A STUDY ON AUTOMATA AND FINITE STATE MACHINES

Ege Sorguç sorguc.ege@metu.edu.tr
Computer Science and Engineering

Yuşa Emek myusa@sabanciuniv.edu
Computer Science and Engineering

Berk Çirişçi
Computer Science and Engineering berkcirisci@sabanciuniv.edu

Project Supervisors:

Hüsnü Yenigün
Computer Science and Engineering

Kamer Kaya
Computer Science and Engineering

Abstract

Computing a shortest synchronizing sequence of an automaton is an NP-Hard problem. There are well-known heuristics to find short synchronizing sequences. Finding a shortest homing sequence is also an NP-Hard problem. Unlike existing heuristics to find synchronizing sequences, homing heuristics are not widely studied. In this paper, we discover a relation between synchronizing and homing sequences by creating an automaton called homing automaton. By applying synchronizing heuristics on this automaton we get short homing sequences. Furthermore, we adapt some of the synchronizing heuristics to construct homing sequences.

Keywords: FSM, Automaton, Synchronizing, Homing

1. Introduction

A Deterministic Finite Automaton (DFA) (or simply an automaton) is a triple $A = (S, X, \delta)$ where S is a finite set of states, X is a finite set of alphabet (or input) symbols, and $\delta : S \times X \rightarrow S$ is a transition function. When δ is a complete (resp. partial) function, A is called complete (resp. partial). In this work we only consider complete DFAs unless stated otherwise. A Deterministic Finite State Machine (FSM) is a tuple $M = (S, X, Y, \delta, \lambda)$ where S is a finite set of states, X is a finite set of alphabet (or input) symbols, Y is a finite set of output symbols, $\delta : S \times X \rightarrow S$ is a transition function, and $\lambda : S \times X \rightarrow Y$ is an output function. In this work, we always consider complete FSMs, which means the functions δ and λ are total.

An automaton and an FSM can be visualized as a graph, where the states correspond to the nodes and the transitions correspond to the edges of the graph. For an automaton the edges of the graph are labeled by input symbols, whereas for an FSM the edges are labeled by an input and an output symbol. In Figure 1 and Figure 2, an example automaton and an example FSM are given.

An input sequence $\bar{x} \in X^*$ is a concatenation of zero or more input symbols. More formally, an input sequence $\bar{x} = x_1, x_2 \ldots x_k$ for some k ≥ 0 where $x_1, x_2 \ldots x_k \in \Sigma$. As can be seen from the definition, an input sequence may have no symbols; in this case it is called the empty sequence and denoted by ϵ.

For both automata and FSMs, the transition function δ is extended to input sequences as follows. For a state s ∈ S, an input sequence $\bar{x} \in X^*$ and an input symbol x ∈ X, we let $\bar{\delta}(s, \epsilon) = s$, $\bar{\delta}(s, x\bar{x}) = \bar{\delta}(\delta(s, x), \bar{x})$. Similarly, the output function of FSMs is extended to input sequences as follows: $\bar{\lambda}(s, \epsilon) = \epsilon$, $\bar{\lambda}(s, x\bar{x}) = \lambda(s, x)\,\bar{\lambda}(\delta(s, x), \bar{x})$. By abusing the notation we will continue using the symbols δ and λ for $\bar{\delta}$ and $\bar{\lambda}$, respectively.

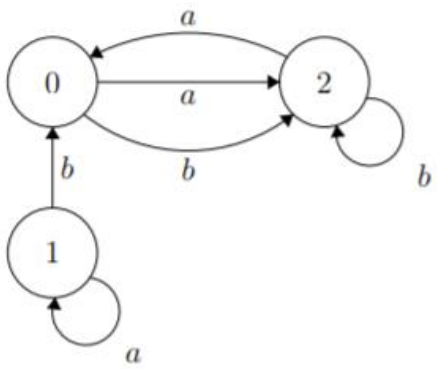

Figure 1: An automaton A_0

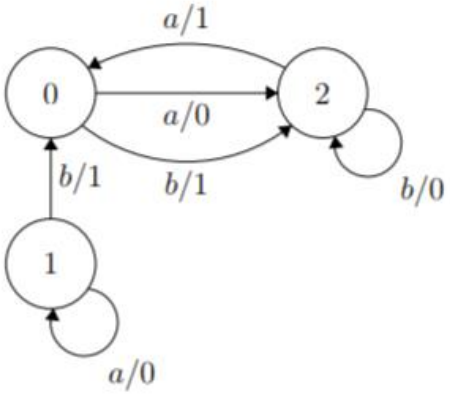

Figure 2: An FSM M_0

Finally for both automata and FSMs, the transition function δ is extended to a set of states as follows. For a set of states S' ⊆ S and an input sequence $\bar{x} \in X^*$, $\delta(S', \bar{x}) = \{\delta(s, \bar{x}) \mid s \in S'\}$. An FSM M = (S, X, Y, δ, λ) is said to be minimal if for any two different states $s_i, s_i \in S$, there exists an input sequence $\bar{x} \in \Sigma^*$ such that $\lambda(s_i, \bar{x}) \neq \lambda(s_j, \bar{x})$.

Definition 1. For an FSM M = (S, X, Y, δ, λ), a Homing Sequence (HS) of M is an input sequence $\bar{H} \in X^*$ such that for all states s, s' ∈ S, $\delta(s, \bar{H}) \neq \delta(s', \bar{H}) \Rightarrow \lambda(s, \bar{H}) \neq \lambda(s', \bar{H})$.

Intuitively, an HS $\bar{H}$ is an input sequence such that for all states output sequence to $\bar{H}$ uniquely identifies the final state. For FSM M_0 given in Figure 2 aa is an HS. If M is minimal, then there certainly exists an HS for M [2]. If M is not minimal, there may also be an HS for M. Furthermore, when M is not minimal, it is always possible to find an equivalent minimal FSM M' such that M' is minimal [2].

Definition 2. For an automaton A = (S, X, δ), a Synchronizing Sequence (SS) of A is an input sequence $\bar{R} \in X^*$ such that $|\delta(S, \bar{R})| = 1$.

A synchronizing sequence is also called a reset sequence in the literature. An automaton does not necessarily have an SS. It is known that the existence of an SS for an automaton can be checked in polynomial time [1]

For a set of states C ⊆ S let $C^2 = \{\{s, s'\} / s,s' \in C^2\}$ be the set of multisets with cardinality 2 with elements from C, i.e C^2 is the set of all subsets of C with cardinality 2, where repetition is allowed. An element {s,s'} ∈ C^2 is called pair. Furthermore it is called singleton pair (or an s-pair or simply a singleton) if s= s', otherwise it is called different pair (or a d-pair). The set s-pairs and d-pairs denoted in C^2 denoted by C_d^2 and C_s^2respectively. A sequence w is said to be merging sequence for a pair {s,s'} ∈ S^2 if δ({s,s'}, w) is a singleton. Note that for a s-pair, every sequence (including ε) is a merging sequence.

2.1 The Relation between Homing Sequences and Synchronizing Sequences

In this section, we will derive an automaton AM from a given FSM M. We call A_M the the homing automaton of M. The construction of A_M from M is similar to the construction of product automaton (as exists in the literature) from a given automaton to analyse the existence of and to find synchronizing sequences.

Definition 3. Let M = (S, X, Y, δ, λ) be an FSM. The homing automaton (HA) A_M of M is an automaton A_M = (S_A, X, δ_A) which is constructed as follows:

The set S_A consists of all 2-element subsets of S and an extra state q* . Formally we have S_A = {{s, s'} | s, s' ∈ S ∧ s ≠ s'} ∪ {q*}.

- For a state q ∈ S_A and an input symbol x ∈ X, the transition function δ_A is defined as follows:

$$\delta_A(q,x) = \begin{cases} \{\delta(s,x), \delta(s',x)\} & , \ (q = \{s,s'\}) \wedge (\delta(s,x) \neq \delta(s',x)) \wedge (\lambda(s,x) = \lambda(s',x)) \\ q^\star & , \ (q = \{s,s'\}) \wedge (\delta(s,x) = \delta(s',x)) \\ q^\star & , \ (q = \{s,s'\}) \wedge (\lambda(s,x) \neq \lambda(s',x)) \\ q^\star & , \ (q = q^\star) \end{cases}$$

As an example, the HA for FSM M_0 given in Figure 2 is depicted in Figure 3.

Lemma 1. Let M = (S, X, Y, δ, λ) be an FSM, A_M = (δ_A, X, δ_A) be the HA of M. For an input sequence $\bar{x}$∈ X* , (λ(s_i , $\bar{x}$) ≠ λ(s_j , $\bar{x}$)) ∨ (δ(s_i , $\bar{x}$)= δ(s_j , $\bar{x}$)) ⟸⟹ δA({s_i , s_j}, $\bar{x}$)= q*.

Proof. Let $\bar{x}$ = $\bar{x}$'x$\bar{x}$'' where $\bar{x}$' , $\bar{x}$'' ∈ X* and x ∈ X such that (δ(s_i , $\bar{x}'$) ≠ δ(s_{ij} , $\bar{x}$ ')) ∧ (λ(s_i , $\bar{x}'$) = λ(s_j , $\bar{x}'$)). So the new state according to δA is q' = {δ(s_i, $\bar{x}'$), δ(s_j , $\bar{x}'$) }. If λ(s_i , $\bar{x}'$x) ≠ λ(s_j , $\bar{x}'$ x) then δA(q' , x) = q* and δA(q* ,$\bar{x}''$) = q* . If λ(s_i , $\bar{x}'$x) = λ(s_j , $\bar{x}'$ x) but δ(s_i , $\bar{x}'$x) = δ(s_j , $\bar{x}'$x) then δA(q' , x) = q* and δA(q* , $\bar{x}''$) = q* .

For the reverse direction, again writing $\bar{x}' = \bar{x}'x\bar{x}''$ where $\bar{x}'$, $\bar{x}''$∈ X* and x ∈ X such that δA({s_i , s_j}, $\bar{x}'$) = {s_i' , s_j'} for some states s_i' , s_j' and δA({s_i , s_j}, $\bar{x}'$x) = q* . This means that λ(s_i , $\bar{x}'$) = λ(s_j , $\bar{x}'$) and δ(s_i , $\bar{x}'$) ≠ δ(s_j , $\bar{x}'$) but after consuming input x, λ(s_i , $\bar{x}'$x) ≠ λ(s_j , $\bar{x}'$x) where x is the

first input which forces FSM to produce different outputs. Or δ(s_i , $\bar{x}'x$) = δ(s_j , $\bar{x}'$x), in this case x is the merging input. This proves that (λ(s_i , $\bar{x}$) ≠ λ(s_j , $\bar{x}$)) ∨ (δ(s_i , $\bar{x}$) = δ(s_i, $\bar{x}$)).

We can now give the following theorem that states the relation between HSs of M and SSs of A_M. Theorem 1. Let M = (S, X, Y, δ, λ) be an FSM and A_M = (δ_A, X, δ_A) be the HA of M. An input sequence $\bar{x}$ ∈ X* is an HS for M iff $\bar{x}$ is an SS for A_M.

Proof. If $\bar{x}$ is an HS of M, for any two states si and sj in M, λ(s_i , $\bar{x}$) ≠ λ(s_j , $\bar{x}$) or δ(s_i , $\bar{x}$) = δ(s_j , $\bar{x}$). Lemma 1 states that δA({s_i , s_j }, $\bar{x}$) = q* for any {s_i , s_j}∈ SA. For q* , δA(q* , $\bar{x}$) = q* . Hence $\bar{x}$ is an SS for A_M.

If $\bar{x}$ is an SS for AM, first note that δA(q* , $\bar{x}$) = q* . For any state of AM of the form {s_i , s_j }, we must also have δA({s_i , s_j }, $\bar{x}$) = q* . From the other direction of Lemma 1, we have for any pair of states λ(s_i , $\bar{x}$) ≠ λ(s_j , $\bar{x}$) or δ(s_i , $\bar{x}$) = δ(s_j , $\bar{x}$).

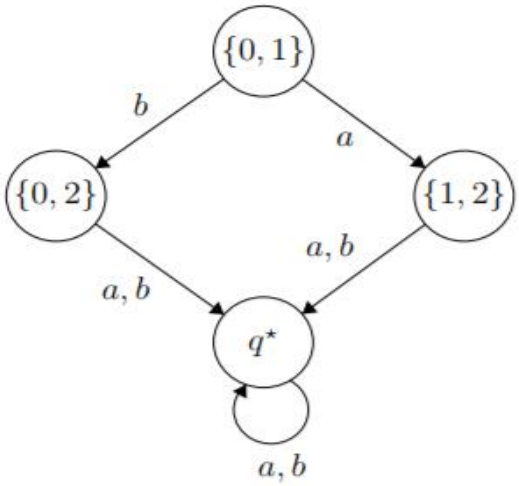

Figure 3: A_M of M_0 in Figure 2

2.2 Synchronizing Heuristics for Homing Automaton

Although finding a shortest SS is known to be an NP-Hard problem, finding a short SS by applying heuristics is studied widely. As shown in Theorem 1, finding an HS for an FSM M is equivalent to finding an SS for its corresponding homing automaton A_M. Based on Theorem 1, we can apply widely known SS heuristics to A_M in order to find a possibly short HS for FSM M.

Among such SS heuristics, Greedy is one of the fastest and the earliest that appeared in the literature [1]. Other than Greedy heuristic, SynchroP is one of the best known heuristics in terms of

finding short SSs [3]. Although, SynchroP is good in terms of length, it is slow compared to Greedy since it performs more analysis on the automaton.

Both Greedy and SynchroP heuristics have two phases. Phase 1 is common in these heuristics and given as Algorithm 1 below. In Phase 1, a shortest merging word $\tau_{\{i,j\}}$=for each {i, j} ∈ S^2 is computed by using a breadth first search. Note that $\tau_{\{i,j\}}$= is not unique.

Algorithm 1 performs a breadth first search(BFS), and therefore constructs a BFS forest, rooted at s-pairs {i, i} ∈ S_s^2, where these s-pair nodes are the nodes at level 0 of the BFS forest. A d-pair {i, j} appears at level k of the BFS forest if $|\tau_{\{i,j\}}|$ = k. When the automaton has a synchronizing sequence, each d-pair should have a merging word since it is a necessary condition for synchronizing automata. . It requires Ω(n^2) time since each {i, i} ∈ S_s^2 pushed to Q exactly once.

Algorithm 1 Phase 1 of Greedy and SycnhroP

Input : An automaton $A(S, \Sigma, \delta)$
Output : A merging word for all $\{i, j\} \in S^2$

```
1: Q ← an empty queue                          ▷ Q: BFS frontier
2: P ← ∅                                       ▷ P: the set of nodes in the BFS forest constructed so far
3: for {i, i} ∈ S_s^2 do
4:     push {i, i} onto Q
5:     insert {i, i} into P
6:     set τ_{i,i} ← ϵ
7: end for
8: while P ≠ S^2 do
9:     {i, j} ← pop next item from Q
10:    for x ∈ Σ do
11:        for {k, l} ∈ δ^{-1}({i, j}, x) do
12:            if {k, l} ∉ P then
13:                τ_{k,l} ← x τ_{i,j}
14:                push {k, l} onto Q
15:                P ← P ∪ {{k, l}}
16:            end if
17:        end for
18:    end for
19: end while
```

Next phase of these heuristics differ from the selection criterion for the current d-pair to be merged. Greedy's Phase 2 (given as Algorithm **2** below) constructs a synchronizing sequence by using the information from Phase 1. It keeps track of a set of active states C, which is the set of states that are not merged yet. A d-pair {i, j} that has shortest merging word in the active states is selected to be merged and current(active) state set is updated by applying $\tau_{\{i,j\}}$. This process will iterate until all states are merged so that C becomes singleton.

Similar to the second phase of Greedy, the second phase of SynchroP also constructs a synchronizing sequence iteratively. It also keeps track of a set of active states C of states. In each iteration, the cardinality of C is reduced at least by one since $\tau_{\{i,j\}}$ of a selected d-pair {i, j} is applied

to C so that at least {i, j} is merged in each iteration. Rather than selecting the state with the shortest merging word, it uses a special cost function to determine the d-pair to be merged. For a set of states C ⊂ S, let the cost φ(C) of C be defined as :

$$\varphi(C) = \sum_{i,j \in C} |\tau_{\{i,j\}}|$$

φ(C) is a heuristic indication of how hard is to bring set C to a singleton. The intuition here is that, the larger the cost φ(C) is, the longer a synchronizing sequence would be required to bring C to a singleton set. Based on this cost function, the second phase of SynchroP is given in Algorithm **3**.

Algorithm 2 Phase 2 of Greedy

Input : An automaton $A(S, \Sigma, \delta), \tau(i, j)$ for all $\{i, j\} \in {S_s}^2$
Output : Synchronizing sequence Γ for A

1: $C \leftarrow S$ ▷ C: current set of states
2: $\Gamma \leftarrow \epsilon$ ▷ Γ: synchronizing sequence to be constructed, initially empty
3: **while** $C \neq \emptyset$ **do** ▷ There are pairs still not merged
4: $\{i, j\} \leftarrow \underset{<k,l> \in C_d^2}{\operatorname{argmin}} |\tau(k, l)|$
5: $\Gamma \leftarrow \Gamma\tau(i, j)$
6: $C \leftarrow \delta(C, \tau(i, j))$ ▷ update the current state set with $\tau(i, j)$
7: **end while**

Algorithm 3 Phase 2 of SynchroP

Input : An automaton $A(S, \Sigma, \delta), \tau(i, j)$ for all $\{i, j\} \in {S_s}^2$
Output : Synchronizing sequence Γ for A

1: $C \leftarrow S$ ▷ C: current set of states
2: $\Gamma \leftarrow \epsilon$ ▷ Γ: synchronizing sequence to be constructed, initially empty
3: **while** $C \neq \emptyset$ **do** ▷ There are pairs still not merged
4: $\{i, j\} \leftarrow \underset{<k,l> \in C_d^2}{\operatorname{argmin}} \varphi(\delta(C, \tau(k, l)))$
5: $\Gamma \leftarrow \Gamma\tau(i, j)$
6: $C \leftarrow \delta(C, \tau(i, j))$ ▷ update the current state set with $\tau(i, j)$
7: **end while**

2.3 Adaptation of Synchronizing Heuristics to Homing Sequences

Taking our cue from the synchronizing heuristics that were mentioned in the previous section, we have implemented various homing heuristics. We experimented with three of these heuristics, Fast HS, Greedy HS and SynchroP HS.

Phase 1 is common in these heuristics and given as Algorithm 2. In Phase 1, a shortest homing word τ{i,j} for each {i, j} ∈ S^2 is computed by using a breadth first search. Generation of the BFS forest is very similar to what is done in SS heuristics except that a d-pair {i, j} that gives different outputs for its states, λ(i, x) ≠ λ(j, x) for an input symbol x ∈ Σ, is located at level 1 of the forest by setting $\tau_{i,j}$=x.

Algorithm 2 Phase 1 of HS Heuristics

Input : An FMS $M(S, \Sigma, O, \delta, \lambda)$
Output : A homing word for all $\{i, j\} \in S^2$

$Q \leftarrow$ an empty queue ▷ Q: BFS frontier
$P \leftarrow \emptyset$ ▷ P: the set of nodes in the BFS forest constructed so far
for $\{i, i\} \in S_s^2$ **do**
  push $\{i, i\}$ onto Q
  insert $\{i, i\}$ into P
  set $\tau_{\{i,i\}} \leftarrow \epsilon$
end for
for $(\{i, j\} \in S^2) \wedge (\{i, j\} \notin P)$ **do**
  for $x \in \Sigma$ **do**
    if $\lambda(i, x) \neq \lambda(j, x)$ **then**
      push $\{i, j\}$ onto Q
      insert $\{i, j\}$ into P
      set $\tau_{\{i,j\}} \leftarrow x$
    end if
  end for
end for
while $P \neq S^2$ **do**
  $\{i, j\} \leftarrow$ pop next item from Q
  for $x \in \Sigma$ **do**
    for $\{k, l\} \in \delta^{-1}(\{i, j\}, x)$ **do**
      if $\{k, l\} \notin P$ **then**
        $\tau_{\{k,l\}} \leftarrow x\tau_{\{i,j\}}$
        push $\{k, l\}$ onto Q
        $P \leftarrow P \cup \{\{k, l\}\}$
      end if
    end for
  end for
end while

Algorithm 5 Phase 2 of Fast HS

Input : An FMS $M(S, \Sigma, O, \delta, \lambda), \tau(i, j)$ for all $\{i, j\} \in {S_s}^2$
Output : Homing sequence Γ for M

$P \leftarrow C_d^2$ ▷ P: current set of pairs
$\Gamma \leftarrow \epsilon$ ▷ Γ: homing sequence to be constructed, initially empty
while $P \neq \emptyset$ **do** ▷ There are pairs still not homed
  Let $\{i, j\}$ be a random pair from P
  $\Gamma \leftarrow \Gamma\tau(i, j)$
  $C \leftarrow \emptyset$
  for $\{k, l\} \in P$ **do**
    if $\delta(k, \tau(i, j)) \neq \delta(l, \tau(i, j))$ **and** $\lambda(k, \tau(i, j)) = \lambda(l, \tau(i, j))$ **then**
      insert $\delta(\{k, l\}, \tau(i, j))$ into C
    end if
  end for
  $P \leftarrow C$
end while

Similar to the synchronizing heuristics, Phase 2 updates the current state set after applying the homing word for the selected d-pair {i, j}. But rather than proceeding with all d-pairs, it divides set of states into blocks such that each block contains the states that give the same output after applying

$\tau_{\{i,j\}}$ and selects the d-pair from these blocks. It iterates until there are no more blocks to home. criterion differs according to the heuristic used.

Phase 2 of Fast HS does not perform a search in the blocks(set of pairs) for a d-pair. It selects a random pair from the current pair set to home without losing time so that it gets rid of the iterative pair selection process.

Unlike Fast HS, in Phase 2 of Greedy HS the d-pair to be selected is searched in the blocks and the d-pair that has a shortest homing word is selected.

SynchroP HS performs a deeper analysis on the automaton. It iterates through the all pairs in the blocks and determines the d-pair to be homed according to the φ cost.

Algorithm 6 Phase 2 of Greedy HS

Input : An FMS $M(S, \Sigma, O, \delta, \lambda), \tau(i, j)$ for all $\{i, j\} \in {S_s}^2$
Output : Homing sequence Γ for M

1: $P \leftarrow C_d^2$ ▷ P: current set of pairs
2: $\Gamma \leftarrow \epsilon$ ▷ Γ: homing sequence to be constructed, initially empty
3: **while** $P \neq \emptyset$ **do** ▷ There are pairs still not homed
4: $\{i, j\} \leftarrow \underset{<k,l> \in C_d^2}{\text{argmin}} |\tau(k, l)|$
5: $\Gamma \leftarrow \Gamma\tau(i, j)$
6: $C \leftarrow \emptyset$
7: **for** $\{k, l\} \in P$ **do**
8: **if** $\delta(k, \tau(i, j)) \neq \delta(l, \tau(i, j))$ **and** $\lambda(k, \tau(i, j)) = \lambda(l, \tau(i, j))$ **then**
9: insert $\delta(\{k, l\}, \tau(i, j))$ into C
10: **end if**
11: **end for**
12: $P \leftarrow C$
13: **end while**

Algorithm 7 Phase 2 of SynchroP HS

Input : An FMS $M(S, \Sigma, O, \delta, \lambda), \tau(i, j)$ for all $\{i, j\} \in {S_s}^2$
Output : Homing sequence Γ for M

$P \leftarrow C_d^2$ ▷ P: current set of pairs
$\Gamma \leftarrow \epsilon$ ▷ Γ: homing sequence to be constructed, initially empty
while $P \neq \emptyset$ **do** ▷ There are pairs still not homed
  $\{i, j\} \leftarrow \underset{<k,l> \in C_d^2}{\text{argmin}} \ \varphi(\delta(C, \tau(k, l)))$
  $\Gamma \leftarrow \Gamma\tau(i, j)$
  $C \leftarrow \emptyset$
  for $\{k, l\} \in P$ **do**
    if $\delta(k, \tau(i, j)) \neq \delta(l, \tau(i, j))$ **and** $\lambda(k, \tau(i, j)) = \lambda(l, \tau(i, j))$ **then**
      insert $\delta(\{k, l\}, \tau(i, j))$ into C
    end if
  end for
  $P \leftarrow C$
end while

3. Experiments and Conclusion

All the algorithms are implemented in C++ and the automata used for these experiments are randomly generated. Our experiments are conducted with 32, 64 state sizes and 2, 4, 8 input and output sizes. For each input, output and state size combinations, we performed experiments on 100 randomly generated automata. Times elapsed while performing these algorithms are measured in terms of microseconds.

We implemented 2 SS heuristics which find SSs on the homing automaton(equivalent to HSs on the FSM). These are Greedy and SynchroP heuristics. After creating the HA A_M explained in Section 3, we applied our implemented SS heuristics on the A_M. As can be seen from the column HA Time in Table 1, creation of the homing automaton is a time consuming process.

In order to enhance the time performance, we adapted these SS heuristics to homing sequences so that we can work on the initial FSM M_0 without converting it to A_M. We implemented 3 HS heuristics which find HSs on the FSM. These are Greedy HS, SynchroP HS and Fast HS heuristics as explained in Section 5.

In all experiments we measured time to create a BFS forest which is Phase 1 of heuristics and and time to home or merge all pairs in the pair set which Phase 2 of heuristics.

Results of SS heuristics show that SycnhroP finds sequences which have shorter lengths, that is an expected results since it performs a deeper analysis because of the φ cost. Greedy, on the other hand, is much more cheaper in terms of time but cannot find such short sequences. The difference might look slight while experimenting with small number of states or large number of outputs but when state sizes increase and number of outputs decreases, which means in fact when the shortest

SS length increases, difference between lengths of sequences found by SycnhroP and Greedy becomes huge.

					SS based					
			Shortest		HA Time	P1 Time	Greedy SS		SynchroP SS	
States	Inputs	Outputs	Length	Time			Length	P2 time	Length	P2 time
32	2	2	6,1	2410,83	46,39	11480,64	7,59	10,36	6,69	1925547,43
32	2	4	4,02	320,36	27,18	7372,63	4,79	5,68	4,32	1134028,55
32	2	8	3,04	96,07	16,99	5493,58	3,54	3,42	3,21	465578,61
32	4	2	5,2	10253,94	54,23	10298,58	7,15	7,21	6,21	2341436,82
32	4	4	3,7	580,23	53,43	10158,08	4,69	5,84	4,07	1306012,45
32	4	8	3	154,82	33,38	7649,16	3,56	3,73	3,05	516154,69
32	8	2	5	41776,44	127,54	16875,96	7,07	7,78	5,62	2922721,11
32	8	4	3,19	1590,49	80,34	11880,44	4,67	4,71	3,92	1328281,55
32	8	8	2,97	394,53	69,37	11249,46	3,56	3,84	3,1	561097,23
64	2	2	7,65	24051,55	185,61	240952,53	9,46	24,16	8,45	559041289,51
64	2	4	4,93	2281,29	161,1	203926,01	5,81	22	5,13	359077761,35
64	2	8	3,73	695,04	123,11	202362,82	4,28	21,52	3,98	145568201,2
64	4	2	6,95	278938,97	447,14	365706,26	9,18	24,12	7,75	795062568,44
64	4	4	4,25	10815,85	298,89	280419,83	5,69	16,92	5	428513432,51
64	4	8	3,12	1052,35	237,58	367358,17	4,24	12,77	3,81	154977902,12
64	8	2	6,24	6746717,82	806,23	525592,55	9,04	23,63	7,33	1000322128,91
64	8	4	4	19652,27	628,88	400603,53	5,65	16,96	4,9	456867018,86
64	8	8	3	1222,58	459,7	411719,73	4,24	13,03	3,78	154973825,85

Table 1: Experiments with SS heuristics on HA

HS heuristics work faster than the SS heuristics since there is no need f a homing automaton. Like in the SS heuristics, sequences found by SynchroP HS are shorter than the others. Fast HS selects a random pair from active set of pairs where as Greedy HS selects the pair that has the shortest homing word among others. Greedy HS generally selects pairs that stay in the level 1 of the generated BFS forest, pairs that have homing words of length 1. For small state sizes Fast HS also selects pairs which have merging word that has a length of 1 or 2. So in our experiments conducted with 32 and 64 state sizes, Fast HS does not seem to be fast. But when the state size increases (especially when the number of inputs is small), speed of Fast HS becomes apparent. Table 3 shows the experiment that reveals the difference between speeds of Fast HS and Greedy HS.

			HS based						
				Fast Homing		Greedy Homing		SynchroP Homing	
States	Inputs	Outputs	P1 Time	Length	P2 time	Length	P2 time	Length	P2 time
32	2	2	81,66	7,69	181,93	7,71	149,62	7,22	3019,61
32	2	4	52,08	4,8	120,58	4,8	104,72	4,65	2127,39
32	2	8	33,31	3,58	86,7	3,57	55,12	3,41	1594,03
32	4	2	56,83	7,2	117,17	7,14	85,84	6,76	1850,09
32	4	4	49,48	4,7	126,91	4,7	88,79	4,59	2272,51
32	4	8	16,27	3,59	89,11	3,59	56,87	3,49	1784,65
32	8	2	66,46	7,1	132,64	7,09	99,72	6,7	2167,34
32	8	4	28,43	4,7	98,81	4,7	67,25	4,54	1823,6
32	8	8	18,41	3,58	95,27	3,58	63,24	3,52	2004,3
64	2	2	278,64	9,53	313,19	9,66	302,35	9,14	25852,08
64	2	4	278,61	5,68	282,08	5,84	253,67	5,64	27990,49
64	2	8	231,76	4,27	231,24	4,27	194,09	4,24	26139,92
64	4	2	424,43	9,24	385,75	9,24	351,03	8,59	30794,32
64	4	4	292,72	5,69	270,94	5,73	243,5	5,57	26754,09
64	4	8	222,99	4,29	217,17	4,29	193,86	4,27	25266,44
64	8	2	474,91	9,15	356,29	9,14	320,09	8,72	27811,68
64	8	4	328,66	5,69	298,41	5,69	260,82	5,54	27936,65
64	8	8	164,02	4,29	217,94	4,29	182,97	4,24	24772,96

Table 2: Experiments with HS heuristics on FSM

			Greedy HS		Fast HS	
States	Inputs	Outputs	Length	Time	Length	Time
4096	2	2	21.47	63327.2	21.62	52520.84

Table 3: Averages of 100 experiments

References

[1] D. Eppstein. Reset sequences for monotonic automata. SIAM J. Comput., 19(3):500–510, 1990.

[2] Z. Kohavi. Switching and Finite Automata Theory. McGraw–Hill, New York, 1978.

[3] A. Roman. New algorithms for finding short reset sequences in synchronizing automata. In IEC (Prague), pages 13–17, 2005.