**COMP 3002 Winter 2021 Assignment #6**

**Creating a Regular Expression Translator**

**Due: Tuesday Midnight March 2**

**What I’ve already done**

I have created a regular expression translator called “finiteStateMachineBuilder.st” which can convert a series of regular expressions into corresponding FSMs. You will see that it contains a class method “promptForFiniteStateMachines” to run it (where you will select one of the following two files, see below, to process). Each of those files contains FSMs in the form

name1 = regularExpression1;

name2 = regularExpression2;

namen = regularExpressionn;

**The files of finite state machines to work with…**

parserFSM.txt (use this one first)

scannerFSM.txt (use this one second)

The reason for the two files is that parser FSMs have different default attributes than scanner FSMs. There is a semantic action called “#processTypeNow:” which will run telling you what kind of FSM you are building. It will automatically halt but there is no code in it yet. You just need to record what kind of FSM you are building so that when you need to choose which kind of default transition attributes to build, you can choose the right kind.

**More details about the builder…**

The builder is already set up to process the root of the tree generated by the parser, called walkList:. You should have a look at it before you start running. Your task is to add

(a) all the other walk routines you will need to create FSMs

(b) all the FSMs building routines such as concatenate:, or:, star, plus, etc.   
 that you will need for building to work.

**What the walkList: method currently does…**

This builder already contains code to process the regular expressions one by one (expecting to get an FSM back) and stores them in a dictionary called map with the names as keys. It will also print this information so that you can decide whether or not the FSM built is correct.

A name in a regular expression that matches one of the keys in the map is expected to be replaced by it’s corresponding FSM (i.e., it plays the role of a macro). MAKE SURE THAT FSMS USED AS MACROS ARE NOT BUGGERED UP BY your routines. You can check that by simply printing the FSM again after it’s been used to see if it has changed.

As you encounter unimplemented walk routines, you will be forced to implement FSM building routines somewhere in the appropriate FSM class, sometimes instance methods and sometimes class methods. You will be done when you can process the two files without encountering any walk routines that are NOT UNDERSTOOD.

Rather than prebuilding all the routines that you THINK you will need, just build them when you get a “doesNotUnderstand” message. That way, you will be forced to implement the easier ones first.

Of course, you will be making use of the **FiniteStateMachine**, **FiniteStateMachineState**, **Transition**, and **TransitionName** classes that you implemented in the previous assignment.

**One last Issue…**

We will not be implementing & (operation AND) and – (operation MINUS) in this assignment. We will leave it for the next assignment.