Evolvable Computational Machines: Formal Approach

Lukáš Sekanina

Faculty of Information Technology at Brno University of Technology
Address: Božetěchova 2, 612 66 Brno, Czech Republic; sekanina@fit.vutbr.cz

Abstract. The paper introduces an original formal definition of the evolvable computational machine. Mathematical properties as well as impact to application design are investigated. The proposed approach is demonstrated on an evolvable non-uniform cellular automaton for generation of sequences.

1 Introduction

Evolutionary algorithms (EA) are traditionally used for optimization, but many works were devoted to evolutionary design [1] in recent years. The paper deals with evolutionary design of *computational machines* (CM). There are some typical examples of CM evolved successfully in past years: cellular automata [2], Turing machines [3], Boolean circuits [4], finite state machines [5], or artificial neural networks [6]. Also specialized types of EA suitable for a given computational model have been developed: cartesian genetic programming for digital circuits [7], cellular programming for cellular automata [2] or genetic programming for evolution of computer programs [8].

Theoretical computer science operates with formally defined objects. Various CM were formally defined in history of the field (there is large overview in [9]). Some formalisms have been also introduced in the field of EA [10,11]. The reasons for formal approaches are straightforward since: (i) problem specification is rigorous, (ii) a mathematical apparatus can be applied to investigate properties, and (iii) tools for automatic analysis, verification and design can be developed.

Taking in account all the reasons of the previous paragraph, we have tried to determine a minimal collection of mathematical objects to establish formal definition of an *evolvable computational machine* (ECM). The definition should allow us to investigate basic properties of ECM formally. Then we were interested to back-impact of such definition to the fields of theoretical computer science, evolutionary algorithms and practical implementations.

2 Formal definitions

2.1 Computational Machine: Non-Uniform Cellular Automaton

Every CM holds its formal definition and definition of its computation. As an example, the following definition (designed with inspiration in [2,9]) of a non-uniform cellular automaton is introduced.

Definition 1: One-dimensional binary **non-uniform cellular automaton** with the finite number of cells is 8-tuple $\mathsf{A} = (d, Q, N, R, z, b_1, b_2, c_0)$, where: $d = 1$ is a dimension, $Q = \{0, 1\}$ is a binary set of states, N denotes a neighbourhood, z denotes the number of cells, b_1 and b_2 are boundary values, c_0 is an initial configuration, and a mapping $R : C \rightarrow (Q^N \rightarrow Q)$ assigns to each cell in $C = \{1, 2, \ldots, z\}$ a *local transition function* $\delta_1, \ldots, \delta_z$, where $\delta_i : Q^N \rightarrow Q$. □

If only a single neighborhood N = {-1, 0, 1} is considered, then *global transition function* $G : Q^C \rightarrow Q^C$ is defined as:

$$G(c(i)) = \begin{cases} \delta_i(c(i-1), c(i), c(i+1)) & \text{for } i = 2 \ldots z-1, \\ \delta_1(b_1, c(1), c(2)) & \text{for } i = 1, \\ \delta_z(c(z-1), c(z), b_2) & \text{for } i = z, \end{cases}$$

where δ_i denotes the local transition function (rule) of the i-th cell defined by R. Cells are indexed from 1 to z.

G is used to define a sequence of configurations c_0, c_1, c_2, ... such that $c_j = G(c_{j-1})$, for $j \geq 1$. This sequence represents a *computation* of A.

2.2 General Evolutionary Algorithm

Definition 2: (see [10]) A **general evolutionary algorithm** is defined as an 8-tuple $\mathsf{E} = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda)$ where:
(i) I is the space of individuals (a set of chromosomes),
(ii) $\Phi : I \rightarrow \mathbb{R}$ denotes a fitness function assigning real values to individuals.
(iii) μ is the number of parent individuals, while λ denotes the number of offspring individuals.
(iv) $\Omega = \{\omega_{\Theta 1}, \ldots, \omega_{\Theta k} \mid \omega_{\Theta i} : I^\lambda \rightarrow I^\lambda\} \cup \{\omega_{\Theta 0} : I^\mu \rightarrow I^\lambda\}$ is a set of probabilistic genetic operators $\omega_{\Theta i}$ each of which is controlled by specific parameters summarized in the sets $\Theta_i \subset \mathbb{R}$.
(v) $s_{\Theta s} : (I^\lambda \cup I^{\lambda+\mu}) \rightarrow I^\mu$ denotes the selection operator, which may change the number of individuals from λ or $\lambda + \mu$ to μ, where $\mu, \lambda \in \mathbb{N}$ and $\mu = \lambda$ is permitted. An additional set Θ_s of parameters may be used by the selection operator.
(vi) $\Psi : I^\mu \rightarrow I^\mu$ is the generation transition function which describes the complete process of transforming a population P into subsequent one by applying genetic operators and selection:

$$\Psi = s \circ \omega_{\Theta_{i_1}} \circ \ldots \circ \omega_{\Theta_{i_j}} \circ \omega_{\Theta_0}$$
$$\Psi(P) = s_{\Theta_s}(Q \cup \omega_{\Theta_{i_1}}(\ldots(\omega_{\Theta_{i_j}}(\omega_{\Theta_0}(P)))\ldots))$$

(vii) The termination criterion is $\iota : I^\mu \rightarrow \{true, false\}$ [1]. □

[1] Here $\{i_1, \ldots, i_j\} \subseteq \{1, \ldots, k\}$, and $Q \in \{\emptyset, P\}$.

Definition 2 is based on high-level description where the population of individuals is manipulated by genetic operators and undergoes a fitness-based selection process. This is captured in the generation transition function Ψ, iterated application of which generates a *population sequence* and leads to definitions of the *running time* and the *result of EA* (see appropriate definitions in [10]). Once the definition of the general EA and its computation is available, we can immediately establish formal definitions of genetic algorithm, evolutionary strategy or evolutionary programming (as seen in [10]). Because the space of individuals I can be arbitrarily complex, e.g. genetic programming may be defined in terms of Definition 2, too.

Definition 2 deals with the space of individuals only. In many cases it is helpful to define search algorithms (i.e. also evolutionary algorithms) with respect to an abstract *representation* of the search space. Then the representation space defines a set of chromosomes which will be manipulated (by genetic operators) during search, and the the *growth function* defines a mapping between chromosomes and solutions [11].

3 Evolvable Computational Machines

To evolve a CM, we have to define which *part* of the CM will be under evolution because some parts of CM are usually required to be invariable (static). For instance, the number of inputs and outputs as well as required logical function of a combinational circuit is given by specification (i.e. is invariable), but its internal structure (i.e. a connection that implements the required function) has to be designed and so it can be the *subject of evolution*. Therefore, the subject of evolution has to be understood as knowledge available a priory.

We introduce two basic construction steps of formal definition of ECM. Let us consider non-uniform cellular automaton (CA) according to Definition 1 for demonstration of the concept:

(i) The subject of evolution (i.e. what is encoded in the chromosome) has to be chosen. Suppose that CA rules have to be evolved and the rest of CA is invariable. Then the growth function $g : I \to A'$ defines how to create a CA (i.e. a machine) from its genetic information (a chromosome). A' denotes a subset of all CA and determines which CA may be constructed using g.

(ii) A *machine fitness function* $f : A' \to \mathbb{R}$ has to be defined since behavior of a given CA must be evaluated in the process of fitness calculation. And because the fitness function is defined as $\Phi : I \to \mathbb{R}$, the fitness function is just the composition $f \circ g$.

The previous steps can be generalized for any CM. Let M denote a set of all CM. Then g will take the form $g : I \to M'$, where $M' \subset M$. And finally, general evolvable computational machine can be defined as follows:

Definition 3: An **evolvable computational machine** EM is a quadruple $\mathsf{EM} = (M', E, g, f)$, where M' denotes a set of computational machines, whose part is the subject of evolution, performed by an evolutionary algorithm E. The growth function $g : I \to M'$ assigns to each individual x ($x \in I$) in E a machine in M'. Function $f : M' \to \mathbb{R}$ is a machine fitness function. Fitness function Φ in the E is a composition $\Phi = f \circ g$. □

Computation of ECM is determined by definitions of computation of a given EA and CM.

Definition 3 deals with a *single* fitness function. However, EA working in time-dependent environments were also investigated [12]. In many applications of ECM (e.g. in the field of evolvable hardware [13]), the fitness function is changed dynamically to reflect environmental changes. To provide simple formal framework for the machine evolution in dynamic environments, the following definition, which is based on definition of a set of fitness functions (called environmental functions) and on a mechanism of transition between them is given. (The set of all mappings from M' to $\mathbb{R}$ will be denoted $\mathbb{R}^{M'}$).

Definition 4: Machine environment is a triplet $\Xi = (\Gamma, \varphi_0, \varepsilon)$ where $\Gamma \subseteq \mathbb{R}^{M'}$ denotes the set of *environmental functions* specifying quality of a machine in a given environment. $\varphi_0 \in \Gamma$ is an *initial environmental function*. $\varepsilon : \Gamma \to \Gamma$ denotes a relation on the set of environmental functions that determines successive environmental function. □

4 Consequences of the proposed approach

(i) Binding the concepts of the ECM and the machine environment together, one can formally describe really evolvable system with dynamically changing requirements for evolutionary design.

Definition 5: Evolvable machine based application is a pair (EM, Ξ) where EM denotes an evolvable machine and Ξ is a machine environment. An initial environmental function φ_0 in the Ξ is matched with the machine fitness function f, i.e. $f = \varphi_0$. □

(ii) From a practical viewpoint, Definition 5 implies the architecture of evolvable components [14]. The idea of the evolvable component is based on the following schema: If E, M' and g are designed in EM properly, then their definition (and mainly implementation) may be encapsulated and so reused in some *class* of applications. For instance, it was shown in [15] that various image filters can be successfully evolved at hardware level only by re-using a single component and redefinition of the fitness function.

(iii) The function g may be viewed as employing of a genotype-phenotype mapping. For example, in case of genetic algorithm and CA, a binary string represents a genotype while CA represents a phenotype. It is evident that different phenotypes may obtain the same fitness value. In general, the growth function has to be surjective. Moreover, if g is a bijective function then the representation is *faithful* and Lemma 1 holds:

Lemma 1: Φ is an equivalence relation on the set I, and its equivalence classes consist of genotypes whose fenotypes share the same fitness values. $\square$

In a very natural way, this result can be exploited for study of landscape neutrality [4] or in Surry's approach which demonstrates how to start from precise statements of beliefs about the relationship of problem structure to fitness and then mathematically derive representation along with appropriate operators in order to construct a problem-specific EA [11].

(iv) Assume that some Turing machine (TM) is evolved. So called halting problem of TM is undecidable [9]. It implies that an implementation of ECM must contains a mechanism to halt a candidate TM. It is also impossible to obtain a fitness value by an algorithm that analyses a candidate solution. Therefore, all candidate solutions must be executed to be evaluated.

(v) From a purely theoretical viewpoint, Definition 2 determines the fitness function as $\Phi : I \to \mathbb{R}$. But the requirement for $\mathbb{R}$ as an infinite uncountable set is strong since a set of all TM is infinite but countable [9].

(vi) The proof of the following Lemma 2 is evident:

Lemma 2: If an environmental function φ_i is used to define the machine fitness function f at least two times, then the graph of the relation ϵ contains a loop. $\square$

Open problems: (i) Does exist some principal limitation (like No Free Lunch theorems [11]) determining the number of environmental functions which evolution is efficient for in the context of a given ECM? (ii) Can be the relation ε in Definition 4 replaced by a function? Is it case of real world applications of ECM? (iii) Are all the ECM isomorph in some sense? If so, software tool could be developed to support (semi)automatic design of ECM. (iv) Is it possible to consider evolvable component as a part of system theory?

5 An example: Evolvable Non-Uniform Cellular Automaton as a Generator of Sequences

This section demonstrates proposed approach on the formal definition of an evolvable non-uniform CA with periodic boundary values (i.e. extreme cells are adjacent to each other), which is evolved to operate as a generator of the sequence: seq = 5-6-7-8-9-10-11-12-13-14-15-0-1-2-3-4. The CA is considered in the style of Definition 1. Simple genetic algorithm is derived from Definition 2 according to [10].

Evolvable non-uniform cellular automaton: $ECA = (A', GA, g, f)$
(i) **Cellular automaton** $\mathsf{A} = (d, Q, N, R, z, b_1, b_2, c_0)$
$A' = \{(d, Q, N, R, z, b_1, b_2, c_0) \mid d = 1, Q = \{0, 1\}, N = \{-1, 0, 1\},$
$z = 4, b_1 = c(4), b_2 = c(1), c_0 = 0101\}$
R is the subject of evolution and R is represented as an 8bit table for each cell. The cardinality of A' is 2^{32}.

(ii) **Genetic algorithm** $GA = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda)$
$I = \{0,1\}^{32}$ (i.e. an individual is a 32bit string – 8bits per cell)
$\Phi = f \circ g$
Ω contains two operators: mutation $m_{\{p_m=0.18\ per\ bit\}}$ and one-point crossover $r_{\{p_c=0.7\}}$ (see definition in [10])
s: 2-tournament selection with elitism (see definition in [10])
$\iota : t_{max} = 10^6$ (i.e. finish when the generation t_{max} is reached)
$\Psi = s \circ m \circ r$
$\lambda = \mu = 32$
(iii) **Growth function** $g : \{0,1\}^{32} \rightarrow A'$, $e(\boldsymbol{a}) = \mathsf{A} = (d = 1, Q = \{0,1\},$ $N = \{-1,0,1\}, R = \boldsymbol{a}, z = 4, b_1 = c(4), b_2 = c(1), c_0 = 0101)$ for $\boldsymbol{a} \in I$
(iv) **Machine fitness function** $f : A' \rightarrow \mathbb{R}$ and $f(e(\boldsymbol{a})) = \sum_{i=1}^{15} cmp(c_i, seq_i)$, where

$$cmp(c_i, seq_i) = \begin{cases} 1 & c_i = seq_i, \\ 0 & c_i \neq seq_i. \end{cases}$$

The parameters chosen in the example were set up after exhaustive experimental analysis (more than 25000 runs) where it was tested: population size (8-256), crossover probability (0-100%), mutation probability (1-10bits/chromosome) and selection mechanism (roulette wheel, 2-tournament, deterministic). Every run was repeated 100 times. The best solution (fitness = 10) appeared in 100 cases of 100 runs in generation 25505 in average. The evolution is very sensitive to change of parameters.

After an analysis of the whole state space of the problem (2^{32} states) using another program, it was recognized that the best fitness value 10 is shared by two solutions (CA rules $4E7A6633_h$ and $4A7A6633_h$). It means there is not any CA that is able to approximate more than 10 numbers of the sequence seq. Thus all CA can be divided into 11 equivalence classes for this task.

For instance, if the sequence is $seq0 = 0 \ldots 15$ then there are 9 equivalence classes or for the sequence $seq2 = 2, 3, \ldots$ there are 10 equivalence classes. However, new machine fitness functions have to be designed for these sequences. All these machine fitness functions can be captured in Γ and provide *machine environment* according to Definition 4. Then *ECA* tries to adapt to changing requirements on production of sequences that are specified using the relation ϵ.

6 Conclusions

We have established formal definitions of ECM and machine environment to investigate basic theoretical properties of ECM formally. Definition 3 shows that general ECM can be defined independently of a given CM, EA and application and thus all ECM operates exactly in the same way.

The ideas of reusability and evolvable component are important from implementation viewpoint. Many problems have been introduced and the se open problems will be the subject of future research.

Acknowledgment

The research was performed with the Grant Agency of the Czech Republic under No. 102/01/1531 *Formal approach in digital circuit diagnostic – testable design verification*. The author would like to thank Alexander Meduna for his fruitful comments.

References

1. Bentley, P. J. (1999) Evolutionary Design by Computers. Morgan Kaufmann Publisher
2. Sipper, M. (1997) Evolution of Parallel Cellular Machines: The Cellular Programming Approach. Springer-Verlag, Berlin
3. Vallejo, E., Ramos, F. (2001) Evolving Turing Machines for Biosequence Recognition and Analysis. In: Proc. of Genetic Programming European Conference EuroGP 2001, Springer-Verlag, 36–50
4. Miller J., Job, D., Vassilev, V. K. (2000) Principles in the evolutionary design of digital circuits – Part I and II. Journal of Genetic Programming and Evolvable Machines, Vol. 1(1–2,3), 8–35, 259–288
5. Sanchez, E., Pérez-Uribe, A., Mesot, B. (2001) Solving Partially Observable Problems by Evolution and Learning of Finite State Machines. In: Evolvable Systems: From Biology to Hardware ICES 2001, Springer-Verlag, 267–278
6. Yao, X. (1999) Evolving Artificial Neural Networks. Proceedings of IEEE, Vol. 87 (9), 1432–1447
7. Miller, J., Thomson, P. (2000) Cartesian Genetic Programming. In: Proc. of the Genetic Programming European Conference EuroGP 2000, LNCS 1802, Springer-Verlag, Berlin, 121–132
8. Koza, J. et al. (1999) Genetic Programming III : Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers
9. Gruska, J. (1997) Foundations of Computing. International Thomson Publishing Computer Press
10. Bäck, T. (1996) Evolutionary Algorithms in Theory and Practice. Oxford University Press
11. Surry, P. D. (1998) A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms. PhD thesis, University of Edinburgh
12. Branke, J. (2001) Evolutionary Approaches to Dynamic Optimization Problems – Update Survey. In Proc. of the GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, 27–30
13. Sanchez, E. et al. (1997) Phylogeny, Ontogeny, and Epigenesis: Three Sources of Biological Inspiration for Softening Hardware. In proc. of Evolvable Systems: From Biology to Hardware ICES96, Springer-Verlag, 35–54
14. Sekanina, L., Sllame, A. (2000) Toward Uniform Approach to Design of Evolvable Hardware Based Systems. In Proc. of Field Programmable Logic and Applications FPL2000, Springer-Verlag, 814–817
15. Sekanina, L. (2002) Image Filter Design with Evolvable Hardware. To appear in Proc. of the 4th European Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP2002, Springer-Verlag, p. 12