

## Reset Sequences for Monotonic Automata

David Eppstein

Xerox Palo Alto Research Center  
3333 Coyote Hill Rd.  
Palo Alto, CA 94303

*Abstract:* Natarajan reduced the problem of designing a certain type of mechanical parts orienter to that of finding reset sequences for monotonic deterministic finite automata. He gave algorithms that in polynomial time either find such sequences or prove that no such sequence exists. In this paper we present a new algorithm based on breadth first search that runs in faster asymptotic time than Natarajan's algorithms, and in addition finds the shortest possible reset sequence if such a sequence exists. We give tight bounds on the length of the minimum reset sequence. We further improve the time and space bounds of another algorithm given by Natarajan, which finds reset sequences for arbitrary deterministic finite automata when all states are initially possible.

*Keywords:* finite automata, reset sequences, breadth first search, automated design, parts orienters.

*ACM subject classification:* F2.2 Nonnumerical Algorithms and Problems (Computations on Discrete Structures).

February 7, 1990

## Introduction

Natarajan [5] considered the design of automated parts orienters; that is, devices that accept mechanical parts in any orientation or in a wide class of orientations, and output them in some predetermined orientation. One such orienter is a pan handler, in which the part slides around on a tray as that tray is tilted, turning in a well-defined way when it hits the walls of the tray. These devices had been previously been described in [2] and [4].

For a given tray and object, and for a given set of possible initial orientations for the object, one desires to determine whether there exists a sequence of tilt angles that will cause the object to always end up in the same orientation. Natarajan made the assumptions that the set of angles is finite, that the set of orientations in which the part can rest on a tray face is also finite, that tilting the tray with a given angle and with the object in a given initial orientation always results in the same final orientation, and that this relation between angles, initial orientations, and final orientations is known. He also made the assumption that all faces of the tray are identical; that is, that the shape of the tray is a regular polyhedron. We will show later how to remove this last assumption. With these assumptions he reduced the problem to the following combinatorial one.

Let  $S = \{s_1, s_2, \dots, s_n\}$  be a set of states, corresponding to orientations of the part to be oriented. Let  $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$  be an alphabet, corresponding to the possible ways of tilting the orienter tray. Finally, let  $\delta(\sigma, s)$  be a function indicating the orientation resulting when tilt  $\sigma$  is applied to state  $s$ . Then  $(S, \Sigma, \delta)$  forms a finite automaton. One is further given a set of initial states, or orientations,  $X \subset S$ . Because the tray is assumed to be symmetric, nothing in the states or transition functions need identify the edge of the tray at which the object is resting.

In what follows, sequences of input symbols to the automaton will be denoted using the letter  $\tau$ . As with the input symbols themselves, we let  $\delta(\tau, s)$  denote the effect of sequence  $\tau$  on the states of the automaton. Thus, if  $\tau = \tau_1\tau_2$ , then  $\delta(\tau, s) = \delta(\tau_2, \delta(\tau_1, s))$ . If  $\tau$  is the empty input sequence,  $\delta(\tau, s) = s$ . We denote the set of all possible input sequences by  $\Sigma^*$ . Further, if  $X \subset S$ , we let  $\delta(\tau, X)$  stand as a shorthand for  $\{\delta(\tau, x) | x \in X\}$ . Finally,  $\delta^{-1}(\tau, X)$  will be taken to mean  $\{s \in S | \delta(\tau, s) \in X\}$ .

Suppose one is given a set of states, or orientations,  $X \subset S$ ; it is assumed that the initial orientation of the object is one of the states in  $X$ . For instance, without assumption we can let  $X = S$ , because  $S$  encodes all the possible orientations of the object. Then the pan handler orientation problem becomes simply that of finding a sequence  $\tau \in \Sigma^*$  such that  $|\delta(\tau, X)| = 1$ ; that is, such that the application of  $\tau$  will leave the automaton in one particular state no matter which state in  $X$  it started at. We call  $\tau$  a *reset sequence* for  $(S, \Sigma, \delta)$  and  $X$ .

Alternately, the problem can be phrased in terms of finite functions, instead of automata. In this formulation, one is given a collection of functions from a finite set  $S$  to itself, and the problem becomes finding a composition of those functions that takes the subset  $X$  to some constant. If  $X = S$ , the problem becomes that of composing the functions to get a constant function.

Natarajan gave an algorithm for solving this problem for any automaton when  $X = S$ . This algorithm takes  $O(kn^4)$  time, and either produces a reset sequence or proves no such sequence exists. The sequence produced is not guaranteed to be the shortest possible, but Natarajan bounded its length by  $O(kn^3)$ .

It turns out that for general automata and general  $X$ , finding a reset sequence is PSPACE-

complete [5]. However Natarajan observed that the automata arising in the pan handler problem have a property which he called monotonicity, and that with this property the problem becomes solvable in polynomial time. He gave algorithms with asymptotic time complexity  $O(kn^4)$  (or  $O(kn^3 \log n)$  when  $X = S$ ), which find sequences of length at most  $O(kn^3)$  (respectively  $O(kn^2 \log n)$ ). The sequences found are again not guaranteed to be optimal.

## New Results

This paper presents a new algorithm for finding reset sequences on monotonic automata, for any  $X \subseteq S$ , which takes time  $O(kn^2)$  and is guaranteed to find the shortest possible sequence. Further, this leads to tight worst case bounds of  $n^2 - 2n + 1$  on the number of input symbols in the optimal reset sequence. The algorithm works by defining a new automaton, the states of which correspond to intervals in the cyclic order of the original automaton's states. Reset sequences in the original automaton correspond to paths in the new automaton leading to a singleton interval. Therefore we can find our desired sequence using a simple breadth first search technique.

As another result of this paper, we extend our technique to certain classes of non-monotonic automata. In particular, we can use this result to solve the pan-handler problem optimally for asymmetric as well as symmetric pans. The search takes time  $O(k^2n^2)$ , and the resulting sequence has length at most  $kn^2 - 2kn + 1$ .

Finally, we improve Natarajan's algorithm for any non-monotonic automaton and for  $X = S$ , to take time  $O(n^3 + kn^2)$ , and working space bounded by  $O(n^2)$ . We also prove a tighter bound of  $O(n^3)$  on the length of the resulting sequence, and show that finding the minimum length reset sequence is NP-complete.

## Definitions and Lemmas

First, we define monotonic automata. Assume that the states of a given deterministic finite automaton  $(S, \Sigma, \delta)$  are arranged in some known cyclic order  $s_1, s_2, \dots, s_n$ . The transition function  $\delta$  is *monotonic* if it preserves the cyclic order of the states. Formally, for any input symbol  $\sigma$ , the sequence of states  $\delta(\sigma, s_1), \delta(\sigma, s_2), \dots, \delta(\sigma, s_n)$ , after removal of possible adjacent duplicate states, must be a subsequence of a cyclic permutation of  $s_1, s_2, \dots, s_n$ . Clearly, if  $\delta$  is monotonic, all compositions  $\delta(\tau, s)$  will also preserve the cyclic order of the states. From now on in this section, when we refer to the automaton  $(S, \Sigma, \delta)$  we will assume that it is monotonic.

Next let us define an interval  $[s_i, s_j]$ . This consists of all those states between  $s_i$  and  $s_j$  (inclusive) in the cyclic order of the states; e.g.,  $[s_1, s_3] = \{s_1, s_2, s_3\}$ . Note that there are  $n$  different ways of representing the full set of states  $S$  as an interval  $[s_i, s_{i-1}]$ ; any other set of states that can be represented as an interval has exactly one such representation. We say that an interval  $J = [s_h, s_i]$  is contained in another interval  $I = [s_g, s_j]$ , and write  $J \prec I$ , when the endpoints of the intervals appear in the cyclic order  $s_g, s_h, s_i, s_j$ . Containment as an interval implies containment as a set of states, but the reverse may be false in the case that the containing interval is all of  $S$ .

**Lemma 1.** For all  $\tau \in \Sigma^*$ , and for any interval  $I$ ,  $\delta^{-1}(\tau, I)$  is an interval.

Proof: If not, there would be  $s_{i_1}, s_{i_2}, s_{i_3}$ , and  $s_{i_4}$  in cyclic order such that  $\delta(\tau, s_{i_1})$  and  $\delta(\tau, s_{i_3})$  are in  $I$  but  $\delta(\tau, s_{i_2})$  and  $\delta(\tau, s_{i_4})$  are not; but this violates monotonicity. •

Unlike their inverses, the transition functions of the DFA do not necessarily take intervals to intervals. However we can define a new transition function, corresponding to the original one, that does take intervals to intervals. Let  $S' = \{[s_i, s_j] | s_i, s_j \in S\} \cup \{\infty\}$ ; i.e.  $S'$  consists of all the possible intervals together with the extra symbol  $\infty$ . Let  $\sigma \in \Sigma$  and  $I = [s_i, s_j] \in S'$ , and define  $d_x = \delta(\sigma, s_x)$ . Then we define  $\delta'$  as follows. (1) If  $d_i \neq d_j$ , let  $\delta'(\sigma, I) = [d_i, d_j]$ . (2) If  $d_i = d_{i+1} = d_{i+2} = \dots = d_j$ , let  $\delta'(\sigma, I) = [d_i, d_i]$ . (3) Otherwise, or if  $I = \infty$ , let  $\delta'(\sigma, I) = \infty$ .

The new transition functions we have defined give us a new DFA  $(S', \Sigma, \delta')$  whose states are the intervals of the original automaton, and which takes the same input alphabet as the original automaton. Note that this DFA, which is of size  $O(kn^2)$ , can be constructed in time linear in its size. The only complication is how to determine whether the result of a transition in which the two endpoints are mapped to a single point should be that point or  $\infty$ . This can be done by first constructing for each  $\sigma \in \Sigma$  and  $s \in S$  the interval  $\delta^{-1}(\sigma, s)$ , which must exist by lemma 1. This construction takes time  $O(n)$ , and there are  $O(kn)$  intervals to construct, so all such intervals can be constructed in time  $O(kn^2)$ . Then if the endpoints of interval  $I$  are mapped by  $\sigma$  to the same state  $s$ ,  $\delta'(\sigma, I) \neq \infty$  if and only if  $I \subset \delta^{-1}(\sigma, s)$ . The containment above should be interpreted as being between sets rather than as the interval containment defined earlier; it can be calculated using a constant number of comparisons between interval endpoints to determine interval containment, together with a test for the special case that  $\delta^{-1}(\sigma, s) = S$ , which is the only case in which set and interval containment can differ.

If  $\tau$  is an input sequence  $\sigma_i \sigma_j \sigma_k \dots$ , we define  $\delta'(\tau, I)$  to be the corresponding composition of interval transition functions. If any of the individual transitions in  $\delta'(\tau, I)$  gives  $\infty$ , the result as a whole is also  $\infty$ . If  $\tau$  is the empty input sequence we define  $\delta'(\tau, I) = I$ . Then the following facts follow by an easy induction. Define  $d_x = \delta(\tau, s_x)$ . (1) If  $d_i \neq d_j$ , then  $\delta'(\tau, I) = [d_i, d_j]$ . (2) If  $d_i = d_{i+1} = d_{i+2} = \dots = d_j$ , then either  $\delta'(\tau, I) = [d_i, d_i]$  or  $\delta'(\tau, I) = \infty$ . (3) Otherwise, or if  $I = \infty$ , then  $\delta'(\tau, I) = \infty$ . I.e. the definitions of  $\delta'$  from  $\delta$  for single input symbols also hold as theorems for sequences of input symbols, except that occasionally the result will be  $\infty$  in case (2).

**Lemma 2.** For all  $\tau$  in  $\Sigma^*$ , and all intervals  $I$ , if  $\delta'(\tau, I) \neq \infty$ , then  $\delta(\tau, I) \subset \delta'(\tau, I)$ .

Proof: This follows from the characterization of  $\delta'(\tau, I)$  above, together with the assumption of monotonicity. •

**Lemma 3.** For all  $\tau$  in  $\Sigma^*$ , and all intervals  $I$ , if  $\delta'(\tau, I) \neq \infty$ , then for all intervals  $J \prec I$ ,  $\delta'(\tau, J) \neq \infty$  and  $\delta'(\tau, J) \prec \delta'(\tau, I)$ .

Proof: Let  $J = [s_i, s_j]$ . If  $s_i = s_j$ , then  $\delta'(\tau, [s_i, s_j])$  is easily seen to satisfy the conditions of the lemma, so assume the two states are different. If  $\delta(\tau, s_i) \neq \delta(\tau, s_j)$ , then by monotonicity these two states appear in the correct order within  $\delta'(\tau, I)$ , and again the lemma is satisfied. The remaining case to check is that  $\delta(\tau, s_i) = \delta(\tau, s_j)$ , and that there is some state  $s$  in  $[s_i, s_j]$  such that  $\delta(\tau, s) \neq \delta(\tau, s_i)$ . But this either violates monotonicity or forces  $\delta'(\tau, I)$  to equal  $\infty$ . •

**Lemma 4.** For all  $\sigma \in \Sigma$  and all intervals  $J$ , if  $I$  is a representation of  $\delta^{-1}(\sigma, J)$  as an interval (which by lemma 1 must exist), and if  $I$  is not all of  $S$ , then  $\delta'(\sigma, I) \neq \infty$  and  $\delta'(\sigma, I) \prec J$ .

Proof: Let  $I = [s_i, s_j]$ . If  $\delta'(\sigma, I)$  were equal to  $\infty$ , then  $\delta'(\sigma, [s_j, s_i])$  would be a singleton interval by monotonicity. But then  $\delta(\sigma, S) = \delta(\sigma, I \cup [s_j, s_i]) = \delta(\sigma, I) \cup \delta(\sigma, [s_j, s_i]) \subset J \cup \{\delta(\sigma, s_j)\} = J$ , which contradicts the assumption that  $\delta^{-1}(\sigma, J) \neq S$ . Therefore  $\delta'(\sigma, I) \neq \infty$ .

By the definition of  $I$ ,  $\delta(\sigma, s_i)$  and  $\delta(\sigma, s_j)$  are both in  $J$ ; by monotonicity, they must appear in the correct cyclic order within that interval. It follows that  $\delta'(\sigma, I) \prec J$ . •

We now prove the main lemma, which shows the equivalence between reset sequences in the original automaton and paths to a singleton in the interval automaton.

**Lemma 5.** Given  $\tau \in \Sigma^*$ , and  $X \subseteq S$ , then  $|\delta(\tau, X)| = 1$  if and only if there is a representation of  $\delta^{-1}(\tau, \delta(\tau, X))$  as an interval  $I$  such that  $\delta'(\tau, I) \neq \infty$  and  $|\delta'(\tau, I)| = 1$ .

Proof: If there is some such  $I$  then, by lemma 2,  $\tau$  must be a reset sequence for  $X$ . In the other direction, assume we are given  $\tau$  and  $X$  with  $\delta(\tau, X) = \{s\}$ . We want to find a representation of  $\delta^{-1}(\tau, s)$  as an interval meeting the terms of the lemma. We prove the lemma by induction on the length of  $\tau$ ; as a base case, if  $\tau$  is the empty input sequence,  $|X| = 1$  and the lemma clearly holds. Otherwise, assume  $\tau = \sigma\bar{\tau}$  for some  $\sigma \in \Sigma$  and  $\bar{\tau} \in \Sigma^*$ . By the induction hypothesis there is a representation of  $\delta^{-1}(\bar{\tau}, s)$  as an interval  $J$  with  $\delta'(\bar{\tau}, J) = [s, s]$ .

First assume  $I = \delta^{-1}(\sigma, J)$  is not all of  $S$ . By lemma 4 we see that  $\delta'(\tau, I) = [s, s]$ , as was to be shown.

The remaining case is that  $\delta^{-1}(\sigma, J) = S$ . Choose the interval  $[s_i, s_j]$  such that  $s_i$  and  $s_j$  are both in  $\delta(\sigma, S)$  (and therefore also in  $J$ ), and also such that  $\delta(\sigma, S) \subset [s_i, s_j] \prec J$ . This can be done by taking  $s_i$  to be the first member in  $J$  that is also in  $\sigma(S)$ , and  $s_j$  to be the last such member.

Now if  $s_i = s_j$ , then  $\delta(\sigma, S) = \{s_i\}$ , and any representation of  $S$  as an interval  $I$  will give us  $\delta'(\tau, I)$  a singleton interval, satisfying the lemma. So assume  $s_i \neq s_j$ . This implies that  $\delta^{-1}(\sigma, s_j)$  is not all of  $S$ , so by lemma 1 this set has a unique representation as an interval  $[s_h, s_k]$ . Using monotonicity and the fact that  $\delta(\sigma, S) \subset [s_i, s_j]$ , it can be shown that that  $\delta(\sigma, s_{k+1}) = s_i$ . Therefore  $\delta'(\sigma, [s_{k+1}, s_k])$  is equal to  $[s_i, s_j]$ . By lemma 3 we see that  $\delta'(\tau, [s_{k+1}, s_k]) = \delta'(\bar{\tau}, [s_i, s_j]) \subset \delta'(\bar{\tau}, J)$  is a singleton interval. •

## Reset Sequences for Monotonic Automata

**Theorem 1.** A minimum length reset sequence for monotonic automaton  $(S, \Sigma, \delta)$  and initial states  $X = \{s_{i_1}, s_{i_2}, \dots\}$  can be found in time bounded by  $O(kn^2)$ .

Proof: First one constructs the automaton  $(S', \Sigma, \delta')$  as described above. By lemma 5, a minimum reset sequence for the original automaton will also be a minimum length path in the new automaton from some interval containing  $X$  to a singleton interval, and vice versa. By lemma 3, we need only consider starting from the minimal intervals containing  $X$ , rather than all intervals containing  $X$ ; these minimal intervals can be found as  $[s_{i_j}, s_{i_{j-1}}]$ . Finally, the shortest path from one of these intervals to a singleton can be found using the standard breadth first search algorithm. •

**Theorem 2.** If a minimum length reset sequence for monotonic automaton  $(S, \Sigma)$  and initial states  $X$  exists, its length is  $\leq n^2 - 2n + 1$ .

Proof: The path constructed by the breadth first search in theorem 1 will visit each state of the constructed automaton at most once, and there are  $n^2 + 1$  states. But the sequence need involve at most one interval representing all of  $S$ , and at most one singleton; further, it need never involve state  $\infty$ . Thus there are at least  $2n - 1$  states not included in the minimum length path. •

**Theorem 3.** For any  $n$ , there exists a monotonic automaton  $(S, \Sigma, \delta)$  with  $|S| = n$ , and a set of initial states  $X$ , such that the minimum length reset sequence for  $(S, \Sigma, \delta)$  and  $X$  has length  $n^2 - 2n + 1$ .

Proof: Name the states  $s_1, s_2, \dots, s_n$ , in that cyclic order. Let  $\Sigma$  consist of only two input symbols,  $\sigma_1$  and  $\sigma_2$ . Let the transition function always take  $s_n$  to  $s_1$ , but let  $\sigma_1$  take all states  $s_i$  other than  $s_n$  to  $s_{i+1}$ , and let  $\sigma_2$  take all states  $s_i$  other than  $s_n$  to themselves. I.e.,

$$\begin{aligned}\delta(\sigma_1, s_i) &= s_{i+1} \text{ for } 1 \leq i < n \\ \delta(\sigma_2, s_i) &= s_i \text{ for } 1 \leq i < n \\ \delta(\sigma_i, s_n) &= s_1 \text{ for } 1 \leq i \leq 2\end{aligned}$$

We take  $X = S$ , so that a reset sequence for  $X$  must take all  $n$  states to a single state.

Assume we have a reset sequence  $\tau$ , and define  $\tau_i$  to be the prefix of  $\tau$  consisting of the first  $i$  symbols of  $\tau$ . Also define  $\ell(i)$  to be the length of the shortest interval containing all the states in  $\delta(\tau_i, S)$ . If by  $|\tau|$  we denote the number of input symbols in  $\tau$ , then clearly  $\ell(|\tau|) = 1$ . Finally, define  $t(j)$ , for each  $j$ , to be the least  $i$  such that  $\ell(i) \leq j$ .

We prove below that, for each  $j < n - 1$ ,  $t(j) \geq t(j + 1) + n$ ; that is, there must be at least  $n$  input symbols processed between each point at which the shortest interval containing the states becomes shorter. The theorem then follows, because the total number of steps in the reset sequence must be at least  $n(n - 2)$  for the  $n - 2$  gaps of  $n$  steps each, plus one initial step to reduce  $\ell(t)$  from  $n$  to  $n - 1$ .

First note that, if  $j \neq 1$ , the  $i$ th input symbol is  $\sigma_1$ , and  $\delta(\tau_i, S) \subset [s_j, s_k]$ , then  $\delta(\tau_{i-1}, S) \subset [s_{j-1}, s_{k-1}]$ . If  $j = 1$ , the  $i$ th input symbol is  $\sigma_2$ , and  $\delta(\tau_i, S) \subset [s_j, s_k]$ , then  $\delta(\tau_{i-1}, S) \subset [s_j, s_k]$ . Therefore, no matter what the input symbols of  $\tau$  are, if  $\delta(\tau_i, S) \subset [s_j, s_k]$ , we can see using induction that  $\delta(\tau_{i-j+1}, S) \subset I$  for some interval  $I$  of length  $k - j + 1$ .

Next observe that if the  $i$ th input symbol is  $\sigma_1$ , then  $\ell(i - 1) = \ell(i)$ ; therefore for each  $j$  the input symbol at position  $t(j)$  must be  $\sigma_2$ , and further it must be the case that  $\delta(\tau_{t(j)-1}, S) \subset [s_n, s_j]$ . Using the previous observation we see that  $\delta(\tau_{t(j)-n}, S) \subset I$  for some interval  $I$  of length  $j + 1$ , and therefore  $t(j + 1) \leq t(j) - n$  as was to be proved. The theorem then follows as described above. •

### Extensions of the Monotonic Reset Sequence Technique

Various generalizations of the algorithms and bounds above may be taken. For instance, let us consider the case that what is desired as the result of the reset sequence is a particular state rather than just any single state. The same algorithm as in theorem 1, but with the breadth first search terminating only when it reaches the singleton interval corresponding to the desired state, will always find the minimum such reset sequence when it exists, again taking time  $O(kn^2)$ . The upper bound of theorem 2 must be relaxed to  $n^2 - n$ , because it is now possible for the path in the interval automaton to go through all singleton intervals before it gets to the desired one. And the example used in theorem 3, with the desired singleton state being  $s_n$ , requires  $n^2 - n$  steps for a reset sequence, showing that this new bound is tight.

Another generalization allows us to remove the assumption, used in reducing the pan-handler problem to that of computing reset sequences, that the shape of the pan-handler is a regular

polyhedron. Let us say that the pan-handler has  $k$  faces, not necessarily all alike. We assume that the face on which the object is originally resting is known. Then we can form an automaton with  $kn$  states, which encode both the position of the object and also the face on which it rests. Again there will be  $k$  transition functions; function  $i$  will correspond to tilting the pan-handler so that the object moves to face  $i$ . For any given tilt of the tray from face  $i$  to face  $j$ , the change in the orientations of the object will be monotonic; however, because the states also encode the face the object is lying on, the automaton as a whole will not typically be monotonic. Nevertheless we can use the methods above, building a new automaton the states of which are intervals of object positions together with a single face that the object rests on, and the transition functions of which are the result of applying the original transition functions to these ranges of positions. As above, a breadth first search through the new automaton results in a reset sequence for the original automaton. The search takes time  $O(k^2n^2)$ , and the resulting sequence has length at most  $kn^2 - 2kn + 1$ .

### Reset Sequences for General Automata

In this section we will relax the requirement that the automaton  $(S, \Sigma, \delta)$  be monotonic, and instead restrict our attention to reset sequences for all of  $S$ ; that is, we will assume that the automaton may initially be in any of its states, rather than in a state drawn from some subset  $X$  of its states. The reason that the case we study is easier than the general case is that we can never get stuck: if there exists a reset sequence  $\tau$ , then no matter what sequence  $\bar{\tau}$  we have chosen already,  $\bar{\tau}\tau$  will still be a reset sequence for the whole set. We can proceed by reducing the size of the set  $\delta(\bar{\tau}, S)$  a step at a time, without ever having to worry about backtracking.

The following algorithm, due to Natarajan, works in the above manner to find a reset sequence for any automaton  $(S, \Sigma, \delta)$ , with the initial set of states being all of  $S$ . The reset sequence it finds is not necessarily the shortest possible such sequence. We will put off describing the implementation details of some of the steps until later.

```

Algorithm 1:
begin
   $X \leftarrow S$ ;
   $\tau \leftarrow$  the empty sequence;
  while  $|X| > 1$  do begin
    pick  $s_i, s_j \in X$  with  $s_i \neq s_j$ ;
    find a sequence  $\bar{\tau}$  taking  $s_i$  and  $s_j$  to the same state;
     $X \leftarrow \delta(\bar{\tau}\tau, X)$ ;
     $\tau \leftarrow \tau\bar{\tau}$ ;
  end;
end

```

**Theorem 4** [5]. Assuming the steps in the loop of algorithm 1 can be computed, the algorithm terminates after  $O(n)$  repetitions of the loop, and finds a reset sequence for  $(S, \Sigma, \delta)$  if such a sequence exists. If the algorithm ever chooses a pair of states  $s_i, s_j$  such that no sequence  $\bar{\tau}$  takes the two states to a single state, then no reset sequence exists.

Proof: Each time through the loop, the size of  $X$  decreases by at least one; therefore the loop can be executed at most  $n$  times. When the size of  $X$  has decreased to one,  $\tau$  will then be a reset

sequence. If any reset sequence  $\tau$  exists, it will a fortiori satisfy the conditions for  $\bar{\tau}$ . •

Natarajan described an implementation of the above algorithm that takes time  $O(kn^4)$ . The two steps of the algorithm that take the most time are finding  $\bar{\tau}$  and applying it to  $X$ . We now describe some preprocessing that allows these steps to be done quickly, therefore improving Natarajan's result. A naive implementation of these preprocessing steps would take space  $O(n^3)$ , which is worse than the previous  $O(n^2)$  bound; we later show how our preprocessing may be performed within a space bound of  $O(n^2)$ .

**Theorem 5.** Algorithm 1 can be executed in time  $O(n^3 + kn^2)$ .

Proof: As in the monotonic case, we first form a new automaton of size  $O(kn^2)$  and perform a breadth first search in it. The states of the new automaton consist of each (unordered) pair of states from the original automaton, together with one state for each of the original automaton's states. The result of applying any the original automaton's input symbols  $\sigma$  to a pair of states  $(s_i, s_j)$  will be  $(\delta(\sigma, s_i), \delta(\sigma, s_j))$ ; if  $\delta(\sigma, s_i) = \delta(\sigma, s_j)$  then the result will be that singleton state.

Before we run algorithm 1 itself, we perform a breadth first search on the new automaton, finding for each pair of original states  $(s_i, s_j)$  a shortest input sequence  $\tau_{i,j}$  taking that pair to a singleton state. This can be performed in time  $O(kn^2)$ , and the result can be represented as a shortest path forest in space  $O(n^2)$ ; paths in this forest lead from each pair to a singleton, along the sequence of pairs found by applying each transition function in  $\tau_{i,j}$  successively to the pair  $(s_i, s_j)$ .

In the following description we will call the above breadth first search stage 1. If we only desire to know whether there is a reset sequence, without needing to know what that reset sequence is, then we may stop now, having taken time  $O(kn^2)$ : a reset sequence exists if and only if for every pair  $(s_i, s_j)$  such a sequence  $\tau_{i,j}$  leading to a singleton can be found.

Next, as stage 2 of our preprocessing, for each pair of states  $(s_i, s_j)$  and each state  $s_k$  of the original automaton, we compute  $\delta(\tau_{i,j}, s_k)$ . This is done by performing a pre-order traversal of the shortest path forest computed in stage 1. Whenever we visit a pair  $(s_i, s_j)$ , we compute in constant time  $\delta(\tau_{i,j}, s_k)$ , for all states  $s_k$ , as follows. Let  $\tau_{i,j} = \sigma\tau_{g,h}$ , where  $\sigma$  is the first transition function in  $\tau_{i,j}$ ,  $\delta(\sigma, s_i) = s_g$ , and  $\delta(\sigma, s_j) = s_h$ . If  $s_g = s_h$  let  $\tau_{g,h}$  be the empty sequence of transition functions, which corresponds to the identity function. Then  $\delta(\tau_{i,j}, s_k) = \delta(\tau_{g,h}, \delta(\sigma, s_k))$  can be computed as one function evaluation of  $\delta(\sigma, s_k)$  followed by a table lookup of the value of  $\delta(\tau_{g,h}, \delta(\sigma, s_k))$ ; because we are performing a pre-order traversal the latter value will have already been computed. Since there are  $O(n^3)$  computations to be performed, each taking constant time, the total time for this stage is  $O(n^3)$ .

Now we show how to perform the steps of the main algorithm described above. To find  $\bar{\tau}$  for  $s_i$  and  $s_j$ , we simply look up  $\tau_{i,j}$  in the forest we calculated in the first stage; there are  $O(n^2)$  pairs, so the shortest sequence  $\tau_{i,j}$  resulting in a singleton is at most  $O(n^2)$  symbols long, and therefore this step takes time bounded by  $O(n^2)$ . To find  $\delta(\bar{\tau}, X)$  we simply look up, for each member  $s$  of  $X$ ,  $\delta(\bar{\tau}, s)$  as calculated in the second stage;  $|X| \leq n$  so this step takes time  $O(n)$ . The inner loop is executed  $O(n)$  times, so the execution of algorithm 1 as a whole takes time  $O(n^3 + kn^2)$ , which is also the time taken by the preprocessing stages. •

Recall that we claimed that we could reduce the working space used to  $O(n^2)$  while keeping the time bounds described above. We do not count the length of the output sequence, for which



the best bound we have is  $O(n^3)$ , as part of this space bound.

The pair automaton we constructed would seem to take  $O(kn^2)$  space, but in fact we need only to use constant storage space for each pair of the automaton, and construct the outgoing arcs from each pair as needed from the original automaton. A more serious obstacle to reducing the space is that the space required to store  $\delta(\tau_{i,j}, s_k)$  is  $\Omega(n^3)$ . However it turns out to be possible to reduce the space required, by keeping  $\delta(\tau_{i,j}, s_k)$  only for certain pairs  $(s_i, s_j)$  rather than all such pairs. First let us describe an algorithm to compute the pairs for which we will calculate the values of  $\tau_{i,j}$ . This algorithm is given as input a forest of size  $x$ , and another integer parameter  $y$ . It calculates a partition of the forest into  $O(x/y)$  subtrees, each of depth at most  $y$ .

**Algorithm 2:**

```

for each vertex  $v$  of the forest, in a post-order traversal, do begin
   $size(v) \leftarrow 1$ ;
  for each vertex  $w$  such that  $(w, v)$  is an edge in the forest do
    if  $mark(w) = 0$  then  $size(v) \leftarrow size(v) + size(w)$ ;
  if  $size(v) < y$  then  $mark(v) \leftarrow 0$ ;
  else  $mark(v) \leftarrow 1$ ;
end

```

**Lemma 6.** Algorithm 2 takes time linear in  $x$ , the number of vertices in the forest it processes. After it has been executed, there will be at most  $x/y$  vertices  $v$  of the forest with  $mark(v) = 1$ . Further, if we break the outgoing link of each such marked vertex, no tree in the new forest so created will have depth greater than  $y$ .

Proof: The post-order traversal guarantees that  $size(w)$  and  $mark(w)$  in the inner loop of the algorithm will have been calculated before we process vertex  $v$ . For each vertex  $v$ ,  $size(v)$  computes the number of vertices in the subtree of unmarked vertices rooted at  $v$ . Each marked vertex has at least  $y - 1$  unmarked vertices in its subtree, so there can be at most  $x/y$  marked vertices. If any tree of unmarked vertices rooted at a marked vertex had depth greater than  $y$ , the number of unmarked vertices on any path of length greater than  $y$  to the root would be enough to have caused one of the vertices along that path to have been marked; therefore the depth of each such tree is at most  $y$ . •

**Theorem 6.** Algorithm 1 can be executed in time  $O(n^3 + kn^2)$  as in theorem 5, using working space bounded by  $O(n^2)$ .

Proof: We compute stage 1 as before. But before performing stage 2, we run algorithm 2 on the shortest path forest computed in stage 1, with  $x$  being the number  $n(n + 1)/2$  of pairs and singletons in the forest, and  $y$  equal to  $n$ , the number of states in the original automaton.

In stage 2 we now only compute  $\delta(\tau_{i,j}, s_k)$  for those pairs  $(s_i, s_j)$  that were marked by algorithm 2. Again we will process each such pair in order by a pre-order traversal of the forest. We first compute the shortest prefix of  $\tau_{i,j}$  that takes  $(s_i, s_j)$  to another marked pair  $(s_g, s_h)$ ; call this shortest prefix  $\bar{\tau}$ . By lemma 6, the number of input symbols in  $\bar{\tau}$  is at most  $n$ . Then  $\delta(\tau_{i,j}, s_k) = \delta(\tau_{g,h}, \delta(\bar{\tau}, s_k))$ , which can be computed with at most  $n$  function evaluations followed by a table lookup. Using lemma 6 again we see that there are at most  $O(n)$  marked pairs, and for each such pair we have to perform  $n$  computations each taking time  $O(n)$ , so the total time for the new version of stage 2 is again bounded by  $O(n^3)$ . We store  $\delta(\tau_{i,j}, s_k)$  for only  $O(n)$  pairs  $(s_i, s_j)$ ,

so the total space used is bounded by  $O(n^2)$ .

In algorithm 1 itself, the only changed step is in computing  $\delta(\tau_{i,j}, X)$ . Here  $(s_i, s_j)$  might not be marked, but as in stage 2 we can find a shortest prefix  $\bar{\tau}$  of  $\tau_{i,j}$  taking  $(s_i, s_j)$  to a marked pair  $(s_g, s_h)$ . Again  $\bar{\tau}$  has length at most  $n$ , so for each member  $s$  of  $X$  we can find  $\delta(\tau_{i,j}, s) = \delta(\tau_{g,h}, \delta(\bar{\tau}, s))$  by  $O(n)$  function evaluations followed by a table lookup. The entire computation of  $\delta(\tau_{i,j}, X)$  takes time bounded by  $O(n^2)$ , which does not reduce the running time of the algorithm from that of theorem 5 by more than a constant factor. •

### Bounds on the Length of Reset Sequences

In his paper, Natarajan claimed a bound of  $O(kn^3)$  on the length of the reset sequences produced by algorithm 1. This can be tightened:

**Theorem 7.** The reset sequence found by algorithm 1, as implemented in theorems 5 and 6, has length at most  $O(n^3)$ .

Proof: There are  $O(n^2)$  pairs and singletons in the derived automaton, so each  $\tau_{i,j}$  has length bounded by  $O(n^2)$ . The reset sequence as a whole is the concatenation of at most  $n$  such sequences, so its length is bounded by  $O(n^3)$ . •

In fact, the algorithm can be modified so that  $(s_i, s_j)$  is always chosen to have the shortest sequence  $\tau_{i,j}$  among all pairs remaining in  $X$ , within the same asymptotic time and space bounds as before. Using this fact, we can bound the constant factor in the reset sequence length formula above.

If there are  $x$  states left in set  $X$ , then those states form  $x^2/2 + O(x)$  possible pairs; this, combined with the arrangement of all  $n^2/2 + O(n)$  possible pairs into a breadth first search tree, shows that the length of  $\tau_{i,j}$  can be at most  $(n^2 - x^2)/2 + O(n)$ . Therefore, the length of the entire reset sequence will be at most  $n^3/3 + O(n^2)$ .

This bound can be further tightened. We omit detailed proofs of the following facts because it seems unlikely that the bound they give is tight. The key observation is that, as the sequence  $\tau_{i,j}$  transforms  $X$ , each pair on the path from  $(s_i, s_j)$  to a singleton appears first as the image of  $(s_i, s_j)$ , and not as the image of any other pair; otherwise  $\tau_{i,j}$  would not be minimal. As a consequence, the length of  $\tau_{i,j}$  can be bounded by  $\min((n - x)^2, n(n - x)/2) + O(n)$ . This leads to a bound on the length of the entire reset sequence of  $11n^3/48 + O(n^2)$ .

### Difficulty of Computing Optimal Sequences

The algorithm we described for monotonic automata will always find the shortest possible reset sequence; in contrast, the algorithm for non-monotonic automata will always find a reset sequence if one exists, but will not necessarily find the shortest such sequence. This raises the question whether it is possible to efficiently find optimum length reset sequences in non-monotonic automata. We now show that this is unlikely.

**Theorem 8.** Finding the shortest possible reset sequence for an automaton is NP-complete.

Proof: More precisely the problem is, given an automaton and an integer parameter  $m$ , to test whether the automaton has a reset sequence of length less than or equal to  $m$ . By theorem 7, such a sequence need have at most polynomial length, so the problem is in *NP*. We prove completeness by reducing 3-SAT [1] to the problem.

Assume we are given a satisfiability problem as a Boolean formula in conjunctive normal form, with  $m$  variables  $x_1, x_2, \dots, x_m$ , and with  $n$  clauses. The automaton we construct will need only two transition functions  $\sigma_1$  and  $\sigma_2$ . There will always be a reset sequence of length  $m + 1$  (in fact any input sequence of that length will reset the automaton), but any reset sequence of length  $m$  or less will correspond to a satisfying assignment. The assignment is constructed by letting  $x_j$  be true if the  $j$ th input symbol of the reset sequence is  $\sigma_1$ , or false if the  $j$ th input symbol is  $\sigma_2$ . Conversely the opposite transformation will produce a reset sequence of length  $m$  from any satisfying assignment.

The automaton itself is constructed as follows. It will have one special state  $r$ , and  $mn + m$  other states  $s_{i,j}$  for  $1 \leq i \leq n$  and  $1 \leq j \leq m + 1$ . For all states  $s_{i,m+1}$ , and for state  $r$ , both transition functions will lead to state  $r$ . If the  $i$ th clause of the formula contains  $x_j$ ,  $\sigma_1$  will take state  $s_{i,j}$  to  $r$ ; if that clause contains  $\bar{x}_j$ ,  $\sigma_2$  will take  $s_{i,j}$  to  $r$ . We call these transitions to  $r$  from states other than  $s_{i,m+1}$  *shortcuts*. All remaining transitions take  $s_{i,j}$  to  $s_{i,j+1}$ .

It can be seen that, as we stated above, any input sequence will take all states to  $r$  in at most  $m + 1$  steps. Further, all states except  $s_{i,1}$  will always be taken to  $r$  in  $m$  steps, so we need only concern ourselves with the former states. If a reset sequence of length  $m$  or less exists, then each of these initial states  $s_{i,1}$  must be taken by that sequence across a shortcut transition, because otherwise an initial state  $s_{i,1}$  would progress through all the states  $s_{i,j}$  before reaching  $r$ , and that would take  $m + 1$  steps.

The variable assignment computed from the reset sequence must have a true variable in each clause, corresponding to the shortcut taken by the initial state corresponding to that clause. Thus we see that a satisfying assignment to the formula can be derived from a short reset sequence. Conversely, if an assignment satisfies the formula, the derived input sequence would cause each initial state  $s_{i,1}$  to take a shortcut corresponding to the first true variable in the corresponding clause, and we would have a reset sequence of length  $m$ . Thus we see that the formula will be satisfiable if and only if the derived automaton has a short reset sequence. •

## Conclusions and Open Problems

We have shown that, given a monotonic DFA and a set of initial states it may be in, we can construct a minimum length sequence (if one exists) that takes all the initial states to one particular state. This construction can be performed in time bounded by  $O(kn^2)$ . Further, we have shown that the length of the resulting sequence is at most  $n^2 - 2n + 1$ ; there are DFAs for which the minimum reset sequence exists and is this long, so this bound is tight.

We have also shown that, in the general case in which the automaton is not monotonic, we can still find a reset sequence for all of  $S$  in time bounded by  $O(n^3 + kn^2)$  and working space bounded by  $O(n^2)$ . The length of the resulting sequence is not necessarily optimal, but is bounded by  $O(n^3)$ .

Some questions remain open. For instance, the algorithm for monotonic automata may be performed in polylogarithmic parallel time using Kučera's breadth first search algorithm [3], and similarly we may perform the preprocessing in stages 1 and 2 of our algorithm for general automata in  $NC$ . But the main part of the latter algorithm seems to be inherently sequential; a natural question is whether it too can be performed in parallel, or whether some other algorithm exists that can find reset sequences in parallel. A partial result in this direction is that a reset sequence

can be found in random  $NC$ ; this can be done by choosing a long random sequence of pairs of states  $(s_i, s_j)$  and concatenating the sequences  $\tau_{i,j}$  that take each random pair to a singleton. If there are at least  $n^3$  pairs in the random sequence, then with very high probability the corresponding input sequence will be a reset sequence, and this can be tested in parallel.

Another open problem is the gap between the  $O(n^3)$  upper bound on the length of reset sequences for general automata, and the  $\Omega(n^2)$  lower bound given for the special case of monotonic automata.

### Acknowledgements

This research was performed while the author was a student in the Computer Science Department at Columbia University. It was supported in part by an NSF student fellowship, by NSF grants DCR-85-11713 and CCR-86-05353, and by DARPA contract N00039-84-C-0165.

I would like to thank Prof. John Kender, who led me to this problem and encouraged me to look for better solutions, and my advisor, Prof. Zvi Galil, for many helpful discussions.

### References

- [1] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979.
- [2] D.D. Grossman and M.W. Blasgen, *Orienting Parts by Computer Controlled Manipulation*, IEEE Trans. Systems, Man and Cybernetics 5(5), 1975, pp. 561–565.
- [3] L. Kučera, *Parallel Computation and Conflicts in Memory Access*, Info. Proc. Letters 14(2), April 1982, pp. 93–96.
- [4] M.T. Mason and M. Erdmann, *An Exploration of Sensorless Manipulation*, 3rd IEEE Intl. Conf. Robotics and Automation, San Francisco, April 1986.
- [5] B.K. Natarajan, *An Algorithmic Approach to the Automated Design of Parts Orienters*, Proc. 27th Annual Symp. Foundations of Computer Science, IEEE, 1986, pp. 132–142.