

Structurally Unambiguous Finite Automata^{*}

Hing Leung

Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, U.S.A.
`hleung@cs.nmsu.edu`

Abstract. We define a structurally unambiguous finite automaton (SUFA) to be a nondeterministic finite automaton (NFA) with one starting state q_0 such that for all input strings w and for any state q , there is at most one path from q_0 to q that consumes w . The definition of SUFA differs from the usual definition of an unambiguous finite automaton (UFA) in that the new definition is defined in terms of the transition logic of the finite automaton, and is independent of the choice of final states. We show that SUFA can be exponentially more succinct in the number of states than UFA and MDFA (deterministic finite automata with multiple initial states). Some interesting examples of SUFA are given. We argue that SUFA is a meaningful concept, and can have practical importance as it can be implemented efficiently on synchronous models of parallel computation.

1 Introduction

The descriptonal complexity of finite automata have been extensively studied since 1970's ([9], [11]). A recent survey on the descriptonal complexity of automata can be found in [3].

While deterministic finite automata (DFA) are more suitable for implementation, nondeterministic finite automata (NFA) can be exponentially more succinct in denoting regular languages. NFA are classified according to the amount of ambiguity used. Given an NFA M , we define the ambiguity of a string w to be the number of different accepting paths for w in M . An NFA is said to be k -ambiguous if every string in the language is accepted with at most k different accepting computations. An unambiguous NFA (UFA) is a 1-ambiguous NFA. An NFA is said to be finitely ambiguous (FNA) if the NFA is k -ambiguous for some positive integer k . There is a special class of FNA called deterministic finite automata with multiple initial states (MDFA) ([5] [4] [2] [13]) which is an NFA with deterministic transition logic. An MDFA is k -ambiguous where k is the number of starting states.

An NFA is polynomially ambiguous (PNA) if there exists a polynomial p such that every string x in the language is accepted with at most $p(|x|)$ accepting computations. Given an NFA of k states, any input string of length n can have

^{*} The research is partially supported by NSF MII grant CNS-0220590.

at most k^n different accepting computations. Thus, it follows that every NFA is exponentially ambiguous (ENA).

In Section 2, we define a variant of UFA which we call structurally unambiguous finite automata (SUFA). We present examples of interesting SUFA. We prove descriptional complexity tradeoffs results between SUFA, UFA, MDFA and reversal of MDFA. After we have established the technical results, we argue in Section 3 that the new model SUFA is a meaningful concept. It can also have practical importance as SUFA can be implemented efficiently on a synchronous model of parallel computation.

2 Structurally Unambiguous Finite Automata

A SUFA is an NFA $(Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, Σ is the alphabet set, $\delta \subseteq Q \times \Sigma \times Q$, q_0 is the starting state and F is the set of final states, such that for any string $w \in \Sigma^*$ and for any $q \in Q$, there is *at most* one path that goes from q_0 to q for processing w .

Note that SUFA is defined by referring to the transition logic of the finite automaton, and is independent of the way the set of final states is defined. That is, SUFA is a property of the structure of the transition logic, independent from the choice of the set of final states.

If there is only one final state, that is $|F| = 1$, then a SUFA is also a UFA. However, SUFA may differ from UFA when there is more than one final state.

Similarly, we define a generalized structurally unambiguous finite automaton (GSUFA) in the same way as SUFA except that we allow more than one starting states. Specifically, a GSUFA is an NFA $(Q, \Sigma, \delta, S, F)$ such that for any $q, q' \in Q$ and for any string $w \in \Sigma^*$, there is *at most* one path that goes from q to q' for processing w . It is possible that in processing the same input w , two different paths beginning from different states may arrive at the same state q' .

One can see that both GSUFA and SUFA are subclasses of FNA as the amount of ambiguity of GSUFA and SUFA are bounded by n^2 and n respectively, where n is the number of states. It is interesting to compare the descriptional complexity of the new models with UFA and MDFA, which are also subclasses of FNA.

We can see that the descriptional complexity of SUFA and GSUFA in terms of the number of states are polynomially related. It is clear that a SUFA is a GSUFA. Given an n -state GSUFA $(Q, \Sigma, \delta, S, F)$ where $S = \{s_1, s_2, \dots, s_k\} \subseteq Q$ is the set of starting states, we replicate from the GSUFA logic k disjoint copies of SUFA $M_i = (Q, \Sigma, \delta, s_i, F)$, where $1 \leq i \leq k$. Next, we introduce a new starting state s where $s \notin Q$ and create ϵ transitions from s to the starting state s_i of each of the k SUFA. By substituting the ϵ -moves with direct non- ϵ -moves, we obtain a $O(n^2)$ -state SUFA equivalent to the given GSUFA. As a MDFA is a GSUFA, consequently neither MDFA and GSUFA can offer significant (bigger than polynomial) advantage in descriptional sizes over SUFA.

Consider an n -state NFA M with $Q = \{q_1, q_2, \dots, q_n\}$, the alphabet set Σ and the transition function δ . Let $1 \leq i, j \leq n$. We define $M_{i,j} = (Q, \Sigma, \delta, q_i, q_j)$. Then M is a GSUFA iff $M_{i,j}$ is a UFA for all $i, j \in \{1, \dots, n\}$. Stearns and

Hunt [12] showed that there is a polynomial time algorithm for checking if a given NFA is a UFA. To test if a given n -state NFA is a GSUFA, we can apply the polynomial time UFA-testing algorithm n^2 times. Similarly, we can apply the UFA-testing algorithm n times to check if a given NFA is a SUFA. Therefore, whether a NFA is a SUFA (or, GSUFA) can be determined in polynomial time.

In [12], it is shown that equivalence and containment problems for UFA can be decided in polynomially time. However, this is not the case for SUFA in general. We can show that the equivalence problem for SUFA is PSPACE-complete. In [1] (p. 266), it is known that the DFA intersection problem is PSPACE-complete. Since DFA are efficiently closed under complementation, the following *union-universe* problem [4] is also PSPACE-complete: Given a number of DFA G_1, G_2, \dots, G_n over an input alphabet Σ , we ask whether the union of the languages accepted by G_1, G_2, \dots, G_n is Σ^* . One can consider the disjoint union of the n DFA as a MDFA G , which is a SUFA. We can thus reduce the union-universe problem to an instance of the SUFA equivalence problem with the instance consisting of the SUFA G and a single state DFA (which is a SUFA) accepting Σ^* . Therefore, the equivalence problem for SUFA is PSPACE-hard. As the equivalence problem for NFA is in PSPACE, we conclude that the equivalence problem for SUFA is PSPACE-complete. It is not difficult to see that the containment problem is also PSPACE-complete since the equivalence problem can be easily reduced to the containment problem, which is also PSPACE-solvable. On the other hand, if the number of final states in a SUFA is bounded by a constant k (independent of the number of states n), the equivalence and containment problems are solvable in polynomial time as Stearns and Hunt [12] had showed that the corresponding problems are polynomial time solvable for k -ambiguous finite automata.

In [8], we introduced a language of “some-register-on”. Suppose there are n registers. Each register holds a value of either 0 or 1 (‘off’ or ‘on’). Initially, register 1 is on. All other registers are off. Consider an instruction

Copy i to j

Executing the instruction will copy the current value of register i to register j . In short, the instruction is given as $C_{i,j}$.

We define an input string to consist of a sequence of copy instructions. As copying register i to itself is a dummy instruction, we assume that $C_{i,i}$ is not allowed. As there are $n(n-1)$ possible copy instructions, the input alphabet has $O(n^2)$ letters.

An example input is $C_{1,4}C_{4,2}C_{3,1}C_{4,3}C_{1,2}$. We say that an input is in the language of some-register-on if some register is on after the sequence of copy instructions have been performed.

Consider the example input given before. Initially, register 1 is on. The first copy instruction $C_{1,4}$ will turn register 4 to on. The next instruction $C_{4,2}$ will turn on register 2. The next instruction $C_{3,1}$ sets register 1 to off as register 3 is off. Next, instruction $C_{4,3}$ turns on register 3. The last instruction $C_{1,2}$ turns register 2 to off. Thus, after all copy instructions are performed, registers 3 and 4 are on whereas registers 1 and 2 are off. Since not all registers are off, we conclude that the input belongs to the language of some-register-on.

We define an n -state SUFA for the language of some-register-on. State 1 is an initial state. The design intuition is that state i is loaded when register i is on. With respect to the input symbol $C_{i,j}$, there are transitions going from state i to state j , and transitions going from state k to state k where $k \neq j$. All states are final states. Formally, the NFA is $A = (Q, \Sigma, \delta_A, q_0, F)$ where $Q = \{1, 2, \dots, n\}$, $\Sigma = \{C_{i,j} \mid 1 \leq i, j \leq n, i \neq j\}$, $q_0 = 1$, $F = Q$ and $\delta_A = \{(k, C_{i,j}, k) \mid 1 \leq i, j, k \leq n, i \neq j \neq k\} \cup \{(i, C_{i,j}, j) \mid 1 \leq i, j \leq n, i \neq j\}$. Nondeterminism occurs when the NFA is at state i given that the current input symbol is $C_{i,j}$. The NFA can remain at state i or go to state j . Other transitions are self loops.

To see that A is an SUFA, we reverse the transition directions and the roles of starting and final states from A . We denote the reversal of A as A^R . The logic after reversing the transitions is deterministic. But it differs from a DFA in that all the n states are starting states. Thus, A^R is a MDFA. Since a MDFA is a GSUFA, the reversal automaton of a MDFA is also a GSUFA. That is, A is a GSUFA with one starting state; hence, A is a SUFA.

We can show that the smallest DFA for the language of some-register-on has 2^n states. This is because all subsets of states considered by the subset construction are reachable, and any two subsets of states are distinguishable in the sense of Myhill-Nerode theorem.

In the next theorem, we show that a UFA for the language of some-register-on has at least $2^n - 1$ states.

Theorem 1. *Let $n \geq 3$. The smallest UFA for the language L of some-register-on with n registers has at least $2^n - 1$ states.*

Proof. (Sketch) The technique for proving lower bound on the size of a UFA is introduced by Schmidt [11].

For $\emptyset \neq Q' \subseteq Q = \{1, 2, \dots, n\}$, we want to define $x_{Q'}$ such that A reaches the subset Q' of states when processing $x_{Q'}$ from the starting state 1. Let $Q' = \{q_1, q_2, \dots, q_k\}$.

Case 1. Suppose $1 \in Q'$. Define $x_{Q'} = C_{1,q_1} C_{1,q_2} \dots C_{1,q_k}$.

Case 2. Suppose $1 \notin Q'$. Let $q \in Q - \{1, q_1\}$. Define $x_{Q'} = C_{1,q_1} C_{q_1,q_2} C_{q_1,q_3} \dots C_{q_1,q_k}$.

Similarly, for $\emptyset \neq Q'' \subseteq Q = \{1, 2, \dots, n\}$, we want to define $y_{Q''}$ such that A^R reaches the subset Q'' of states when processing the symbols of the string $y_{Q''}$ from right to left. Let $Q'' = \{q_1, q_2, \dots, q_k\}$.

Case 1. Suppose $Q'' = Q$. Define $y_{Q''} = \epsilon$.

Case 2. Suppose $Q'' \neq Q$. Let $Q - Q'' = \{q'_1, q'_2, \dots, q'_h\}$ where $h + k = n$. Define $y_{Q''} = C_{q'_2,q'_1} C_{q'_3,q'_2} \dots C_{q'_h,q'_{h-1}} C_{q_1,q'_h}$.

We can see that $x_{Q'} y_{Q''} \in L$ iff $Q' \cap Q'' \neq \emptyset$. We define a matrix M indexed by nonempty subsets of states such that entry $[Q', Q'']$ has the value 1 if $Q' \cap Q'' \neq \emptyset$, otherwise the entry has the value 0. It has been shown in [7] that M has rank $2^n - 1$. Then, by Schmidt's technique ([11] [8]), the smallest UFA equivalent to A has $2^n - 1$ states. \square

Theorem 1 shows that we can achieve the biggest tradeoff between SUFA and UFA. However, the language of some-register-on is over an alphabet of size

$O(n^2)$ whereas the number of states in A is only n . Using a binary encoding for the $O(n^2)$ letters, the number of states in A becomes $n + O(n^2) \cdot O(\log n^2) = O(n^2 \log n)$.

On the other hand, in [8], we have shown that there exists a family of n -state MDFA, where all states are starting states, over a binary alphabet such that the smallest UFA has $2^n - 1$ states. As explained before, a MDFA can be transformed into a $O(n^2)$ -state SUFA. Therefore, we have another family of SUFA demonstrating exponential succinctness in the number of states over UFA.

As a consequence, we have

Corollary 2. *SUFA can be exponentially more succinct in the number of states than UFA.*

Not only that SUFA can be exponentially more succinct in descriptioinal sizes than UFA, it can also be exponentially more succinct than MDFA.

Lemma 3. *The smallest MDFA for the language L of some-register-on with n registers has at least $2^n - 2$ states.*

Proof. Suppose the contrary that there exists a k -entry MDFA with less than $2^n - 2$ states for L . The MDFA can be considered as a nondeterministic union of k DFA (named D_1, D_2, \dots, D_k) each having less than $2^n - 2$ states. We consider each DFA D_i as an incompletely specified DFA such that every state in D_i are reachable from the start state and can reach some final state. Moreover, we can assume that every state in D_i is indeed an accepting state. This is because a state that can reach some final state is reached by the processing of some prefix of a string in L , where L has the property that all prefixes of strings in L are also in L .

From the subset construction of A , the state that corresponds to the subset Q is a state that once entered, the subset construction automaton will never leave the state. A string w that causes the subset construction automaton to go into this accepting “sink” state satisfies the property that $w^{-1}L = \Sigma^*$. In the following discussion, we deliberately avoid constructing strings that belong to $\{w \mid w^{-1}L = \Sigma^*\}$.

On the other hand, for all nonempty subsets of states $Q' \subsetneq Q$, there exists a string that will cause the subset construction automaton to return to the state that corresponds to a set consisting only of the starting state 1. Thus, for any string $u \in L$ such that A reaches a nonempty subset $Q' \subsetneq Q$ of states, there exists a string v such that $(uv)^{-1}L = L$.

Recall that the smallest DFA for L has 2^n states where one of the state is a non-accepting dead state and another state is an accepting sink state. Moreover, the rest of the $2^n - 2$ states in the DFA obtained by the subset construction are strongly connected as there is a resetting mechanism which we have discussed.

Suppose D_i arrives at state q on processing the string u from the start state. Recall that u is designed such that $u \in L - \{w \mid w^{-1}L = \Sigma^*\}$. Let v be a string such that $(uv)^{-1}L = L$. We resume the processing of D_i from state q to process v . It is possible that D_i aborts, or it may arrive at a state q' . As D_i does not have $2^n - 2$ states and the language accepted by D_i is a proper subset of

L , together with the fact that all states in D_i are accepting states, we deduce that there must exist a string $y \in L$ such that D_i aborts in processing y from state q' . We conclude that from any state q that D_i reaches on processing a string $u \in L - \{w \mid w^{-1}L = \Sigma^*\}$, there is a string z such that D_i aborts when processing z from state q , whereas $uz \in L - \{w \mid w^{-1}L = \Sigma^*\}$.

We consider DFA D_1 . There exists some string $w_1 \in L - \{w \mid w^{-1}L = \Sigma^*\}$ such that D_1 aborts in processing w_1 from the starting state. On processing w_1 , DFA D_2 may abort or may arrive at a state q_2 . There exists w_2 such that $w_1w_2 \in L - \{w \mid w^{-1}L = \Sigma^*\}$ but D_2 aborts when processing w_2 from q_2 . Similarly, on processing w_1w_2 , DFA D_3 may abort or may arrive at a state q_3 . There exists w_3 such that $w_1w_2w_3 \in L$ but D_3 aborts when processing w_3 from q_3 . We can repeat this process to obtain a string $w = w_1w_2 \dots w_k \in L - \{w \mid w^{-1}L = \Sigma^*\}$ such that every D_i aborts in processing w . Thus, the MDFA cannot recognize L , a contradiction to the assumption that the MDFA has less than $2^n - 2$ states. \square

In fact, the above lemma can be strengthened to show that the smallest MDFA for the language of some-register-on has at least $2^n - 1$ states.

Theorem 4. *The smallest MDFA for the language L of some-register-on with n registers has at least $2^n - 1$ states.*

Proof. (Sketch) We continue with the analysis given in the previous lemma. Suppose the contrary that there is a MDFA of $2^n - 2$ states for L . Each DFA D_i (as defined in the previous proof), where $1 \leq i \leq k$, can be assumed to have at least $2^n - 2$ states. Otherwise, we can show that any DFA D_i with less than $2^n - 2$ states is not needed as we can use the same technique as in the previous proof to ‘attack’ D_i by a string $w \in L$ that possesses the resetting property $w^{-1}L = L$. Next, as the previous proof considers strings $w \in L - \{w \mid w^{-1}L = \Sigma^*\}$, we can argue that the $2^n - 2$ states of each D_i can be identified with the $2^n - 2$ non-sink (accepting) states of the DFA obtained by applying the subset construction to A . Moreover, all the different D_i ’s are functionally equivalent. Therefore, we can reduce them to only one DFA which is of $2^n - 2$ states. But this is a contradiction as we know that the smallest incompletely specified DFA for L has $2^n - 1$ states. \square

As a consequence, we have

Corollary 5. *SUFA can be exponentially more succinct in the number of states than MDFA.*

Note that the SUFA A is the reversal of a MDFA. Theorem 4 also shows that the reversal of a MDFA can be exponentially more succinct than a MDFA. This should not be a surprise as the reversal of a DFA can also be exponentially more succinct than a DFA [8]. Let L^R denote the reversal of L . It is clear that L^R can be recognized by the n -state MDFA A^R . On the other hand, the statement that a MDFA requires at least $2^n - 1$ states to recognize L can be restated as L^R requires $2^n - 1$ states for the reversal of a MDFA to accept. Therefore, we have the next corollary:

Corollary 6. *The reversal of a MDFA can be exponentially more succinct than a MDFA. A MDFA can be exponentially more succinct than the reversal of a MDFA.*

In [8], it is shown that MDFA can be exponentially more succinct than UFA. On the other hand, we are going to show that UFA can also be exponentially more succinct than MDFA.

We modify the language of some-register-on. Instead of accepting a string when some register is on, we accept a string only if the register that we query at the end of the input is on. That is, the end of an input is augmented by a query instruction.

Assert: Register i is on

An input is accepted if the register i queried is on. In short, the query instruction is denoted as Q_i . We denote the new language L_1 .

We extend the previous example input by a query Q_3 . The example input becomes $C_{1,4}C_{4,2}C_{3,1}C_{4,3}C_{1,2}Q_3$. Since register 3 is on after the copy instructions are performed, the input is accepted as the query is about register 3. If we query about register 1 at the end of the input as in $C_{1,4}C_{4,2}C_{3,1}C_{4,3}C_{1,2}Q_1$, then the input is not accepted since register 1 is off.

To handle the newly added query feature, we modify the SUFA for the language of some-register-on. We introduce a new state called f , which is the only final state. New transitions are added: from each state i , on processing Q_i , it will go to state f . The resulting $(n + 1)$ -state NFA is a UFA. We can see this as the reversal of the NFA is a DFA. In fact, it has been shown [8] that the UFA is the smallest UFA for the language.

We can argue that the smallest MDFA for L_1 has at least $2^n - 1$ states.

Corollary 7. *The smallest MDFA for L_1 has at least $2^n - 1$ states. Hence, UFA can be exponentially more succinct in the number of states than MDFA.*

Proof. Suppose the contrary that there is a MDFA A_1 for L_1 with less than $2^n - 1$ states. We can assume without loss of generality that all states in A_1 are useful in the sense that each state in A_1 can reach some final state. We can modify A_1 to give a MDFA for L . The modifications are as follows: Remove all transitions labelled with queries and define every state to be an accepting state. It is easy to see that the resulting modified automaton is a MDFA for L with less than $2^n - 1$ states. But this contradicts with the result of Theorem 4, which states that the smallest MDFA for L has at least $2^n - 1$ states. \square

In the literature, we have seen UFA designed as the reversals of DFA and FNA designed as the reversals of MDFA. Our example of SUFA, the language of some-register-on, is also the reversal of a MDFA.

As we have shown that the reversal of a MDFA is a SUFA, and a n -state MDFA can be converted to an equivalent SUFA with $O(n^2)$ states. One may wonder if SUFA is just the study of finite automata that are MDFA, or the reversals of MDFA. We answer the question by constructing a SUFA such that

any equivalent MDFA, or reversal of MDFA, requires an exponential blow up in the number of states.

Recall that L denotes the language of some-register-on over the alphabet $\Sigma = \{C_{i,j} \mid 1 \leq i, j \leq n, i \neq j\}$. We define a language $L' = L^R \cdot b \cdot L \subseteq (\Sigma \cup \{b\})^*$, where b is a new symbol not in Σ . Note that the reversal of L' is L' itself.

An initial design of a finite automaton B for L' does not give us a SUFA. We connect A^R with a transition labelled by b , followed with A . Specifically, the copy of A^R is a MDFA with n starting states and a final state 1 (which is considered as a final state of A^R , but it is not exactly a final state of B). From the state 1 of A^R , we have a transition labelled by b that goes to the state 1 of A which is a SUFA with all n states accepting. Together, B has $2n$ states.

We replicate n copies B_1, B_2, \dots, B_n of B , each with a different starting state. That is, B_i is a SUFA with state i of A^R as the starting state. We introduce a new starting state for the nondeterministic union of B_1, B_2, \dots, B_n . We call the resulting SUFA B' , which has $O(n^2)$ states over the alphabet $\Sigma \cup \{b\}$ of size $O(n^2)$.

Theorem 8. *The smallest MDFA for L' has at least $2^n - 1$ states. The smallest NFA for L' that is the reversal of a MDFA has at least $2^n - 1$ states.*

Proof. Suppose there is a MDFA N for L' with less than $2^n - 1$ states. We derive from N a finite automaton for the language of some-register-on by removing all transitions that are encountered before N processes the symbol b . We define the starting states to be $\{q \mid (q', b, q) \text{ is a transition in } N\}$. The resulting finite automaton is a MDFA for the language of some-register-on. As N has less than $2^n - 1$ states by assumption, the resulting finite automaton is a MDFA for the language of some-register-on with less than $2^n - 1$ states, which contradicts the statement of Theorem 4.

Suppose there is a finite automaton N' , which is the reversal of a MDFA, that recognizes L' with less than $2^n - 1$ states. The reversal of N' is a MDFA that recognizes the reverse of L' , which is again L' . But this contradicts the previous result. \square

As a consequence, we have

Corollary 9. *SUFA can be exponentially more succinct in the number of states than MDFA and the reversal of MDFA simultaneously with respect to the same language family.*

Using a binary encoding for the alphabet symbols, we obtain from B' a SUFA for L' with $O(n^3 \log n)$ states.

3 Why SUFA?

In Section 2, we have shown that SUFA can be exponentially more succinct than UFA and MDFA (also, reversals of MDFA) for denoting some family of regular languages. On the other hand, SUFA will not do worse than equivalent MDFA (or, reversals of MDFA) by more than a quadratic blow up in sizes.

From the descriptonal complexity results, we see that SUFA is a stronger model than UFA and MDFA (also, reversals of MDFA). But, one may wonder whether SUFA has practical significance for the practitioners implementing finite automata for online processing of input strings.

SUFA can be implemented efficiently on a synchronous model of parallel computation. One process thread can be assigned to each state. When the state is off, the process thread is waiting to be woken up by another thread. As there is only one path arriving at a state at any moment, there will not be two messages sending to a process thread at the same time. Thus, the process thread will not require any buffer to hold the incoming messages in the synchronous computation.

NFA have been classified into UFA, FNA, PNA and ENA according to the ambiguity levels exhibited. As the amount of ambiguity is defined in terms of the number of accepting computations, the classification depends on the choice of the set of final states, which determines the language denoted.

Structural properties are obtained that offer equivalent characterizations of FNA, PNA and ENA. Suppose all states in an NFA are useful; that is, every state can reach some final state, and can be reached from some starting state. It is shown ([6] [10] [14]) that an NFA is strictly exponentially ambiguous if and only if there exists a state q and a string w such that there are more than one path from q to q processing w ; an NFA is strictly polynomially ambiguous if and only if the NFA is not strictly exponentially ambiguous and there exists different states p, q and a non-empty string w such that there are paths for processing w that goes from p to itself, from q to itself and from p to q ; an NFA is finitely ambiguous if and only if the NFA is not strictly polynomially ambiguous.

Observe that the structural properties are defined in terms of the transition logic of an automaton, but not on the set of final states. The characterizations for ENA, PNA and FNA show that one can replace the semantic definition of ambiguity levels exhibited by a NFA by the structural definition.

In this paper, we have shown that the structural definition of unambiguous finite automata differs from the semantic definition. That is, SUFA and UFA are not the same class. Unlike the undesirable effect that ambiguity has on the parsing of programs, ambiguity in NFA are used to reduce the descriptonal size. It is therefore not necessary to demand an unambiguous finite automaton to allow only one accepting path for each string accepted. Moreover, from a practitioner's point of view, there is no drawback in adopting SUFA as the definition of unambiguous finite automata as it allows efficient synchronous parallel processing.

The classes of SUFA, FNA, PNA and ENA are forming a nice proper hierarchy in that the next model in the hierarchy is more general and could be exponentially more succinct than the previous model. Note that it is still a conjecture that PNA can be exponentially more succinct than FNA.

On the other hand, the models UFA, MDFA and reversal of MDFA are incomparable to each other as it has been shown that UFA can be exponentially more succinct than MDFA (Corollary 7), MDFA can be exponentially more succinct

than UFA [8], MDFA can be exponentially more succinct than reversal of MDFA (Corollary 6) and reversal of MDFA can be exponentially more succinct than MDFA (Corollary 6). As the reversal of a UFA is a UFA, we can also conclude from Corollary 7 and [8] that UFA can be exponentially more succinct than reversal of MDFA, and reversal of MDFA can be exponentially more succinct than UFA.

Finally, the models UFA, MDFA and reversal of MDFA are proper subclasses of SUFA. It is shown that SUFA can be exponentially more succinct than UFA (Corollary 2), and SUFA can be exponentially more succinct than MDFA and the reversal of MDFA simultaneously (Corollary 9).

Acknowledgements

The author would like to thank Enrico Pontelli for valuable discussions.

References

1. Garey, M.R., Johnson, D.S.: Computers and intractability, W. H. Freeman and Company, San Francisco, 1979.
2. Gill, A., Kou, L.T.: Multiple-entry finite automata, *J. Comput. System Sci.* **9** (1974) 1–19.
3. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptive complexity of machines with limited resources, *J. Univ. Comput. Sci.* **8** (2002) 193–234.
4. Holzer, M., Salomaa, K., Yu, S.: On the state complexity of k -entry deterministic finite automata, *J. Autom. Lang. Comb.* **6** (2001) 453–466.
5. Kappes, M.: Descriptive complexity of deterministic finite automata with multiple initial states, *J. Autom. Lang. Comb.* **5** (2000) 269–278.
6. Ibarra, O., Ravikumar, B.: On sparseness, ambiguity and other decision problems for acceptors and transducers, In *Proc. 3rd STACS*, Orsay, France, (1986) 171–179.
7. Leung, H.: Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata, *SIAM J. Comput.* **27** (1988) 1073–1082.
8. Leung, H.: Descriptive complexity of NFA of different ambiguity, *International Journal of Foundations of Computer Science* **16** (2005) 975–984.
9. Meyer, A.R., Fischer, M.: Economy of description by automata, grammars, and formal systems, *IEEE Twelfth Annual Symposium on Switching and Automata Theory*, (1971), 188–191.
10. Ravikumar, B., Ibarra, O.: Relating the type of ambiguity of finite automata to the succinctness of their representation, *SIAM J. Comput.* **18** (1989) 1263–1282.
11. Schmidt, E.M.: Succinctness of descriptions of context-free, regular and finite languages, PhD Thesis, Cornell University, Ithaca, NY (1978).
12. Stearns, R.E., Hunt, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata, *SIAM J. Comput.* **14** (1985) 598–611.
13. Veloso, P.A.S., Gill, A.: Some remarks on multiple-entry finite automata, *J. Comput. System. Sci.* **18** (1979) 304–306.
14. Weber A., Seidl, H.: On the degree of ambiguity of finite automata. *Theor. Comput. Sci.* **88** (1991) 325–349.