Minds & Machines
DOI 10.1007/s11023-007-9071-5

Evolved Computing Devices and the Implementation Problem

Lukáš Sekanina

Received: 25 November 2005 / Accepted: 13 August 2007

Abstract The evolutionary circuit design is an approach allowing engineers to realize computational devices. The evolved computational devices represent a distinctive class of devices that exhibits a specific combination of properties, not visible and studied in the scope of all computational devices up till now. Devices that belong to this class show the required behavior; however, in general, we do not understand how and why they perform the required computation. The reason is that the evolution can utilize, in addition to the "understandable composition of elementary components", material-dependent constructions and properties of environment (such as temperature, electromagnetic field etc.) and, furthermore, unknown physical behaviors to establish the required functionality. Therefore, nothing is known about the mapping between an abstract computational model and its physical implementation. The standard notion of computation and implementation developed in computer science as well as in cognitive science has become very problematic with the existence of evolved computational devices. According to the common understanding, the evolved devices cannot be classified as computing mechanisms.

Keywords Computing mechanism · Evolutionary design · Evolvable hardware · Implementation problem

Introduction

Our brains are physical devices, products of nature. Their behavior is often interpreted as computing (see a thorough review in Piccinini 2003). In comparison with the

L. Sekanina (✉)
Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic
e-mail: sekanina@fit.vutbr.cz
URL: www:http://www.fit.vutbr.cz/∼sekanina

ordinary computers, they differ in many features. One of them, especially important for this paper, is their origin. Our brains were evolved; their increasing complexity has emerged as a consequence of selection pressure (Dawkins 1991). They are constructed in the process of development, in the course of cellular divisions and cellular differentiations, starting from the zygote. In contrast, the ordinary computers are built according to abstract computational models introduced to satisfy the a priori given specification. The conventional process of implementation is well-developed and has achieved an amazing success. In the process of top-down approach, engineers are able to construct a chip consisting of 10^9 transistors; every of them perfectly controlled. However, it seems that the traditional top-down design methods do not allow engineers to create systems above a certain level of complexity. Furthermore, it is very difficult to achieve self-adaptation and self-recovery using conventional techniques. This contrasts very strongly with the mechanisms which have produced the extraordinary diversity and sophistication of living creatures.

Getting inspiration from nature, *evolutionary algorithms* have been utilized to design hardware in the recent decade (Higuchi et al. 1993; Sekanina 2004; Zebulum et al. 2002). *Evolvable hardware* is an emerging field that applies evolutionary algorithms to automate design and adaptation of physical, reconfigurable and morphable structures such as electronic systems, antennas, microelectromechanical systems and robots. Figure 1 demonstrates the difference between the traditional implementation scheme and the evolutionary design approach. In contrast to the conventional implementation, the evolutionary approach is based on the generate&test approach that modifies properties of the target physical system in order to obtain the required behavior. Candidate solutions are iteratively generated by the evolutionary algorithm and evaluated using the fitness function which assigns higher scores to the behaviors better satisfying the specification.

In particular, the most promising outcomes of the evolutionary approach are: (1) the artificial evolution can produce intrinsic designs for electronic circuits which lie

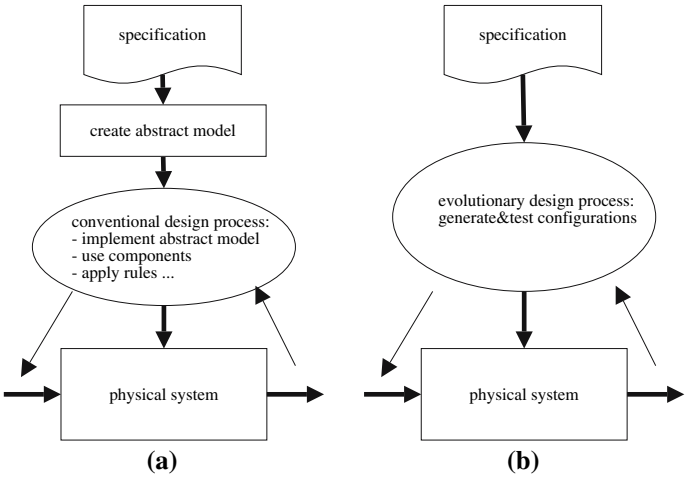

Fig. 1 Conventional design (**a**) vs evolutionary design (**b**)

outside the scope of conventional methods (Thompson et al. 1999) and (2) the challenge of conventional design is replaced by that of designing an evolutionary process that automatically performs the design in our place. This may be harder than doing the design directly, but makes autonomy possible (Stoica 2004).

The evolved computational devices represent a distinctive class of devices that exhibits a specific combination of properties, not visible in the scope of all computational devices up till now. The main properties of this class of devices are as follows:

1. We can specify behavior of these devices beforehand.
2. We can usually obtain their implementations.
3. In general, we are not able to understand why and how their internal implementation causes the computation.

Properties (1) and (2) mean that these devices are useful for practice. Property (3) is sometimes problematic from a practical point of view. It basically means that we are not able to see any relation between the internal physical processes and the required behavior which could be formulated via an abstract model. These devices are totally different from those devices and concepts studied in the context of theoretical computer science (Gruska 1997), computationalism (Piccinini 2003) and the so-called "implementation problem" (Copeland 1996; Piccinini 2003; Scheutz 1999). Similarly to other classes of computational devices (e.g., abstract computing machines or processors), we should address fundamental questions such as what computational power these devices possess, whether they can be universal and what is limiting for their complexity. However, because of the specific nature of the process utilized to create them, we have to ask a more basic question: Are the evolved devices the computers (computing mechanisms) in sense we usually understand the computers? The notions of 'computations' and 'implementation' are significant for both cognitive science and foundations of computing. We could ask: Could the analysis of evolved computing devices help for cognitive science?

This paper should contribute to the explanation of what it means for a computational system to be designed evolutionarily and what we can expect from the evolutionary approach that works on the position of human designer. In particular, we claim that the evolved computing devices are not necessarily computing mechanisms, although they can perform useful computations. The standard notion of computation and implementation developed in computer science as well as in cognitive science has become very problematic with the existence of evolved computational devices. The computational power of evolvable computing systems was analyzed in (Sekanina 2004); this analysis is not a part of this paper.

Evolutionary Design and Evolvable Hardware

Evolutionary Algorithms

Evolutionary algorithms (EAs) are stochastic search algorithms inspired by Darwin's theory of evolution. The *search space* is a space that contains all possible

considered solutions to the problem, and a point in that space defines a solution. Instead of working with one solution at a time (as random search, hill climbing and other search techniques do (Michalewicz and Fogel 2000)), these algorithms operate with the *population* of *candidate solutions* (individuals). Every new population is formed using genetically inspired operators (like *crossover* and *mutation*) and through a selection pressure, which guides the evolution towards better areas of the search space. The evolutionary algorithms receive this guidance by evaluating every candidate solution to define its *fitness value*. The fitness value, calculated by the *fitness function*, indicates how well the solution fulfills the problem objective (specification). A higher fitness value implies a greater chance that an individual will live for a longer while and generate offspring, which inherit parental genetic information. This leads to the production of novel genetic information and so to novel solutions to the problem. The fundamental structure of an evolutionary algorithm is captured in the following pseudocode:

Algorithm 1: Fundamental structure of an evolutionary algorithm

```
set time t = 0
create initial population P(t)
evaluate P(t)
WHILE (not termination condition) DO
  t = t + 1
  P(t) = create new population using P(t − 1)
  evaluate P(t)
END WHILE
```

Because the objects in the search space can be arbitrary structures (e.g., real-valued vectors, circuits, buildings, artefacts etc.), it is often helpful to distinguish between the *search space* (i.e., solution or phenotype space) and the *representation space* (i.e., chromosome or genotype space). The encoded solutions (*genotypes*) must be mapped onto actual solutions (*phenotypes*).

The fitness function is applied to evaluate phenotypes. While the fitness function operates with phenotypes, genetic operators are defined over genotypes. This concept of *genotype–phenotype* mapping has not been considered as crucial in the history of this field. It is probably due to the use of evolutionary algorithms as tools for optimization. However, recent studies (especially in the field of evolutionary design) assume this concept.

In order to show how EA can be used we can utilize Bentley's classification (Bentley 1999), which defines four main categories of applications of evolutionary algorithms: evolutionary design optimization, creative evolutionary design, evolutionary art and evolutionary artificial life forms.

Evolutionary design optimization. In general, an optimization problem requires finding a set of free parameters of the system under consideration such that a certain quality criterion (fitness function here) is maximized (or minimized). Designers usually begin the process with an existing design, and parameterize those parts of the design that they feel need improvement. The parameters are then encoded into chromosomes and an evolutionary algorithm tries to find the global or a sufficient

local optimum. The evolutionary optimization places great emphasis upon finding a solution as close to the global optimum as possible. The genotype–phenotype mapping is not practically important. Optimization of the coefficient values of a digital filter or component placement on the chip are typical examples from the electrical engineering industry.

Creative evolutionary design. Bentley specifies what "creativity" means in the computer context (Bentley 1999): "... the main feature that all creative evolutionary design systems have in common is the ability to generate entirely new designs starting from little or nothing and be guided purely by functional performance criteria. ... The emphasis is on the generation of novelty and originality, and not the production of globally optimal solutions. ... A computer is designing creatively when it explores the set of possible design state spaces in addition to exploring parameters within individual design spaces. Typically such systems are free to evolve any form capable of being represented." In terms of evolutionary algorithms, creative design is able to introduce new variables in the chromosome and so to define new search spaces. Among others, evolvable hardware falls into this category.

Evolutionary art. Evolutionary art is focused on creating new forms of images and art. The output from evolutionary art systems is usually attractive, but non-functional. A human evaluator determines the fitness, which is normally based on aesthetic appeal.

Evolutionary Artificial Life forms. This means the usage of evolutionary algorithms in the domain of Artificial Life.

Evolvable Hardware

The idea of evolutionary hardware design was introduced at the beginning of nineties in papers of Higuchi and de Garis (Higuchi et al. 1993; de Garis 1993). Evolvable hardware is usually defined as an approach in which the evolutionary algorithm is utilized to search for a suitable configuration of a reconfigurable device in order to achieve the behavior required by specification.

Reconfigurable Devices

The reconfigurable devices operate according to a configuration bitstream that is stored in the configuration memory. The cells of the configuration memory control a set of transistor switches. The switches define the way in which the programmable elements available on the device operate and the way in which are interconnected. By writing to this configuration memory, the user can physically create new electronic circuits. The advantage of the approach is that new hardware functionality is obtained through a simple reprogramming of the chip, similarly to reprogramming of the universal computer. These configuration bitstreams are generated by a conventional design tool in a way similar to software design using a compiler and builder. In evolvable hardware, these tools are replaced by evolutionary algorithm.

The idea of *reconfigurable hardware*, initially introduced in programmable digital devices such as Programmable Array Logic (PAL) and Field Programmable Gate Array (FPGA), has attracted attention in various other domains in the recent years. The first FPGAs were commercially introduced in middle eighties by Xilinx, Inc. Contemporary FPGAs contain thousands of programmable elements. Furthermore, various additional circuits are integrated on the FPGAs, including RAMs, efficient multipliers and interfaces. Nowadays, there are many different types of reconfigurable devices, including field programmable analog arrays (FPAA), field programmable transistor arrays (FPTA), reconfigurable antennas, reconfigurable nanodevices (e.g., NanoCell, Tour 2003), reconfigurable liquid crystals (Harding et al. 2005), reconfigurable microelectromechanical systems and reconfigurable optics (see survey in Sekanina 2004). We can see that the granularity of reconfigurable elements ranges from molecules to microprocessors.

Evolvable Hardware = Reconfigurable Device + Evolutionary Algorithm

The principle of evolvable hardware is shown in Fig. 2. The chromosome represents either the configuration bitsream directly or a prescription determining how to create the configuration bitstream. A particular variant of evolutionary algorithm and genetic operators are chosen according to the problem to be solved. The quality of the algorithm is evaluated experimentally. Candidate circuits are evaluated in the following way: First, a configuration bitstream is extracted from the chromosome. Then, the bitstream is uploaded into a reconfigurable device and the circuit created is evaluated using fitness function. In the case of digital circuits, some training vectors are applied at the primary inputs, corresponding output responses are collected and compared against the desired vectors. The fitness value is usually determined as the number of bits calculated correctly by the candidate circuit. In the case of analog circuits, training signals are applied at primary inputs for a given time period and circuit responses are compared with the desired signals. The objective here is to minimize the difference between the output signals and desired signals. Other approaches are possible, for example, the frequency characteristic is measured. The evolutionary algorithm (genetic operators etc.) is usually executed in a personal computer because its time requirements are negligible in comparison with the evaluation time of a candidate circuit. However, its implementation as an application-specific integrated circuit or in an FPGA is also possible, especially in case that the complete system should be realized on a single chip.

Intrinsic Evolvable Hardware

Extrinsic evolution means that candidate circuits are evaluated using a circuit simulator, i.e., in software. *Intrinsic evolution* means that every candidate circuit (i.e., circuit configuration) in every population is evaluated in a physical reconfigurable circuit. It has very important consequences.

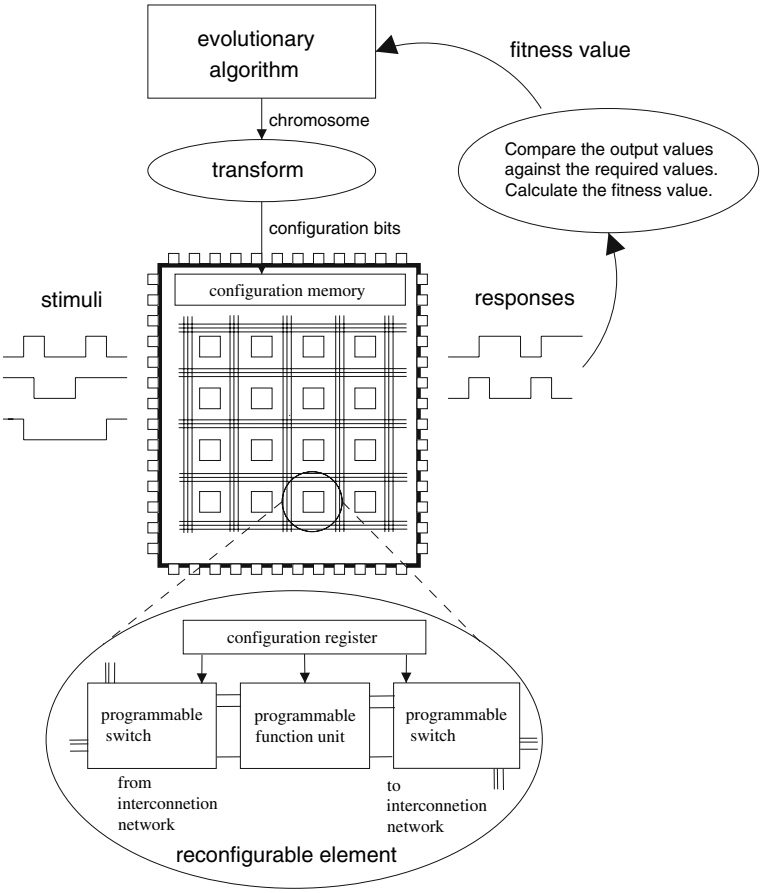

Fig. 2 Evolvable hardware: Candidate configurations are generated by the evolutionary algorithm, uploaded to a reconfigurable device and evaluated using fitness function

Let us suppose that we have evolved a circuit configuration C_1 intrinsically that shows a perfect fitness value x_1 in the reconfigurable circuit RC_1. Then the configuration C_1 is uploaded into RC_2 (which is exactly of the same type as RC_1), but contrary to our presumption (that we must obtain fitness value x_1 because it is the same chip) we get a different fitness value x_2, although the process of the fitness calculation remains formally unchanged! And finally, the same circuit (created according to configuration C_1) is evaluated using the software circuit simulator, and after this evaluation another fitness value x_3 is detected. How is it possible? Adrian Thompson has given the answer (Thompson 1998; Thompson et al. 1999). The evolved circuit (with configuration C_1) requires some special properties that are provided only by RC_1 and/or by some type of environment (e.g., by the environment with constant temperature) to operate perfectly (i.e., with fitness x_1). This approach, sometimes called *unconstrained evolution*, refers to such a situation in which the evolution can also exploit the physical properties of a chip and other environmental

characteristics (such as temperature, electromagnetic field etc.). The evolved circuits operate differently in different environments.

Recently, Miller has used the more general term, *evolvable matter*, to address the usage of evolutionary algorithms for the design on any physical reconfigurable platform (e.g., chemical or biological) in a real environment. The idea behind the concept is that applied voltages may induce physical changes that interact in unexpected ways with other distant voltage-inducted configurations in a rich physical substrate. In other words, it should theoretically be possible to perform the evolution directly *in materio* if the platform is configurable in some way. Miller and Downing reviewed this promising technology in Miller et al. (2002) and showed that liquid crystals can be used as a platform to evolve various circuits including primitive logic functions, robot controllers etc. (Harding and Miller 2005). Tour's group has presented another example—the nanocell (Tour 2003). The nanocell is a 2D network of self-assembled metallic particles connected by molecular switches. The nanocell is surrounded by a small number of lithographically defined access leads. The nanocell is not constructed as a specific logic gate—the logic is created in the nanocell by training it postfabrication by changing the states of the molecular switches. The training algorithm does not know the connections within the nanocell or the locations of the switches. However, the configuration can be changed by voltage pulses applied to the I/O pins. A genetic algorithm was utilized to generate these pulses in order to form the required logic gates.

Applications

It is important to distinguish between two approaches: evolutionary circuit design and evolvable hardware.

In case of the *evolutionary circuit design*, the objective is to evolve (i.e., to design) a single circuit. The circuit is usually designed in a design lab. The aim is typically to find an innovative implementation of a required behavior and, for instance, to reduce the number of utilized gates or to ensure testability of a circuit. The technique was successfully applied to design a number of unique digital as well as analog circuits. The evolved circuits exhibit properties that we have never reached by means of traditional engineering methods. It was demonstrated on small-scale-circuits that creative circuit designers can effectively be replaced by machines (Miller et al. 2000; Sekanina 2004; Zebulum et al. 2002).

In case of *evolvable hardware*, the evolutionary algorithm is responsible for continual adaptation. Evolvable hardware was applied to high-performance and adaptive systems in which the problem specification is unknown beforehand and can vary in time (Higuchi et al. 1999; Stoica 2004; Sekanina 2004). Its main objective is the development of a new generation of hardware, self-configurable and evolvable, environment-aware, which can adaptively reconfigure to achieve optimal signal processing, survive and recover from faults and degradation, and improve its performance over lifetime of operation. Evolutionary algorithm is an integral part of the target system. The use of evolvable hardware to achieve adaptation was mainly demonstrated by Higuchi's group which developed a number of ASIC-based

evolvable systems. Examples include: a prosthetic hand controller chip, image compression chip for electrographic printers, self-reconfigurable neural network chip, analog EHW chip for cellular phones and post-fabrication adjustment technology to maximize the system performance (Higuchi et al. 1999).

Due to the existence of redundancy in reconfigurable devices, the evolvable hardware is *inherently fault tolerant*. It means that in case that a circuit element is damaged, the evolutionary algorithm is usually able to recover the functionality using the remaining elements. If a critical number of elements is damaged, the functionality cannot be recovered and the chip "dies". Hence, evolvable hardware is a method for automatic designing of adaptive as well as fault-tolerant (e.g., self-repairing) systems (Thompson et al. 1999). The use of evolvable hardware to achieve functional recovery was mainly demonstrated by Stoica's group which developed an FPTA and performed evolutionary functional recovery of analog circuits in extreme environments, such as extreme temperatures and radiation (Keymeulen et al. 2000; Stoica et al. 2004; Stoica et al. 2004).

Computing and its Implementation

From Abstract Computing to Physical Computing

The classical computational theory has its roots in Turing's work on the Entscheidungsproblem (Turing 1937) (the decision problem for first order logic). Turing introduced a novel abstract computational device, called *automatic machine* (a-machine). We now know a-machines as Turing machines. Turing, Church and others were interested in effective ways of computing functions, where "effectiveness" was a mathematical notion synonymous with "mechanical" and lacking formal definition. Initially informal notion of algorithm was formalized using Turing machines. Later, it was shown that the computational power of Turing machines, recursive functions and λ-definable functions is equivalent. This result was taken as confirmation that the notion of effective function computation had finally been formally captured and the Church–Turing thesis was formulated as:

A function is effectively calculable if and only if it is Turing-computable.

Turing has offered his model for the study of the computations whose steps can be carried out by "a human being working mechanically with pencil and paper, unaided by machinery, and idealized to the extent of having available unlimited amounts of time, internal memory, and so forth" (Copeland and Sylvan 1999; Turing 1937). Similarly to other models, its fundamental property is being independent from the physical. However, the Turing machine was not (only) a mathematical representation of a computing human, but literally an idealized mechanical device. Furthermore, according to Piccinini, Turing thought his machine could compute any function computable by machines (Piccinini 2003).

The first who incorporated physically motivated mathematical constraints into a formal model of computation was Gandy (Gandy 1978) who attempted to capture in a precise mathematical framework the notion of computation by a *discrete*

deterministic mechanism. Gandy has shown that the machine-computable functions, even in this broader sense, are simply the Turing-computable functions, thus adding a striking bit of evidence for the adequacy and stability of Turing's analysis. Other researchers have analyzed the Church–Turing thesis with respect to the physical implementation and in terms of *computing mechanism*. A computing mechanism is a mechanism whose proper function is to obtain certain output strings of tokens from certain input strings (and internal states), according to a general rule that applies to all inputs and outputs. For purposes of this paper, we will mention two other variants of the Church–Turing thesis, Modest Physical and Bold Physical Church–Turing thesis, reflecting two different opinions on computational properties of physical computing systems. According to Piccinini (Piccinini 2003), Modest Physical Church–Turing thesis states that

> A function is computable by a mechanism if and only if it is Turing-computable.

On the other hand the Bold Physical Church–Turing thesis states:

> Any function whose values are generated by a physical system is Turing-computable.

Modest Physical Church–Turing thesis does not apply to all physical systems, but only to mechanisms that perform computations. It says that if a physical system computes, then it computes a Turing-computable function. Bold physical Church–Turing thesis applies to all physical systems. It is out of scope of this work to review all the views on the Church–Turing thesis and its variants; for a detailed analysis, see (Piccinini 2003).

There are many computational *models* provably more powerful than a standard Turing machine, for example, real-number computation, coupled Turing machines, accelerating Turing machines, various neural networks and site machine (Copeland 1998; Copeland and Sylvan 1999; van Leeuwen and Wiedermann 2001; Stannett 2003; Wegner and Goldin 2003). The research field dealing with computational models and systems more powerful than Turing machine is called *hypercomputation* or *super-Turing computation*. Eberbach et al. have summarized three principles that allow us to derive computational models more expressive than Turing machines: interaction with the world, infinity of resources or evolution of system (Eberbach et al. 2004).

It is easy to extend an abstract computational device to obtain a super-Turing computational device. However, the question is whether *physical* computational devices can exhibit super-Turing computational power. A well known machine of this type should be Hogarth machine that exploits the properties of a special kind of spacetime, called Malament–Hogarth spacetime, which is physically possible in the sense of constituting a solution to Einstein's field equations for General Relativity. From a computation point of view, the idea is to use the infinity of certain discrete processes in finite physical time (Wiedermann and van Leeuwen 2002). However, the validity of the model has not been experimentally verified.

There is another group of computational devices that is often counted among super-Turing computers. Those machines do not aspirate to calculate a priory

defined non-computable functions; rather, they perform computations that do not follow the computational scenario of Turing machines. Hence, Turing machines cannot simulate them. Typically, non-uniform, interactive and infinite computations play a crucial role in these machines. Examples include Internet (van Leeuwen and Wiedermann 2001), interactive agents (Eberbach et al. 2004) and evolvable computational systems (Sekanina 2004).

Van Leeuwen and Wiedermann have shown that such computations may be realized by an *interactive Turing machines with advice*. They have proposed the following extension of the Church–Turing thesis (van Leeuwen and Wiedrmann 2001):

> Any (non-uniform interactive) computation can be described in terms of interactive Turing machines with advice.

Interactive Turing machine with advice is a classical Turing machine endowed with three important features: the advice function (which is a form of oracle), interaction and infinity of operation. The advice function is to change the machine's control unit when it is necessary. Van Leeuwen and Wiedermann discuss their thesis in (van Leeuwen and Wiedrmann 2001), in particular, they ask whether: "... we are able to solve some concrete, a priori given undecidable problems with them? The answer is negative. What we have shown is that some computational evolutionary processes, which by their very definition are of an interactive and unpredictable nature, can be modeled a posteriori by interactive Turing machines with advice. Yet none of these systems is seen as violating the Church–Turing thesis. This is because none of them fits the concept of a *finitely describable algorithm* that can be mechanically applied to data of arbitrary and potentially unbounded size."

The standard classical computation theory (including complexity and computability analysis) is philosophically and terminologically closely bounded to the theory of Turing machines. The recent development in the areas of artificial intelligence, molecular biology, computer engineering and some others together with considering concepts such us interaction, evolution over time, infinite computation and embodiment has led to a different understanding and explanation of computation (Maclennan 2003).

The Implementation Problem

The Church–Turing thesis explains what it means to compute from a formal-mathematical point of view. We can ask: What exactly is the relationship between the formal structure and the physical system? What is for a physical system to be a computing mechanism? The process of building a physical computational device according to a formal model is called *implementation*. This section deals with *the implementation problem*. This problem is closely related to another fundamental issue: What is a computing mechanism and, more generally, computation? Copeland (1996), Scheutz (1999), Piccinini (2003), Johnson (2004) and others have surveyed previous works in this area and provided different answers. In particular, they usually begin with Putnam's view in which a system is a computing

mechanism if and only if there is a mapping between a computational description and a physical description of the system.

The approach developed by Scheutz (1999) differs significantly from the Putnam's "state-to-state correspondence view": it does not require a notion of physical state, but determines directly the function which is realized by a physical system. It does not have to assume a particular computational formalism (to which the physical system exhibits a state-to-state correspondence), but can be related to computations described by any computational formalism via the function that these computations give rise.

Piccicini claims that an optimal account of computing mechanisms should satisfy the six desiderata proposed in (Piccinini 2003) ((1) Paradigmatic computing mechanisms compute. (2) Paradigmatic non-computing systems don't compute. (3) Computation is observer-independent. (4) Computations can go wrong. (5) Some computing mechanisms are not computers. (6) Program execution is explanatory.) and that his account of computing mechanisms "... allows us to formulate the question of whether a mechanism computes as an empirical hypothesis, to be decided by looking at the functional organization of the mechanism. It allows us to formulate a clear and useful taxonomy of computing mechanisms and compare their computing power".

For Copeland, to compute is to execute an algorithm (Copeland 1996). More precisely, to say that a device or organ computes is to say that there exists a modeling relationship of a certain kind between it and a formal specification of an algorithm and supporting architecture. Then (physical) entity e is computing function f if and only if there exist a labeling scheme L and a formal specification SPEC (of an architecture and an algorithm specific to the architecture that takes arguments of f as inputs and delivers values of f as outputs) such that $\langle e, L \rangle$ is an "honest" model of SPEC. To describe a physical entity as a computing machine of a certain kind is to envisage being able to predict aspects of its physical behavior on the basis of its architecture-algorithmic specification and its labeling scheme. He suggested two necessary conditions for honesty: (1) the labeling scheme must not be *ex post facto* (from a thing done afterward) and (2) the interpretation associated with the model must secure the truth of appropriate counterfactuals concerning the machine's behaviors.

According to Johnson's view, formulated from the position of natural computing, an important characteristic of computing is "... that symbols within the system have a consistent interpretation throughout the computation, or at least if they do not there is a component of the system which explains how the interpretation of the symbols changes as the computation progresses. That is, any external system which observes and/or initiates a computation must declare in advance how it is going to interpret those symbols. This seems to be a key characteristic of computing which can be applied to natural systems. If there is not a consistent allocation of symbols then transformations are meaningless. In particular, if we are completely free to assign any symbol to any meaning at any point in the computation then we can say that any transformation is doing any computing" (Johnson 2004).

Searle (1990) recalls that on the standard textbook definition, computation is defined syntactically in terms of symbol manipulation. He argues that "... syntax

and symbols are not defined in terms of physics. Syntax, in short, is not intrinsic to physics. This has the consequence that computation is not discovered in the physics, it is assigned to it. Certain physical phenomena are assigned or used or programmed or interpreted syntactically. Syntax and symbols are observer relative." On the question "Is the brain a digital computer?" he answers that this question is ill defined and does not have a clear sense because "... you could not discover that the brain or anything else was intrinsically a digital computer, although you could assign a computational interpretation to it as you could to anything else." In case of ordinary computers it is reasonable to interpret the physics in both syntactical and semantic terms.

We have learned from the previous, surprisingly, that such the fundamental terms as computing and implementation of computing do not have a uniform interpretation.

Evolved Computing Devices

Basic Characteristics

The traditional process of implementation of a computing device, which is supposed to work according to a given specification, can be understood as a process of manipulating/fabricating a suitable physical system whose behavior can easily be interpreted in the required way. In this implementation process, the specification is transformed onto implementation in well-defined steps using well-defined methods, tools and components. The engineer usually knows the abstract model of the target system at any desired level. The engineer is primarily guided by knowledge of methodology established for solving the given type of problem. For that process, it is crucial that the engineer understands the principles that the resulting system is based on. The design process is completely under engineer's control. There exists a well-established consensus (e.g., voltage levels for logic '0' and '1') allowing us to interpret the input/output behavior of the system. Furthermore, it is usually required to have a sufficient interpretation of internal physical processes of that system in terms of computation ("two pn-junctions form a transistor and two transistors make up an inverter").

Similarly to the conventional design process, we have to declare our required interpretation of input/output behavior in advance when the evolutionary design method is used to find a physical implementation of a computational process. This interpretation is used to define the fitness function. When not constrained, the evolution can utilize all the resources available, including normally unused characteristics of the reconfigurable platform and environment. Similarly to biological evolution, artificial evolution promotes those candidates ("organisms") that are better adapted to the given environment. Although the evolution is often able to find an implementation perfectly satisfying the specification, we have problems to understand how and why the solution works. In other words, we are not able to interpret the internal behavior as a computational process which produces the required responses.

Table 1 Comparison of different types of computational devices

Property	Processor	The brain	Evolved device
I/O behavior can be interpreted as computing	Yes	Yes	Yes
Device is a computing mechanism	Yes	Unsure	Unsure
An abstract model exists before implementation	Yes	No	No
The required behavior is specified beforehand	Yes	No	Yes
Engineers can design&build	Yes	No	Yes

We can assemble Table 1 that compares ordinary computers, evolved computational systems and the brain (the brain represents here all biological systems that are often studied as computational devices). This table shows that the evolved computational devices represent a distinctive class of devices that exhibits a specific combination of properties, not visible in the scope of all computational devices up till now.

The Embodiment Phenomenon

We have explained in Sect. 'Intrinsic Evolvable Hardware' that the evolutionary approach is able to find a solution outside the space of conventional designs. Every physical object which can be designed by means of the evolutionary algorithm is of finite size. Assume that we have a reconfigurable digital circuit consisting of K programmable elements whose function is defined by the configuration bitstream stored in the B-bit configuration memory. There are 2^B different configurations, i.e., up to 2^B behaviors that can be realized in the reconfigurable device. However, the number of *distinguishable* behaviors is usually much smaller. Although every two different configurations determine two physically different electronic circuits, the two configurations can produce the same (digital) behavior because of inherent redundancy of reconfigurable circuits (Fig. 3a).

It was shown that under some conditions, some configurations can exhibit *useful* digital behavior that *none* of the 2^B configurations can do under normal conditions (see Thompson's experiment (Thompson et al. 1999)). Note that the fitness function has remained formally unchanged. The evolutionary analog circuit design or

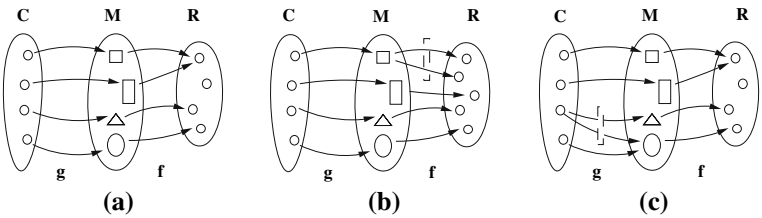

Fig. 3 Properties of g and f when implemented physically: (**a**) a physical implementation corresponds to an abstract model, (**b**) the fitness function is influenced by environment, (**c**) the genotype-phenotype mapping is influenced by environment

evolutionary robotics show that it is impossible to build a simulator of a given physical entity in order to completely avoid doing experiments with physical devices. In many cases, simulators predict different values than physical devices generate simply because computer models cannot cover everything about the reality. The physical embodiment is a crucial feature of these systems (see also Brooks 2001).

We can explain this "embodiment phenomenon" by means of mappings g and f shown in Fig. 3. Mapping g represents the transformation from the genotype space C to phenotypes M (i.e., from configuration bitstreams to physical circuits, i.e., computational machines). Mapping f is the fitness function assigning fitness values (e.g., real numbers, R) to phenotypes. We are offering two explanations of the "embodiment phenomenon":

In the first viewpoint, f is important. In fact, f assigns two different fitness values to a single circuit (machine) $m \in M$ (see Fig. 3b). Note again that the fitness function has remained formally unchanged. In reality, m is represented by an electronic circuit (configuration) whose behavior depends on something missing in the fitness calculation. In this case, f can not be seen as a function, rather f is a *relation*.

In the second viewpoint, g is important. Fitness function has never assigned different fitness values to a single machine (assuming invariable fitness function). All phenotypes are seen as different (but they can obtain the same fitness value). However, a single genotype can generate two or more different phenotypes (circuits), because it is assumed that environment influences the genotype-phenotype mapping. The situation is known from nature where phenotypes depend on the environment where they are constructed (developed). For instance, the development depends on gradient concentrations nearby a dividing cell (Alberts et al. 1998).

Evolution is able to effectively exploit physical properties of a given *materio* in order to build a target system, as demonstrated by Thompson (Thompson et al. 1999), Harding (Harding and Miller 2005), Tour (Tour 2003) and others. Probably the best example is the brain evolved by nature. However, there are physical limits that define what can ultimately be done by an arbitrary physical system when it is considered as a computational device, independently of the method utilized to build that system (Lloyd 2000). As the conventional implementation approach is constrained by practical limits that are determined by contemporary technological development, there are, in principle, no limits of this kind in case of the evolutionary approach. The main problem of the current (artificial) evolutionary design approach is that it is not able to produce complex and simultaneously innovative designs because the fitness calculation is not usually scalable.

The Implementation Problem and Evolutionary Design

Various versions of the Church–Turing thesis are often used to analyze computational capabilities of physical and biological systems (such as the brain). The Church–Turing thesis does entail that if the brain follows an effective procedure, then that procedure is Turing–computable. The Modest Physical Church–Turing

thesis does entail that if the brain performs computations then those computations are Turing-computable. But neither the Church–Turing thesis nor Modest Physical Church–Turing thesis do say whether the brain follows effective procedure or performs computations (Piccinini 2003). In the brains we can observe elementary cells and higher-level units composed of many cells; however, we are not able to exactly determine which properties are responsible for the behavior that we can interpret as computation. Also in the case of artificially evolved computational devices we can observe elementary components but, in general, we do not know which properties of these components the evolution has used to ensure the behavior which can be interpreted as computation. As the brains and computational devices are very similar in this point neither the Church–Turing thesis nor Modest Physical Church–Turing thesis does say whether the evolved computational devices follow effective procedure, i.e., whether they perform computations.

A given computing can be implemented in different ways in different physical systems (this property is called *multiple realizability*). Searle pointed out that "the multiple realizability of computationally equivalent processes in different physical media is not just a sign that the processes are abstract, but that they are not intrinsic to the system at all. They depend on an interpretation from outside ... computation is not discovered in the physics, it is assigned to the physics" (Searle 1990). We asked whether the evolved physical entities that perform some useful computations are computers. The previous argument clearly shows that the question does not have a clear sense. The computation is not discovered in the physics; however, we (humans) are able to interpret this behavior as computation.

Assume that a simple sequential circuit has been evolved in an FPTA or liquid crystals (see, for example Sekanina and Zebulum 2005). Moreover, assume that the environment is stable (i.e., temperature, radiation and all the phenomena that could influence its behavior are stable). The circuit has digital inputs and outputs. In principle, we are able to interpret the signals at these ports as "zeroes" and "ones" because we had to declare our interpretation in the fitness evaluation process (i.e., in advance, before the circuit was evolved). However, we don't exactly understand how the platform performs the computation. We have specified only the required input/output relation in the fitness function, i.e., we have not introduced any abstract model indicating that some internal states must exist in order to implement that behavior (in general, the behavior is not combinational). It is the role of evolution to invent, introduce and implement the internal states in some way. Therefore, we are not able to say anything about the relation between the abstract model (because it does not exist) and the resulting implementation created using the evolutionary algorithm.

Most scientists cited in Sect. 'The Implementation Problem' would probably agree that the evolved systems really perform computation because we can interpret their input/output behavior as computation. However, there are problems with the interpretation of their internal behavior. Copeland, Johnson and others would like to see that symbols within the system have a consistent interpretation throughout the computation and that interpretation is specified beforehand. It makes no sense to establish the mapping between computational states and physical states before the evolution is performed. The evolution can use totally different physical properties of

the physical device to implement the required behavior. On the other hand, establishing the mapping at the end of evolution could solve the problem because (under some assumptions, such as that the environment does not influence the physical device at all) it could be possible to identify physical states that correspond to abstract computational states. Here, it is reasonable to follow Scheutz approach ("starting in physical", Scheutz 1999). Unfortunately, the discovery of computational states/functions within the physical system is not quite sure. Although we can apply all the physical theories we currently know to analyze and explain the evolved system, we could obtain no satisfactory answer. As Bartles et al. have shown (Bartels et al. 2004), evolution is able to utilize those physical behaviors that are physically impossible from the point of view of current physical theories.[1]

Finally, we have to consider again that no abstract model is available because our specification is defined in terms of input/output relations. Hence we are able neither to establish the relation between physical and abstract states nor to "distill a (matter-independent) mathematical mapping—the function realized by evolved system *S*—between inputs and outputs of *S* by abstracting over every physical dimension" (Scheutz 1999).

If we agree that (1) it does not matter how a system realizes a function as long as the system is digital from the outside (a digital system can be treated as a "black box" with digital inputs and outputs), (2) we are able to interpret its input/output behavior as computation and (3) this interpretation is defined in advance then *the evolved systems are computational systems* (computing mechanisms). On the other hand, if the correspondence between the abstract states and physical states is crucial to qualify a system as computational one then *the evolved systems are not computing mechanisms*.

Conclusions

Similarly to nature, we can build physical systems whose behavior can be interpreted as computing. Furthermore, we can specify this behavior in advance. Evolutionary circuit design and evolvable hardware traditionally belong to the area of electrical engineering. In this paper, we have interpreted the evolutionary design of computational systems from the perspective of computer science. In this view, the evolutionary circuit design is an approach allowing engineers to realize computational devices. The evolved computational devices represent a distinctive class of devices that exhibits a specific combination of properties, not visible and studied in the scope of all computational devices up till now. Devices that belong to

[1] Bartels used an evolutionary optimization algorithm to shape laser pulses to distort molecules in specific ways to catalyze chemical reactions, with the ultimate goal of manipulating large molecules, for example proteins and enzymes, in a biological cell. The method has also been used to create new quantum behaviors at the atomic level. Human can probe quantum systems, but are not capable of exploring different quantum behaviors in a fast automated fashion. The results are completely unexpected and amazing from a physical point of view: behaviors are being evolved that were not known to be physically possible. This includes anti-correlated attosecond harmonics in quantum systems. The ability to move beyond the nanoscale to the attoscale is a major breakthrough, and the potential applications of controlling the behavior of materials at atomic level are enormous (Bartels et al. 2004).

this class show the required behavior; however, in general, we do not understand how and why they perform the required computation. The reason is that the evolution can utilize material-dependent constructions and properties of environment (such as temperature, electromagnetic field etc.) and, furthermore, unknown physical behaviors to establish the required functionality. Therefore, nothing is known about mapping between an abstract model and its physical implementation and the evolved computing devices are not necessarily computing mechanisms.

Acknowledgements This research was partially supported by the Grant Agency of the Czech Republic under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* and the Research Plan No. MSM 0021630528 *Security-Oriented Research in Information Technology*.

References

Alberts, B., et al. (1998). *Essential cell biology – An introduction to the molecular biology of the cell*. New York: Garland Publishing.

Bartels, R., et al. (2004). Learning from learning algorithms: Applications to attosecond dynamics of high-harmonic generation. *Physical Review A, 70*(1), 1–5.

Bentley, P. (Ed.) (1999). *Evolutionary design by computers*. San Francisco CA: Morgan Kaufmann Publishers.

Brooks, R. (2001). The relationship between matter and life. *Nature, 409*(6818), 409–411.

Copeland, B. J. (1996). What is computation? *Synthese, 108*, 335–359.

Copeland, B. J. (1998). Super-Turing machines. *Complexity, 4*(1), 30–32.

Copeland, B. J., & Sylvan, R. (1999). Beyond the universal Turing machine. *Australasian Journal of Philosophy, 77*(1), 46–66.

Dawkins, R. (1991). *The blind watchmaker*. London: Penguin Books.

Eberbach, E., Goldin, D., & Wegner, P. (2004). Turing's ideas and models of computation. In Ch. Teuscher (Eds.), *Alang Turing: Life and Legacy of a Great Thinker* (pp. 166–173). Berlin Heidelberg New York: Springer.

Gandy, R. (1978). Church's thesis and principles for mechanisms. In K. J. Barwise, H. J. Keisler, & K. Kunen (Eds.), *The Kleene symposium* (pp. 123–148). New York: North Holland.

de Garis, H. (1993). Evolvable hardware – Genetic programming of a Darwin machine. In ICANNGA'93: *International Conference on Artificial Neural Networks and Genetic Algorithms*, Innsbruck, Austria. Berlin Heidelberg New York: Springer.

Gruska, J. (1997). Foundations of Computing. International Thomson Publishing Computer Press.

Harding, S., & Miller, J. (2005). Evolution in materio: Evolving logic gates in liquid crystal. In Proceedings of the workshop on unconventional computing at ECAL 2005 VIIIth European conference on artificial life. To appear in International Journal of Unconventional Computing, pp 12.

Higuchi, T., et al. (1993). Evolving hardware with genetic learning: A first step towards building a Darwin machine. In SAB'92: *Proceedings of the 2nd International Conference on Simulated Adaptive Behaviour* (pp. 417–424). Cambridge MA: MIT Press.

Higuchi, T., et al. (1999). Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation, 3*(3), 220–235.

Johnson, C. G. (2004). What kinds of natural processes can be regarded as computations? In T. Paton (Eds.), *Computation in cells and tissues: Perspectives and tools of thought*. Berlin Heidelberg New York: Springer.

Keymeulen, D., Stoica, A., & Zebulum, R. (2000). Fault-tolerant evolvable hardware using field programmable transistor arrays. *IEEE Transactions on Reliability. Special Issue on Fault-Tolerant VLSI Systems, 49*(3), 305–316.

van Leeuwen, J., & Wiedermann, J. (2001). The turing machine paradigm in contemporary computing. In *Mathematics unlimited – 2001 and beyond* (pp. 1139–1155). Berlin Heidelberg New York: Springer.

Lloyd, S. (2000). Ultimate physical limits to computation. *Nature, 406*, 1047–1054.

Maclennan, B. J. (2003). Transcending turing computability. *Minds and Machines, 13*(1), 3–22.

Michalewicz, Z., & Fogel, D. B. (2000). *How to Solve It – modern heuristics*. Berlin Heidelberg New York: Springer.

Miller, J., Job, D., & Vassilev, V. (2000). Principles in the evolutionary design of digital circuits – Part I. *Genetic Programming and Evolvable Machines, 1*(1), 8–35.

Miller, J., & Downing, K. (2002). Evolution in Materio: Looking beyond the silicon box. In Stoica, A. et al. (Ed.), EH'02: *Proceedings of the 4th NASA/DoD conference on evolvable hardware*. Alexandria, Virginia, USA, 2002 (IEEE Computer Society, Los Alamitos 2002) pp 167–176.

Piccinini, G. (2003). Computations and Computers in the Sciences of Mind and Brain. PhD thesis, University of Pittsburgh, p 323.

Scheutz, M. (1999). When physical systems realize functions. *Minds and Machines, 9*(2), 161–196.

Searle, J. (1990). Is the brain a digital computer? *Proceedings and Addresses of the American Philosophical Association, 64*, 21–37.

Sekanina, L. (2004). *Evolvable Components: From Theory to Hardware Implementations. Natural Computing Series*. Berlin Heidelberg New York: Springer.

Sekanina, L. (2004). Evolvable computing by means of evolvable components. *Natural Computing, 3*(3), 323–355.

Sekanina, L., & Zebulum, R. (2005). Evolutionary discovering of the concept of the discrete state at the transistor level. In EH'05: *Proc. of the 2005 NASA/DoD Workshop on Evolvable Hardware*, Lohn, J. et al. (Eds.), Washington DC, USA, 2005. IEEE Computer Society, Los Alamitos, pp. 73–78.

Stannett, M. (2003). Computation and hypercomputation. *Minds and Machines, 13*(1), 115–153.

Stoica, A., Keymeulen, D., Arslan, T., Duong, V., Zebulum, R., Ferguson I., & Guo, X. (2004). Circuit Self-Recovery Experiments in Extreme Environments. In Zebulum, R. et al. (Eds.), EH'04: *Proc. of the 2004 NASA/DoD Workshop on Evolvable Hardware* (pp. 142–145). Seattle USA, 2004. IEEE Computer Society, Los Alamitos.

Stoica, A., Zebulum, R., Keymeulen, D., Ferguson, I., Duong, V., & Guo, X. (2004). Evolvable hardware techniques for on-chip automated reconfiguration of programmable devices. *Soft Computing – Spec. Issue on Evolvable Hardware, 8*(5), 354–365.

Stoica, A. (2004). Evolvable Hardware for Autonomous Systems. Tutorial at IEEE Congress on Evolutionary Computation. http://ehw.jpl.nasa.gov/Content/Public/TutorialCEC2004/TutorialCEC2004.pdf

Thompson, A. (1998). *Hardware evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution*. Distinguished Dissertation Series. Springer, London.

Thompson, A., Layzell, P., & Zebulum, R. S. (1999). Explorations in design space: unconventional electronics design through artificial evolution. *IEEE Trans. on Evolutionary Computation, 3*(3), 167–196.

Tour, J. M. (2003). *Molecular electronics*. World Scientific.

Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, 42(2) (1936–37), 230–265 (with corrections from Proceedings of the London Mathematical Society, Series 2, 43 (1937), 544–546).

Wiedermann, J., & van Leeuwen, J. (2002) Relativistic Computers and Non-uniform Complexity Theory. In Calude C. S. (Ed.), UMC'02: *Proceedings of 3rd Conference on Unconventional Models of Computation*. Kobe, Japan, 2002. Lecture Notes in Computer Science, vol 2509. Springer, Berlin Heidelberg New York, pp. 287–299.

Wegner, P., & Goldin, D. (2003). Computation beyond Turing machines. *Communications of the ACM, 46*(4), 100–102.

Zebulum, R., Pacheco, M., & Vellasco, M. (2002). *Evolutionary electronics – Automatic design of electronic circuits and systems by genetic algorithms*. Boca Raton: CRC Press.