The program was written in Racket. A parser was created by using the brag module language which gives tools for easily constructing a tokenizer that extracts the meaningfull segments of the input regex and for constructing a grammar that organizes the segments into an AST in the form of a Lisp S-expression. One and the same struct was used to represent each kind of automata with elements for states, transitions, start and end states and the alphabet. A transition was represented as a triple, a list of three elements (s,d,a) where s is source state, d is destination state and a the alphabet element connecting the states.

The course book describes the basic algorithm of how to expand the e-NFA for each kind of AST node. This implementation uses a similar algorithm but it skips many empty transitions to reduce the NFA size. For instance, the union expression would produce 4 empty transitions but it produces no empty transitions in this implementation. It simply recursively passes on the two subexpressions of the union for further construction, linking the source and destination states by transitions in the subexpression.

One could not simulate the e-NFA directly since it is non deterministic and the program would not know which transition edge to pick or in this implementation remember which transition edges have been picked if all correct routes would be attempted. The DFA could on the other hand be simulated directly, although it would be slow in many cases.

The most intereseting test case was number 10 that in the implementation is much slower than the other test cases. A way to make it much faster is to store transitions in vectors that are indexed by states. Now they are just organized flat in a list which necessitates traversal of the entire list rather than just the part represented by the state that is the source of the transition.

The reason why the test case even terminates is because the e-NFA is created more economically as described above, and an alternative implementation for the equivalence table is used than the one in the course book. This implementation attempts to mark as many state relations as distinguishable as possible at the outset before attempting the algorithm from the book. This means that most states are marked as distinguishable when the algorithm from the course book begins which stops most equivalence searches at the outset for the state pairs. The premarking phase is implemented by following the transitions that lead to the end states from source states that are distinguishable from the end states. By remembering the alphabet element and source state that leads to the end state, most of a row can be marked as distinguishable in the table by equivalence testing that source state on that alphabet element to every other state. This process is recursively repeated taking the adjacent source states to the end states as a new set of destination states that are distinguishable from most other states in the table.