CS250 Lab 4 – Equivalence of Regular Expressions and Finite Automata

Objectives: In this lab you will learn how to

- analyze the equivalence of regular expressions and FAs

To see that regular expressions and Finite Automata are equivalent we must show that every regular expression must have an equivalent FA and that every FA must have an equivalent regular expression.

We will start by taking a regular expression and see how we can construct an FA to recognize the same language. The process is explained in detail on pp 80-82 of your text. Open JFLAP and select the Regular Expression button. Enter the expression `a*+ab`. We will see how we can construct an NFA to recognize this language:

1. Select `Convert to NFA`. You are given a machine that has a single initial and final state with the transition labeled with the regular expression. Clearly this is not a legal NFA since it has a regular expression on a transition. This is a generalized transition graph (GTG). We will need to spit the transition up so that it contains simple input values rather than a regular expression.
2. Select `Do Step` This will "De-or" the regular expression by removing the `+`.

Assignment 1:
Explain why the resulting "De-ORed" machine accepts either `a*` or `ab`. How does de-oring work in general?

3. Select `Do Step` again. This will "De-concatenate".

Assignment 2:
Explain how de-concatenate works in general.

4. Select the `Do Step` a final time to "De-star".

Assignment 3:
Explain ho de-star works in general.

We now must show that every FA must have an equivalent regular expression. The process is explained in detail on pp82-86 of your text and involves building a GTG with only two states (an intial and final state) and a regular expression of the transition between them.

Select the Finite Automata button and open the file ex4.1. Under the Convert menu select convert FA to RE.

1. The first step in the conversion is to make sure that every state has exactly one transition to every other state. This may involve combining transitions with an OR or adding transitions with the empty set as a label (meaning the transition does not lead to acceptance of any string). Add the empty transitions using the transition tool.

2. Next we need to remove non-initial and final states. Select the state collapser tool and click on state $q1$. A table of transitions will be shown. This table contains how the transition labels of the machine need to change (by regular expressions) if state $q1$ is removed. For example the transition from $q0$ to itself will be either the old transition of `b` OR the labels of $q0$ to $q1$ concatenated with $q1$ to itself as many times as we like, concatenated with the transition from $q1$ back to $q0$: `(b + ab*a)`.

Assignment 4:
Explain the new transition from $q0$ to $q2$.

3. Click `finalize` on the table window. If we had more states to remove we would continue this process until we have only the initial and final states left. Now the regular expression expressed by the FA is displayed at the top of the window.

Assignment 5:
Explain why the constructed FA represents the given regular expression.

Since we can convert a FA to an RE and we can convert an RE to an FA, it must be the case that these two models of computation represent the same family of languages – the Regular Languages!

Submit your files in goucherLearn for grading.