1 Finite state machine minimization

- *Algorithm:* Finite state machine minimization (algo. 1)
- *Input:* A finite state machine
- *Complexity:* Average complexity $O(n \log \log n)$
- *Data structure compatibility:* Finite state machine (Deterministic finite automaton)
- *Common applications:* compilers, network protocols, theory of computation

Finite state machine minimization

Given a finite state machine, minimize the states.

Description

1. Definition of FSM

   The formal definition of a finite state machine (deterministic finite automaton) is

   $$M: \ (Q, \Sigma, \delta, q_0, F)$$

   (a) Q: finite set of states

   (b) Σ: finite set of input symbols

   (c) $\delta : Q \times \Sigma \to Q$: transition function

   (d) $q_0 \in Q$: initial state

   (e) $F \subseteq Q$: accept state

   The automaton will accept a string w if it starts at start state q_0, and given each character in w, the transition rule will transit state to state according to δ, and the final state shall halt at F states.

2. Input

   The input of the algorithm shall be a well-defined finite state machine. Any illegal input should not be considered.

3. Complexity

   For the complexity calculation, it is given by 2 theorems. The first theorem is described as :

   > For any fixed integer $k \geq 2$ and for the uniform distribution over the deterministic and complete automata with n states over a k-letter alphabet, the average complexity of Moores state minimization algorithm is $\mathcal{O}(n \log \log n)$.

The proof of this theorem is generally as: The average number of iteration is given by

$$N_n = \frac{1}{A_n}(\sum_{i=0}^{n-2}(i+1) * |A_n|)$$

Then define $\lambda_n = \lceil \log_k \log_2 n^3 + 2$, then an upper bound can be obtained as

$$N_n \leq \frac{\lambda_n + 1}{|A_n|}\sum_{i \leq \lambda_n}|A_n^i| + \frac{5\log_2 n + 1}{|A_n|}\sum_{i=\lambda_n+1}^{5\log_2 n}|A_n^i| + \frac{n-1}{|A_n|}\sum_{i=5\log_2 n+1}^{n-2}|A_n^i|$$

And we find the third term to have time complexity $\mathcal{O}(1)$. And the rest term has time complexity $\mathcal{O}(n \log\log n)$

4. Application

   In computer science field, finite state machine can be used to treat string. It can set up several states to check whether to accept an arrival of strings or not.

   In digital circuits field, finite state machine can be used to depict the behavior of a certain circuit, and it could link the combinatoric circuits with a desired function.

   Finite State Machine can also be used to depict a lot of behaviors in natural science or social science. For example, it could be applied to analyze the relation in different social characters, and to analyze how an social event is carried out.

5. Detailed Algorithm

   This algorithm is introduced by *Hopcroft*. The idea behind is called **partition refinement**, which means partition a large set into several small sets by their behavior, and after each iteration, the partition will be improved, till we reach the final result when the partition can no longer be refined. The detailed way of the algorithm works as follows.

   (a) At the very beginning, the are partitioned into two different groups, that is $\{F\}$ and $\{Q\backslash F\}$. These two groups are accepting states and rejecting states, obviously they are inequivalent.

   (b) Then it comes with the magic of this algorithm. That it separates $\{A\}$ with $\{W\backslash A\}$, where W initially to be F. Then it separates the states whose result will lead to different sets which have been separated before, namely, after the δ transition, the next states from the current states are not equivalent.

   (c) And if we keep going with this method, when we reach an iteration after which the result no longer changes, we can tell that we reach the optimized FSM.

Algorithm 1: FSM minimization

Input : A formally defined FSM
Output: Minimized FSM

Function `MinimizeFSM`(M : $(Q, \Sigma, \delta, q_0, F)$)**:**
  set $P \leftarrow \{F, Q \backslash F\}$; /* Described in Description.5.(a) */
  set $W \leftarrow \{F\}$;
  while $W \neq \varnothing$ **do**
    Choose $A \in W$;
    $W \leftarrow W \backslash A$;
    for s *in* Σ **do**
      $X \leftarrow \{X : \delta(X \times c) \rightarrow A\}$; /* In natural language, it means that X is a state that: the transition rule takes c in the state X will go to a state in set A */
      for $Y \subset P$ *s.t.* $X \cap Y \neq \varnothing$ **and** $Y \backslash X \neq \varnothing$ **do**
        /* These two steps partition one set into two other inequivalent sets according to the definition of X */;
        remove Y in P;
        add $X \cap Y$ and $Y \backslash X$ to P;
        if Y *in* W **then**
          remove Y in W;
          add $X \cap Y$ and $Y \backslash X$ to W;
        else
          /* These two steps add the newly found smaller set of equivalent states to W */
          if $|X \cap Y| \leq |Y \backslash X|$ **then**
            add $(X \cap Y)$ to W;
          else
            add $(Y \backslash X)$ to W;
          end if
        end if
      end for
    end for
  end while
end
return P

References

- Foundation of Data Science, 2017, https://www.cs.cornell.edu/jeh/book.pdf
- Lawson, Mark V. (2004). Finite automata. Chapman and Hall/CRC. ISBN 1-58488-255-7.
- Sakarovitch, Jacques (2009). Elements of automata theory. Translated from the French by Reuben Thomas. Cambridge: Cambridge University Press. ISBN 978-0-521-84425-3.