P1: NFA to DFA Converter

Connor Minton | CS 461 | September 16, 2014

Introduction

In the lexical analysis phase of program compilation, special care is taken to ensure that each lexeme constitutes a valid token under the language. One can simplify this task by representing valid patterns as regular expressions, which in turn can be represented by nondeterministic finite automata (NFAs). The drawback of the NFA is that an automaton simulator would invariably have to try multiple transitions when one input symbol satisfies more than one of them, or when states branch on the empty string ε—thus the term *nondeterministic*. Obviously, this is very costly, and such is the motivation behind an algorithm for converting these structures to a deterministic form.

In this project, I created a program in C++ to convert a given NFA to an equivalent deterministic finite automaton (DFA).

Implementation

The method of design I chose for this project featured a bottom-up approach, whereby I designed and tested skeletal, specific, fundamental structures and algorithms before composite algorithms. The keystone of my implementation revolves around the classes `nfa2dfa`, which contains the NFA and DFA data, and `nfa2dfa::fsa`, which outlines the structure for the data. In the latter class is an object `transitions` of type `map<int, map<char, vector<int>>>`. This is effectively a function that relates each unique state ID to another function that relates an input symbol to the IDs of subsequent states under that symbol. In other words, `transitions[5]['a']` would give us the vector of IDs to which it is possible to transition from state 5 under the character 'a'. A vector of integers in `nfa2dfa` named `inputs` stores the alphabet of the language, or at least the symbols defined by the input NFA.

The conversion algorithm is implemented using an object of type `map<int, vector<int>>` to relate unique DFA keys to sets of NFA keys. Unmarked states are kept via a queue. When a unique DFA state is discovered, it is added to the queue. Thus, to "mark" a state is to simply pop it out, disabling it for reuse. When it is necessary to identify a unique set of states in the NFA, I combine the set and vector data structures. The set structure allows the insertion of elements without accidental duplication. Then, the sets can be copied into vectors and compared to see if a set already exists. (I might have implemented this differently if I had not already based my interface algorithms on vectors.)

Debugging

Debugging was a very simple process, as I was sure to thoroughly test every fundamental component before proceeding to use them in aggregate structures. I used Git to track the development of the project. Each of the following stages was separated and rigorously tested before moving on to the next:

1. Parse input and print input to stdout
2. Implement skeletal data structures for FSAs
3. Redirect input into data structures and write print function. This print function accomplishes the additional purpose of debugging the input functions.
4. Implement epsilon closure algorithm
5. Implement conversion algorithm
6. Finalize project (clean up output, comment and organize code, remove debug constructs, etc.)

The debug tools I used were primitive (statements to stdout), but with careful separation of components, the bugs I faced were trivial.

Difficulties

The most intricate and time-consuming of the fixes ended up being the addition of one line of code (line 258). At the time just before this fix, my output was close to the solution for DFA IDs introduced earlier in the algorithm, but the algorithm broke down after about the third ID. I hypothesized that this was because the algorithm had no method of stopping the insertion of an empty set as a unique ID. Such an empty set was creatable by generating the set *move(T, a)* where there exist no transitions on/from *T* under *a*. This allowed subsequent sets generated in the same manner to be matched to the new "null ID." Line 258 detects the null set and skips the insertion, thus preserving the integrity of the map from DFA states to NFA states.