Improvements in FSM Evolutions from Partial Input/Output Sequences

Sérgio G. Araújo, A. Mesquita, Aloysio C. P. Pedroza

Electrical Engineering Dept.
Federal University of Rio de Janeiro
C.P. 68504 - CEP 21945-970 - Rio de Janeiro - RJ - Brazil
Tel: +55 21 2260-5010 - Fax: +55 21 2290-6626
{granato,aloysio}@gta.ufrj.br mesquita@coe.ufrj.br

Abstract. This work focuses on the synthesis of finite-state machines (FSMs) by observing its input/output behaviors. Evolutionary approaches that have been proposed to solve this problem do not include strategies to escape from local optima, a typical problem found in simple evolutionary algorithms, particularly in the evolution of sequential machines. Simulations show that the proposed approach improves significantly the state space search.

1 Introduction and Related Works

The task of modeling existing systems by exclusively observing their input/output (I/O) behavior is of interest when the purpose is (1) to uncover the states of natural systems, (2) to model synthetic systems implemented from scratch, in an ad hoc manner or for which the documentation has been lost and (3) to generate automata from scenario (trace) specifications, a fruitful area in the telecommunications domain. At present, most of these problems are outside the scope of conventional techniques.

Evolutionary Algorithms (EAs), and variations, have been used in the synthesis of sequential machines to explore the solutions space. Early attempts dating back to the 60's [1] used some of the today EAs concepts such as population, random initialization, generation, mutation and reproduction (cloning) to evolve an automaton that predicts outputs based on known input sequences. However, these early approaches have shown poor performance by lack of the crossover operator [7].

More recent attempts have been successful in synthesizing sequential systems with the aid of Genetic Algorithms (GAs) [2, 3, 4]. In [1] an approach to synthesize synchronous sequential logic circuits from partial I/O sequences by using *technology-based* representation (a netlist of gates and flip-flops) is presented. The method is able to synthesize a variety of small FSMs, such as serial adders and 4-bit sequence detectors. With the same purpose other works [3, 4] used technology-independent *state-based* representation in which the next-state and the output, corresponding to each current-state/input pair of the state-transition table, are coded in a binary string

defined as the *chromosome*. Because, in this case, the number of states of the finite-state automaton is unknown, a large number of states must be used at the start.

The proposed approach differs from the previous ones by the use of the fitness gradient to improve the system performance in the presence of evolutionary stagnation phenomenon. Stagnation problems refer to a situation in which the optimum seeking process stagnates before finding a global optimal solution. Stagnation at local optima where all the neighboring solutions are non-improving is frequently found in simple EAs [13], particularly in the evolution of complex systems such as state-machines. This gap in the automata evolution may persevere for thousandths of generations due to an "unproductive" population. To overcome this problem, the proposed approach penalizes the best individual and its variants and a new population emerges in order to surpass the previous best fitness value. Simulations show that the proposed approach improves significantly the state space search.

2 Definitions

2.1 FSMs

Finite-state machines (FSMs) are commonly used for specifying reactive systems due to its simplicity and precise definition of the temporal ordering of interactions [5]. From the two traditional Moore/Mealy FSMs models it is found that in the Mealy machines, since the outputs are associated with the transitions, some behaviors can be implemented with fewer states than Moore machines.

The Mealy FSM model M is formally defined as a 7-tuple $\{Q,V,t,q_0,V',o,D\}$ where Q $\varnothing$ is a finite set of states of M, V is a finite input alphabet, t is the state transition function, q_0 $\hat{\mathbf{I}}$ Q is the initial state, V' is a finite output alphabet, o is the output function and D is the specification domain, which is a subset of $Q \times V$. t and o together characterize the behavior of the FSM, i.e., $t(q,v)$: $Q \times V$ Q and $o(q,v)$: $Q \times V$ V'. If $D = Q \times V$, then t and o are defined for all possible state/input combinations and therefore the FSM is said to be *completely specified*. Deterministic and completely specified FSMs are preferred strategies for automata modeling since they tell it exactly how to react in every situation it perceives [11].

2.2 Evolutionary Computation

Evolutionary Computation (EC) concerns the design and analysis of probabilistic algorithms inspired by the principles of Darwinian natural selection. The Genetic Algorithm (GA) and the Genetic Programming (GP) are the most familiar instances of EC algorithms.

The *chromosome* is the basic component of the GA [6]. It represents a point (an individual) in the problem solution space. The fitness value measures how close the

individual is to the solution. By iteratively applying the genetic operators (fitness evaluation, fitness-based selection, reproduction, crossover and mutation) to a randomly started population of individuals, it evolves in order to breed at least one offspring with a given target behavior. GAs usually employ *fixed length* chromosome strings in which one or more parameters are coded. GA solutions are best applied to poorly understood or highly non-linear problems for which deterministic solutions are not available. GP [7] is a branch of the GA, the solutions representation being the main difference. Unlike GA, GP can easily code chromosomes of *variable length*, which increases the flexibility in structure production.

3 Methodology

The proposed methodology is aimed to solve the problem of finding a completely specified deterministic FSM consistent with a given sample of I/O sequences called *training sequences* (*TSs*). A *TS* is defined as a finite sequence of *correct* input-output pairs $< v_1/v_1', v_2/v_2' \ldots v_L/v_L' >$, where $v \in V$, $v' \in V'$ and L is the length of the sequence (L inference is discussed in subsection 3.1). The methodology execution flow is shown in Fig. 1a. The population of FSMs supplied by the GP is evaluated for the fitness by means of the *TSs*; if at least one individual reproduces the I/O behavior specified by the *TSs*, the algorithm stops. The resulting FSM describes the observed system.

The *black-box* approach depicted in Fig. 1b is adopted for fitness evaluation. In this case the automaton to be evolved interacts with the environment through an event-driven interface. In each evolutionary step (generation) the system probes a population of FSMs with input sequences and records the corresponding output sequences. These output sequences are compared with the *correct* (desired) output sequences (the outputs of the *TSs*) and a fitness value is assigned to each candidate solution.

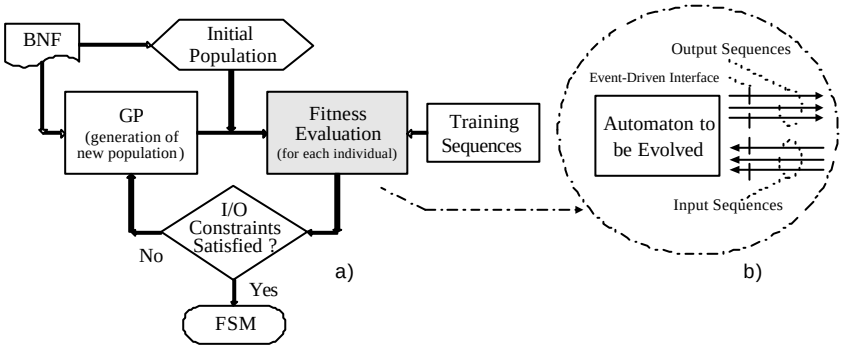


Fig. 1. a) Execution flow of the methodology; b) Model for fitness evaluation

Unlike other approaches [3, 4], the proposed methodology includes a heuristic to help the algorithm to evade from local optima. One of the main causes of premature convergence is loss of genetic diversity due to selection pressure [13]. Moreover, population diversity tends to get worst at local optima. To support the emergence of new populations, carrying out alternative solutions, the heuristic penalizes the fitness of the best individual and of its variants each time a local-optimum condition is detected. The penalty applied to outstanding fitness FSMs is defined by a *penalty factor Pf* ($0.0 \leq Pf < 1.0$) and variants of the best individual are defined by a *similarity factor* based on the Hamming distance concept. The classic binary Hamming distance was extended to a more adequate notation system, being redefined as the number of components (genes) by which two vectors (chromosomes) differ. Given a reference chromosome with *K*-genes, C_R, the similarity factor *Sf* of any chromosome with respect to C_R is defined as $Sf = (K-H)/K$, where *H* is the Hamming distance, in *K*-dimensional Hamming space, between them. For instance, the similarity factor between 130212 012001 140120 (C_R coding a 3-state FSM) and 130**10**2 012023 140122 is $Sf = (18-5)/18 = 0.72$.

The intention of the heuristic is to eliminate the building blocks (or schemas), which push the individuals to the local optimum, from the current population. *Pf* is triggered as the gradient of the best individual fitness (*GR*) drops below a pre-defined threshold. At this time, the system reduces diversity-decreasing operators (i.e., crossover and cloning - see [12]) rate and raises the mutation probability (p_m) to increase the population diversity. *Pf* remains active for sufficient time to break the "bad" building blocks. However, the genetic operators only return to their original values as the system breeds a solution improving the previous best fitness value.

3.1 Training Sequence (*TS*) Length

The *TSs* must be long enough to exercise all paths of the FSM that describes the observed system. Ref. [2] gives an approximation formula to estimate the length of the *TSs* that yields a correct FSM, based on the *waiting times in sampling* problem solution [10]. This formula defines the length of the input sequence as $L = E(S) \times E(I)$, where $E(S)$ and $E(I)$ are the *expected number of state transitions* and *expected number of inputs*, respectively. As example, $E(N)$ can be computed using $E(N) = N\,(1 + {}^1\!/_2 + \ldots + {}^1\!/_N)$. However, since the number of states *S* required to describe the system is unknown, it must be overestimated a priori.

3.2 Chromosome Coding

The chromosome, which encodes a FSM, uses state-based representation (SBR). The resulting string (chromosome) with *S* states and *I* inputs is shown in Fig. 2.

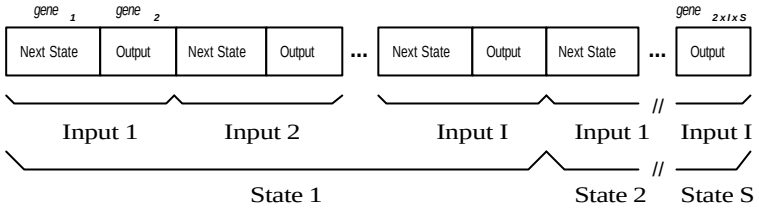


Fig. 2. Chromosome coding using SBR

The GP Kernel (GPK) [8], a complex G^3P (Grammar-Guided GP) system [9], was used in the present work to evolve automata encoded as syntactic trees. GPK system requires a Backus-Naur form (BNF) for chromosome structuring. The BNF that allows the generation of chromosomes representing six-state FSMs is defined as:

```
S          := <stat>;
<stat>                                                              :=
<in_even><in_even>><in_even>><in_even>><in_even>><in_even>;
<in_even>   :=   <next_st><out>   <next_st><out>   <next_st><out>
             <next_st><out>
             <next_st><out> <next_st><out>;
<next_st> := "0"|"1"|"2"|"3"|"4"|"5";
<out>      := "0"|"1"|"2"|"3"|"4"|"5"|"6";
```

Note that <*in_even*> fixes the number of input events (six, in the above BNF, each one yielding a next-state/output pair, i.e., <*next_st*> <*out*>). In contrast to the classic GA, where genes are simple bit-sequences and crossover may be performed at any point, GPK only permits crossover by swapping sub-trees (of a couple) starting with the same non-terminal symbols (i.e., symbols between corner brackets). Moreover, the mutation operation is implemented as a crossover operation between the selected individual and a temporary random-derived tree.

3.3 Fitness Evaluation

The fitness value assigned to a given FSM behavior, here evaluated through an input/output sequence perspective, is defined by the fitness function *F* in the form:

$$F = \sum_{i=1}^{N} w_i H_i \qquad (1)$$

Where w_i is a weighting factor for fitness case i, N is the number of fitness cases (*TSs*) and H_i is the number of *output hits* due to the fitness case i (TS_i). H_i is evaluated as follows. Initially, the FSM must be in the reset (idle) state. In the sequence, for each

input of the TS_i, its output is compared with the *correct* output and an output hit is signed in case of a match.

4 EXPERIMENT

The goal is to generate a protocol entity specification, in FSM model, for the sender entity of a connection-oriented protocol (PS_{SND}) from given I/O event samples. The *TS* length was evaluated using six inputs (see coding table of Fig. 4) and an estimated value of five for *S*, leading to a 168-input/output *TS* (see subsection 3.1). Moreover, it is desirable to have multiple *TSs* to improve the performance of the learning system (see [4]). In fact, eighteen *TSs* (N = 18) were used, each with 32 bits in length, which corresponds to more than three 168-length sequences. w_i was set to 1 for all *i* since the *TSs* have the same length.

The population size (M) was set to 200 and the maximum number of generations (G_{MAX}), 5,000. The crossover (two-point shaped), the mutation and the reproduction probabilities were set to p_c = 0.65, p_m = 0.05 and p_r = 0.30, respectively. Linear rank selection was used considering the elitist strategy. The population of FSMs was shaped using the BNF described in subsection 3.2, which defines six-states FSMs.

Individuals with $Sf \geq 0.38$ were defined as the variants of the best individual and, therefore, will be penalized at local optima. *Sf* comprised only next-state genes (see Fig. 2). In fact, the heuristic did not consider the similarity factor with respect to output genes, since they have no influence in the state-machine graph. *GR* was evaluated over 50 generations and *GR* threshold for heuristic activation was set to 0.1. In case of heuristic activation, the system works with p_m = 0.15, p_c = 0.60 and p_r = 0.25. At generation corresponding to 95% of the maximum fitness value the heuristic was disabled since at this evolution stage *GR* is naturally close to zero. *Pf* was defined as 0.5 (fitness value penalized in 50%), remaining active throughout the next 50 generations.

Fig. 3 depicts the fitness curves of the best FSM of a typical run for three setups: **1: full heuristic**, **2: change genetic operators only** at local optima (no penalty applied) and **3: without heuristic**. As noted, until G = 266 the three curves are the same, as the heuristic isn't yet activated. For setup 1 (full heuristic), from G = 290 to G = 1028, as GR drops below 0.1 five times, the fitness value of the best FSM "backtracks" five times (at G = 268, G = 560, G = 760, G = 904 and G = 1028) and the system finally converges to global optimum (F_{MAX} = 576) at G = 1230. For setup 2, the system only converges to global optimum at G = 2944. For setup 3, the system did not escape from local optimum (F = 554) considering G_{MAX} = 5,000.

The resulting FSM using the full heuristic, which successfully describes the PS_{SND}, is given using state-transition graph (STG) in Fig. 4, after 576 output hits at G = 1230 (note: label v/v' represents an edge e_{ij} between two states q_i and q_j iff $o(q_i,v)=q_j$ and $t(q_i,v)=v'$). This FSM has one redundant state (state 4) and one unreachable state (state 1). Conventional methods may be used for state minimization. Table 1 compares the performance among the three setups. It shows that the full heuristic, which

penalizes the best individual and its variants, yields 38 global optimum convergences (using $Sf > 0.27$) among 50 runs, each evolving up to a maximum of 1,500 generations. This result improves setups 2 and 3 in 46% and 280%, respectively. This table also indicates that the heuristic is quite sensitive to *Sf*.

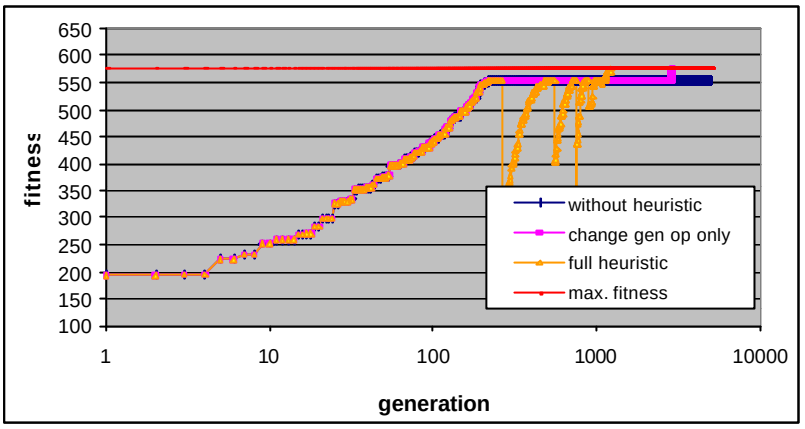
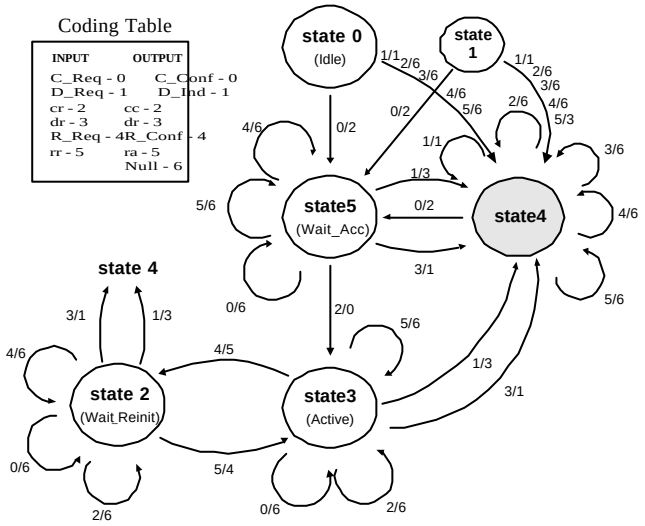


Fig. 3. Fitness curves of the best individual for three setups



Chromosome: 524146464646 524146464643 264326412634 364336412536 524146464646 564330415656

Fig. 4. PS_{SND}, using STG, of the fittest individual

Table 1. Comparison among three different setups

	Setup 3: Without	Setup 2: Change genetic	Setup 1: Full heuristic Penalty ($Pf = 0.5$) applied to:

	heuristic	operators only	$Sf > 0.16$	$Sf > 0.27$	$Sf > 0.38$	$Sf > 0.55$
Convergence to Global Optimum for $G_{MAX} = 1{,}500$ (50 runs)	10	26	32	38	32	27

5 CONCLUSION

A methodology for generating state machines from given input/output sequences was proposed. Application fields range from reverse engineering approaches, focusing on creating representations for already implemented systems, to forward engineering ones, by moving from high-level abstractions (e.g., specifications by means of user cases or traces) to physical implementations of the system.

The proposed approach has as distinctive feature a specific strategy for escaping from local optima. Preliminaries results show improvements of about 46% in the convergence to the global optimum, considering a maximum number of generations. However, the right most values for the penalty and similarity factors, as well as for the number of generations the penalty factor is maintained active, still has to be chosen upon experiments. Our main focus on the future is to quantify these parameters regarding to the problem size. In addition, improvements that may be done on the presented search algorithm contemplate a heuristic with memory cells to retain all previous penalized schemas, guiding new populations to explore a refined state space.

REFERENCES

[1] L. Fogel, *Autonomous Automata*, Industrial Research, 4:14-19, 1962.

[2] C. Manovit, C. Aporntewan and P. Chongstitvatana, *Synthesis of Synchronous Sequential Logic Circuits from Partial Input/Output Sequences*, ICES'98, pp. 98-105, 1998.

[3] R. Collins and D. Jefferson, *Representation for Artificial Organisms*, in Proc. of the 1st Int. Conf. on Simulation of Adaptive Behavior, MIT Press, 1991.

[4] P. Chongstitvatana and C. Aporntewan, *Improving Correctness of Finite-State Machine Synthesis from Multiple Partial Input/Output Sequences*, in Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware, pp. 262-266, 1999.

[5] G. Bockmann and A. Petrenko, *Protocol Testing: A Review of Methods and Relevance for Software Testing*, ISSTA'94, ACM, Seattle, U.S.A., pp. 109-124, 1994.

[6] J. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan, 1st Edition, 1975.

[7] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[8] H. Hörner, *A C++ Class Library for Genetic Programming*, Release 1.0 Operating Instructions, Viena University of Economy, 1996.

[9] P. Whigham, *Grammatically-Based Genetic Programming*, in Proc. of the Workshop on G.P.: From the Theory to Real-World Applications, pp. 33-41, Morgan Kaufman, 1995.

[10] W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. I, Wiley, pp. 224-225, 1968.

[11] W. Spears and D. Gordon, *Evolving FSM Strategies for Protecting Resources*, in Proceedings of the, 2000.

[12] W. Langdon, *Evolution of Genetic Programming Populations*, University College London Technical Report RN/96/125, 1996.

[13] R. Ursem, *Diversity-Guided Evolutionary Algorithms*. In Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002), p. 462-471, 2002.