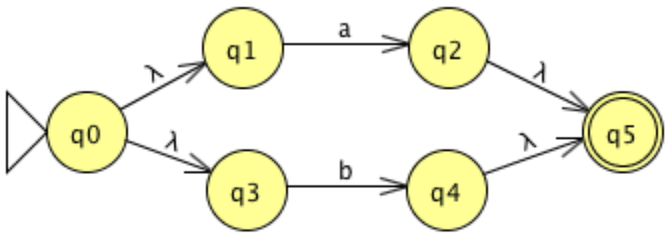
b



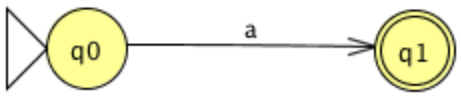
- The language comprised of the union of two languages, R and S, is represented by an NFA whose initial state is attached by ε-transitions to the initial states of R and S, and whose accepting state is attached by ε-transitions from the accepting states of R and S.

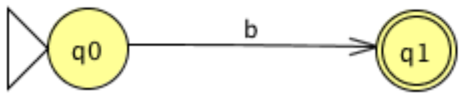


a|b

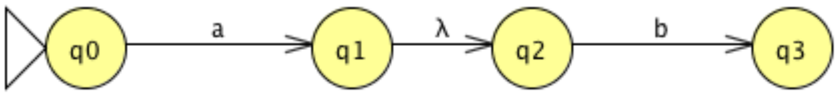


- The language comprised of a concatenation (sequence) of two languages, R and S, is an NFA with the accepting state of R attached to the initial state of S by an ε-transition.

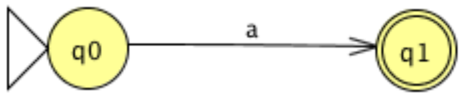




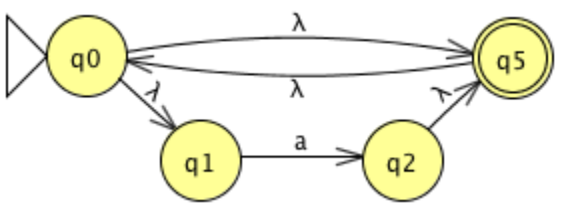
ab



- The language comprised of the Kleene star of a language R is an NFA with an ε-transition from its initial state to the initial state of R, an ε-transition to its accepting state from the accepting state of R, and ε-transitions between its initial and accepting states.

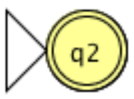


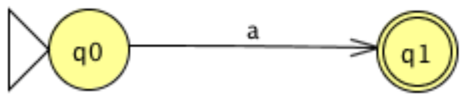
a*



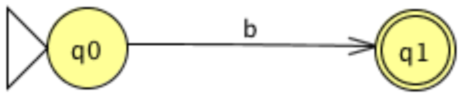
Since each of these results in a valid NFA, each can be subjected to the NFA to DFA conversion algorithm described in unit *"Explain algorithm and Convert NFA to DFA"*. For example, here are equivalent DFA for the previously developed NFA.

ε

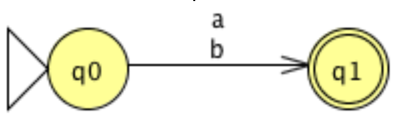




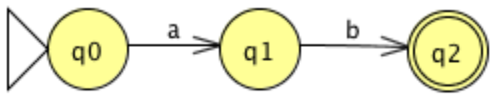
b



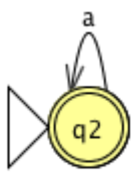
a|b



ab



a*

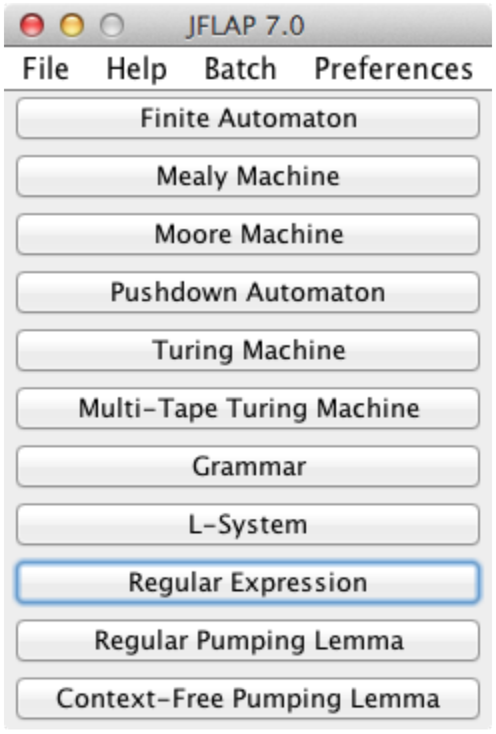


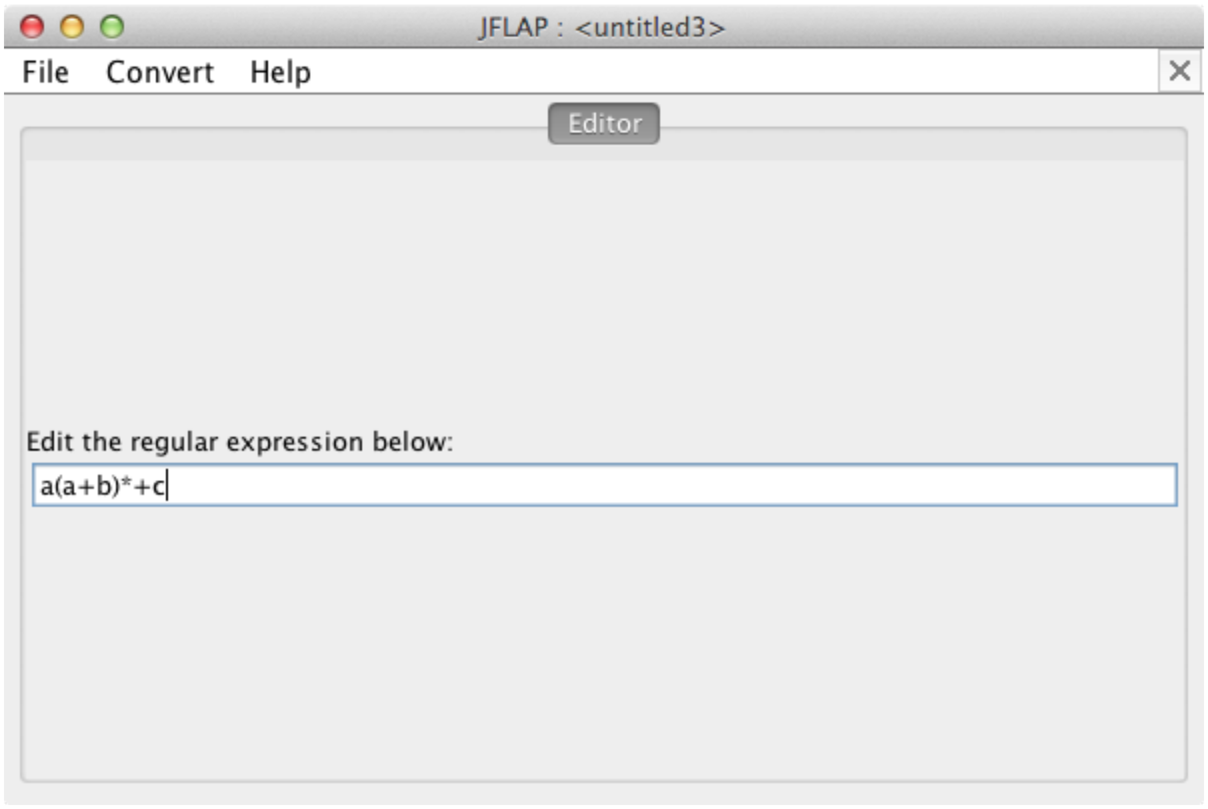
Example: Conversion from RE to DFA (RE NFA DFA)

Consider the RE developed in the previous example: **a(a|b)*|c**

1. Specifying the RE

First, enter the RE into the Regular Expression Editor in JFLAP using “+” for the union operator. (See: `regex_abc.jff`)



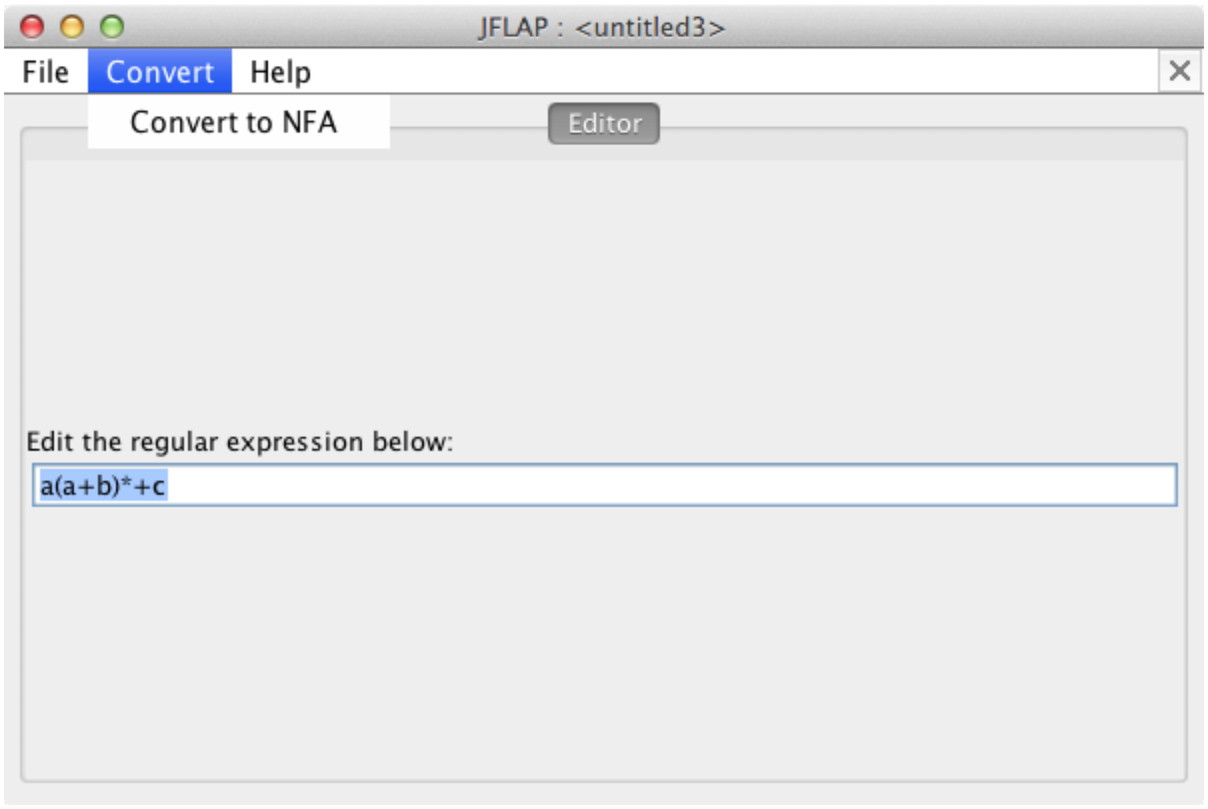


Edit the regular expression below:

a(a+b)*+c

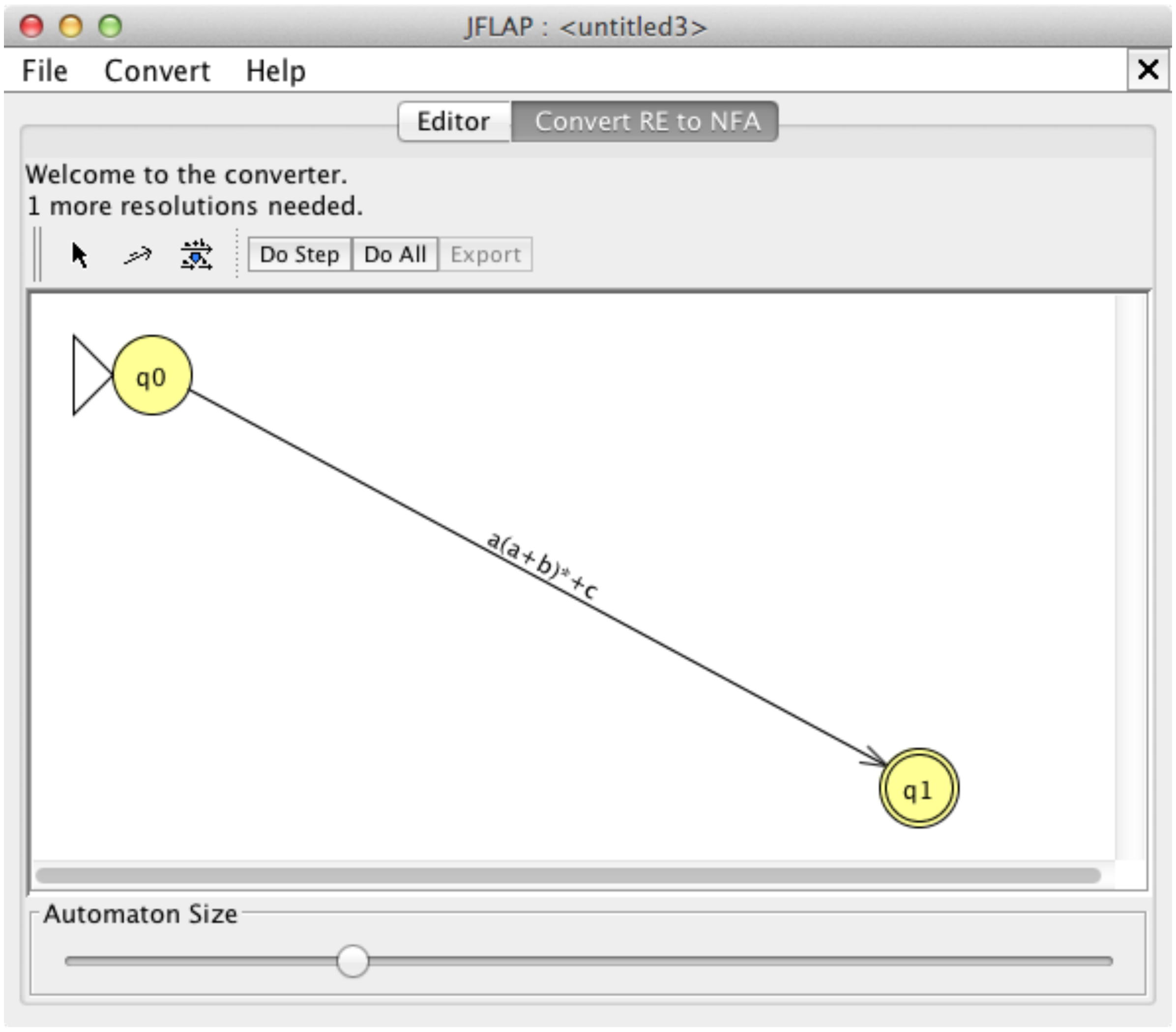
2. Converting RE to NFA

We begin the RE to NFA conversion process by creating a state diagram with an initial state, an accepting state, and a transition labeled with the specified RE. The JFLAP *Convert:Convert to NFA* menu item produces such a diagram.



Edit the regular expression below:

a(a+b)*+c



Welcome to the converter.
1 more resolutions needed.

Note that this diagram does not represent an executable automaton within JFLAP and serves only as an intermediary representation. Thus, for example, you cannot run this machine or save this representation. Once conversion is complete, you will be able to export and use the resulting NFA (see: `regex_abc_nfa.jff`).

There are an infinite number of NFA for a given regular language. JFLAP restricts the space by adhering to a specific set of conversion steps. You can always choose to develop an equivalent FA in a separate JFLAP window to compare with JFLAP's conversion results.

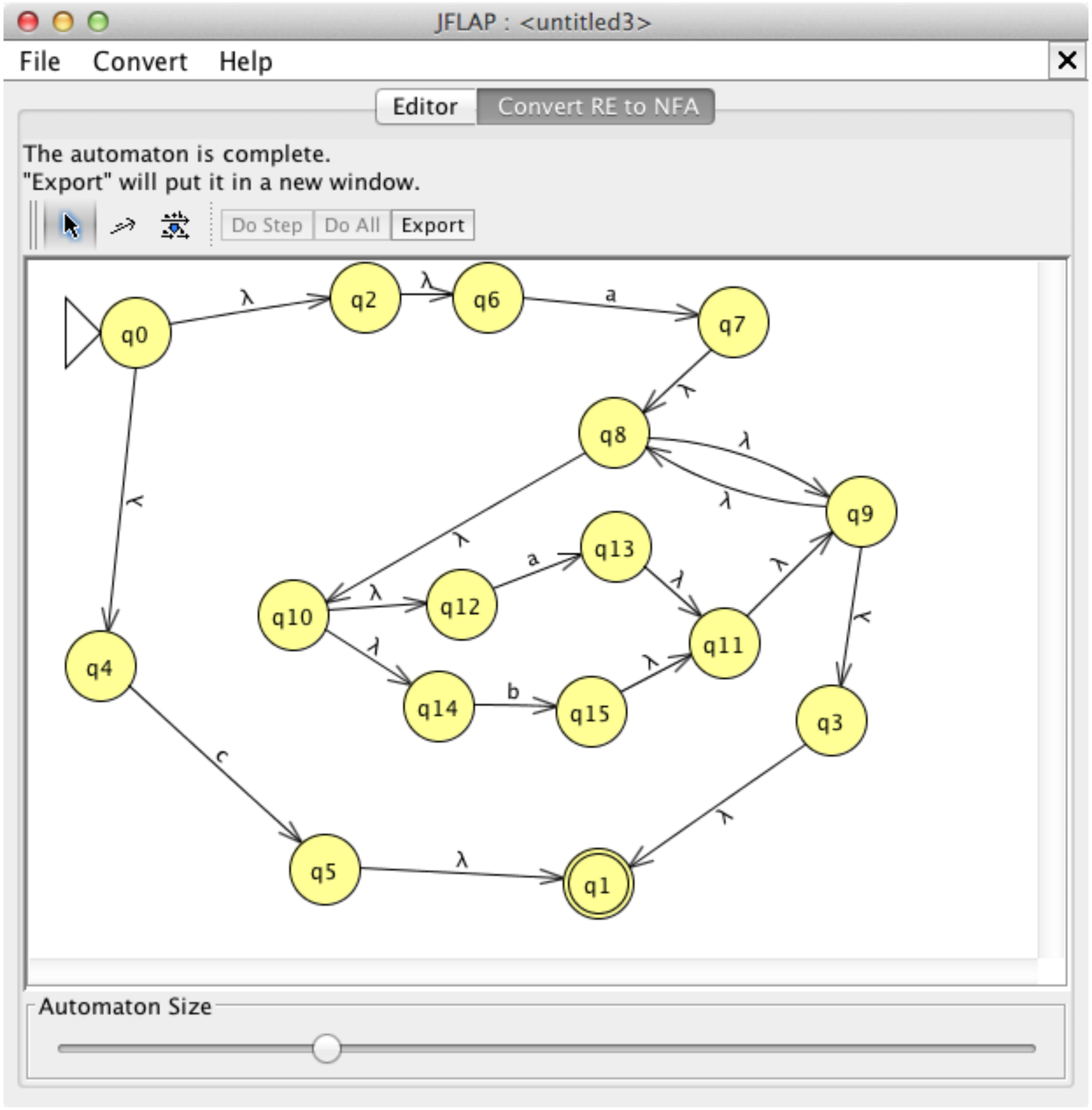
There are three options within JFLAP for assisting with the conversion from RE to NFA.

Option 1: Have JFLAP do the conversion all at once

Option 2: Follow along as JFLAP demonstrates step-by-step conversion

Option 3: Manually select resolutions and add ε-transitions during step-by-step conversion

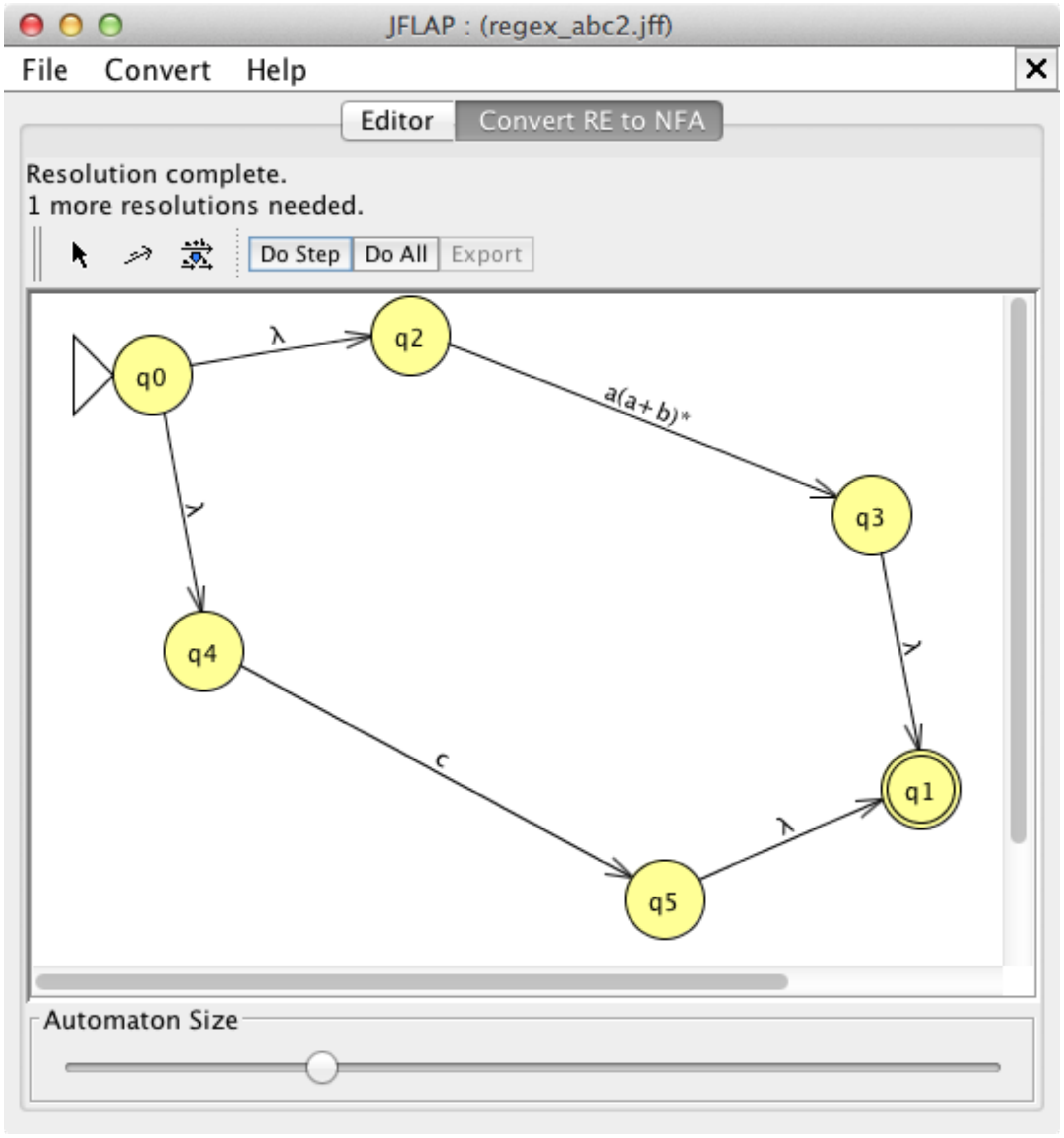
Option 1: Choose "Do All" from the initial diagram



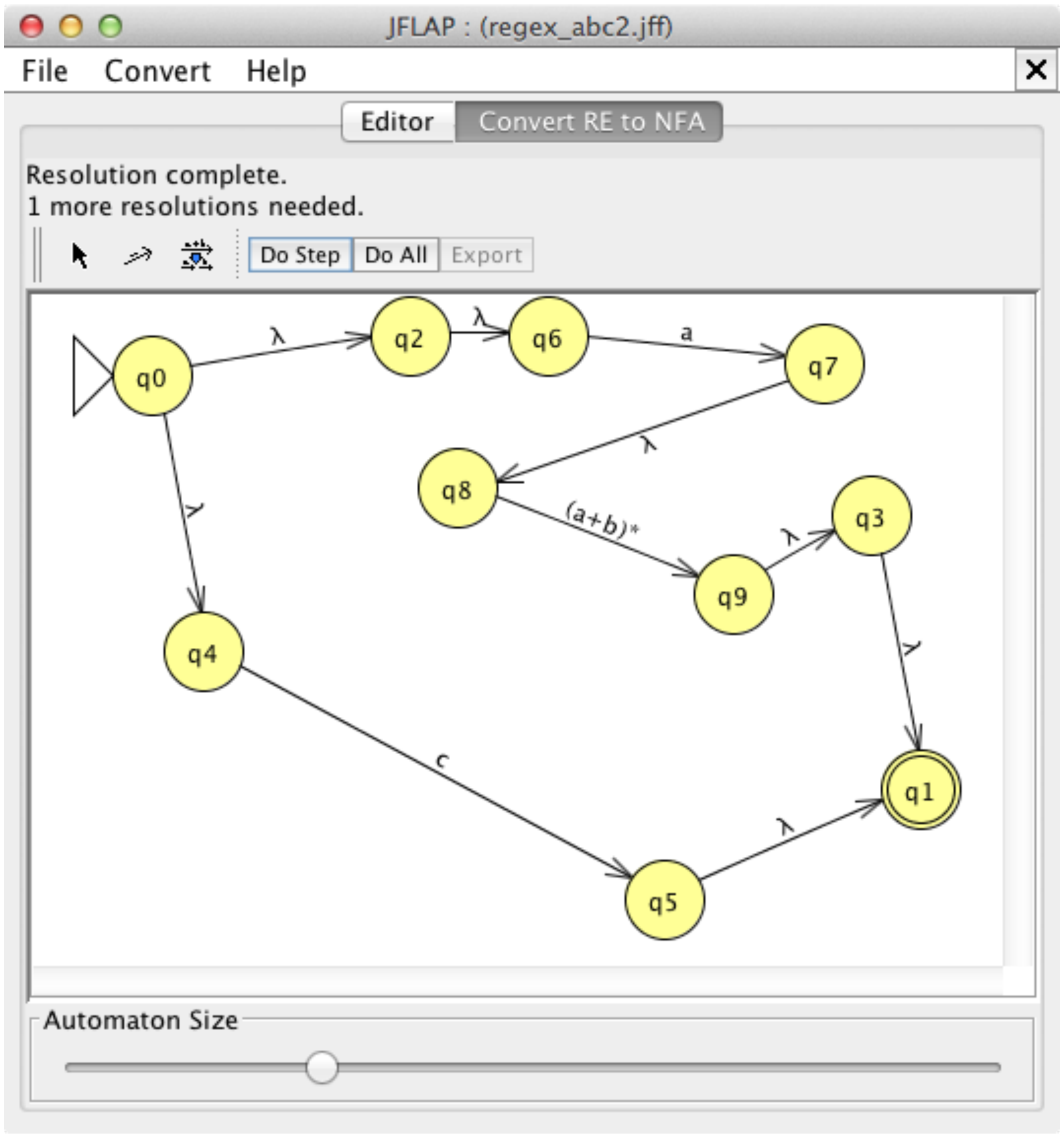
The automaton is complete.
"Export" will put it in a new window.

The result is a well-defined NFA that can be exported (choose "Export"; see: `regex_abc_nfa.jff`) and then used for conversion to DFA.

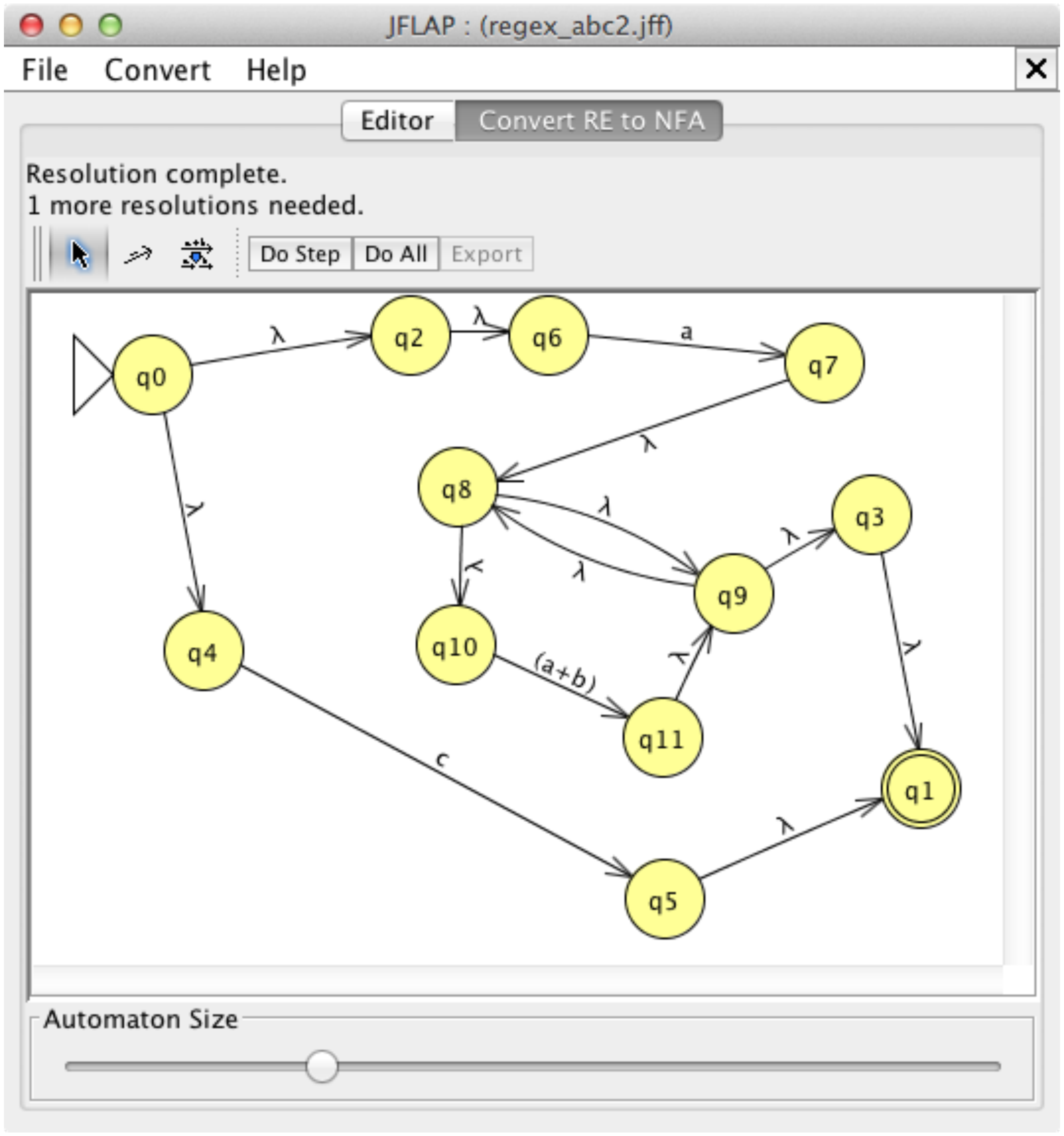
Option 2: Follow along as JFLAP demonstrates step-by-step conversion
Simply choose "Do Step" repeatedly and observe as JFLAP goes through the steps of conversion from RE to NFA.



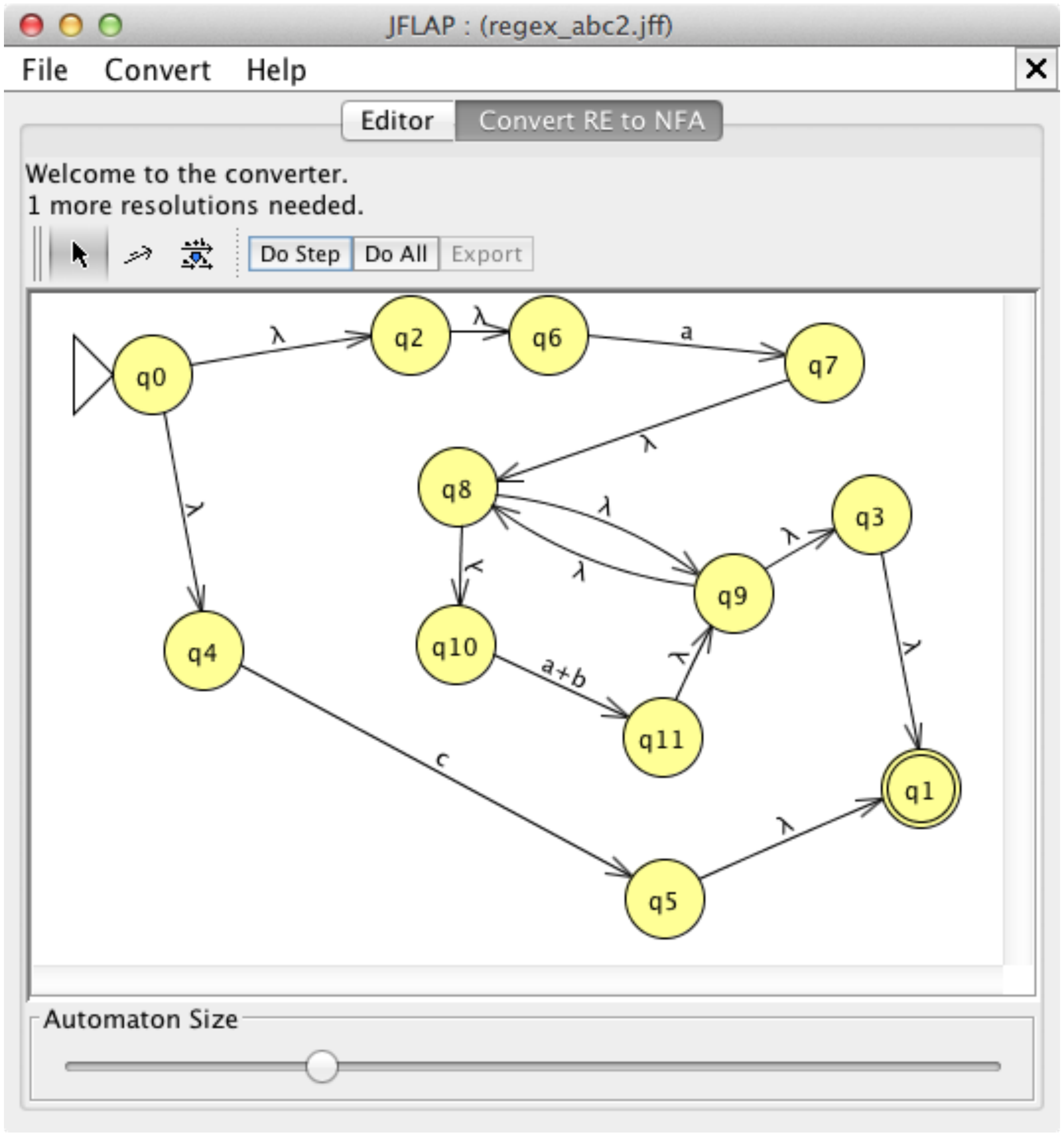
Resolution complete.
1 more resolutions needed.



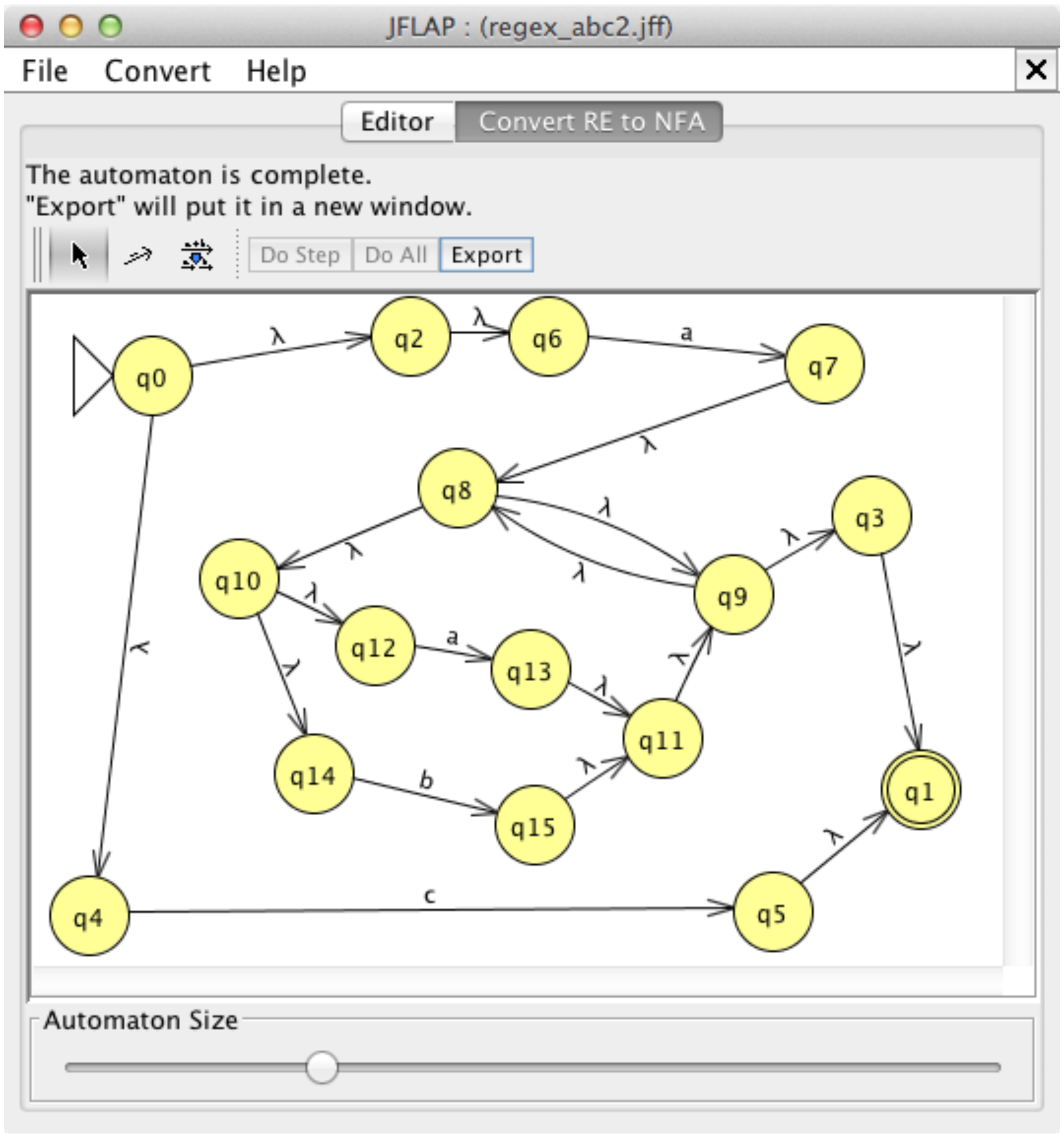
Resolution complete.
1 more resolutions needed.



Resolution complete.
1 more resolutions needed.



Welcome to the converter.
1 more resolutions needed.



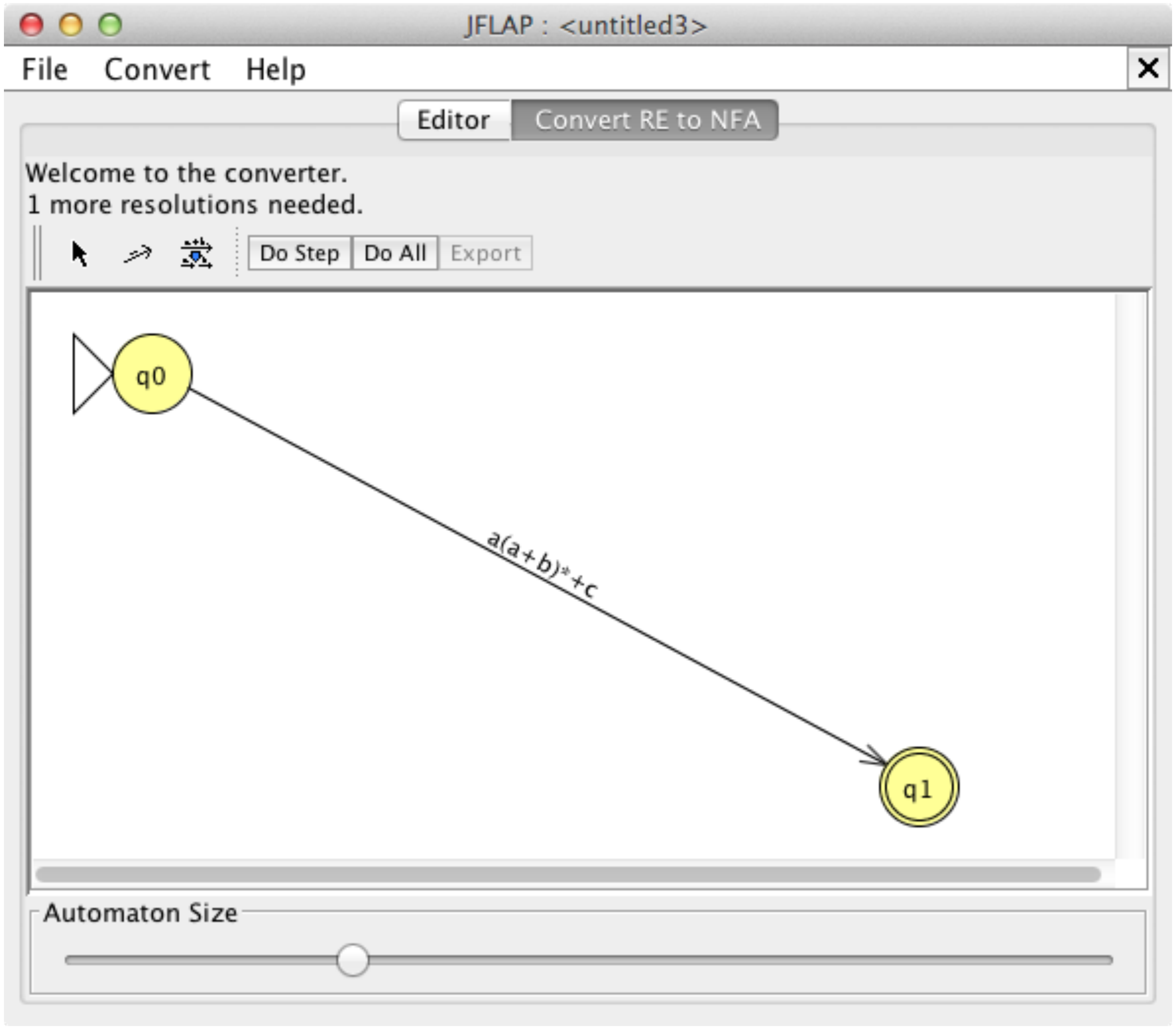
The automaton is complete.
"Export" will put it in a new window.

You now have a well-defined NFA that can be exported (choose “Export”; see: `regex_abc_nfa.jff`) and then used for conversion to DFA.

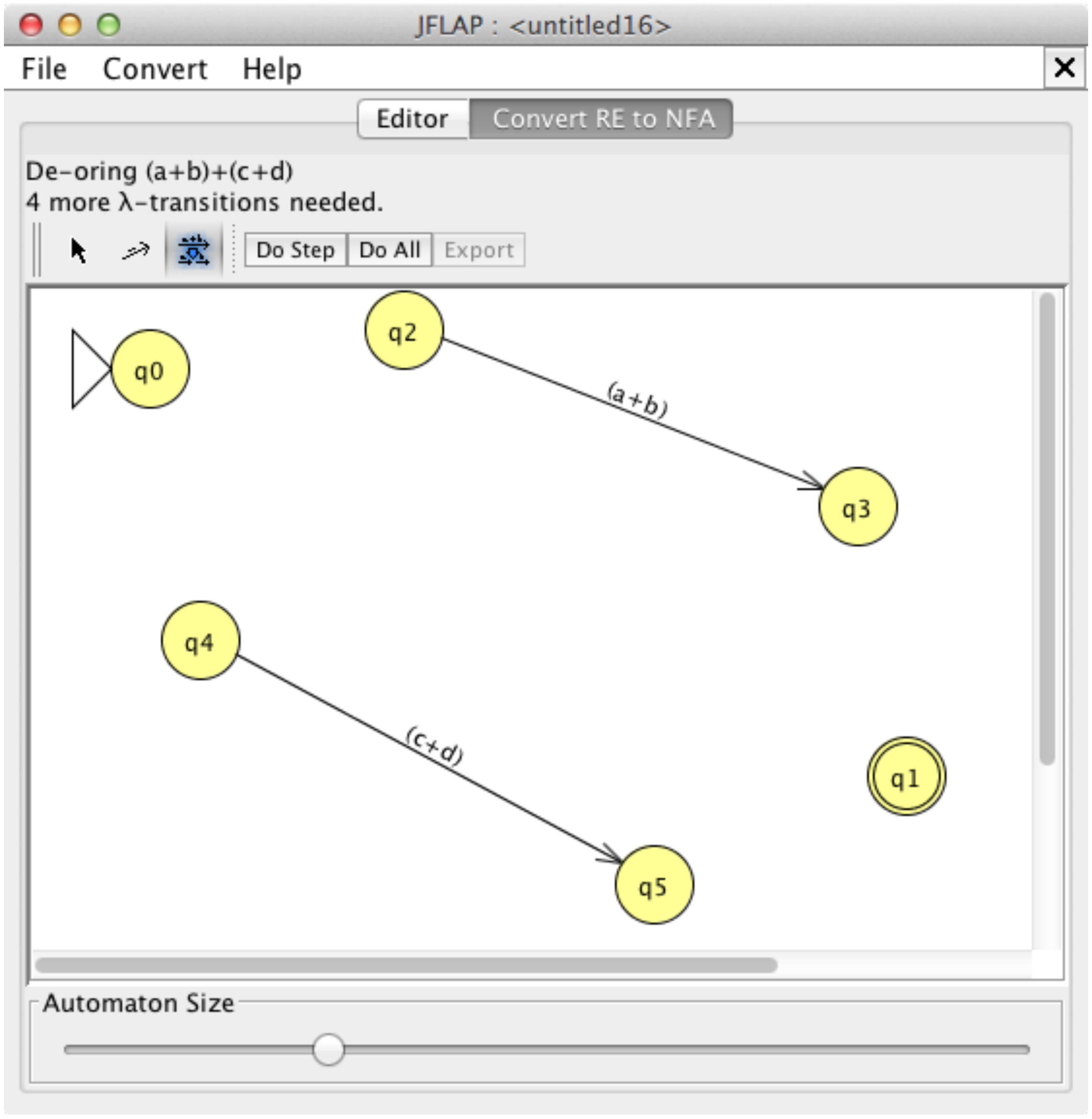
Option 3: Select resolutions and add ε-transitions manually during conversion

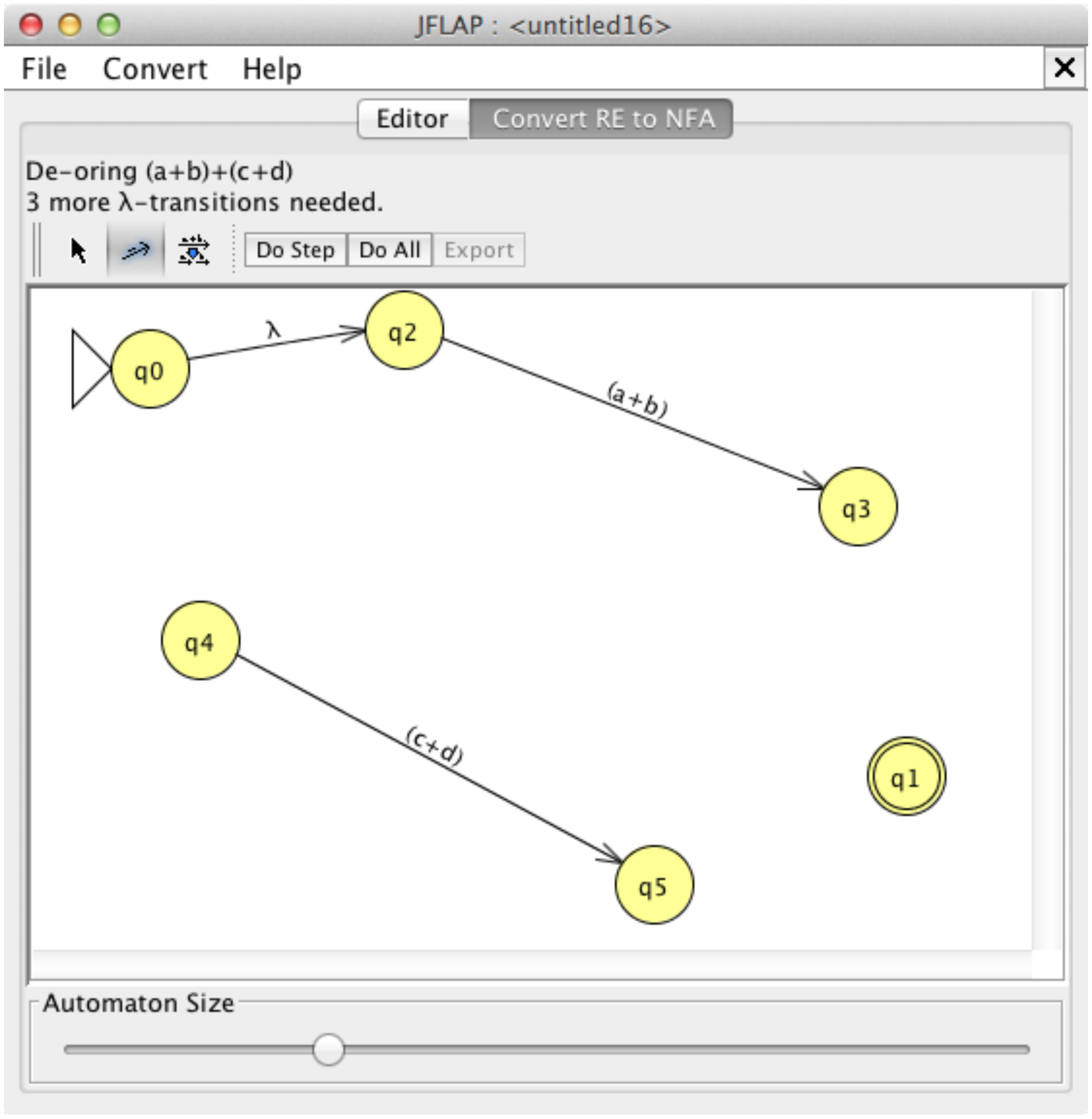
JFLAP provides a "(D)e-expressionify Transition" button that enables you to select which transition is to be converted next. Having chosen a transition, you then indicate all required ε-transitions associated with that conversion step. Once all associated ε-transitions have been specified, you can again use the (D)e-expressionify Transition button. This process continues until the result is a valid NFA.

Here are the steps in the conversion of the example RE.

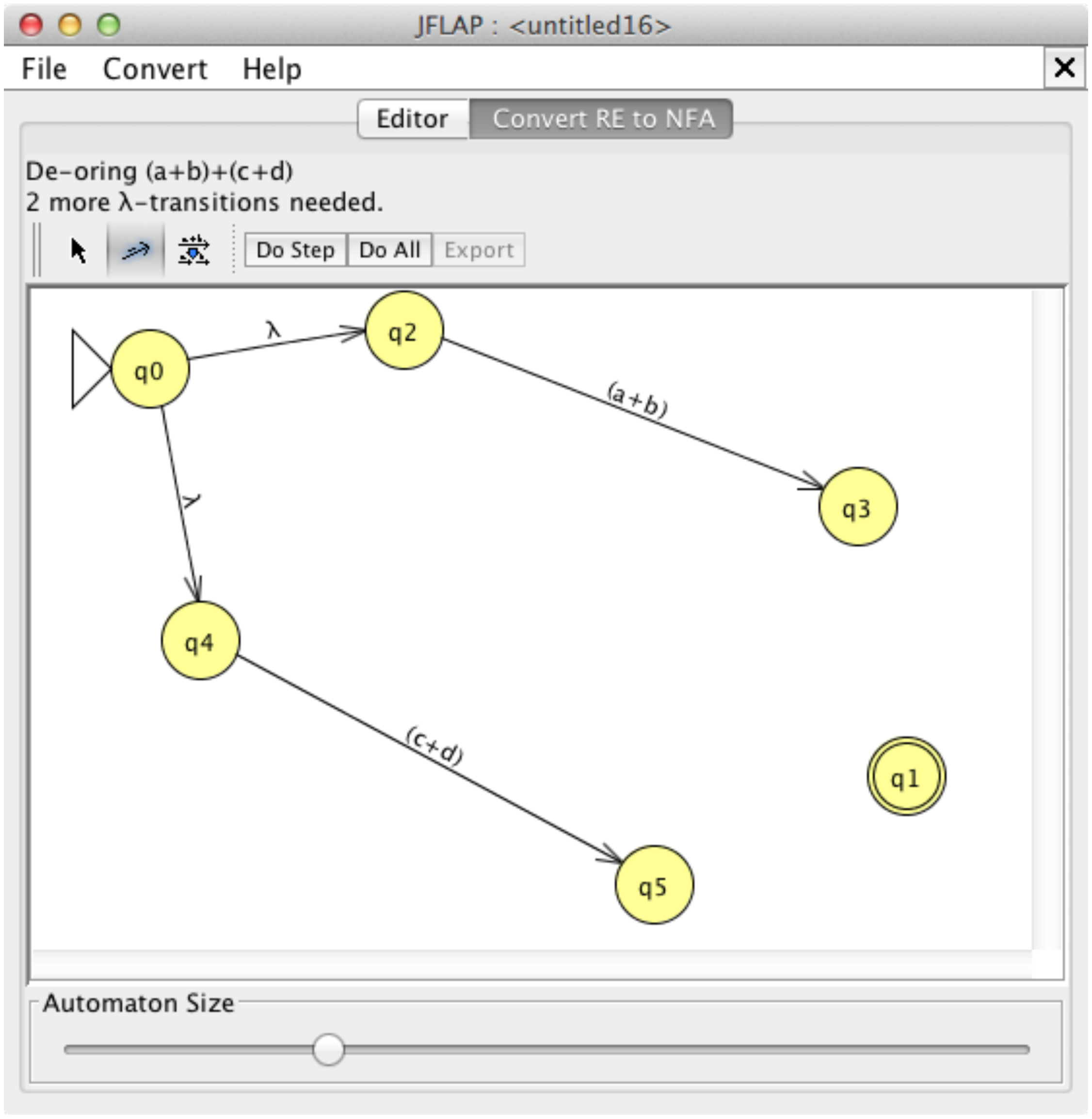


Welcome to the converter.
1 more resolutions needed.

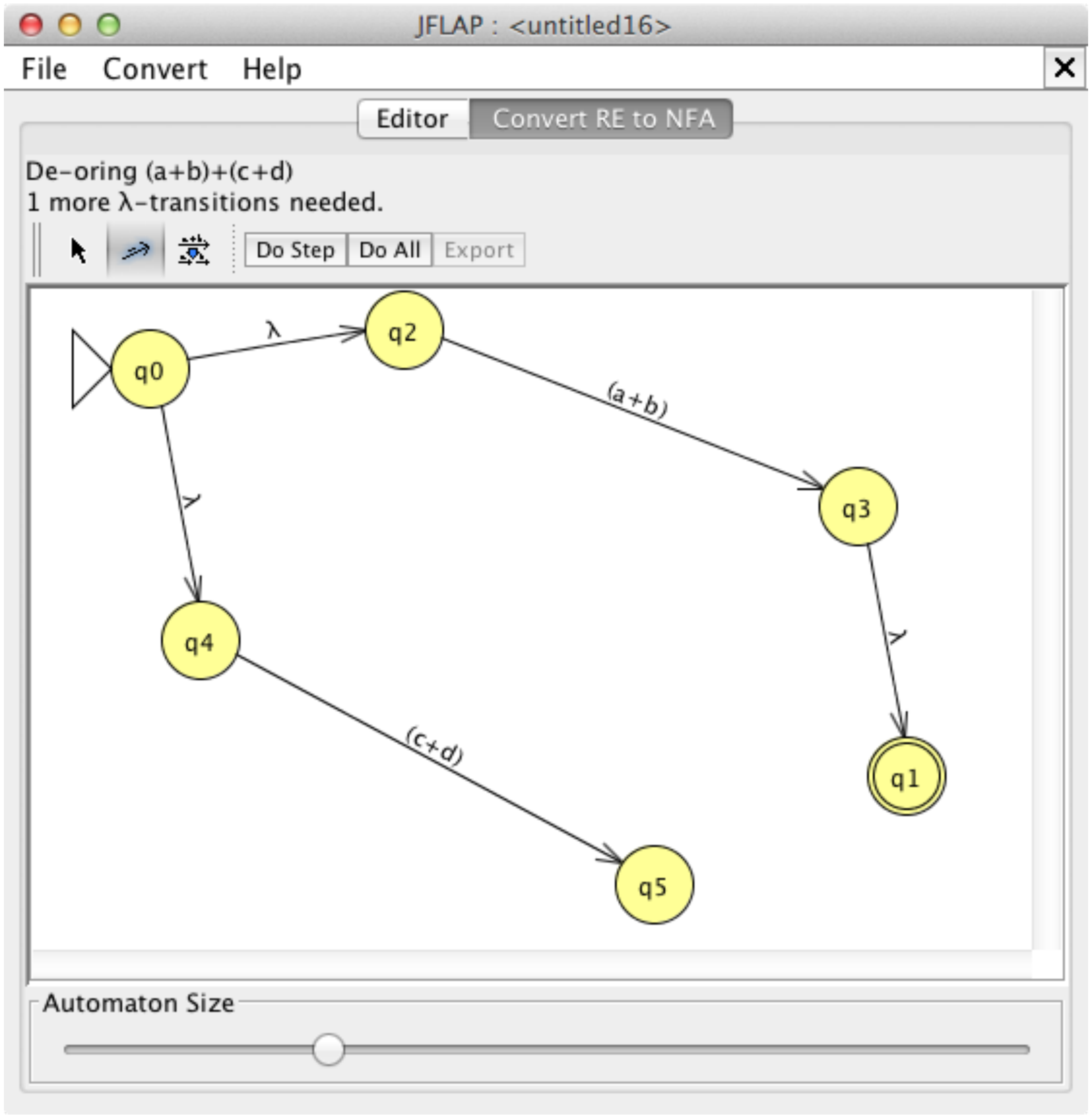

De-oring (a+b)+(c+d)
4 more λ-transitions needed.



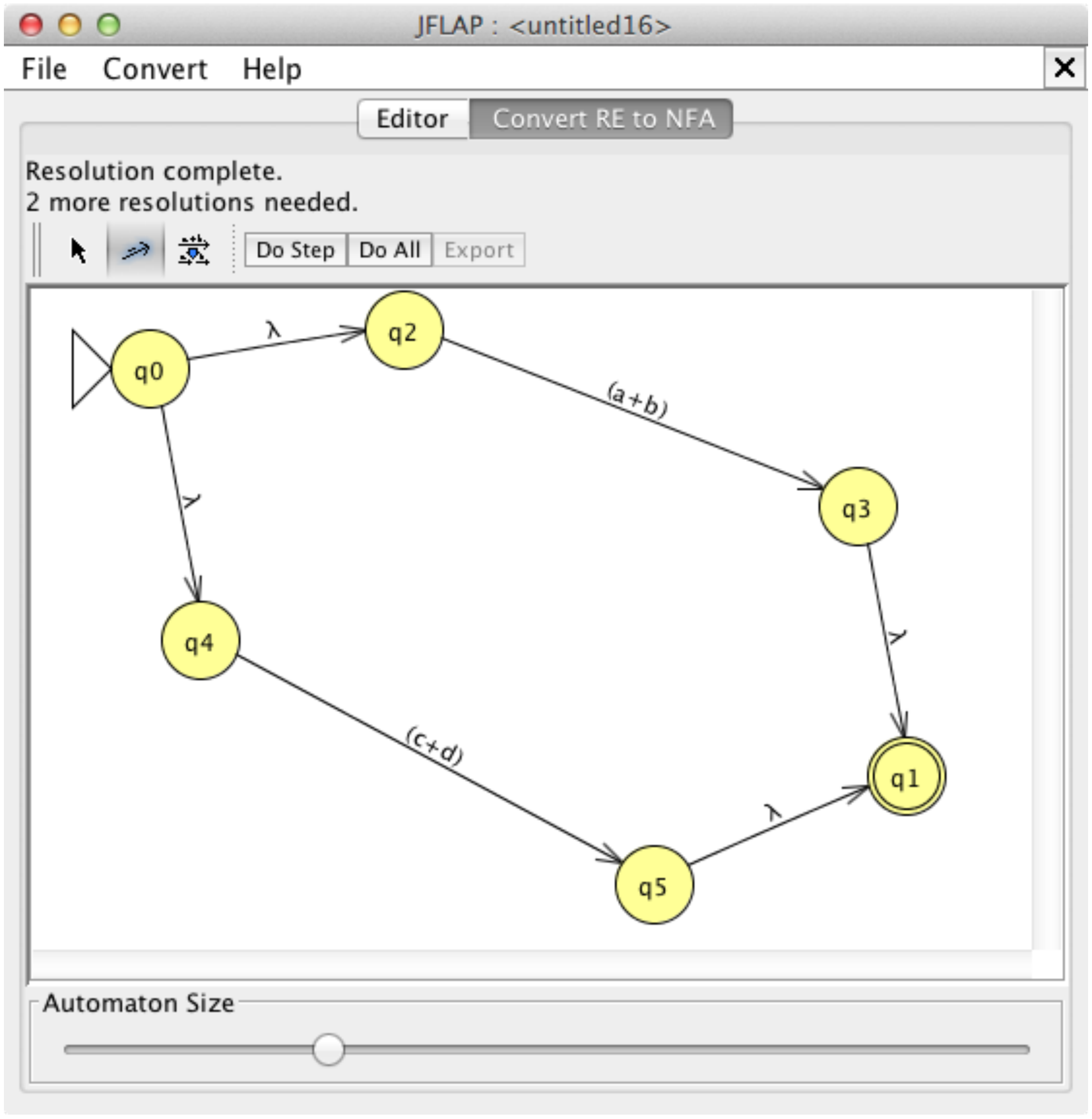
De-oring (a+b)+(c+d)
3 more λ-transitions needed.



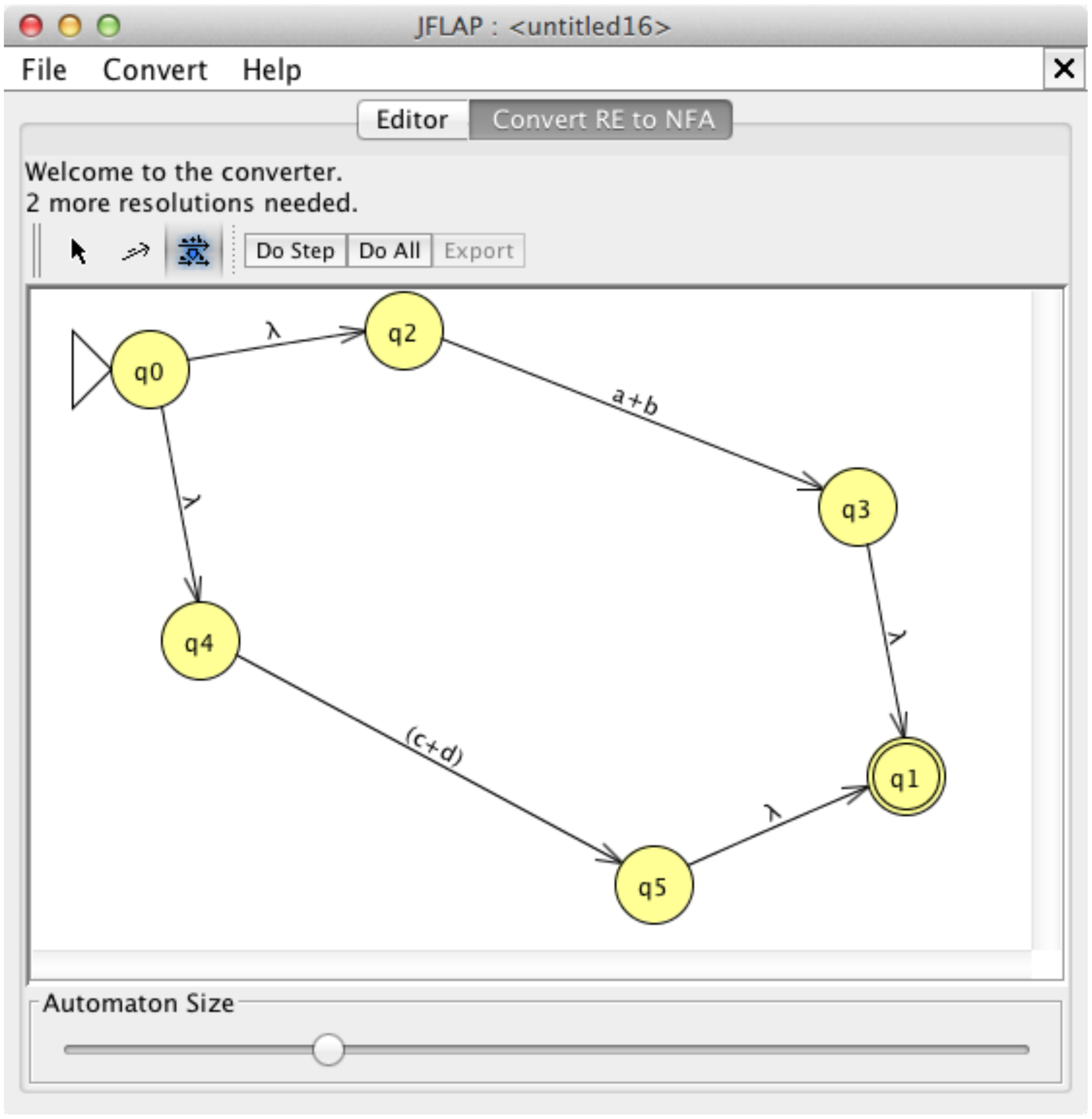
De-oring (a+b)+(c+d)
2 more λ-transitions needed.



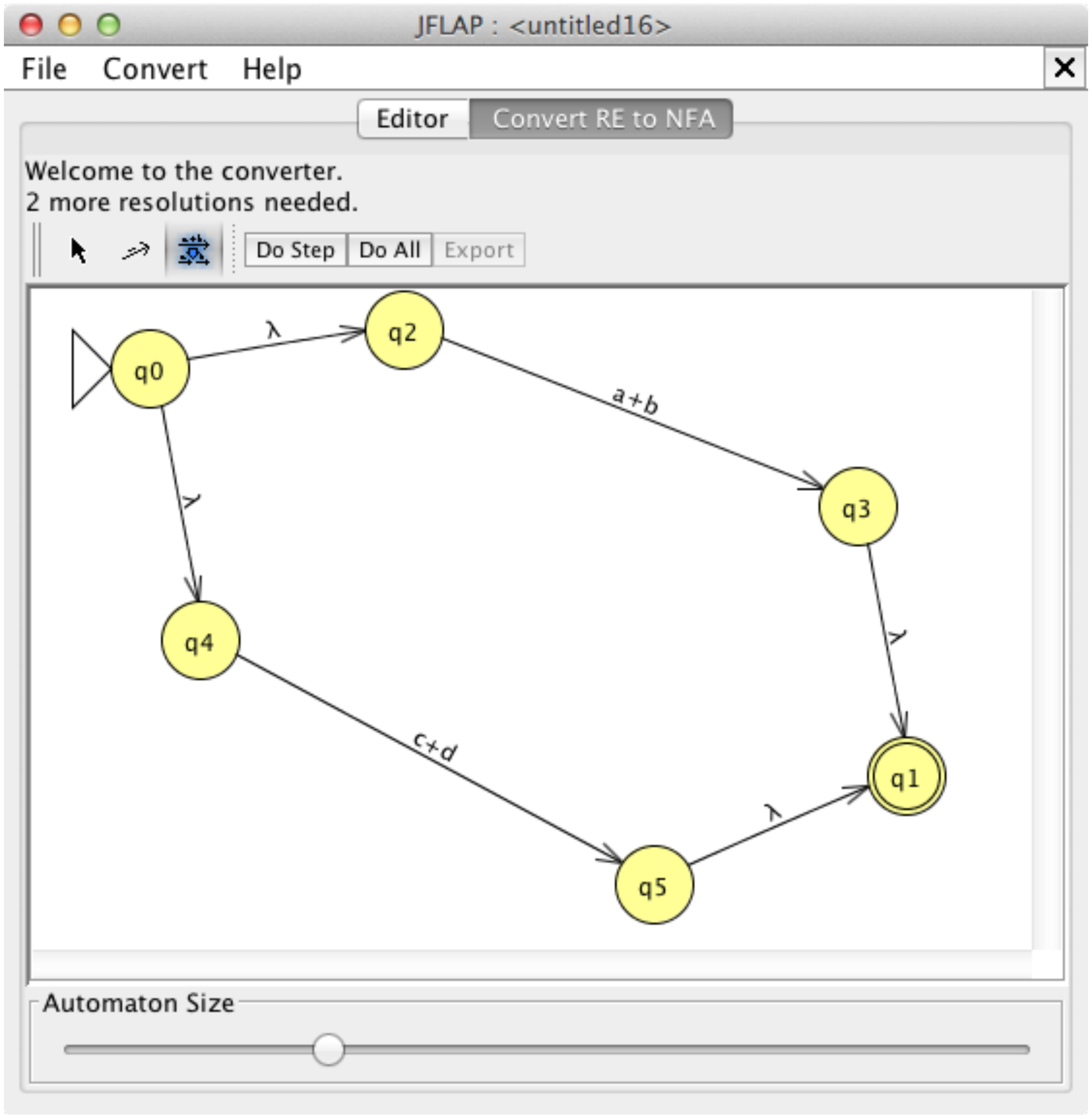
De-oring (a+b)+(c+d)
1 more λ-transitions needed.

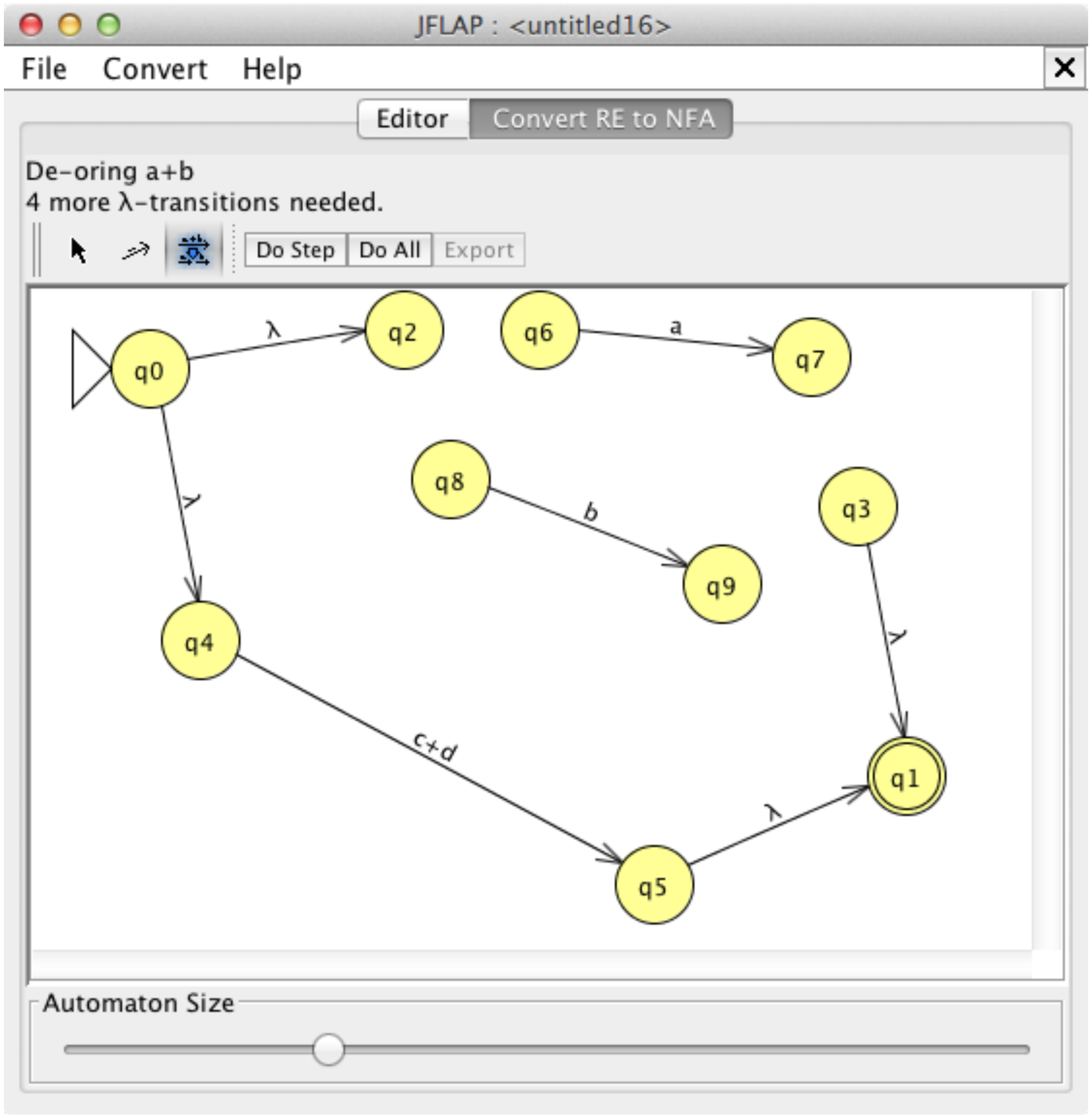


Resolution complete.
2 more resolutions needed.

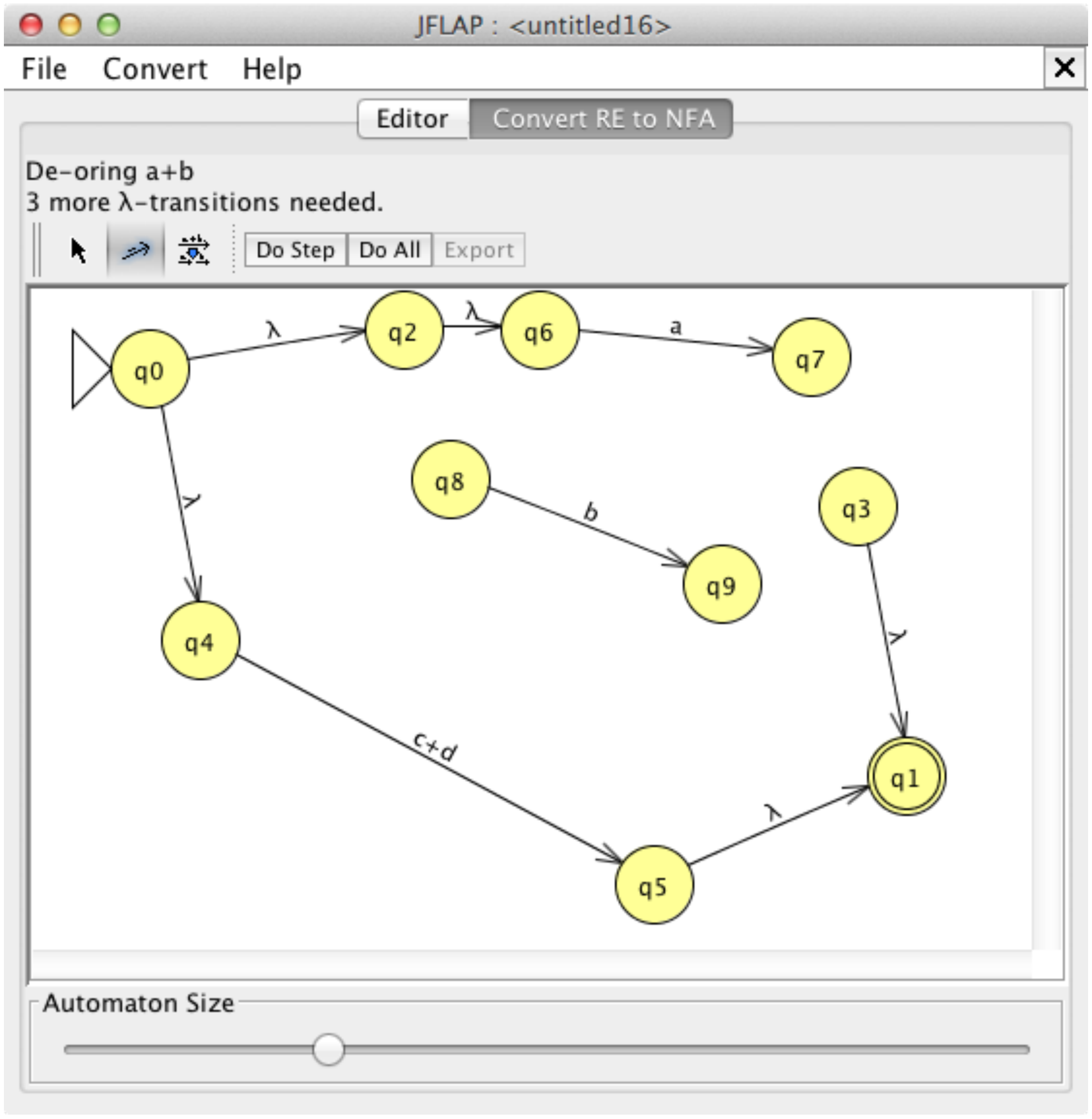


Welcome to the converter.
2 more resolutions needed.

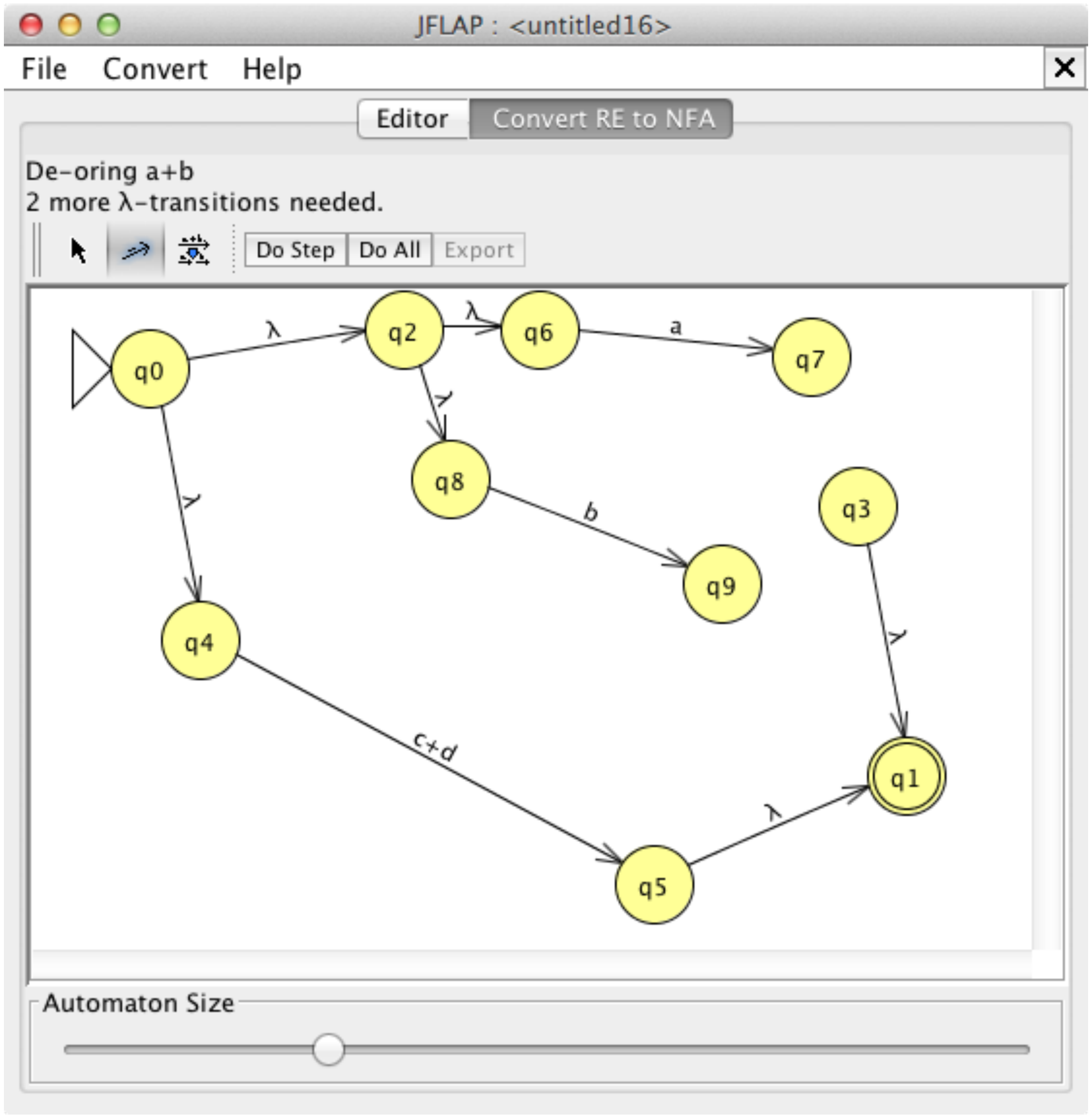

Welcome to the converter.
2 more resolutions needed.



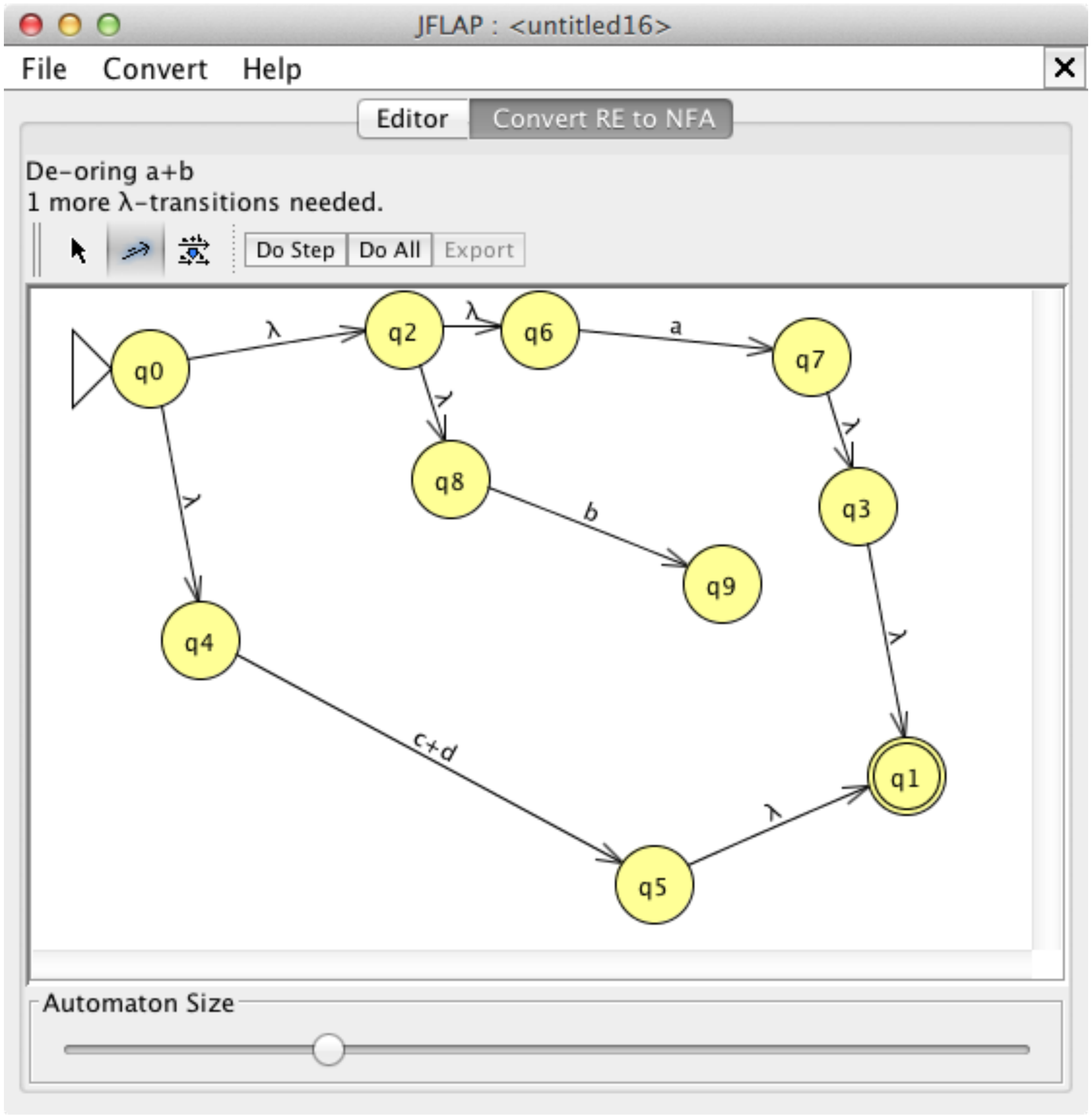
De-oring a+b
4 more λ-transitions needed.



De-oring a+b
3 more λ-transitions needed.

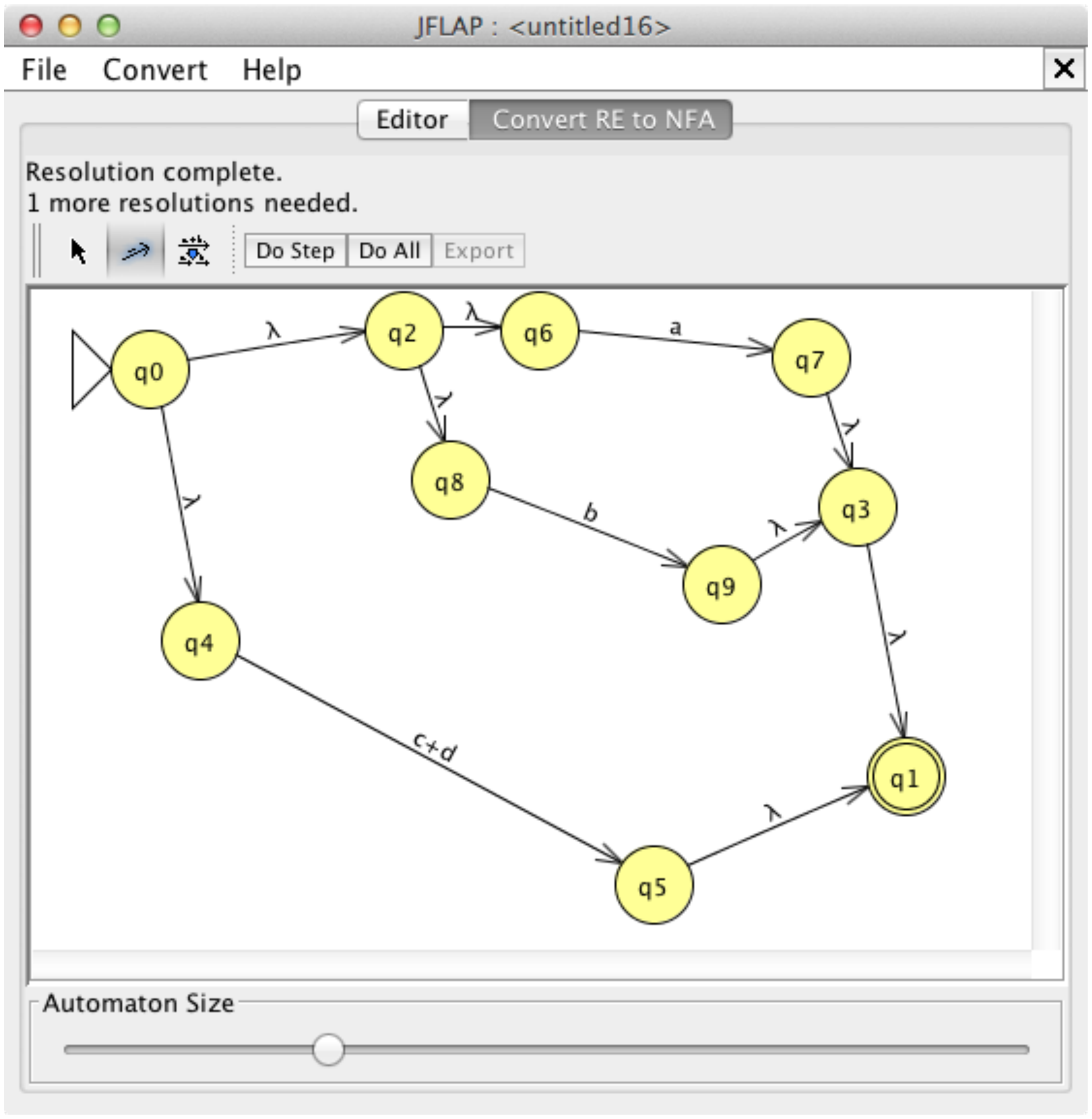


De-oring a+b
2 more λ-transitions needed.

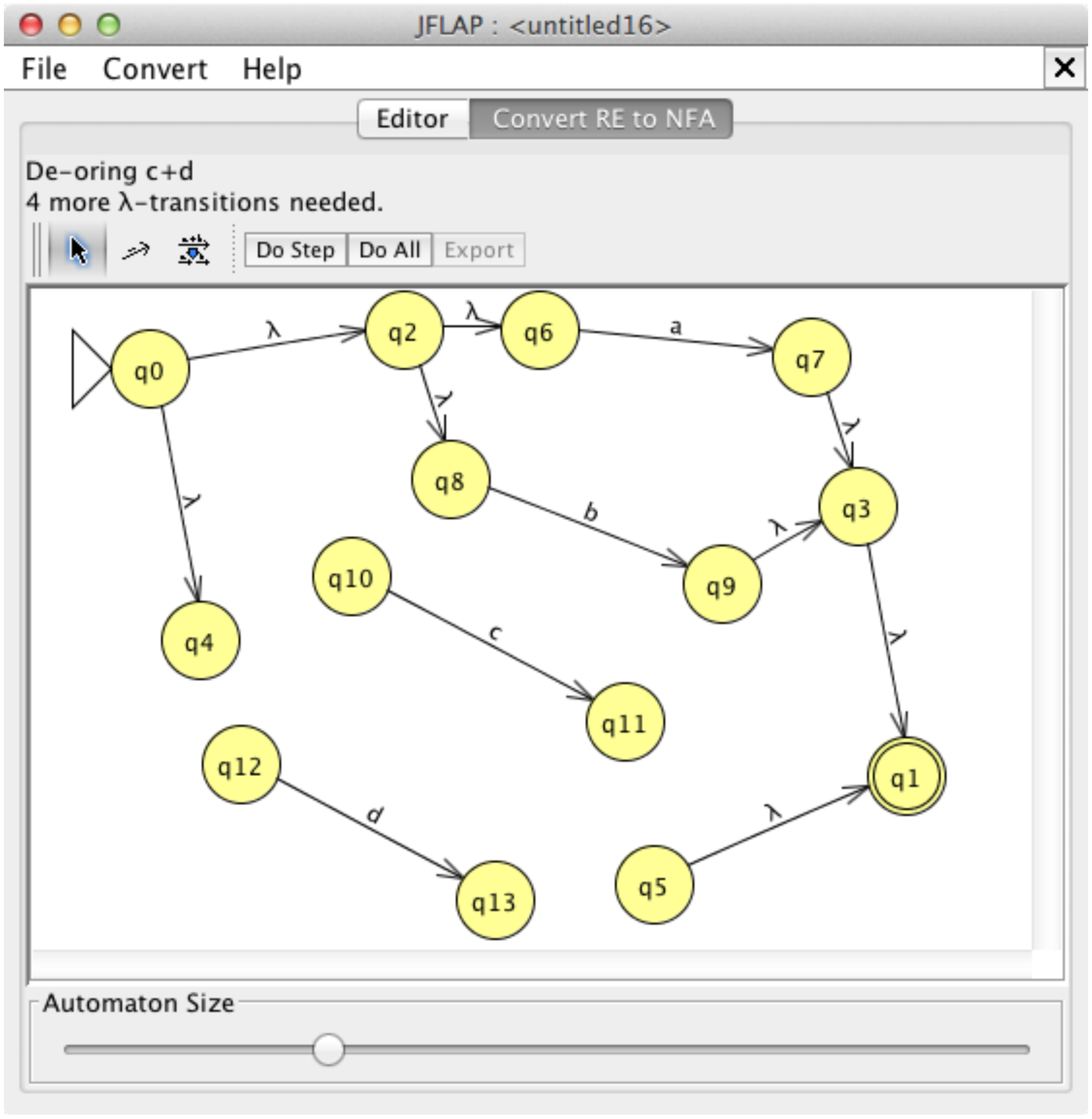


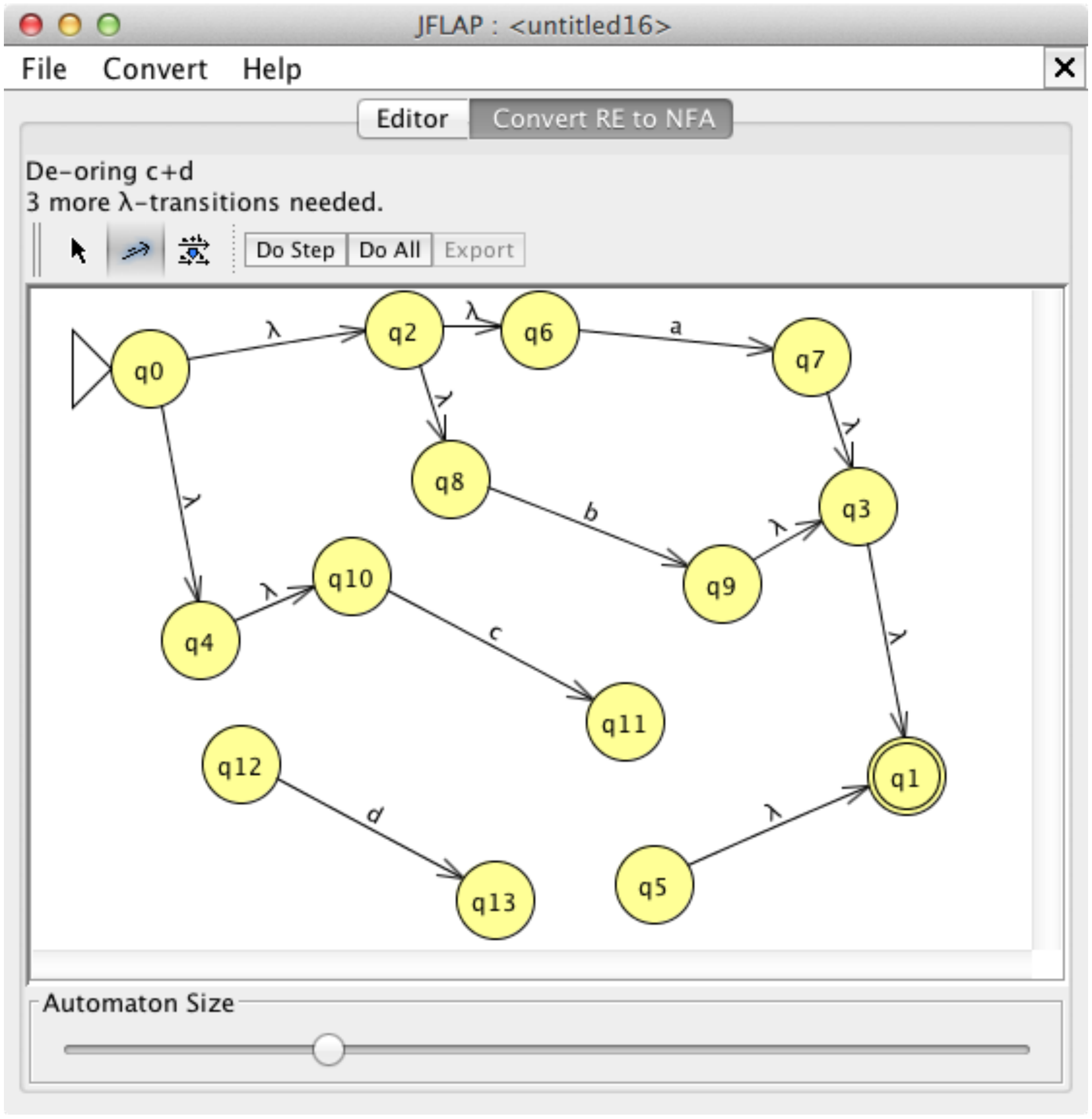
De-oring a+b
1 more λ-transitions needed.

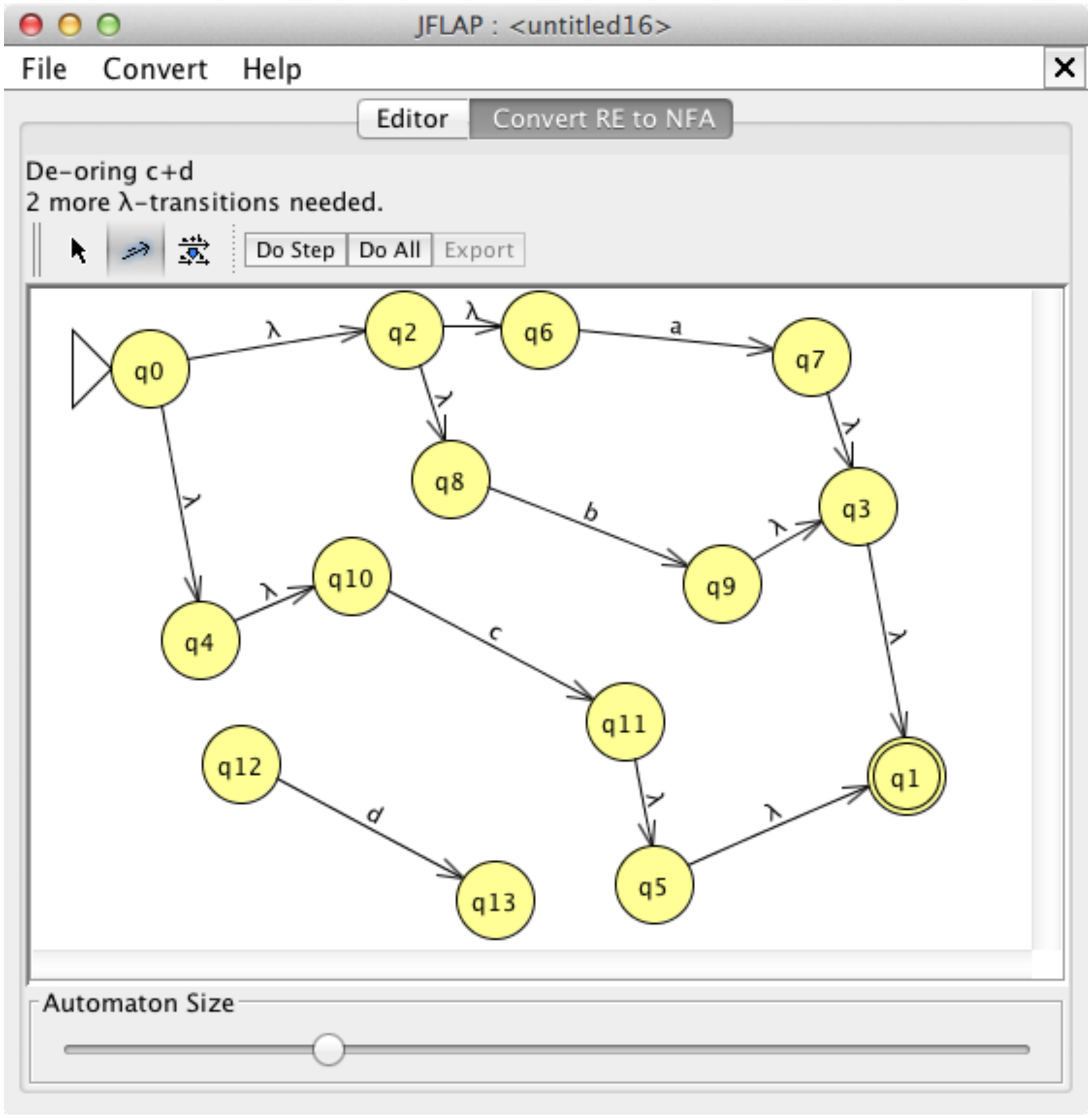
$$\begin{tikzcd}
q0 \arrow[r, "\lambda"] \arrow[d, "\lambda"] & q2 \arrow[r, "\lambda"] \arrow[d, "\lambda"] & q6 \arrow[r, "a"] & q7 \arrow[d, "\lambda"] \\
q4 \arrow[r, "c+d"] & q8 \arrow[r, "b"] & q9 & q3 \arrow[d, "\lambda"] \\
& & q5 \arrow[r, "\lambda"] & q1
\end{tikzcd}$$



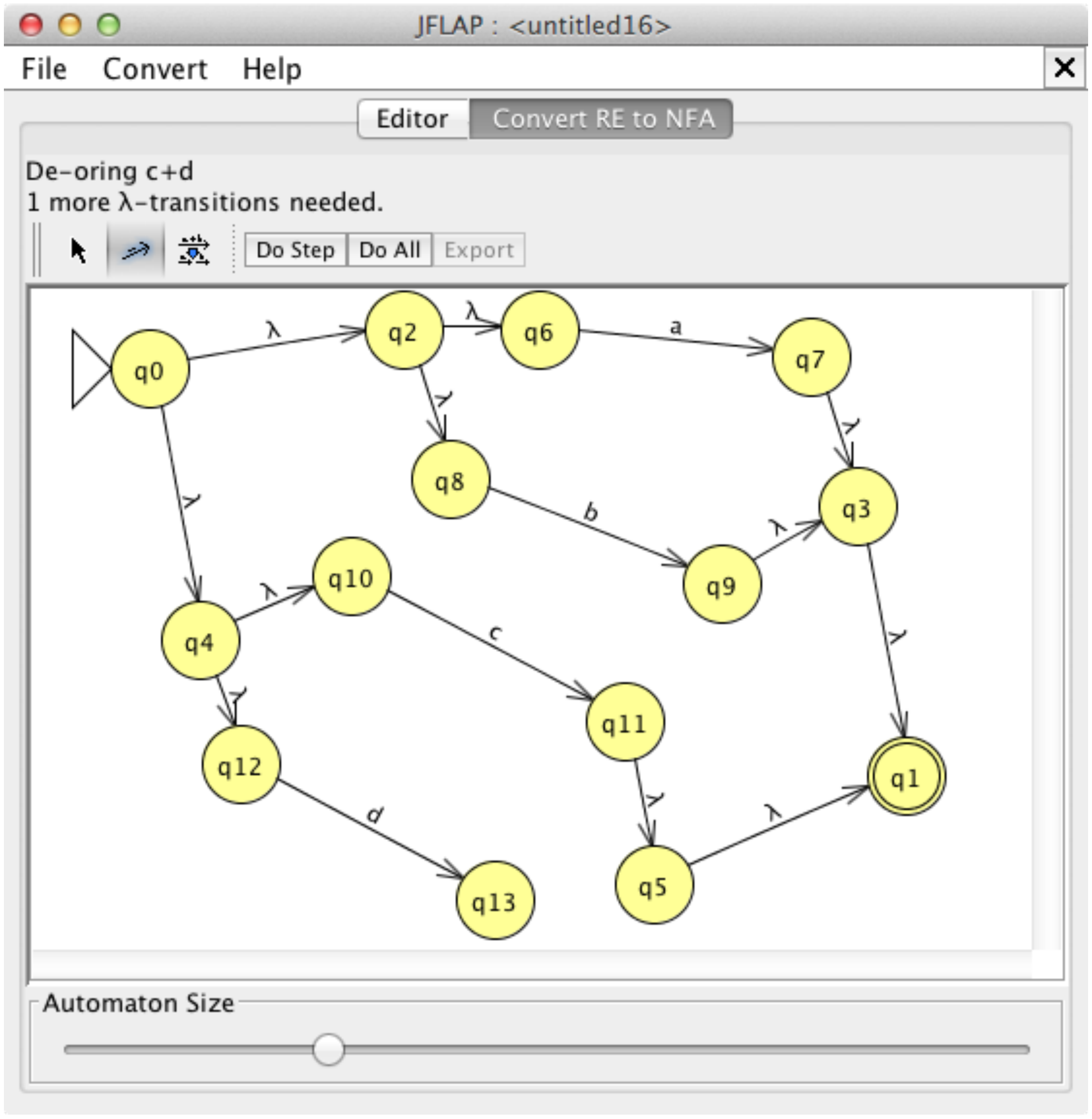
Resolution complete.
1 more resolutions needed.


De-oring c+d
4 more λ-transitions needed.

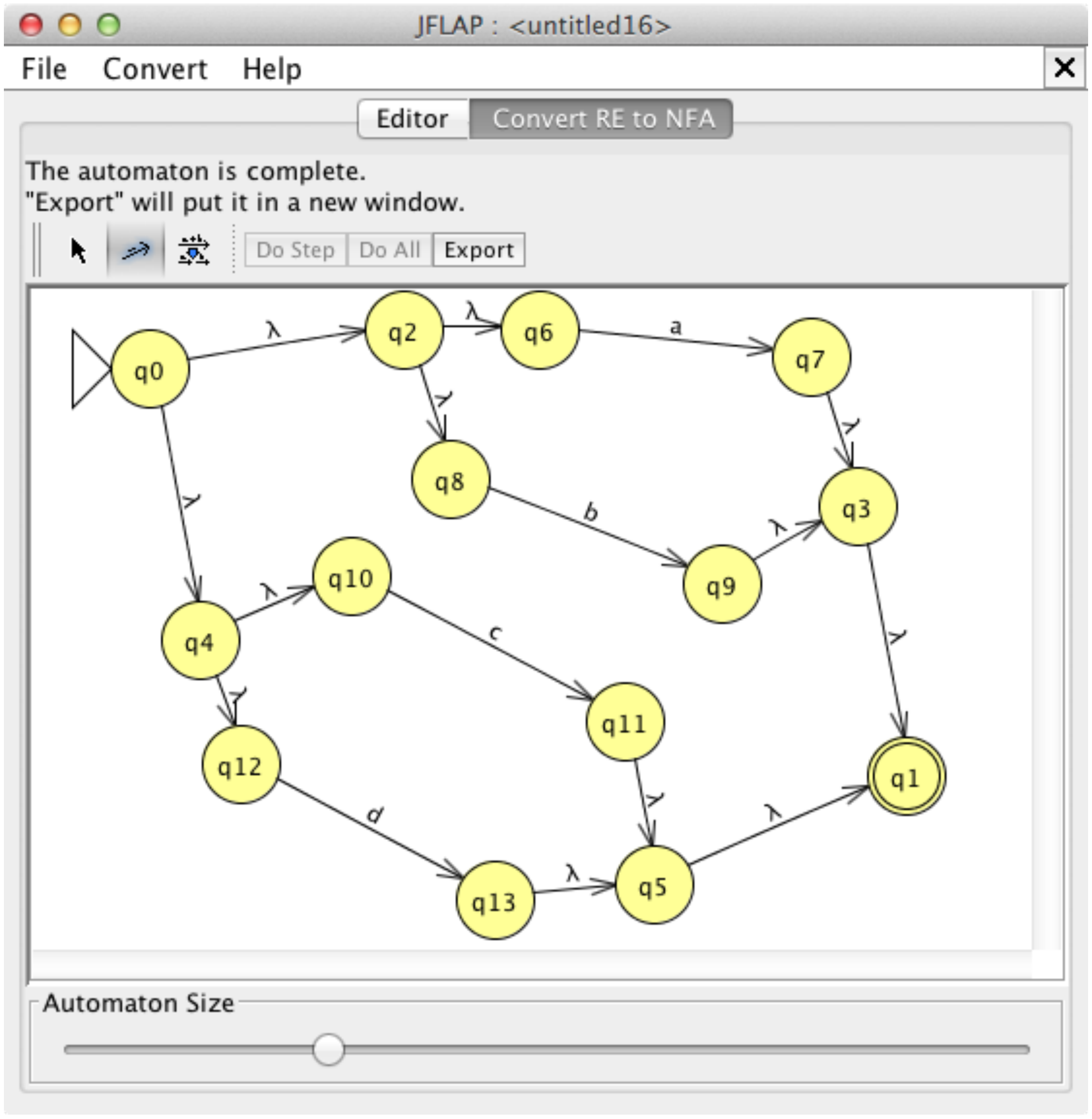

De-oring c+d
3 more λ-transitions needed.



De-oring c+d
2 more λ-transitions needed.



De-oring c+d
1 more λ-transitions needed.

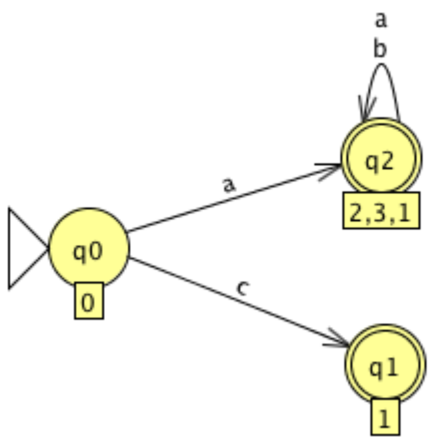


The automaton is complete.
"Export" will put it in a new window.

You now have a well-defined NFA that can be exported (choose "Export"; see: `regex_abc_nfa.jff`) and then used for conversion to DFA.

3. Convert NFA to DFA

However you arrived at the equivalent NFA for the RE, you can now apply the transformation process to convert the exported NFA to a DFA described in unit *“Explain algorithm and Convert NFA to DFA”* (see: `regex_abc_dfa.jff`).

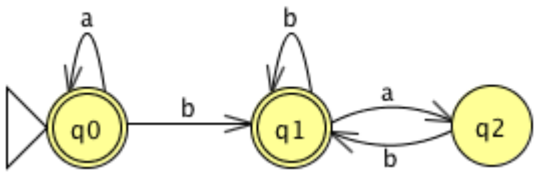


Questions to Think About

1. What is a regular expression for the following language L?
   L = { *w* | *w* □ {**a**, **b**}* where every occurrence of **a** is followed by a **b** }
   Answer: b*(ab)*b*

2. How many strings are in the language defined by the regular expression?
   (a|b)a(a|b)b
   Answer: 4 {aaab, baab, aabb, babb}

3. Specify a DFA for the language defined by the following regular expression.
   a*(b|ab)*



   Answer:
   (see `DFA_astar_babstar.jff`)

References

Wikipedia, *Regular Expression*
http://en.wikipedia.org/wiki/Regular_expression
[13 June 2014; Accessed on 17 June 2014]

Brown, Barry, *Convert Regular Expression to DFA*
https://www.youtube.com/watch?v=dlH2pIndNrU
[14 May 2011; Accessed on 26 June 2014]

JFLAP Tutorial, *Regular Expressions and Converting to a NFA*
http://www.jflap.org/tutorial/regular/index.html
[Accessed on 27 June 2014]