CS461 PA 1: NFA to DFA

Ksenia Burova

September 8th, 2017

Abstract:
In this programming assignment, the goal was to implement ‘the subset construction‘ algorithm to convert NFA to the equivalent DFA. In addition to major algorithm, ‘e-closure‘ and ‘move‘ algorithms had to be implemented to be used in the subset construction.

Input / Output:
Program takes an NFA transition table on **stdin** along with the information about *start state, accept states, total number of states.*
Program outputs DFA on **stdout** in a same format as NFA. In addition, an intermediate information about what sets of states form NFA represent what states in DFA has to be displayed.
In transition table, 'e' transition is represented with symbol 'E'.

Language used: C++

Programming approach:
I've built my program incrementally by trying to identify what information I would have to store and operate on, and what methods I would need to use for parsing data and implementing algorithms. I've done couple of examples by hand on paper to understand the logic first.

- **Design**
  I've created one struct to hold the information about finite automata, NFA and DFA. It holds transition table, start state, accept states and number of states.

  ```
  struct FA {
      int initState;
      set <int> finalStates;
      int numOfStates;
      vector < unordered_map < char, set <int> > > transTable;
  };
  ```

  My transitions table is a vector that represents all the states in FA, each vector has an unordered_map (hash table) that has transitions symbols ('a', 'b',...'E') as keys and sets of states corresponding to that transition as values.I've decided to use unordered_map

to store transitions because of constant access time, whereas keys don't have to be numbers. And when I search for 'E' transition for example, it takes O(1) to return me a set of states for that transition. I use set data structure for sets because I want them to be sorted.
In addition to my struct, I have a class that holds mutual data for nfa and dfa, like intermediate Dtrans table and symbols for transitions, and that implements methods for converting from one FA to another. There are also some helper functions implemented.

```
class nfa2dfa{
    private:
        FA nfa, dfa;
        vector < set <int> > Dstates;
        vector < char > symbols;
    public:
        void ParseNFA();
        set<int> ConstructEclosure( const set <int> &states,
                bool & isFinal );
        set<int> ConstructEclosure( int state, bool & isFinal );
        void closureHelper(
                vector < unordered_map <char, set <int> > > & tTable,
                int state, set <int> & eClosureSet, bool & isFinal);
        set<int> move(set<int> & T, char symbol);
        void constructSubset();
        void printFA(bool option);
};
```

My **Parse()** function parses stdin NFA information into an FA object.
I have a recursive method that goes through all the e-transitions from some state to calculate e-closure for that state. This method is used as a helper method for **eClosure()** methods. I have two of those, one calculates e-closure for a state, another calculates an e-closure for a set of states.
Move() function detects in what set of states you may end up starting from some other set on some transition symbol.
constructSubset() implements an algorithm for constructing Dtran table, and outputs intermediate data.

Finally, I have set to string and string to set methods that are helpers for working with input and output:

```
string SetToString(const set<int> &s);
void StringToSet(set<int> &s, const string &setStr);
```

- **Testing:**
  I've tested my program with provided to us input and output. I tried to match the output format for easier check. I've also tested each function individually by writing some small unit tests.

- **Debugging:**
  There were no that many instances when I had to debug my code, but I've used CLion IDE and its debugger for that purpose.
- **Issues:**
  The were 2 issues I ran into while implementing this algorithm. First, I forgot that it is possible that one state can return to itself through multiple e-transitions, and I ran out of stack in my recursive function. Second, I kept forgetting about 1-indexing for states instead of 0-indexing. That messed up my search algorithm a few times.