Regular Expression Identities Exercise
Martha Kosa

While each regular language is a unique set, there are many valid regular expressions that can describe it, and there are many equivalent finite automata that accept it.

Your textbook may have a section in it describing various regular expression identities. To show formally that two regular expressions are equivalent, we must show that their corresponding languages are equal as sets. To show that two sets are equal, we must show that each is a subset of the other. To show that a set S is a subset of another set T, we must show that an arbitrary element of S belongs to T, using properties of S and logic.

We can also use JFLAP's **Compare Equivalence** feature to validate these regular exprssion identities by building automata for the expressions on both sides of the identity and verifying their equivalence.

Consider the identity (xy)*x ≡ x(yx)* for arbitrary regular expressions x and y. We can choose relevant expressions to substitute for x and y. Let's say that we have a test in the next few days. We may alternate between studying and procrastination. We will substitute **study** for x and **procrastinate** for y. So we are verifying the identity (**study procrastinate)* study** ≡ **study** (**procrastinate study**)*.

Try It!

1. Build a DFA corresponding to the regular expression on the left-hand side of the identity. If you are not running JFLAP with multi-character input as the default, set multi-character input as the default by choosing ***Help > Preferences ...*** from the starting JFLAP window and selecting the appropriate radio button. Your automaton should have three states.
2. Save your DFA with a descriptive file name.
3. Build a new DFA corresponding to the regular expression on the right-hand side of the identity. Your automaton should have three states, and should look different from your previous automaton.
4. Save your DFA with a descriptive file name.
5. Now you can test for equivalence. Select ***Test > Compare Equivalence***. A dialog similar to the following should appear. Choose the file name corresponding to your first automaton built in this exercise.

1. Click the **OK** button. The following dialog should appear.

If the message in the dialog box is "They ARE NOT equivalent!", you need to go back and fix your automata.

Your first automaton should look similar to the following:

Your second automaton should look similar to the following:

Let's do another example. We have multiple identities shown below.
(x ≡ y)* ≡ (x* ≡ y)* ≡ x*(x ≡ y)* ≡ (x ≡ yx*)* ≡ (x*y*)* ≡ x*(yx*)* ≡ (x*y)*x*.

Since identities are reflexive (x ≡ x for all regular expressions x) and transitive (if x ≡ y and y ≡ z, then x ≡ z), the above statement is shorthand for a huge number of identities.

Questions To Think About

1. How many non-reflexive identities are shown above?
2. How many unique non-reflexive identities with respect to symmetry (if x ≡ y is counted, do not count y ≡ x)?
3. The identity x*(yx*)* ≡ (x*y)*x* above follows immediately from our first example by a simple substitution. What is the replacement that you need to make?

We will now consider an identity in the middle our large group of identities:

(x ≡ yx*)* ≡ (x*y*)*

This time, we will substitute **work** for x and **play** for y. Thus, we have

(**work** ≡ **play work***)* ≡ (**work***play***)***

Try It!

1. Build and save a FA (can be an NFA, assuming multi-character input as before) corresponding to the regular expression on the left-hand side of the identity.
2. Build and save a FA (can be an NFA, assuming multi-character input as before) corresponding to the regular expression on the right-hand side of the identity.
3. Test for equivalence as before.

Your first automaton should be equivalent to the following:

Your second automaton should be equivalent to the following.

Reference: Languages and Machines by Sudkamp