Student: Mark Gill

Student ID: G00348787

Module: Graph Theory

Internal Examiners: Ian McLoughlin

This project is a submission for graph theory. It revolves around the concept of non-determinate finite automata. I had to build a non-deterministic finite automaton (NFA) from an expression, then use that NFA to check if the expression matches a string. Here are the steps/algorithms I followed:

The Shunting Yard Algorithm:

The shunting yard algorithm is used to convert infix expressions to postfix. I first declare the special characters and their precedence. I then declare stack, which will be the output, as an empty string. I then declare stack, which will be the stack the operators will be on, as an empty string. I then create a for loop with infix. I made infix at the start of the shunt method. I use this for loop to cycle through the string, one character at a time. If there is an open bracket, I push to the stack. If there is a closing bracket, I pop from the stack.

If it’s an operator, push to the stack after popping an operator of equal or lower precedence from the top of the stack, to the output. After he for loop, I pop all remaining operators from stack to output. I then return the function.

Thompson’s construction:

I declare the class state. This will be used to show us the states of the arrows. I then declare the class nfa. NFA’s will be represented by the initial and accept states. I then make the compile function. This compiles postfix regular expressions into an NFA. I create the nfastack and start a for loop. This for loop will be used to convert postfix to an NFA. This goes through the expression one character at a time. If there is a “.”, pop the NFA’s off the stack, convert the first NFA’s accept state to the second NFA’s initial state and then push the stack. This is because “.” means concatenate.

If there is a “|”, pop the NFAs off the stack, create a new initial state and connect the initial state of the 2 NFAs popped off the stack. Then I create a new accept state. I connect this new accept state the accept states of the 2 NFAs popped from the stack. I then push the new NFA to the stack. This is because “|” means or.

If there is a “\*”, pop a single NFA from the stack. I then create new initial and accept states. I join the new initial state to NFA1’s initial state and the new accept state. I then join the old accept state to the new accept state and NFA!’s initial state. I then push the new NFA to the stack.

If there are none of those operators, I create new initial and accept states. I then join the initial and accept state using the c label. I then Push this new NFA to the stack.

I then pop the nfastack and return it.

Follows function:

I use the class “state” for this function. This returns the set of states that follow e arrows. First, I create a new set, with state as it’s only member. I then check if the state has any e arrows(None). Check if edge1 is a state, if it is, follow it. Check if edge2 is a state, if it is, follow it. I then return the states.

Expression matching:

This function is used to match strings to infix regular expressions. I start by shunting and compiling the regular expression. I then declare the current set of states and next set of states. I then add the initial state to the current state. I use a for loop to loop through each character in the string. I use an indented for loop to loop through the current set of states. I then check if the state is labelled with an s, if it is, add the edge1 state to the next state. When I come out of the indented for loop, I set current to next and then clear out next. I then check if the accept state is in the set of current states.

I then run a few tests with strings and infixes returning true or false values. I experimented with zip methods, list methods and tuples but ultimately stuck with the basic print function with 2 for loops. This function matches each infix to every string. I feel it offers the most robust form of testing. I then allow the user to input their own infix and string, and then return a true or false value.

Thanks for taking the time to read my doc. Please find attached, the notes I wrote down during Ian’s video to help me understand infix and postfix and regular expressions.

References:

Ian McLoughlin via LearnOnline and Microsoft Stream,

<https://pythonprogramminglanguage.com/input/>

<http://www.oxfordmathcenter.com/drupal7/node/628>

<https://swtch.com/~rsc/regexp/regexp1.html>

