19115045 Theory of Computation

Kunal Sachdeva Submission Date:

3rd Semester CSE 16/12/2020

TERM PROJECT

Finding non-important states in Automata

Theory:

An automaton with a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM).

Formal definition of a Finite Automaton

An automaton can be represented by a 5-tuple (Q, ∑, δ, q0, F), where −

* Q is a finite set of states.
* ∑ is a finite set of symbols, called the alphabet of the automaton.
* δ is the transition function.
* q0 is the initial state from where any input is processed (q0 ∈ Q).
* F is a set of final state/states of Q (F ⊆ Q).

There are four major families of automaton :

1. Finite-state machine

2. Pushdown automata

3. Linear-bounded automata

4. Turing machine

Finite-state Automata:

Finite automata are used to recognize patterns.

It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.

At the time of transition, the automata can either move to the next state or stay in the same state.

Finite automata have two states, accept state or reject state. When the input string is processed successfully, and the automata reached its final state, then it will accept.

There are two types of finite automata:

1. DFA(deterministic finite automata)
2. NFA(non-deterministic finite automata)

1. DFA:

DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

2. NFA:

NFA stands for non-deterministic finite automata. It is used to transmit any number of states for a particular input. It can accept the null move.

Starting from certain states in a finite state automaton once can never reach a final state. Therefore, removing these states will not make any difference to the language accepted by the automaton. These states are called useless states.

The algorithm is quite straightforward. Rather than constructing the set of useless states, we will calculate its complement: the set of useful states. Clearly, all final states are useful. Similarly, any state which has a transition to a useful state is, itself useful. The algorithm basically repeats this step until no new useful states are discovered.

Algorithm:

1. Initialise the set of useful states U to the set of final states.

2. Calculate the set of states M which can go to a state in U in one step. Thus M will include all states q such that for some input symbol a and state q' in U, (q, a) 🡪 q' is a production rule.

3. Update U to be the union of M and the old value of U.

4. If new elements have been added to U in the last step, jump to step 2.

5. Remove all states in the automaton not in the final value of U and transitions from or into them.

6. If the initial state has been removed, add a new state with no related transitions and set it to be the initial state.

Note that the last step is necessary since any finite state automaton requires to have an initial state.

**Example:**

Consider the following finite state automaton:

b

a

a

**B**

**S**

**A**

a

b

**C**

1. Initialise the set of useful states to the final states:

U={ A }

2. Calculate the set M of all states which can go to a state in {A} in one transition:

M={ S, A }

3. Update U by taking the union of its old value to that of M:

U={ A } U { S, A } = { S, A }

4. U has changed value and we therefore jump back step 2 of the algorithm. Calculate the set M of all states which can go to a state in {S, A} in one transition:

M={ S, A }

5. Update U by taking the union of its old value to that of M:

U={ S, A } U { S, A } = { S, A }

6.Since no new elements have been added to U, we can now calculate the set of useless states by taking the complement of U:

{ B, C }

7. Remove all useless states (and related transitions) from the automaton:

a

a

**S**

**A**

8. The automaton still has an initial state and is therefore valid.