Pass-3 reading section II. BACKGROUND

**Synchronizing Sequences**

As we covered in class, a synchronizing sequence will take different machines to the same final state like homing sequence. But we didn’t cover how to determine whether a FSM (finite state machine) has such a sequence or not and how to create it in polynomial time in class. So, I am most interested in the algorithm to find a synchronizing sequence of a FSM, which is described in D of Part II, Background.

How can we figure out whether the synchronizing sequence exists or not? We need to construct an auxiliary directed graph $G \times G$ with $n(n+1)/2$ nodes firstly. To be specific, we build such auxiliary graph by the following 3 steps.

**Step 1:** State becomes unordered pairs $(s_i, s_j)$ and we can have two identical states such as $(s_i, s_i)$. For example, if we have $s_1, s_2, s_3$, the pairs in auxiliary directed pairs could be:

$$(s_1, s_1), (s_1, s_2), (s_1, s_3), (s_2, s_2), (s_2, s_3), (s_3, s_3)$$

Apparently, we will get $n(n+1)/2$ pairs in total.

**Step 2:** Add directed edges between two states $(s_i, s_j) \rightarrow (s_p, s_q)$ if and only if there is a transition from $s_i$ to $s_j$ and a transition from $s_p$ to $s_q$ and these two transitions are both labeled by a.

**Step 3:** Synchronizing sequence exists if and only if there is a path in $G \times G$ that every pairs $(s_i, s_j)$ $(i \neq j)$ has a directed link to the same node $(s_r, s_r)$.

We can determine whether a synchronizing sequence exists or not for FSM by finding such $s_r$ after we construct such an auxiliary graph. The running time is $O(pn^2)$ to check if such a sequence exits because we use BFS (breadth-first-search) to check if a node is reachable or not.

Now, we can find a synchronizing sequence with the help of such an auxiliary graph. This algorithm is done by following 3 steps:
Step 1: Find shortest path from \((s_i, s_j)\) \((i \neq j)\) to \((s_r, s_r)\) and denote the input along the path by \(x_i\)

Step 2: Get \(S_i = \delta(S, x_i)\), take two distinct states from \(S_i\) and construct another input sequence \(x_2\) to the same state \((s_r, s_r)\)

Step 3: Stop when we only have one state \(s_r\) and concatenation \(x\) of input sequence \(x_1, x_2, \ldots, x_k\), which is the synchronizing sequence.

I will further explain this algorithm by giving a specific example. Figure 1 shows the transition diagram of a FSM, Figure 2 is the auxiliary graph accordingly.

---

**Fig. 1** Transition diagram of a FSM

**Fig. 2** The auxiliary graph

Step 1: \((s_2, s_3) \overset{a}{\rightarrow} (s_2, s_2)\), \(x_1 = a\)

Step 2: \(S_i = \delta(S, a) = \{s_1, s_2\}\) because via the input \(a\) which takes the machine to state \(S_i\) (if it starts in \(S_i\)) and \(S_2\) (if it starts in \(S_2\) or \(S_3\)).
Step 3: \((s_1, s_2) \xrightarrow{b} (s_2, s_3) \xrightarrow{a} (s_2, s_2)\), \(x_2 = ab\), \(S_2 = \delta(S_1, ba) = (s_2)\), stop,

\(x = x_1x_2 = aba\)

The running time is \(O((n-1)^3)\) with best algorithm, which is introduced by a paper from references.