Providing Information from the Environment for Growing Electronic Circuits Through Polymorphic Gates

Michal Bidlo
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
bidlom@fit.vutbr.cz

Lukas Sekanina
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz

ABSTRACT

This paper deals with the evolutionary design of programs (constructors) that are able to create $(n+2)$-input circuits from n-input circuits. The growing circuits are composed of polymorphic gates considered as building blocks. Therefore, the growing circuit can specialize its functionality according to environment which is sensed through polymorphic gates. The work was performed using a simple circuit simulator. We evolved constructors that are able to create arbitrarily large polymorphic even/odd parity circuits and polymorphic sorting networks.

Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design Styles—*Combinational logic*; B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; I.2.m [**Artificial Intelligence**]: Miscellaneous

General Terms

Experimentation, Design

Keywords

Genetic algorithm, development, digital circuits design, polymorphic circuit

1. INTRODUCTION

During *development*, a multicellular organism is formed from the zygote in a process of cellular division and cellular differentiation. Pattern formation depends on positional information, which instructs competent cells to differentiate in ways characteristic of their position in the structure. Positional information is not only provided by the environmental trigger but also is usually understood to be imparted by the concentration of one or more morphogens emitted from spatially distinct organizers [1].

GECCO'05, June 25–29, 2005, Washington, DC, USA.

The concept of development was adopted by *evolutionary algorithms* community in order to realize non-trivial genotype-phenotype mappings which are necessary to overcome the scalability problem of evolutionary design. Various approa-ches have been investigated (see survey in [9]). In the context of evolutionary algorithms, computational development might be utilized to achieve diverse objectives, including: adaptation, compacting genotypes, reduction of search space, allowing more complex solutions in solution space, regulation, regeneration, repetition, robustness, scalability, evolvability, parallel construction, emergent behavior and decentralized control [9].

The utilization of the developmental process in electronic hardware is a challenging task. The approach is extremely interesting for hardware design because it could provide a solution via the concept of self-organization to the design of electronic circuits of a complexity beyond current layout techniques and introduce into electronics some fault-tolerance and self-repair mechanisms. The following list only summarizes some of the most interesting results presented in this area: Embryonics [11], POEtic [20], CAMBrain [4] and CellMatrix [10] projects deal with new reconfigurable platforms inspired at the level of embryology. Haddow et al. have adopted L-system in order to evolve scalable digital circuits [5]. John Koza introduced an original method in which novel analog circuits have been constructed according to the instructions produced by genetic programming [7]. In another approach, Gordon and Bentley have utilized the interaction of artificial genes and proteins to model development in digital circuits [3]. Miller and Thomson have invented a developmental method for growing graphs and circuits using Cartesian genetic programming [13].

The concept of protein gradients has been examined in detail for electronic circuits in many publications, e.g. [20, 12]. Essentially, gradient-based systems consist of a set of diffusers that release a given protein into the system. Functionality of a cell depends on proteins concentration, i.e. on the distance from the proteins diffusers. As the environment (including proteins) influences development (i.e. provides the positional information), it is possible to control development by means of proper proteins concentrations.

The main feature of all the electronic developmental systems utilizing some information from the environment (e.g. positional information) is that this information is considered as one of many *digital* inputs to the system, typically obtained through a couple of wires.

In this paper we show that there are other options for pro-

viding information for growing digital circuits. For instance, positional information will not be provided in the digital form; rather it will influence the system in an analogue (and perhaps non-electrical) form, for example, as temperature, light, radiation, specific voltage etc. The motivation for this approach is that every electronic circuit operates in a real physical environment and "good" physical environment is crucial for correct behavior of the resulting system (similarly, cells survive and divide only in a "good" environment). The proposed approach should allow the developing system to interact with the environment much closely and to differentiate according to specific real-world situations. For example, one can imagine that a growing circuit could generate some heat, which, in turn, could automatically be reflected by the circuit to regulate its developmental process and so its behavior.

Of course, we do not want to build special sensors. The growing circuits should inherently interact with the physical environment. Technology, which, in principle, allows engineers to build such systems, is called *polymorphic electronics* [17, 18, 19]. It was shown that it is possible to create digital gates whose functionally can be controlled in a non-traditional way: by temperature, power supply voltage (Vdd), some external signals etc. For example, the polymorphic (i.e. multifunctional) NAND/NOR gate operates as NOR in the case that Vdd=1.8V and as NAND in the case that Vdd=3.3V. No conventional design is available with this logic function controlled by Vdd [17].

If some gates of a growing circuit are polymorphic then the process of development can be controlled by the mentioned external signals (in addition to controls derived from expressed genetic information). The objective of this paper is to propose a simple example of a developmental electronic system which consists of polymorphic gates and so which can be influenced by external analogue signals. The proposed research is based on our previous work in which we evolved arbitrarily large sorting networks using an application specific model of development [14, 16]. We evolved a program (constructor) which is able to create a larger instance of a problem (e.g. 5-input sorting network) from a smaller one (e.g. a 3-input sorting network). In general we evolved a constructor which is able to create (n+2)-input solution from the n-input solution. In this paper, we present constructors that are able to work with polymorphic gates (considered as building blocks in growing circuits). Therefore, the growing circuit can specialize its functionality according to the environment which is sensed through polymorphic gates. We assume that arbitrary polymorphic gates exist; however, only bi-functional (polymorphic) are considered in the current version of our program. The results are presented on growing even/odd parity circuits and sorting/median networks.

2. POLYMORPHIC ELECTRONICS

Polymorphic circuits, introduced by Stoica's team at JPL, are in fact multifunctional circuits. The change of their behavior comes from modifications in the characteristics of components (e.g. in the transistor's operation point) involved in the circuit in response to controls such as temperature, power supply voltage, light, etc. [18]. Polymorphic circuits are able to work in several modes of operation corresponding to different operational conditions. Table 1 gives examples of the polymorphic gates reported in literature. Most of them have been designed by means of evolutionary techniques. The mentioned NAND/NOR gate is the most famous example [17]. The circuit consists of 6 transistors and was fabricated in a 0.5-micron CMOS technology. The circuit is stable for $\pm 10\%$ variations of Vdd and for temperatures in the range 20C – 200C.

Table 1: Examples of existing polymorphic gates

Gate	control values	control method	ref.
AND/OR	27/125C	temperature	[19]
AND/OR/XOR	3.3/0.0/1.5V	external voltage	[19]
AND/OR	3.3/0.0V	external voltage	[19]
AND/OR	1.2/3.3V	Vdd	[18]
NAND/NOR	3.3/1.8V	Vdd	[17]

Potential applications are discussed in [18]. Polymorphic electronics should allow engineers to build inherently adaptable digital circuits. By changing the temperature, Vdd or some other conditions a circuit can change its functionality immediately, with no reconfiguration overhead. The potential applications include: special circuits that are able to decrease resolution of digital/analog converters or speed/resolution of a data transmission when a battery voltage decreases, circuits with a hidden/secret function that can be used to ensure security, intelligent sensors and novel solutions for reconfigurable cells and function generators in reconfigurable devices (such as FPGA and CPLD) [18].

The design of polymorphic circuits is considered as main problem because these circuits typically utilize normally unused characteristics of electronic devices and working environment; conventional design techniques are not able to deal with that. A. Thompson has shown that unconstrained evolutionary design is able to produce innovative designs that effectively utilize these characteristics [21]. Sekanina has proposed a method for the evolutionary design of gate-level digital polymorphic circuits in which polymorphic gates are considered as building blocks [15]. For instance, a circuit was evolved operating as 2-bit adder in environment E1. This circuit can also work as 2-bit multiplier in environment E2. A typical feature of polymorphic gate-level circuits is that their topology (i.e. connection of components) is fixed; however, the components can change the functionality.

3. THE PROPOSED APPROACH

All the approaches to development of digital circuits, mentioned in Introduction, can be enriched by using polymorphic gates. In particular, the use of polymorphic gates enables us to implement more complex developmental schemes. Therefore, the growing circuit and environment can interact in more complex ways which, in result, leads to more complex target circuits.

In case that an electronic digital platform contains polymorphic gates then the behavior of a growing circuit (embodied in the platform), in addition to the approaches mentioned in Introduction, can be influenced in two ways:

- Polymorphic gates influence the physical structure of the circuit. For instance, for some environments (e.g. for low temperatures), a functional unit, say X, is created, for other environments (e.g. for high temperatures) functional unit X is not created. Unit X is created because a special mode of polymorphic gate is activated in which a corresponding control digital

Code	Gate	Code	Gate	Code	Gate	Code	Gate	Code	Gate
0	AND/I	4	I/AND	8	OR/XOR	12	XOR/OR	16	NOT/NOT
1	AND/AND	5	OR/I	9	I/OR	13	XOR/XOR	17	I/NOT
2	AND/OR	6	OR/AND	10	XOR/I	14	I/XOR	18	I/I
3	AND/XOR	7	OR/OR	11	XOR/AND	15	NOT/I		

Table 2: List of gates. Some of them are polymorphic. "I" denotes the identity function (buffer).

signal is enabled. Otherwise, the control signal is not enabled and unit X is not created.

- Polymorphic gates do not influence the physical structure of the growing circuit, i.e. circuit topology is independent of the environment. However, because the circuit contains polymorphic gates, its behavior depends on the environment. For instance, the growing circuit operates as adder in low temperatures and as sorter in high temperatures.

This paper deals with the second option.

3.1 Polymorphic Circuits Considered for Development

In our previous research we have used development to evolve arbitrarily large sorting and median networks, parity circuits and adders [16, 14, 22]. In some cases we discovered a new design principle for creating large circuits. Huelsbergen discovered general constructors for parity circuits [6]. In this work we combine our method with polymorphic gates to design arbitrarily large polymorphic circuits, in particular:

- Polymorphic odd/even parity circuits and
- Polymorphic sorting networks (with increasing/decreasing order of the sorted sequences).

3.2 The Developmental Method

We employ genetic algorithm to evolve a sequence of instructions (a program), by means of which an initial simple instance of the problem (an embryo) will *grow* to form a more complex circuit [16]. In our case each instruction of the program consists of three integers (*opcode* $arg1$ $arg2$), where *opcode* represents the operational code of the instruction and $arg1$ and $arg2$ are its arguments. The meaning of arguments depends on the type of the instruction (instructions with no operands are allowed; then $arg1$ and $arg2$ are not interpreted). A *developmental step* is understood as an application of the constructor on a circuit to construct more complex circuit. After its application the number of circuit inputs increases according to the size of the developmental step. The circuit grows from top-left corner to down-right corner.

Figure 1 shows the structure of a basic building block utilized in the developmental system. Each building block consists of two gates – $g1$ and $g2$ represent logic functions performed in the gates. $i1$ and $i2$ denote the input indices. Each building block is then encoded using a 4-tuple $(i1, i2, g1, g2)$. The target circuit is represented as a sequence of building blocks. Table 2 shows the list of gates we utilized. Some of them are standard (e.g. 1 – AND/AND) and some polymorphic (2 – AND/OR). We also included some buffers (e.g. 18 – I/I). The polymorphic gates are bifunctional, i.e. function AND is performed in environment E1 and function OR is

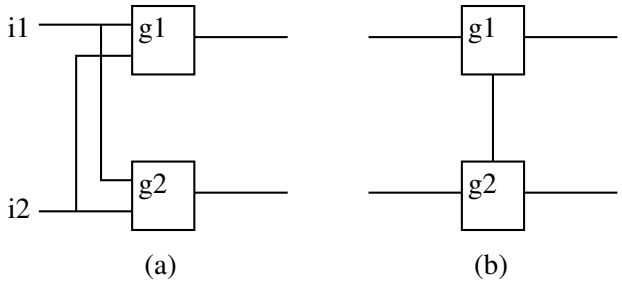

Figure 1: The basic building block: (a) logic structure, (b) symbolic notation. $g1$ **and** $g2$ **represent logic functions performed in the gates.** $i1$ **and** $i2$ **denote the input indices.**

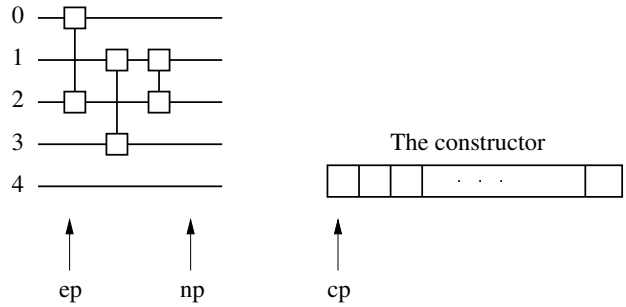

Figure 2: Configuration of development: a growing circuit (left) and constructor which is represented by a chromosome (right).

performed in environment E2 in case of polymorphic gate AND/OR.

A sample configuration of the proposed developmental system is shown in Figure 2. The embryo pointer (*ep*) indicates the building block that is actually processed by the instruction selected through the instruction pointer (*cp*). As the result of application of instruction new gates will be placed on the position denoted by the next-position pointer (*np*). This pointer denotes the first empty position where the circuit will grow to. The instructions of constructor are processed sequentially. The process of construction terminates when either all the instructions of the constructor are executed or the end of embryo is reached. After executing an instruction the pointers *ep*, *cp* and *np* are updated. We are using an "empty" embryo at the beginning of the developmental process, i.e. no particular information is known about the structure of the initial circuit, which is, therefore, considered as a 4-tuple $(0, 0, I/I, I/I)$. Considering that, there must be instructions in the instruction set that are able to set the input signals of the polymorphic gates and their functions (e.g. instruction MODIS and MODFS). These so-called *modify*-instructions only modify the gates denoted by the embryo pointer (*ep*) without copying them and hence the next-position pointer (*np*) remains unchanged after their execution. The instructions which are responsible for growing the circuit are of the types *copy* or *copy-and-modify* (e.g. instructions CPOS or CPMIS). Every instruction is used in two variants (e.g. CPMIS and CPMIN), whose difference lies in updating strategy of the embryo pointer (*ep*) after execution of the instruction. Table 3 lists all the utilized instructions. For example, CPOS copies a pair of gates from position given by *ep* to the position given by *np*; the embryo pointer remains unchanged in this case. The position (indices of inputs) of the newly created or modified gates depends on the position of the gates being processed and the number of inputs of currently constructed circuit (w).

3.3 Experimental Setup

A simple genetic algorithm is utilized to find a constructor whose repeated application on an existing circuit will create a larger circuit. All the experiments were performed with the following settings: standard crossover with the probability 0.55, the probability of mutation 0.04, population size 40, tournament selection mechanism with the base 3, the maximal number of generations 20,000.

The length of the chromosome remains constant during an experiment. We determined this value experimentally in [14]. Note, that all the evolved constructors work only with either even or odd number of inputs.

The objective is to develop arbitrarily large circuits; however, only four developmental steps are considered in the fitness calculation in order to make the time of evolution reasonable. For a single environment the fitness value is determined as $f = f(2) + f(4) + f(6) + f(8)$ for even-input circuits and $f = f(3) + f(5) + f(7) + f(9)$ for odd-input circuits, where $f(i)$ is the number of correctly processed testing sequences by the circuit with i inputs. Therefore, the maximum fitness value that we can reach is $f_{max} = 2^2 + 2^4 + 2^6 + 2^8 = 340$ for even-input circuits and $f_{max} = 2^3 + 2^5 + 2^7 + 2^9 = 680$ for odd-input circuits. For the second environment the fitness value is calculated in the same way. At the end of evolution we have to verify whether the resulting constructors are general, i.e. able to produce arbitrarily large circuits (typically we verify the functionality of circuits up to 28 inputs).

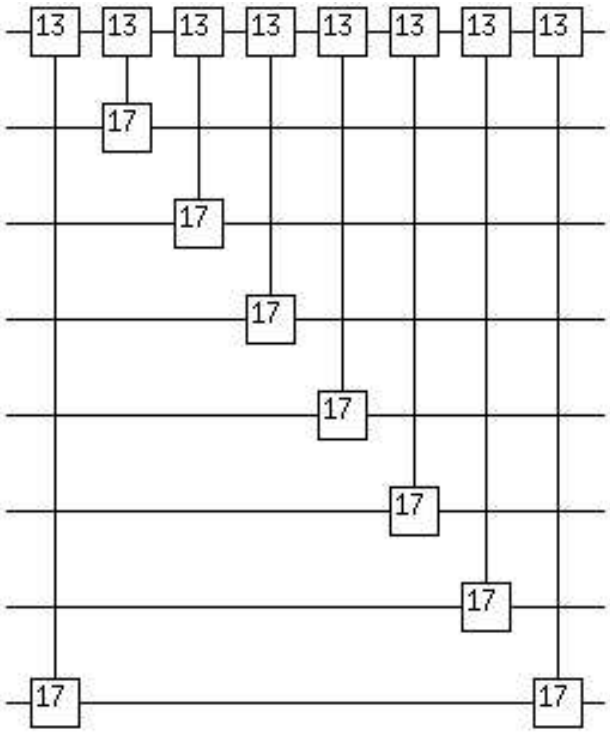

Figure 3: Polymorphic even/odd parity circuit created by means of constructor [MODIS 4 3] [MODFS 13 17] [CPMIS 0 1] [MODIS 0 2] [CPOS 3 1] (a 5-instruction constructor, even number of inputs)

4. THE OBTAINED RESULTS

4.1 Polymorphic Parity Circuits

Some constructors were successfully evolved for the polymorphic even/odd parity circuits. According to the environment the circuits calculate either even or odd parity. The solution is usually based on the conventional XOR gate (no. 13 in Table 2) and a polymorphic NOT switch (no. 17 in Table 2). Its structure is not surprising; however, the structure was designed fully automatically without any supporting domain knowledge (we started with the empty embryo $(0, 0, I/I, I/I)$). Figures 3 and 4 show two polymorphic circuits calculating even/odd parity functions. We have gained 35 general constructors out of 200 independent runs of the evolutionary design process for a five-instruction chromosome. 42 constructors consisting of six instructions were evolved out of 200 independent runs of the evolutionary process from which 36 were recognized as general.

4.2 Median and Sorting Networks

The evolutionary process has succeeded in the design of structurally variable median circuits. These circuit structures, considered as polymorphic circuits, do **not** compute different functions in different environments. However, the way in which the median is calculated depends on the environment. Therefore, the environment determines, through

Op. code	Name	Arg1	Arg2	Meaning
0	CPOS	–	–	copy the pair of gates from ep to np; $cp = cp + 1, np = np + 1$
1	CPON	–	–	copy the pair of gates from ep to np; $cp = cp + 1, np = np + 1, ep = ep + 1$
2	CPNS	p	–	copy $w - p$ pairs of gates; $cp = cp + 1, np = np + w - p$
3	CPNN	p	–	copy $w - p$ pairs of gates; $cp = cp + 1, np = np + w - p, ep = ep + w - p$
4	CPMIS	p	q	copy the pair of gates from ep to np and do $i_1 = (i_1 + p)\ mod\ w, i_2 = (i_2 + q)\ mod\ w, cp = cp + 1, np = np + 1$
5	CPMIN	p	q	copy the pair of gates from ep to np and do $i_1 = (i_1 + p)\ mod\ w, i_2 = (i_2 + q)\ mod\ w, cp = cp + 1, np = np + 1, ep = ep + 1$
6	CPMFS	p	q	copy the gates from ep to np and do $f_1 = p, f_2 = q, cp = cp + 1, np = np + 1$
7	CPMFN	p	q	copy the gates from ep to np and do $f_1 = p, f_2 = q, cp = cp + 1, np = np + 1, ep = ep + 1$
8	MODIS	p	q	modify inputs of the gates at ep as follows: $i_1 = (i_1 + p)\ mod\ w, i_2 = (i_2 + q)\ mod\ w, cp = cp + 1$
9	MODIN	p	q	modify inputs of the gates at ep as follows: $i_1 = (i_1 + p)\ mod\ w, i_2 = (i_2 + q)\ mod\ w, cp = cp + 1, ep = ep + 1$
10	MODFS	p	q	modify functions of the gates at ep as follows: $f_1 = p, f_2 = q; cp = cp + 1$
11	MODFN	p	q	modify functions of the gates at ep as follows: $f_1 = p, f_2 = q; cp = cp + 1, ep = ep + 1$
12	NOP	–	–	an empty instruction: $cp = cp + 1$

Table 3: The instruction set. p and q represent the arguments of the instruction; i_1 and i_2 denote the indices of inputs of the polymorphic gates; f_1 and f_2 are functions of the gates and w is the number of inputs of the circuit being created.

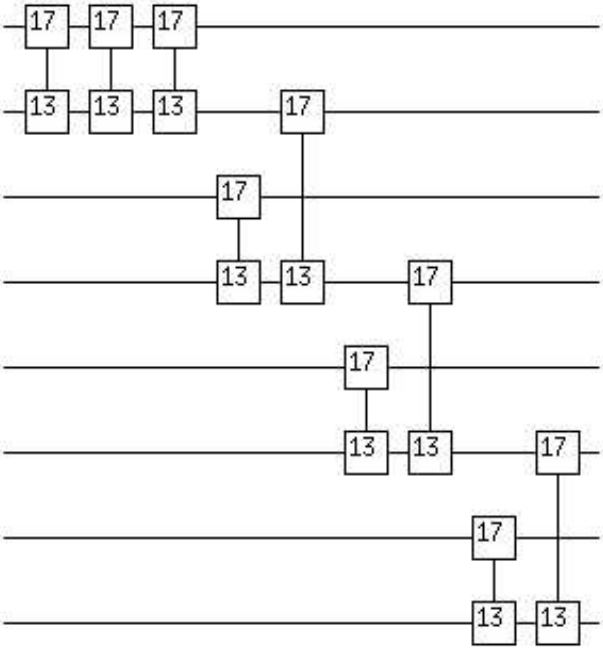

Figure 4: Polymorphic even/odd parity circuit created by means of constructor [MODFS 17 13] [MODIS 2 3] [CPMIS 2 2] [CPMIS 2 0] [CPMIN 1 2] [MODIN 0 3] (a 6-instruction constructor, even number of inputs)

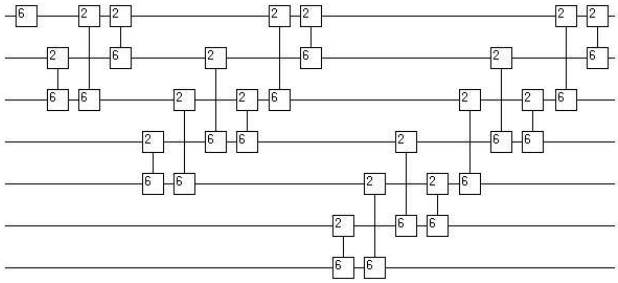

Figure 5: Polymorphic median and sorting network created by means of the constructor [MODFS 2 6] [CPMIS 2 2] [CPMIS 1 2] [CPMIS 3 2] [CPMIS 0 1] [CPMIN 1 1] [CPNN 0 2] (a 7-instruction constructor, odd number of inputs)

polymorphic gates, physical implementation of median circuit.

In addition, the mentioned polymorphic circuits can also work as sorting networks. It is interesting that they sort the input sequences in increasing or decreasing order according to the environment. Figure 5 shows the polymorphic median and sorting network circuit built by means of a seven-instruction constructor. However, this constructor is not general. Median networks can be constructed only up to 13 inputs and sorting networks only up to 11 inputs. However, these circuits exhibit better properties (the number of comparators) than the conventionally constructed circuits (e.g. bubble-sort network) [8]. The evolutionary process has succeeded 68 times in 200 independent runs, from which eight constructors are able to build fully functional median circuits up to 13 inputs.

Figures 6 and 7 show polymorphic median and sorting

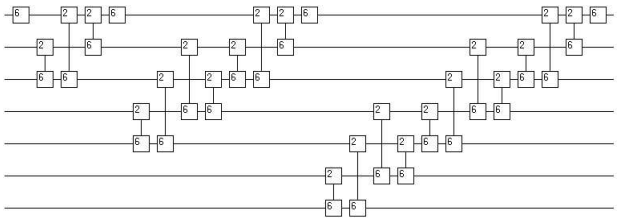

Figure 6: Polymorphic median and sorting network created by means of the constructor [CPMIS 2 2] [MODFS 2 6] [CPMIS 1 2] [CPMIS 3 2] [CPMIS 0 1] [CPMIS 1 1] [CPNN 2 4] [CPNN 4 4] (a 8-instruction general constructor, odd number of inputs)

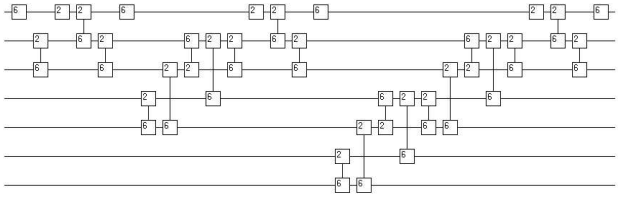

Figure 7: Polymorphic median and sorting network created by means of the constructor [MODFS 2 6] [CPMIS 2 2] [CPMIS 1 2] [CPMFS 6 2] [CPMIS 0 1] [CPNN 3 0] [CPMIS 4 2] [CPNN 1 2] (a 8-instruction general constructor, odd number of inputs)

networks created by means of a seven- and eight-instruction constructor. These constructors are general. We gained 45 constructors out of 200 independent runs of the evolutionary process, from which 5 constructors are general.

The evolved circuits use gate 2 (AND/OR) and 6 (OR/AND) which means that they sort the input sequences in increasing order in the first environment and in decreasing order in the second environment. As the median value is taken from the middle output, it does not depend on signals coming from the environment.

5. DISCUSSION

In case of sorting networks, evolution has discovered that by exchanging AND–OR gates for OR–AND gates, the ordering of sorted sequence can be changed. There is no innovation; human designer would construct the circuit in the same way. Although the evolution could use many types of gates, it has utilized the same gates as a human designer uses. The implementation of AND as well as OR gate costs 6 transistors in the standard CMOS technology. Surprisingly, the cost of polymorphic AND/OR gate controlled by temperature is also 6 transistors [19]. If one were able to build OR/AND gate with the same cost, the resulting polymorphic sorting network would consist of the same number of transistors as the original one whose behavior cannot be changed!

Despite we put effort into evolution of other types of large polymorphic circuits (such as adder/sorting network and parity/Boolean symmetry circuits etc.) we have not obtained any functional result yet. The explanation could be as follows: The number of correct suitable topologies which perform the required behavior is very limited with the proposed encoding. Therefore, the probability is very low that a single topology can represent two different behaviors (e.g. n-bit adder and n-bit multiplier) in two different environments.

Of course, because of the utilized representation, it is always possible to manually merge two different circuits into a single working polymorphic circuit. The method is as follows: Construct the resulting circuit from left to right. If the first circuit requires logic function L_1, create polymorphic gate L_1/I. If the second circuit requires logic function L_2, create polymorphic gate I/L_2. Then the resulting circuit (consisting of polymorphic gates) will perform the first function in the first environment and the second function in the second environment. However, we are not interested in this type of solution, since there is no innovation visible.

In real biological systems as well as in some artificial developmental systems (e.g. in [2]) the interplay between a growing solution and its environment is very complex. In our system the interplay practically does not exist. We are planning to develop more complex models of development and combine them with polymorphic gates to create innovative arbitrarily large polymorphic circuits.

6. CONCLUSIONS

We evolved programs (constructors) that are able to create arbitrarily large polymorphic even/odd parity circuits and polymorphic sorting networks. According to control signals (environment) the evolved circuits perform one of two possible functions in every step of development. Polymorphic gates were considered as building blocks. The work was performed using a simple circuit simulator. Future research will be oriented to designing more complex models of development combined with polymorphic gates to create more complicated arbitrarily large polymorphic circuits.

7. ACKNOWLEDGMENTS

The research was performed with the support of the Grant Agency of the Czech Republic under No. 102/04/0737 *Modern Methods of Digital Systems Design* and No. 102/03/P004 *Evolvable Hardware Based Application Design Methods*. M. Bidlo was partially supported from J. Hlavka foundation.

8. REFERENCES

[1] B. Alberts et al. Essential Cell Biology – An Introduction to the Molecular Biology of the Cell. Garland Publishing, New York, 1998

[2] T. G. W. Gordon. Exploring models of development for evolutionary circuit design. In *2003 Congress on Evolutionary Computation*, pp. 2050–2057. IEEE Press, 2003.

[3] T. G. W. Gordon and P. J. Bentley. Towards development in evolvable hardware. In *Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware*, pp. 241–250, Washington D.C., US, 2002. IEEE Computer Society Press.

[4] H. de Garis et al. Atr's artificial brain (Cam-Brain) project: A sample of what individual "codi-1 bit" model evolved neural net modules can do with digital and analog i/o. In *Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware*, pp. 102–110. IEEE Computer Society Press, 1999.

[5] P. Haddow, G. Tufte, and P. van Remortel. Shrinking the genotype: L-systems for ehw? In *Proc. of the 4th International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, vol. 2210*, pp. 128–139. Springer–Verlag, 2001.

[6] L. Huelsbergen. Finding general solutions to the parity problem by evolving machine-language representations. In *Proc. of the Genetic Programming 1998 Conference*, pages 158–166, San Francisco, CA, 1998. Morgan Kaufmann.

[7] J. R. Koza et al. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, San Francisco, 1999.

[8] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.

[9] S. Kumar. Investigating computational models of development for the construction of shape and form. PhD thesis. Department of Computer Science, University College London, 2004.

[10] N. Macias. The PIG Paradigm: The Design and Use of a Massively Parallel Fine Grained Self-Reconfigurable Infinitely Scalable Architecture. In EH'99: *Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware*, ed by Stoica, A. et al., Pasadena, CA, USA, 1999 (IEEE Computer Society, Los Alamitos 1999) pp. 175–180

[11] D. Mange et al. Towards robust integrated circuits: The embryonics approach. Proc. of IEEE. Vol. 88, No. 4, 2000, pp. 516–541

[12] J. F. Miller. Evolving developmental programs for adaptation, morphogenesis, and self-repair. In *Proc. of the Seventh European Conference on Artificial Life, Lecture Notes in Computer Science, vol 2801*, pp. 256–265, Berlin Heidelberg New York, 2003. Springer.

[13] J. F. Miller and P. Thomson. A developmental method for growing graphs and circuits. In *Proc. of the 5th Conf. on Evolvable Systems: From Biology to Hardware (ICES 2003), Lecture Notes in Computer Science, vol. 2606*, pp. 93–104, Berlin, DE, 2003. Springer–Verlag.

[14] L. Sekanina. Evolving Constructors for Infinitely Growing Sorting Networks and Medians. In Proc. of the Conference on Current Trends in Theory and Practice of Computer Science SOFSEM 2004. LNCS 2932, Springer Verlag, 2004, pp. 314–323

[15] L. Sekanina. Evolutionary Design of Gate-Level Polymorphic Digital Circuits. In Applications of Evolutionary Computing, Lausanne, LNCS 3449, Springer Verlag, 2005, pp. 185-194

[16] L. Sekanina and M. Bidlo. Evolutionary Design of Arbitrarily Large Sorting Networks Using Development. Genetic Programming and Evolvable Machines. Vol. 6, 2005 (to appear)

[17] A. Stoica et al. Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration. IEE Proc.-Comp. Digit. Tech. Vol. 151, No. 4, 2004, pp. 295–300

[18] A. Stoica et al. On Polymorphic Circuits and Their Design Using Evolutionary Algorithms. In Proc. of IASTED International Conference on Applied Informatics (AI2002), Innsbruck, Austria 2002

[19] A. Stoica and R. Zebulum and D. Keymeulen. Polymorphic Electronics. In Proc. of International Conference on Evolvable Systems: From Biology to Hardware, LNCS 2210, Springer Verlag, 2001, pp. 291–302

[20] G. Tempesti et al. Ontogenetic development and fault tolerance in the poetic tissue. In *Proc. of the 5th Conf. on Evolvable Systems: From Biology to Hardware (ICES 2003), Lecture Notes in Computer Science, vol. 2606*, pp. 141–152, Berlin, DE, 2003. Springer–Verlag.

[21] A. Thompson and P. Layzell and R. Zebulum. Explorations in design space: Unconventional electronics design through artificial evolution. IEEE Trans. on Evolutionary Computation. Vol. 3, No. 3, 1999, pp. 167-196

[22] M. Bidlo. A Developmental Method for Construction of Arbitrarily Large Sorting Networks and Adders. Technical report, Department of Computer Systems, Faculty of Information Technology, Brno University of Technology, CZ, 2005