**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Description** | **Page No.** |
|  | Introduction | 4 |
|  | Topics covered in jflap | 5 |
|  | Problem statement | 6 |
|  | Description with screenshots | 7-14 |
|  | Results | 15-16 |
|  | References | 17 |

**1. INTRODUCTION**

- JFLAP (Java Formal Languages and Automata Package) is interactive educational software written in [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) for experimenting with topics in the [computer science](https://en.wikipedia.org/wiki/Computer_science) area of [formal languages](https://en.wikipedia.org/wiki/Formal_languages) and [automata theory](https://en.wikipedia.org/wiki/Automata_theory), primarily intended for use at the undergraduate level or as an advanced topic for high school.

- JFLAP allows one to create and simulate structures, such as programming a finite state machine, and experiment with proofs, such as converting a [nondeterministic finite automaton](https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton) (NFA) to a deterministic finite automaton (DFA).

-JFLAP is developed and maintained at [Duke University](https://en.wikipedia.org/wiki/Duke_University), with support from the [National Science Foundation](https://en.wikipedia.org/wiki/National_Science_Foundation) since 1993. It is [freeware](https://en.wikipedia.org/wiki/Freeware) and the source code of the most recent version is available, but under some restrictions.[[1]](https://en.wikipedia.org/wiki/JFLAP#cite_note-1) JFLAP runs as a Java application.

**2. TOPICS COVERED IN JFLAP**

Topics on [regular language](https://en.wikipedia.org/wiki/Regular_language) include:

* [finite state machine](https://en.wikipedia.org/wiki/Finite_state_machine)
* [regular grammar](https://en.wikipedia.org/wiki/Regular_grammar)
* [regular expression](https://en.wikipedia.org/wiki/Regular_expression)
* Proof on [nondeterministic finite automaton](https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton) to [deterministic](https://en.wikipedia.org/wiki/Deterministic) finite automaton
* Proof on deterministic finite automaton to regular grammar
* Proof on deterministic finite automaton to regular expression
* [pumping lemma](https://en.wikipedia.org/wiki/Pumping_lemma) for [regular languages](https://en.wikipedia.org/wiki/Regular_languages)

Topics on [context-free language](https://en.wikipedia.org/wiki/Context-free_language) include:

* [pushdown automata](https://en.wikipedia.org/wiki/Pushdown_automata)
* [context-free grammar](https://en.wikipedia.org/wiki/Context-free_grammar)
* proof on [with: nondeterministic](https://en.wiktionary.org/wiki/nondeterministic) [pushdown automaton](https://en.wikipedia.org/wiki/Pushdown_automaton) to context-free grammar
* proof on context-free grammar to pushdown automaton
* pumping lemma for context-free language
* [CYK parser](https://en.wikipedia.org/wiki/CYK_parser)
* [LL parser](https://en.wikipedia.org/wiki/LL_parser)
* [SLR parser](https://en.wikipedia.org/wiki/SLR_parser)

Topics on [recursively enumerable language](https://en.wikipedia.org/wiki/Recursively_enumerable_language):

* [Turing machine](https://en.wikipedia.org/wiki/Turing_machine)
* [unrestricted grammar](https://en.wikipedia.org/wiki/Unrestricted_grammar)

Other related topics:

* [Moore machine](https://en.wikipedia.org/wiki/Moore_machine)
* [Mealy machine](https://en.wikipedia.org/wiki/Mealy_machine)
* [L-system](https://en.wikipedia.org/wiki/L-system)

**3. PROBLEM STATEMENT**

- Obtain a regular expression for a language L = {w:|w| mod 3 = 0 where w Ɛ (a, b) \*}

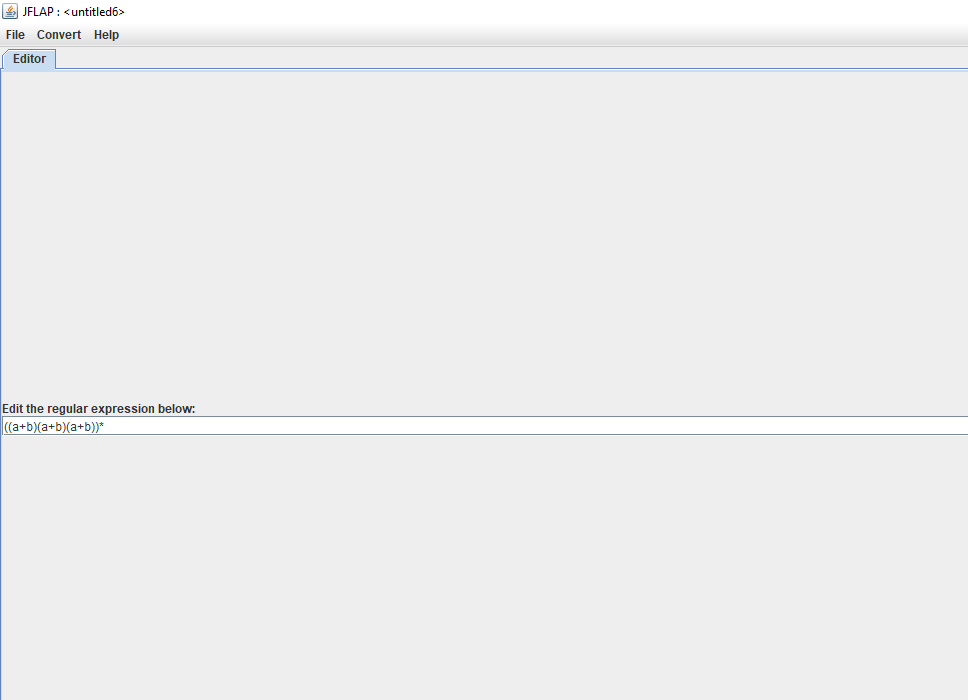
**-**Here we have to obtain a regular expression for the given language in problem statement.

-Regular expression should accept the result for any strings of length equal to multiples of 3 or null and reject strings, whose lengths are not equal to multiples of 3.

-As the solved problem can be seen in description

**4.DESCRIPTION WITH SCREENSHOTS**

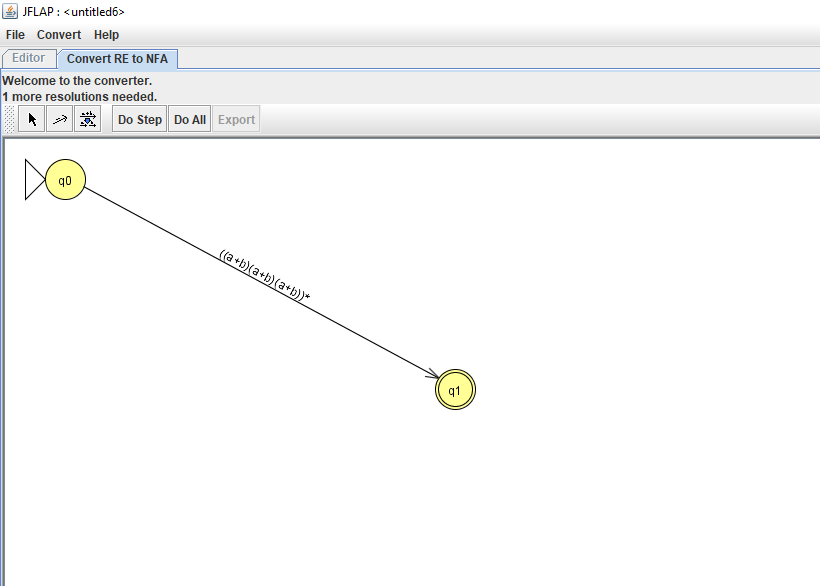
* Now we should enter the expression that we need to obtain. Note that there should be no blanks in the regular expression, as any blank will be processed as a symbol of Σ.



**Fig 4.1**

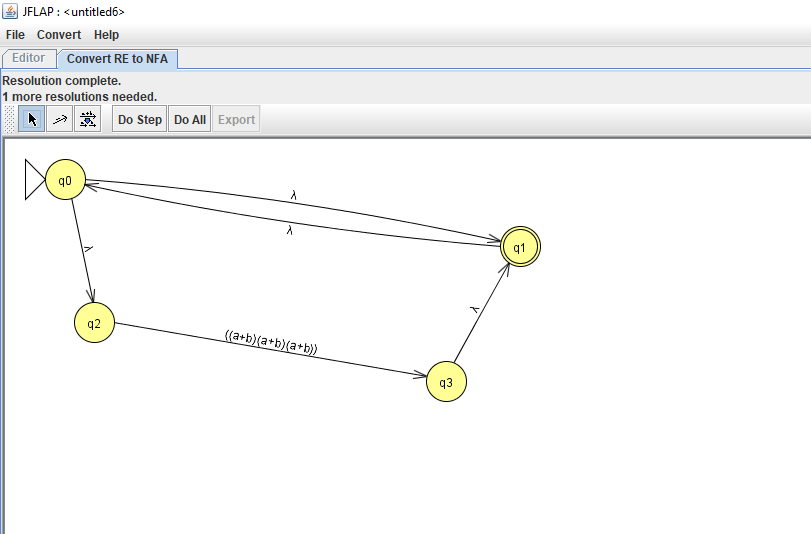
**Converting to NFA**

* After typing in an expression, there is nothing else that can be done in this editor window besides converting it to an NFA.  Click on the “Convert → Convert to NFA” menu option.



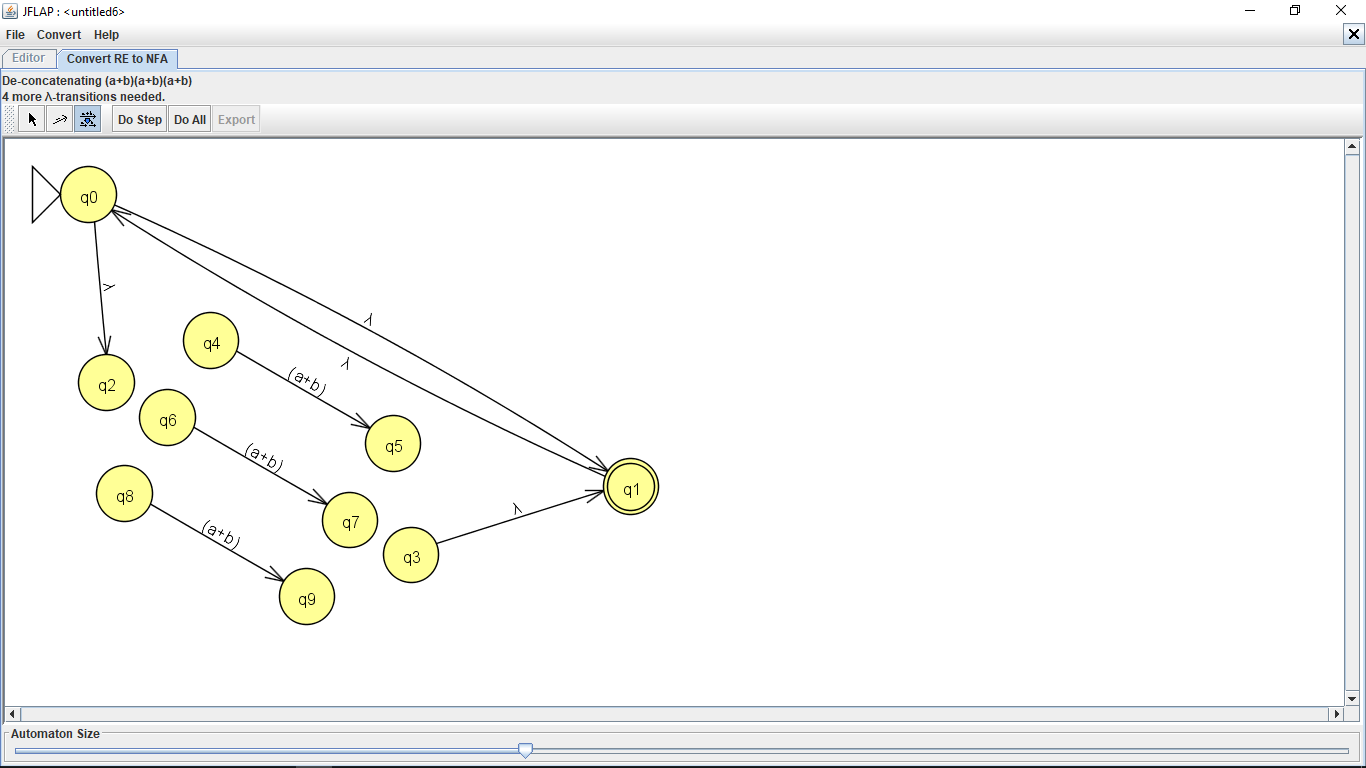
**Fig 4.2**

* Now we get NFA direct from start to final state with the required expression.
* We should expand this to understand step by step transition. Click on “Do step” button.



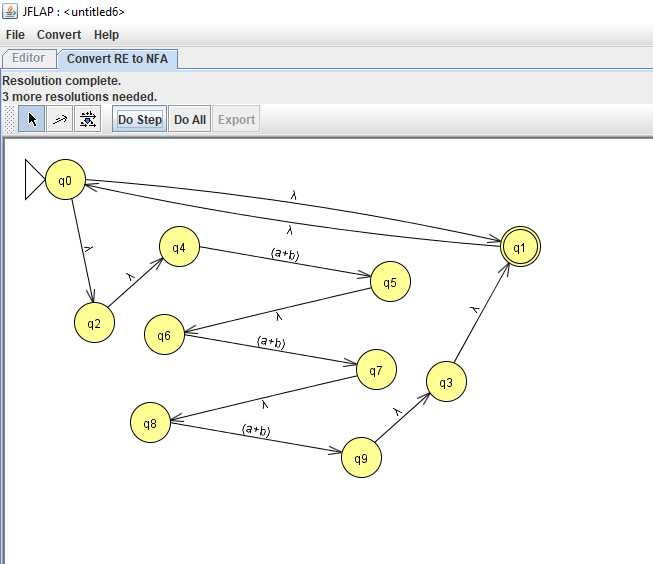
**Fig 4.3**

* Here the NFA is expanded to show the transition of null (lambda here ) strings from state ‘q0’ to ‘q1’ and from ‘q0’ to ‘q2’ and then ‘q3’ to q1’ accepting any strings between ‘q2’ and ‘q3’.



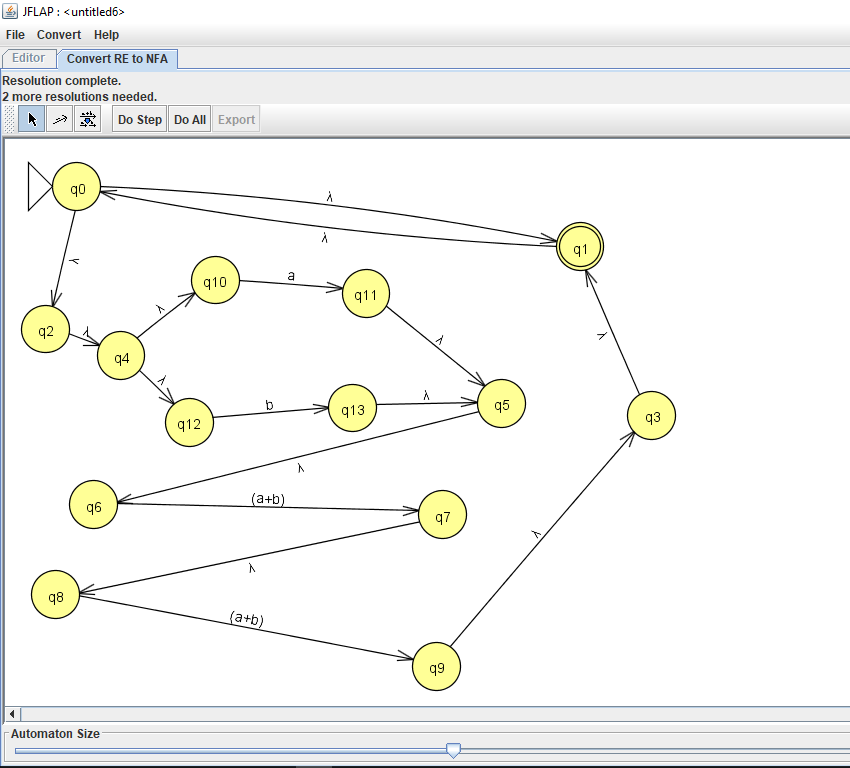
**Fig 4.4**

* In fig 4.3 from state ‘q2’ to ‘q3’ we have the expression ((a+b)(a+b)(a+b)) .
* In fig 4.4 we have divided this exprression to 3 different paths which should be traversed as shown in the figure.

****

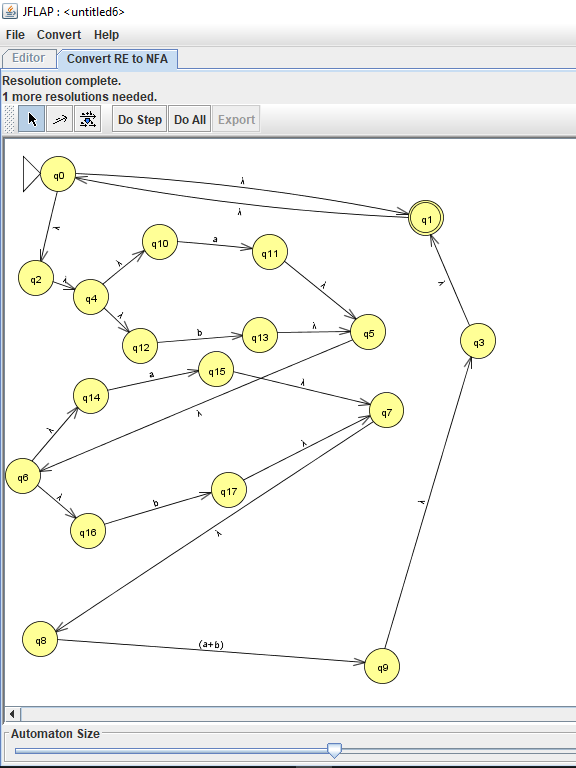
**Fig 4.5**

* In fig 4.5 we show the transition from ‘q2’ state. There is transition from ‘q2’ to ‘q4’ only since it should go through first letter of the string it doesn’t go to ‘q6’ since it wil show invalid transition because it cant go direct to second letter of string without traversing the first and same for ‘q8’ also .
* From ‘q5’ we go to ‘q6’ then ‘q7’ to ‘q8’ like this we traverse first,second and third letter of the string.

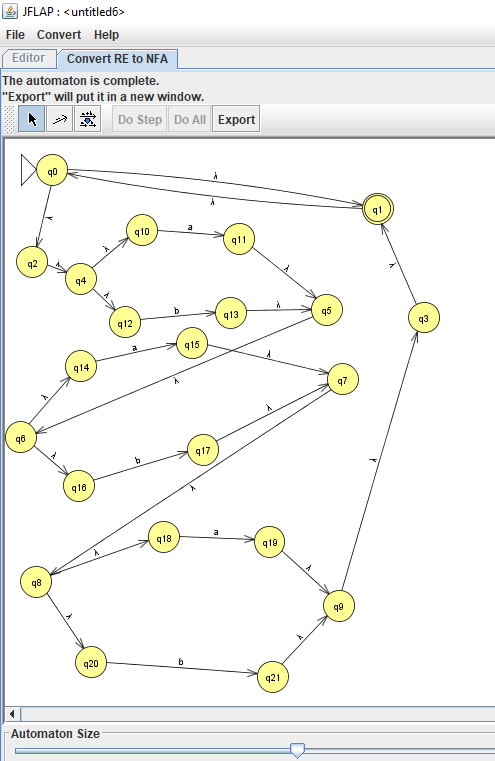
****

**Fig 4.6**

* In fig 4.6 we further divide the first expression of (a+b) which was from ‘q4’ to ‘q5’as ‘a’ and ‘b’ separately .
* ‘q4’ -> ‘q10’ -> ‘q11’ -> ‘q5’ as ‘a’ and ‘q4’ -> ‘q12’ -> ‘q13’ -> ‘q5’ as ‘b’.

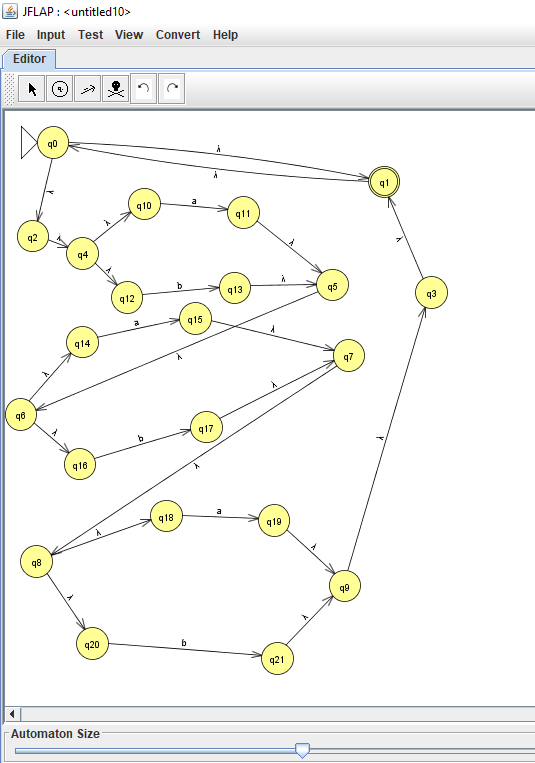


**Fig 4.7**

****

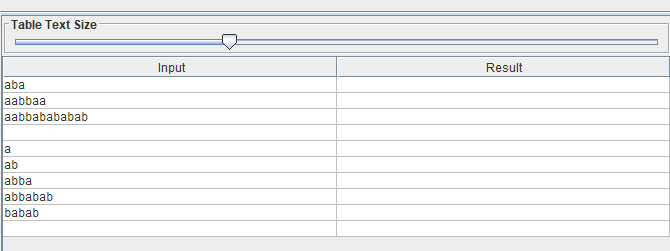
**Fig 4.8**

* Fig 4.7,4.8 shows the expansion of second and third expression of (a+b) to ‘a’ and ‘b’ with null transitions.

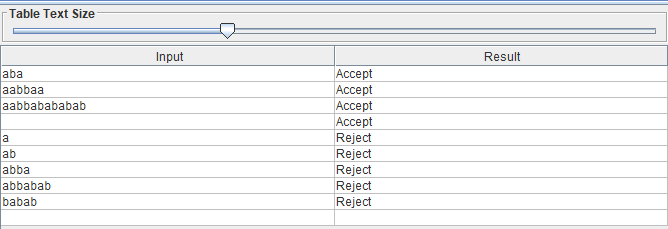


* **Fig 4.9 Complete NFA**

**5.RESULT**

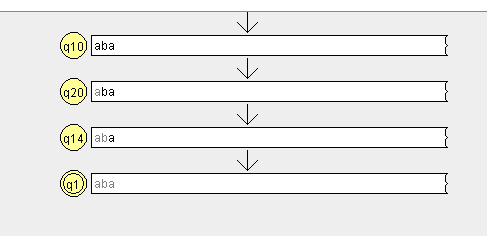
****

**Fig 5.1 Inputs**

****

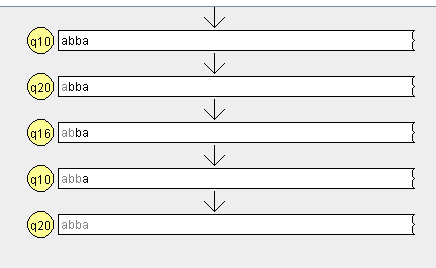
**Fig 5.2 Output**

* For inputs whose length are multiples of 3 it is accepted here aba,aabbaa,aabbabababab and null(lambda).
* For inputs such has a,ab, abba ,abbabab,babab whose lenghts are not multiples of 3 it is rejected.
* Trace for string ‘aba’ which is accepted at state ‘q1’.



**Fig 5.3**

* Trace for string ‘abba’ which is rejected at state ‘q20’ itself .

****

**Fig 5.4**

**6.REFERENCES**

* www.jflap.com
* Formal Languages & Automata Theory (by AM Padma Reddy)
* DFA (deterministic finite automata) :-Having finite number of states
* Jflap tool to solve problem statement