NFA to DFA Homework

Jeremy Poff

The problem I set out to sold was the conversion of a provided NFA to a DFA via the Subset Construction algorithm. The NFA was provided by the user via standard in and the output is expected to be printed to standard out of the conversion process. This goal can be broken down into several smaller goals: parse the input, accurately represent the NFA internally, implement the Move algorithm, implement the Epsilon Closure algorithm, display the generated DFA. These goals were achieved in the following way.

I chose to implement my solution in Python. Using Python made parsing the input a trivial exercise; the parse NFA is held internally in a class. The Move algorithm is shown in the slides as an operation on sets, this is convenient since Python has a Set class and corresponding utilities for it. I used the below relation to implement my Move operation.

Move([1,2], “a”] = Move(1, “a”) U Move(2, “a”)

The Epsilon Closure operation also has this property, so it was implemented in a similar manner. For getting the new DFA state the below equation is used.

Statenew = Closure(Move(S, t))

My implementation has almost one line of code for each line of pseudocode in the slides; this is do to the high level nature of Python. I generally try to use the functional aspects of Python as much as possible to code. This is particularly noticeable in the parsing phase of my program.

For testing my program, I visually examined the results for the first two provided input files. For the third file, I copied all of the expected DFA states into a function that returned them as a set, this allowed me to halt the program if an unexpected state was seen thus decreasing the number of log messages between me and the problematic code. For this to work I also had to implement a do\_sort and a hashable\_list function since Python lists are not sorted and are not hashable so [1,2] != [2,1] even though in our case, since they are really sets, this is not true.

The biggest issue I had was initially thought the referenced figures were in the book. When I couldn’t find them I tried to implement the solution the way we were taught in 312. This didn’t work at all since they algorithms are not at all the same thing. The second biggest issue was initially I was sorting the list of strings instead of sorting them as ints before making them hashable, this gave me incorrect orderings and made comparing my results to the solutions difficult. The last major issue was that my reindexed DFA states had the wrong indexes. This was because the search for the new DFA states was being done in queue order instead of stack order. The search for epsilon closures was being done in stack order. Once I figured this out my solution exactly matched the expected results.